



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
<b>Bibliografia</b>	<b>19</b>

# CAPITOLO 1

---

## Introduzione

---

Il presente documento ha lo scopo di illustrare passo-passo tutte le attività svolte durante il progetto del corso di "*Penetration Testing and Ethical Hacking*". Per lo svolgimento dello stesso è stato necessario scegliere un asset da analizzare e, dunque, è stata scelta una macchina virtuale vulnerabile by-design identificata con il nome **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 scansionare 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**);
- **Maintaining 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 [5] 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:

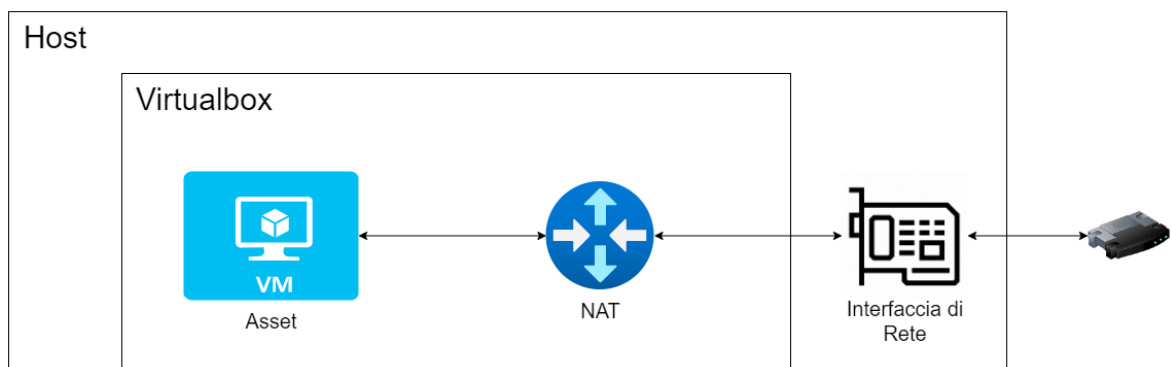
1. Aprire il pannello degli strumenti di VirtualBox;
2. Selezionare il sotto-menù rete;
3. All'interno della pagina, selezionare il pannello "Reti con NAT";
4. Cliccare il pulsante per la creazione di una nuova rete ed impostare i parametri desiderati.

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:

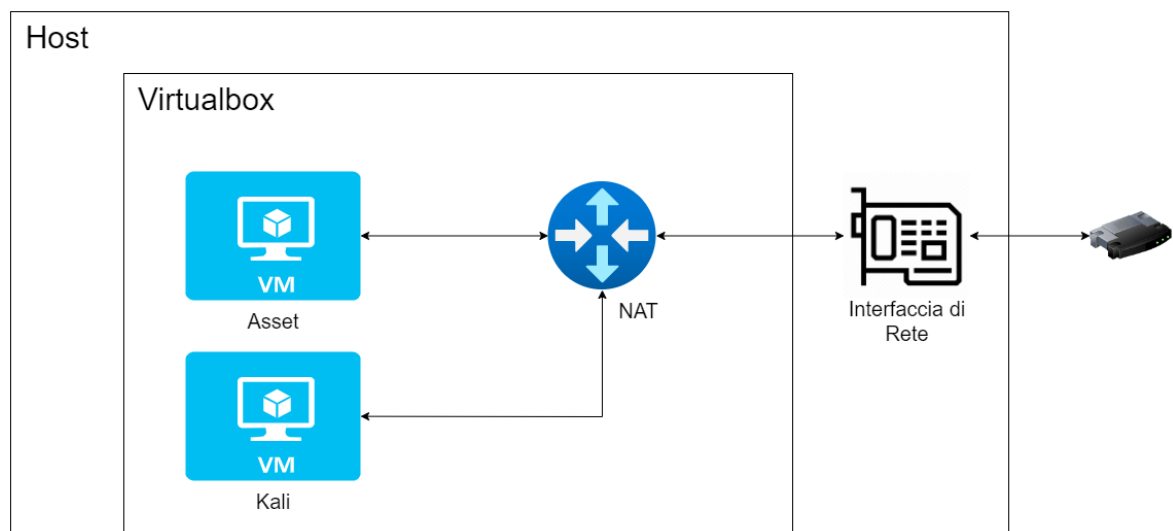


**Figura 1.1:** Infrastruttura di rete

## 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

### 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
    inet6 fe80::4caf:e98c:243e:394 prefixlen 64 scopeid 0x20<link>
    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
```

Figura 2.1: Esecuzione di `ifconfig` su Kali

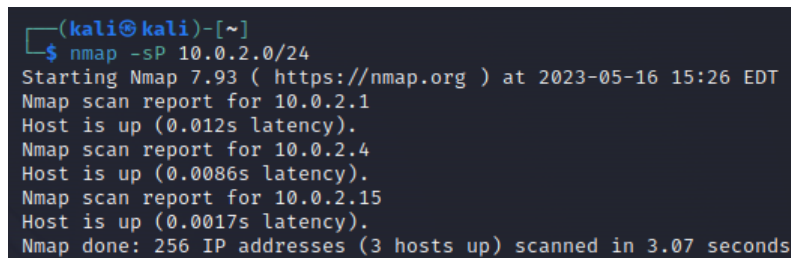
Come si può notare dall'output, l'indirizzo di Kali è *10.0.2.15* e, adesso che si ha quest'informazione, si può cominciare con la scansione della rete.

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

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

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



```
(kali㉿kali)-[~]  
$ nmap -sP 10.0.2.0/24  
Starting Nmap 7.93 ( 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*

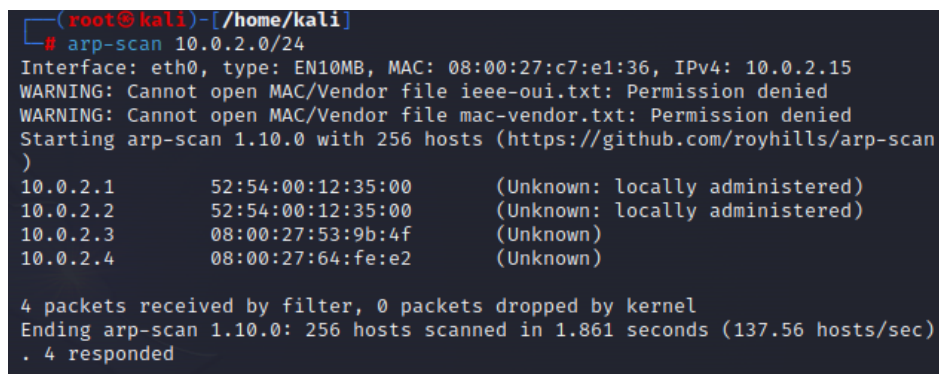
Il primo dato che dovrebbe risaltare è che il numero degli host attivi sulla rete è 3, e non 2 come in realtà ci si aspettava dalla configurazione realizzata. Tuttavia, come specificato a lezione e approfondito anche nella documentazione di *VirtualBox*, all'interno della rete saranno presenti uno o più host "fittizi" che sono necessari allo stesso *VirtualBox* per realizzare la stessa rete con NAT [8]. 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 perché 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 [1]. 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)~[/home/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 (incluso 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* [3]). 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`

```

Starting Nping 0.7.93 ( https://nmap.org/nping ) at 2023-05-17 10:21 EDT
SENT (0.0400s) ICMP [10.0.2.15 > 10.0.2.0 Echo request (type=8/code=0) id=26834 seq=1] IP [t
ttl=64 id=43400 iplen=28 ]
SENT (1.1149s) ICMP [10.0.2.15 > 10.0.2.1 Echo request (type=8/code=0) id=4012 seq=1] IP [t
ttl=64 id=43400 iplen=28 ]
RCVD (1.1160s) ICMP [10.0.2.1 > 10.0.2.15 Echo reply (type=0/code=0) id=4012 seq=1] IP [ttl
=255 id=43400 iplen=28 ]
SENT (2.1608s) ICMP [10.0.2.15 > 10.0.2.2 Echo request (type=8/code=0) id=49804 seq=1] IP [
ttl=64 id=43400 iplen=28 ]
RCVD (2.1639s) ICMP [10.0.2.2 > 10.0.2.15 Echo reply (type=0/code=0) id=49804 seq=1] IP [tt
l=128 id=26 iplen=28 ]
SENT (3.1747s) ICMP [10.0.2.15 > 10.0.2.3 Echo request (type=8/code=0) id=34899 seq=1] IP [
ttl=64 id=43400 iplen=28 ]
  
```

(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`

```

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

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 [2]. 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

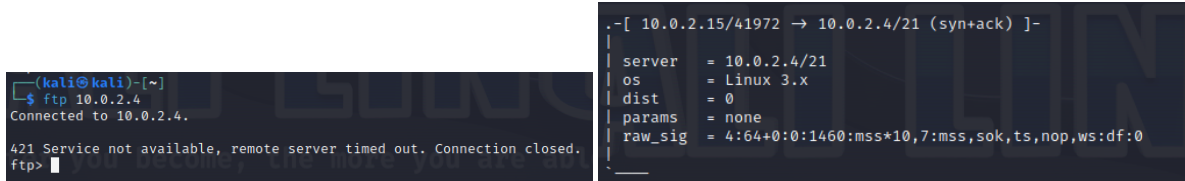
Si può notare che la versione del kernel identificata da `nmap` risulta essere la `2.6.x/3.x`, in particolare potrebbe trattarsi della versione `2.6.32` o `3.13`. Oltre all'**OS fingerprint**, è possibile notare che `nmap` ha eseguito anche una scansione delle porte attive sull'host target. Ovviamente si è limitato solo alle mille più frequenti [2] (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** [10]. 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 estrapolare 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*),

(a) Generazione di traffico *ftp* legittimo(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 [2].

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 **chiusa**. 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 perchè 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 [2]. 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 pacchetto *ICMP* allora la porta è filtrata [2]). 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`



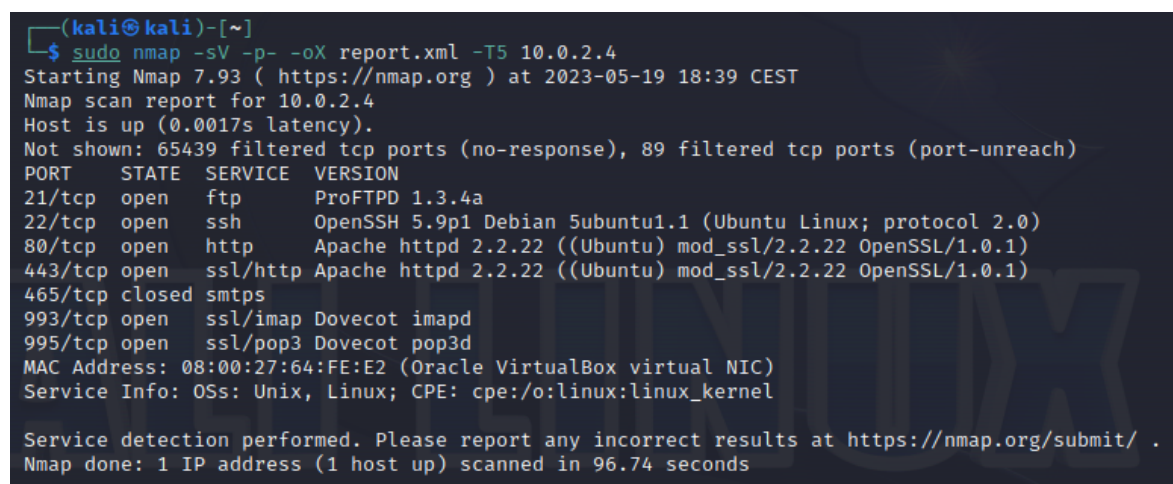
Come è evidente dal risultato ottenuto, non si può che confermare la presenza di un meccanismo di filtraggio sull'asset.

### 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 [2]. 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 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 [2]. 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 (0.0011s 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 AUTH=
01 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 [9]. 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 [4].

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 recieved 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 recieved 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 recieved 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**. 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** [7]. 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 [6]. 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 [9].

### Osservazioni finali

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

---

## Bibliografia

---

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

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