



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

Indice	i
1 Introduzione	1
1.1 Ambiente utilizzato	2
1.2 Strumenti utilizzati	3
2 Pre-Exploitation	4
2.1 Target Scoping	4
2.2 Information Gathering	5
2.3 Target Discovery	6
2.3.1 Esecuzione di ifconfig	6
2.3.2 Scansione con nmap	7
2.3.3 Scansione con arp-scan	8
2.3.4 Ulteriore scansione con nping	8
2.3.5 OS Fingerprinting con nmap	9
2.3.6 OS Fingerprint passivo con p0f	10
2.4 Target Enumeration	11

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;
- (CONTINUA)

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* 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:

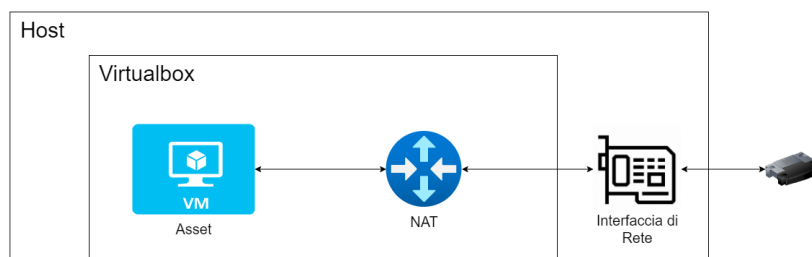


Figura 1.1: Infrastruttura di rete

1.2 Strumenti utilizzati

Per proseguire con l'analisi dell'asset, è necessario ottenere strumenti appositi che permettano 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 distribuito 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. Otteniamo così il seguente schema:

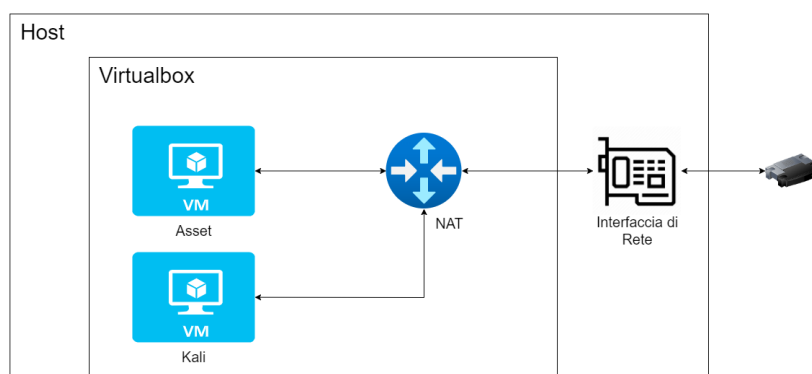


Figura 1.2: Infrastruttura di rete con Kali

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, possiamo 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 avviamo la macchina non possiamo fare nulla tramite *interazione diretta* in quanto **non abbiamo 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 conosciamo apriori l'indirizzo della macchina (sappiamo solo 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 abbiamo accesso alla macchina**;
- Informazioni riguardo il **sistema operativo**. Altra informazione molto utile in quanto adesso sappiamo che l'asset è un sistema Linux e questo ci permetterà di risparmiare tempo in fasi avanzate perché possiamo restringere il campo delle scansioni solo a

sistemi Linux, escludendo tutti gli altri. Tuttavia, non sappiamo ancora la versione precisa del kernel e quindi dobbiamo ricavarla 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 aspettiamo di trovare solo la macchina **De-ICE S1.140** e la macchina **Kali** per come abbiamo impostato l'ambiente.

2.3.1 Esecuzione di `ifconfig`

Prima di cominciare con la scansione effettiva della rete, dobbiamo 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, otteniamo 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

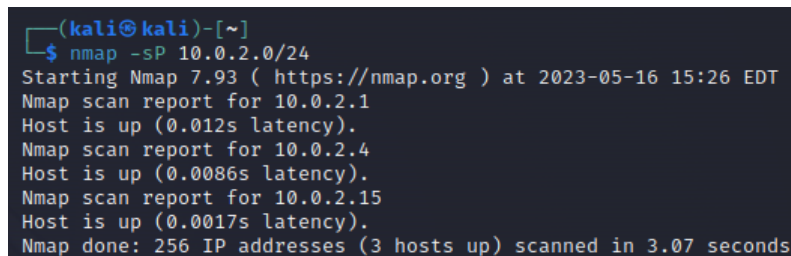
Come possiamo notare dall'output, l'indirizzo di Kali è *10.0.2.15* e, adesso che lo conosciamo, possiamo cominciare con la scansione della rete.

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*) 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, possiamo 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*

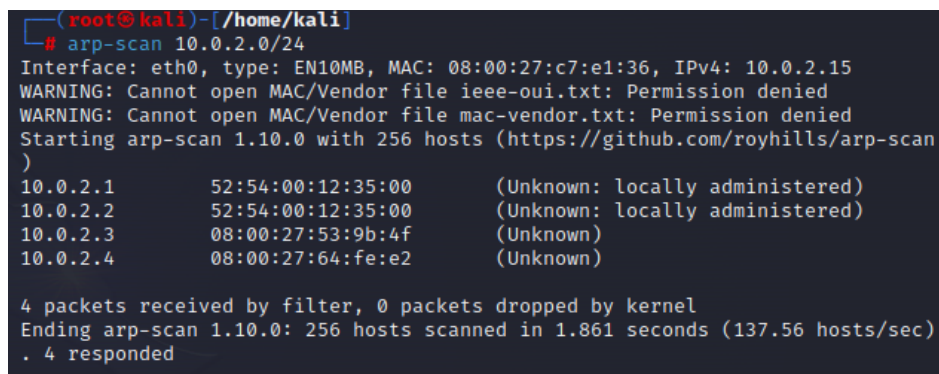
Il primo dato che dovrebbe risaltare è che il numero degli host attivi sulla rete è 3, e non 2 come in realtà ci si aspettava dalla configurazione realizzata. Tuttavia, come specificato a lezione e approfondito anche nella documentazione di *VirtualBox*, all'interno della rete saranno presenti uno o più host "fittizi" che sono necessari allo stesso *VirtualBox* per realizzare la stessa rete con NAT. 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 lo conosciamo dal comando lanciato prima e sappiamo per certo che è proprio l'indirizzo della macchina **Kali**.

2.3.3 Scansione con arp-scan

Grazie ad `nmap` conosciamo 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. 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 possiamo notare subito un'altra anomalia. Con `nmap` sono stati rilevati 3 host invece di 2, mentre adesso con `arp-scan` ne abbiamo rilevati persino 5 (incluso anche la macchina **Kali** nel conteggio). Osservando con attenzione il risultato, possiamo 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 continuiamo a seguire la supposizione che `10.0.2.4` sia l'asset, allora possiamo 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, siamo effettivamente sicuri di aver scovato tutti gli host sia "reali" che "fittizi"? Per essere abbastanza sicuri di averli trovati 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 non fa altro che richiamare il comando `ping` su tutti gli host forniti in input (quindi ancora una *ping scan*). 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 otteniamo 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 [
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, possiamo 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 abbiamo terminato di identificare gli host attivi sulla rete e, per questo motivo, possiamo passare al passo successivo. Durante la fase di *Information Gathering* siamo riusciti a 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. 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

Notiamo 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**, possiamo notare che nmap ha eseguito anche una scansione delle porte attive sull'host target. Ovviamente si è limitato solo alle più frequenti (e per questo sarà necessaria una scansione più approfondita nella fase successiva), tuttavia, possiamo notare che ha delle porte aperte che 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 estrapolando le cosiddette **Informazioni Caratterizzanti**. Quindi, tutto quello che dobbiamo 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

```
(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*

tecniche passive non hanno la stessa accuratezza dei metodi attivi (per ovvi motivi), 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