《软件安全》实验报告

姓名: 陆皓喆 学号: 2211044 班级: 信息安全

实验名称:

格式化字符串漏洞

实验要求:

以第四章示例4-7代码,完成任意地址的数据获取,观察Release模式和Debug模式的差异,并进行总结。

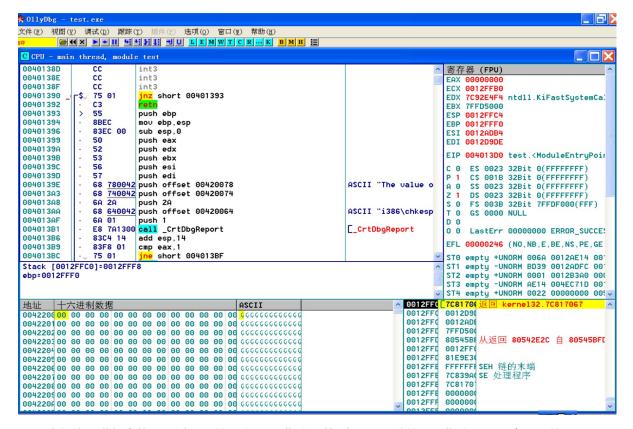
实验代码如下所示:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
    char str[200];
    fgets(str,200,stdin);
    printf(str);
    return 0;
}
```

实验过程:

一.Debug模式

首先,我们在vc6中输入源代码,进行debug模式的调试,并且将exe文件导出,并导入到ollydbg中进行调试。



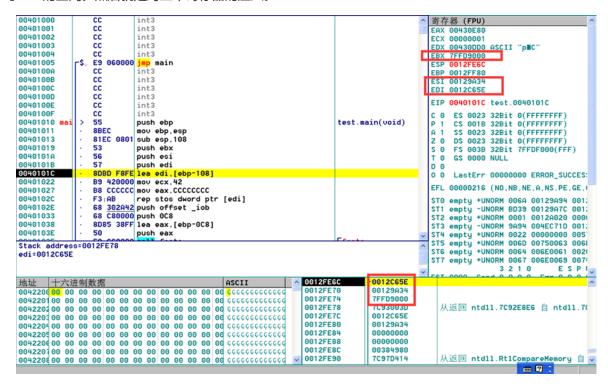
我们往下进行查找,通过明显的三个 mov 指令,找到main函数的调用指令,即图中所示的 call 00401005。

```
00401476
                 H3 <u>EC3B42</u> mov aword ptr [_acmdin],eax
0040147B
                 E8 304A00 call __crtGetEnvironmentStringsA
                                                                                  __crtGetEnvironme
                 A3 B43542 mov dword ptr [_aenuptr],eax
00401480
                 E8 164500 call _setargu
E8 C14300 call _setenup
E8 CC3D00 call _cinit
00401485
                                                                                   _setargv
00401480
                                                                                   _setenup
                                                                                  __cinit
0040148F
                 8B0D 0836 mov ecx,dword ptr [_environ]
890D 0C36 mov dword ptr [_initenv],ecx
00401494
0040149A
004014A0
                 8B15 0836 mov edx, dword ptr [_environ]
004014A6
                 52
                             push edx
004014A7
                 Al 003642 mov eax, dword ptr [__argv]
004014AC
                 50
                             push eax
004014AD
                 8B0D FC35 mov ecx, dword ptr [__argc]
004014B3
                             push ecx
                 51
004014B4
                             call 00401005
                 E8
004014B9
                 83C4 OC
                             add esp, OC
004014BC
                 8945 E4
                             mov dword ptr [ebp-1C],eax
004014BF
                 8B55 E4
                            mo∪ edx,dword ptr [ebp-1C]
```

我们按下 F7 进行main函数的调试,得到以下的汇编语言代码:

```
00401004
                          int3
00401005
               E9 060000
                          imp main
00401000
               CC
                          int3
0040100B
                CC
                          int3
0040100C
                CC
                          int3
0040100D
               CC
                          int3
0040100F
               CC
                          int3
0040100F
               CC
                          int3
00401010
               55
                          push ebp
                                                                          test.main(void)
00401011
               8BEC
                          mo∪ ebp,esp
00401013
                81EC 0801 sub esp,108
00401019
                53
                          push ebx
0040101A
                56
                          push esi
0040101B
                57
                          push edi
               8DBD F8FE lea edi,[ebp-108]
0040101C
00401022
               B9 420000 mov ecx,42
00401027
                B8 CCCCCC mov eax, CCCCCCCC
0040102C
               F3:AB
                          rep stos dword ptr [edi]
0040102E
                68 302A42 push offset _iob
00401033
                68 C80000 push 0C8
00401038
               8D85 38FF lea eax, [ebp-0C8]
                          push eax
0040103E
                50
               E8 CC0000 call fgets
0040103F
                                                                         [fgets
00401044
                83C4 OC
                          add esp,0C
00401047
               8D8D 38FF lea ecx, [ebp-0C8]
```

首先,前六句汇编的意思是,将 ebp 寄存器压入栈中,然后完成栈顶与栈底的变换,注意到此处赋了108的空间。然后就是对三个寄存器的压入。



由上图,我们可以发现,将三个寄存器都压入到了栈的顶端,分别是 EDI, ESI, EBX

接下来的四句话,就是简单的对开辟出的空间全部赋值为 ccccccch。我们在此设置断点进行观察,发现确实整个栈都被赋值成了 ccccccch。

		ST6 empty		
■ 0012FE64	FFFFFFF			
0012FE68	7C947764		从返回	n
0012FE6C	_0012C65E			
0012FE70	00129A34			
0012FE74	7FFD9000			
0012FE78	ccccccc			
0012FE7C	ccccccc			
0012FE80	ccccccc			
0012FE84	ccccccc			
0012FE88	ccccccc			
0012FE8C	ccccccc			
0012FE90	ccccccc			
<u> 10012FF94</u>	l recerer			

接下来,我们将三个参数入栈,调用函数 fgets 函数,lea eax,[ebp-0C8] 就是我们的 str 的起始地址,在 str 的起始地址后压入 eax 寄存器。

分析完成后,我们进行内容的输入,输入AAAA%%%%%x,观察栈区,确认已经将其压入栈中。

```
00401044 从返回 test.fgets 自 t

-0012FEB8 ASCII "AAAA%x%x%x%x*x***

000000BB
00422A30 offset test._iob
00000000
00000000
7FFDE000
```

接下来执行 print 函数,其中 add esp,0c 指令是清除上一个 fgets 函数所使用的栈帧,然后把 str 字符串的起始地址存到 ecx 中,进行入栈。

AAAA007ffd8000ccccccc

接下来,我们分析一下为什么会出现这个结果。

```
| COUNTY |
```

上图是我们栈中的内容,我们在输入AAAA时,程序正常输出四个A;但是当程序遇到后面的%x时,自动读取后面的地址作为输出,因此四个%x对应的输出分别是,00000000(0)、00000000(0)、7FFD8000、CCCCCCCC。所以,对应的程序输出为:AAAA007ffd8000cccccccc。这样我们就完成了在debug模式下的调试运行。

二、Release模式

下面我们开始release模式的分析。在VC6中将执行模式切换为release模型,并继续将生成的可执行 文件导入到ollydbg中,同样通过三个 push 操作来定位到我们的 main 函数。

```
0 mov eax,dword ptr [406900]
0 mov dword ptr [406904],eax
push eax
8 push dword ptr [4068F8]
8 push dword ptr [4068F4]
F call 00401000
add esp,0C

ASCII "�A"

ASCII "�A"

Arg3 => [406900]
Arg2 = 410EA0
Arg1 = 1
```

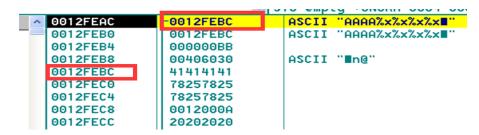
按 F7 进入主函数,我们发现,在release模式下,栈的分配显得比之前要紧凑了一些,每个变量之间都靠在一起。接下来我们对代码进行逐条的分析。

```
sub esp,0C8
lea eax,[esp]
push offset 00406030
push 0C8
push eax
call 00401061
```

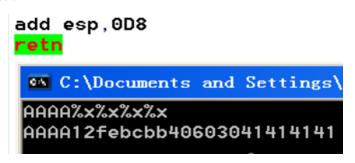
sub esp, 0C8 是直接为 str 字符串分配20字节的内存空间,而后通过三个 push 操作将 fgets 函数的三个参数入栈,进行调用。这里,我们可以看到此模式下与debug模式的一个区别:无过多的栈内存空间分配,无寄存器的旧值保存。

然后,我们执行 fgets 函数,输入与debug模式相同的 AAAA%x%x%x%x,我们在输入之后观察栈的情况,发现 fgets 函数的地址与 str 的地址是紧挨着的,这进一步说明了该模式下,栈分配与debug模式的不同。

然后,我们执行 print 函数,第一行为 print f的参数即 str 字符串的地址 0012FEBC,然后执行的时候先输出 AAAA,接着四个 %x 格式化操作符读取后四行的内容作为四个参数进行输出,因此输出结果为 AAAA12FEBCBB40603041414141。

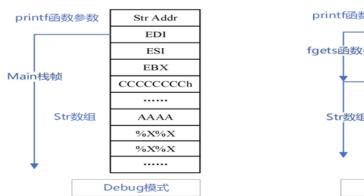


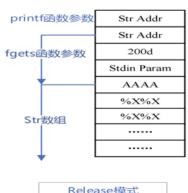
以下是我的输出结果。



在程序函数调用完之后,还有两句汇编语言代码,含义是,将 eax 寄存器置零,然后将 esp 寄存器的位置恢复到原始的位置,完成栈帧改的恢复。

三、Debug模式与Release模式的差异





在完成两个模式下的分析后,我发现这两个模式有以下的不同之处:

- 1. Debug模式 main 函数一开始 sub esp 会分配更大的栈空间, char str[200] 是从靠近 EBP 的地址分配空间, 因此在DEBUG模式下如果要读到 str 的地址, 需要很多的格式化字符。
- 2. Release模式下,main 函数不执行严格的栈帧转换(即 push ebp,mov ebp,esp),也不对栈空间进行统一初始化(即 rep stos 指令),也不通过 push 保存寄存器原来的值。会在程序的最后处完成栈帧的恢复。

心得体会:

- 1. 通过本次实验, 我学会了对格式化字符溢出的使用, 对这个漏洞有了更深的了解。
- 2. 我通过理论课的学习,发现一些看似没有问题的程序中竟然存在这么多漏洞,比如说SQL注入就是一个很严重的漏洞,轻则进入系统,重则更改账户中的内容,而破解方式也很简单易懂,这说明了在一些常用的代码中,还是存在不少问题,需要我们去发现与解决。在本次实验中,我通过对程序的两种调试方法,了解了两种模式下的具体栈差异,了解了具体的利用格式化字符去攻击代码的方式。
- 3. 不仅仅是 %x 符号可以攻击程序,理论课上讲的许多 % 型字符都可以造成一些的攻击性,这告诉我们,在编写代码的时候,不能只考虑代码段正确性,还必须要考虑代码的安全性,目前我了解到的就是关于栈溢出与堆溢出两种常见的攻击方式,我们需要防范,比如说使用一些输入限制,或者在程序堆或者栈异常时直接中断调试,这样就可以避免一些问题。