

《软件安全》实验报告

姓名：禹相祐

学号：2312900

班级：计科

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：

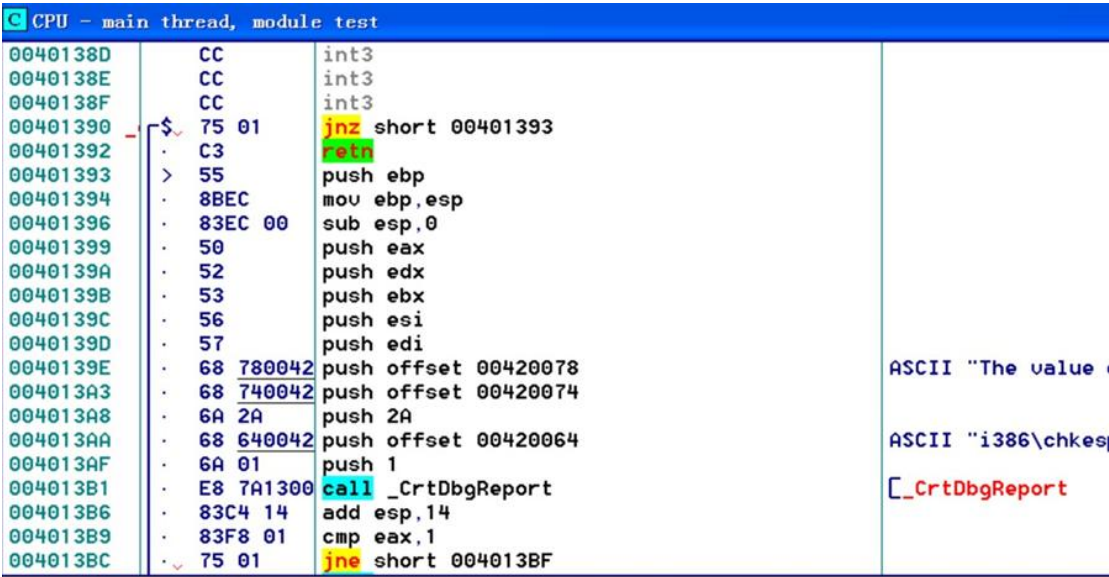
```
#include <stdio.h>

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

1. 首先将该段代码输入进 VC6.0, 进行 debug 的调试，并生成可执行文件 exe，再将该 exe 拖入进 ollydbg 进行调试，如图：



2. 我们进行查找,直至找到三个mov 指令,再往下找到 call 语句:call 00401005, 如图:

00401485	· E8 164500	call _setargv	[_setargv
0040148A	· E8 C14300	call _setenvp	[_setenvp
0040148F	· E8 CC3D00	call _cinit	[_cinit
00401494	· 8B0D 0836	mov ecx,dword ptr [_environ]	
0040149A	· 890D 0C36	mov dword ptr [__initenv],ecx	
004014A0	· 8B15 0836	mov edx,dword ptr [_environ]	
004014A6	· 52	push edx	
004014A7	· A1 003642	mov eax,dword ptr [__argv]	
004014AC	· 50	push eax	
004014AD	· 8B0D FC35	mov ecx,dword ptr [__argc]	
004014B3	· 51	push ecx	
004014B4	· E8 4CFBFF	call 00401005	[main

然后再按下 F7 进入 main 函数, 如图:

00401005	· E9 060000	jmp main	
0040100A	· CC	int3	
0040100B	· CC	int3	
0040100C	· CC	int3	
0040100D	· CC	int3	
0040100E	· CC	int3	
0040100F	· CC	int3	
00401010	· 55	push ebp	test.main(void)
00401011	· 8BEC	mov ebp,esp	
00401013	· 81EC 0801	sub esp,108	
00401019	· 53	push ebx	
0040101A	· 56	push esi	
0040101B	· 57	push edi	
0040101C	· 8DBD F8FE	lea edi,[ebp-108]	
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]	
0040103E	· 50	push eax	
0040103F	· E8 CC0000	call fgets	[fgets
00401044	· 83C4 0C	add esp,0C	
00401047	· 8D8D 38FF	lea ecx,[ebp-0C8]	

由图可知: 从 push ebp 到 push edi 这六句其实就实现了将 ebp 压入, 并且更新 ebp 和 esp, 并且留下了 108 的空间, 紧接着压入了三个寄存器:ebx,esi,edi。

后面的四句就是将开辟的空间全部赋值为 CCCCCCCh。

3. 接下来就是实现 fgets () 的调用:

我们直接输入 AAAA%x%x%x%x:

00401044	从返回 test.fgets 自
0012FEB8	ASCII "AAAA%x%x%x%x"
000000BB	
00422A30	offset test._iob
00000000	
00000000	
7FFDE000	

4. 接下来执行 `print()` :
得到输出:AAAA007ffd8000cccccccc

至此完成, 接下来分析为什么会出现这个结果:
由栈中内容可知:

0012FEB8	ASCII "AAAA%x%x%x%x■"
00000000	
00000000	
7FFD8000	
CCCCCCCC	

这四个%x 分别对应的就是后面的 00000000、00000000、7FFD8000、CCCCCCCC, 与输出对应, 分析结束。

二. RELEASE 模式

1. 在 VC6 执行时从 debug 切换为 release, 然后再次将生成的可执行文件导入 ollydbg 中, 通过三个 push 定位, 然后按着 F7 进入主函数, 依然得到如下:

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

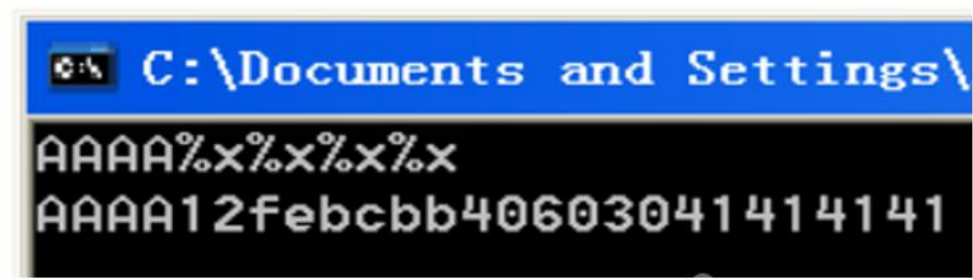
2. 对上面的指令进行分析:

Sub esp,0C8 就是为输入的 str 留下了 200 的空间, 后面通过三个 push 将 fgets() 的参数入栈。

3. 接着执行 fgets() 并输入 AAAA%x%x%x%x :

0012FEB8	ASCII "AAAA%x%x%x%x■"
000000BB	
00406030	ASCII "■n@"
41414141	
78257825	
78257825	

4. 接着执行 print() :



输出为此原因同上。

三. RELEASE 和 DEBUG 的比较:



可以从上图看出:

对于 DEBUG 模式, 其一开始会分配更大的空间 (108), 导致如果想读到输入字符串 str 的地址, 需要更多的 “%x” ;
而对于 RELEASE 模式, 其不会有像 DEBUG 模式那样的 push ebp, mov ebp, esp, 会在程序最后完成栈帧的改变。

心得体会:

通过此次实验, 我通过动手更加深了我对于格式化字符溢出这个漏洞的理解, 且也开始对 Release 和 Debug 模式的差别有了一定的认识。

另外, 上课老师所讲到的像是 SQL Injection 这样的漏洞, 实现不是特别困难, 但引起的问题却非常的大。而且课上还有很多别的格式化字符一样能引起这样的漏洞, 以后编程时需要对字符串的输入作一定规范, 尽量减少这样情况的发生。