

实验三 栈缓冲区溢出

1170300728 汤添凝

一. 实验目的

掌握栈缓冲区溢出原理；

二. 实验环境

吾爱破解虚拟机 WinXP_52Pojie_2.0、Microsoft Visual C++6.0（在虚拟机里安装）、Ollydbg

三. 实验内容

注：由于实验环境、代码编写的不同，使用 Ollydbg 反汇编出来的指令的实际地址可能与本指导的地址有所差异，请按实际情况填写，并给出必要的截图。

(1) 使用 VC++6.0 编写一段 C 程序，源代码如下：

```
#include<string.h>
#include<stdio.h>
#include<windows.h>
//有问题的函数
int test(char *str)
{
    char buffer[8]; //开辟8个字节的局部空间
    strcpy(buffer,str); //复制str到buffer[8],这里可能会产生栈溢出
    return 0;
}
//主函数
int main()
{
    char str[30000]="AAAAAAAA";
    LoadLibrary("user32.dll");
    test(str); //调用test函数并传递str变量
    return 0;
}
```

其中，strcpy 函数是一个不安全函数，无字符串长度检查，执行该函数可能会产生栈溢出。

(2) 编译程序，生成 Debug 版本的 EXE 文件，使用 Ollydbg 打开 EXE，程序断在入口点，首先寻找 call main 指令，向下查找汇编代码，在 401344 处找到一个 call 指令（实际情况可能不同，如图 1），跟踪步入，到达 401070 处（如图 2），从该地址开始 main 的执行。



图 1 call main

00401070	> 55	push ebp	
00401071	8BEC	mov ebp,esp	
00401073	B8 70750000	mov eax,0x7570	
00401078	E8 B3010000	call stack0ve._chkstklllocineAnExAtVaria	
0040107D	53	push ebx	
0040107E	56	push esi	
0040107F	57	push edi	
00401080	8DBD 908AFF	lea edi,[local.7516]	
00401086	B9 5C1D0000	mov ecx,0x1D5C	
0040108B	B8 CCCCCCCC	mov eax,0xCCCCCCCC	
00401090	F3:AB	rep stos dword ptr es:[edi]	
00401092	A1 742F4200	mov eax,dword ptr ds:[0x422F74]	
00401097	8985 D08AFF	mov [local.7500],eax	
0040109D	8B0D 782F42	mov ecx,dword ptr ds:[0x422F78]	stack0ve.00414141
004010A3	898D D48AFF	mov [local.7499],ecx	
004010A9	B9 4A1D0000	mov ecx,0x1D4A	
004010AE	33C0	xor eax,eax	
004010B0	8DBD D88AFF	lea edi,[local.7498]	
004010B6	F3:AB	rep stos dword ptr es:[edi]	
004010B8	8BF4	mov esi,esp	
004010BA	68 F0214200	push stack0ve.004221F0	
004010BF	FF15 6CA142	call dword ptr ds:[<&KERNEL32.LoadLibra	FileName = "user32.dll"
004010C5	3BF4	cmp esi,esp	LoadLibraryA
004010C7	E8 24010000	call stack0ve._chkespBufferstringsWte	
004010CC	8D95 D08AFF	lea edx,[local.7500]	
004010D2	52	push edx	ntdll.KiFastSystemCallRet
004010D3	E8 32FFFFFF	call stack0ve.0040100A	
004010D8	83C4 04	add esp,0x4	
004010DB	33C0	xor eax,eax	

图 2 main 开始

(3)4010D2 处令 edx 入栈，4010D3 使用 call 指令调用 test 函数，因此 edx 寄存器的是 test 函数的参数，即

\$edx=____00128A50____，

该寄存器代表变量名_str____的地址，变量值为____ASCII “AAAAAAA” ____。

(4)跟踪步入 call 0040100A 指令，进入 test 函数，给出 test 开始执行时的截图（从 push ebp 指令开始）。

00401020	> 55	push ebp	
00401021	8BEC	mov ebp,esp	
00401023	83EC 48	sub esp,0x48	
00401026	53	push ebx	
00401027	56	push esi	
00401028	57	push edi	
00401029	8D7D B8	lea edi,[local.18]	
0040102C	B9 12000000	mov ecx,0x12	
00401031	B8 CCCCCCCC	mov eax,0xCCCCCCCC	
00401036	F3:AB	rep stos dword ptr es:[edi]	
00401038	8B45 08	mov eax,[arg.1]	user32.77D10000
0040103B	50	push eax	
0040103C	8D4D F8	lea ecx,[local.2]	kerne132.7C8018FA
0040103F	51	push ecx	
00401040	E8 DB000000	call test.strcpysgzeHeaderListle_pages	
00401045	83C4 08	add esp,0x8	
00401048	33C0	xor eax,eax	user32.77D10000
0040104A	5F	pop edi	test.004010D8
0040104B	5E	pop esi	test.004010D8
0040104C	5B	pop ebx	test.004010D8
0040104D	83C4 48	add esp,0x48	

ebp=0012FF80
跳转来自 0040100A

(5) 子程序首先执行 push ebp;mov ebp,esp;sub esp,0x48，为变量 buffer 分配一定的内存空间，此时函数栈帧结构已经完成，此时右击寄存器窗口的 EBP 寄存器，点击“堆栈窗口中跟随”，在右下角堆栈窗口中可以观察到栈底的情况，给出堆栈窗口截图：

`_001289F0__`，而堆栈中存储返回地址的位置为`_001289FC__`，二者相差 12 字节，因此如果 `src` 的字符串长度超过 12 字节，如果字符串足够长，`src` 的第 13~16 个字符会覆盖函数的返回地址。

001289F0	41414141	
001289F4	41414141	
001289F8	41414141	
001289FC	41414141	
00128A00	41414141	
00128A04	00636D00	
00128A08	0012B880	
00128A0C	7FFDA000	