

《漏洞利用及渗透测试基础》第三次实验报告

课本样例和课外习题的crack过程

1811463 赵梓杰 信息安全

- 课本样例（密码判断机制）

```
#include <iostream>
using namespace std;
#define password "12345678"
bool verifyPwd(char * pwd)
{
    int flag;
    flag=strcmp(password, pwd);
    return flag==0;
}

void main()
{
    bool bFlag;
    char pwd[1024];
    printf("please input your password:\n");
    while(1)
    {
        scanf("%s",pwd);
        bFlag=verifyPwd(pwd);
        if(bFlag)
        {
            printf("passed\n");
            break;
        }
        else
        {
            printf("wrong password,please input again:\n");
        }
    }
}
```

- 实验步骤

在VC6.0下新建一个空的控制台项目，然后添加以上代码的main.cpp文件，单步执行生成反汇编，进行相关的阅读操作

- 对verifyPwd函数调用前后的汇编语言的解释理解

在得到汇编代码后，我们将verifyPwd函数调用前后的汇编代码进行分析（这里我并没有复制全部的汇编代码，而是将verifyPwd函数调用前后的汇编代码写了上来）

```

8:                return flag==0;
0040105C  xor          eax,eax
0040105E  cmp          dword ptr [ebp-4],0
00401062  sete        al

19:                bFlag=verifyPwd(pwd);
004010D8  lea          edx,[ebp-404h]
004010DE  push        edx
004010DF  call         @ILT+15(verifyPwd) (00401014)
004010E4  add          esp,4
004010E7  mov          byte ptr [ebp-4],al

```

首先在调用verifyPwd之前，我们知道该函数有一个参数变量pwd，因此我们需要将参数pwd压入，即lea指令和push指令，将[ebp-404h]对应地址的值取出给数据寄存器edx，然后压入edx，跳转至verifyPwd函数体内部执行，函数体内执行的操作后面一部分会详述，此处只关注函数体结束后返回flag==0，此时flag存储在[ebp-4]中，将eax异或归零后，比较flag和0，对al进行设定，因此我们可以看到主函数对应的汇编代码中，在call结束后，对esp加4后，因为只有一位参数，所以只需要对esp+4即可，最后将al的值，也就是verifyPwd的函数返回值al按位赋值给[ebp-4]这个位置

○ 对flag==0的汇编代码的解释理解

同样，flag==0相关的汇编代码我们此处忽略了其它位置的汇编代码，仅对这一部分进行分析

```

7:                flag=strcmp(password, pwd);
00401048  mov          eax,dword ptr [ebp+8]
0040104B  push        eax
0040104C  push        offset string "12345678" (0043301c)
00401051  call         strcmp (00408220)
00401056  add          esp,8
00401059  mov          dword ptr [ebp-4],eax
8:                return flag==0;
0040105C  xor          eax,eax
0040105E  cmp          dword ptr [ebp-4],0
00401062  sete        al

```

我们从flag的赋值所对应的汇编代码进行查看，首先，将pwd的值赋值给eax，然后将eax (pwd) 和string对应的“12345678”一起压入栈中，去执行strcmp函数，执行结束后，返回的参数保存到了eax中，因为两个参数，所以将esp增加8位，然后将eax赋值给[ebp-4]。最后进行flag==0的判断和返回，将eax归零处理后，对0和[ebp-4]进行比较处理，如果相等的话就设置al

○ 课本上两种crack方式的复现操作

■ 第一种crack方法

1. 查找字符串wrong，定位到分支跳转的代码

<pre> 004010F6 68 5430A300 push offset 00433054 004010FB EB 50720000 call printf 00401100 83C4 04 add esp,4 00401103 EB 0F jmp short 00401114 00401105 68 2830A300 push offset 00433028 0040110A EB 41720000 call printf 0040110F 83C4 04 add esp,4 </pre>	<pre> ASCII "passed" Cprintf ASCII "wrong passuord,please input again:!" Cprintf </pre>
--	--

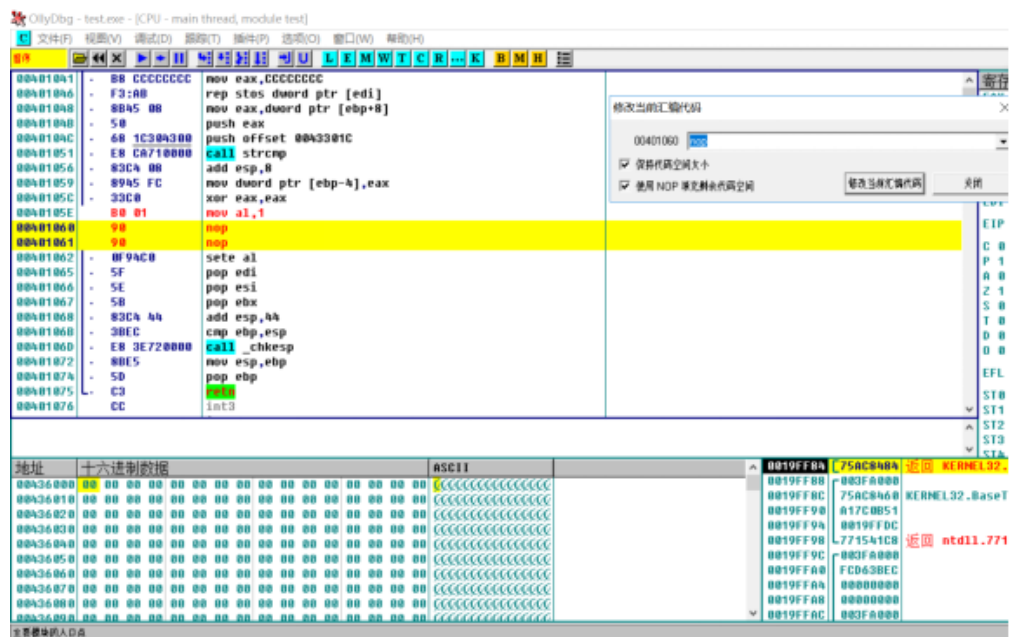
2. 我们看到了jz short 00401105，因为已知jz在ZF=1（ZF为zero flag即零标志）时跳转，当运算结果为0时置1，跳转到显示wrong这段语句的分支中，因此我们只需要将jz改成jnz，jnz为jz的反运算，那么程序跳转结果就截然相反，当输入和12345678不一样的错误密码时，就进入到验证成功的分支中，此时可以直接将jz改成jnz，也可以将机器码74（jz）改成75（jnz）

004010F2	85C0	test eax,eax	
004010F4	74 0F	jz short 00401105	ASCII "passed"
004010F6	68 54304300	push offset 00433054	[printf
004010F8	E8 50720000	call printf	
004010FD	25 7F000000	and eax,0000007F	
004010F2	85C0	test eax,eax	
004010F4	75 0F	jnz short 00401105	ASCII "passed"
004010F6	68 54304300	push offset 00433054	[printf
004010F8	E8 50720000	call printf	
00401100	83C4 04	add esp,4	
00401103	EB 0F	jmp short 00401114	ASCII "wrong password,please input again:"
00401105	68 28304300	push offset 00433028	[printf
0040110A	E8 47200000	call printf	

■ 第二种crack方法

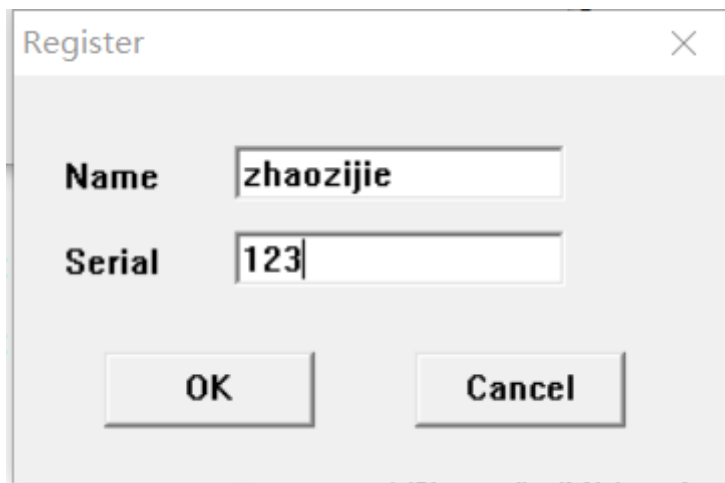
第二种crack方法相比较第一种更改分支较为复杂，第二种主要思路是将verifyPwd函数的返回值进行固定化修改

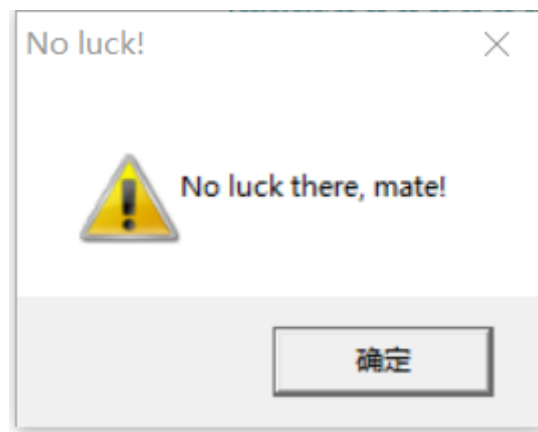
在我们找到函数体后，直接将从其判断flag==0附近的汇编代码全部置为nop，然后再给al的值赋1，然后即可完成crack



● 课外题目

- 因为课本上这个verifyPwd的题目之前自己按照课本有复现过，所以在下课时找到了一位同学发给我的一个题目（附件中的crackme.exe），实验报告的剩下部分我将简单介绍下针对crackme.exe文件的相关处理（以下操作在主机win10系统下完成）
- 首先我们直接尝试打开这个exe文件，发现基本什么都没有，在菜单的help目录下有register注册，因此猜测此为一个注册机，即serial需要为name经过一定算法处理后得到的字符串或数字，随便输入一个内容后发现弹出字符串"No luck....."

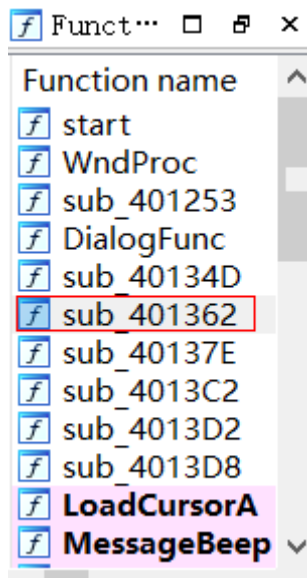




- 然后我们就可以借助IDA和OLD进行操作，我是借助OLD进行查找操作，借助IDA进行伪代码审计和汇编代码审计操作，首先在OLD中查找到字符串，发现这段展示字符串的代码段的进入地址为00401362

00401333	75 11	jne short 00401346	
00401335	6A 00	push 0	
00401337	FF75 08	push dword ptr [ebp+8]	
0040133A	E8 73010000	call <jmp.&USER32.EndDialog>	Result = 0
0040133F	B8 01000000	mov eax,1	hDialog => [ARG.EBP+8]
00401344	EB DF	jmp short 00401325	USER32.EndDialog
00401346	B8 00000000	mov eax,0	
00401348	EB D8	jmp short 00401325	
0040134D	6A 30	push 30	
0040134F	68 29214000	push offset 00402129	ASCII "Good work!"
00401354	68 34214000	push offset 00402134	ASCII "Great work, mate!"
00401359	FF75 08	push dword ptr [ebp+8]	
0040135C	E8 D9000000	call <jmp.&USER32.MessageBox>	Jump to USER32.MessageBox
00401361	C3	ret	
00401362	6A 00	push 0	Type = MB_OK
00401364	E8 AD000000	call <jmp.&USER32.MessageBeep>	USER32.MessageBeep
00401369	6A 30	push 30	
0040136B	68 60214000	push offset 00402160	ASCII "No luck!"
00401370	68 69214000	push offset 00402169	ASCII "No luck there, mate!"
00401375	FF75 08	push dword ptr [ebp+8]	
00401378	E8 BD000000	call <jmp.&USER32.MessageBox>	Jump to USER32.MessageBox
0040137D	C3	ret	
0040137E	8B7424 04	mov esi,dword ptr [esp+4]	
00401382	56	push esi	
00401383	8A06	mov al,byte ptr [esi]	
00401385	84C0	test al,al	

然后我们可以直接在IDA中看到跳转到这个地址的函数，这里去查找一下执行call sub_401362的函数位置，我们可以找到调用这个函数的入口位置



CODE:0040121E E8 7D 02 00 00	call DialogBoxParamA
CODE:00401223 83 F8 00	cmp eax, 0
CODE:00401226 74 BE	jz short loc_4011E6
CODE:00401228 68 8E 21 40 00	push offset String
CODE:0040122D E8 4C 01 00 00	call sub_40137E
CODE:00401232 50	push eax
CODE:00401233 68 7E 21 40 00	push offset byte_40217E
CODE:00401238 E8 9B 01 00 00	call sub_4013D8
CODE:0040123D 83 C4 04	add esp, 4
CODE:00401240 58	pop eax
CODE:00401241 3B C3	cmp eax, ebx
CODE:00401243 74 07	jz short loc_40124C
CODE:00401245 E8 18 01 00 00	call sub_401362
CODE:0040124A EB 9A	jmp short loc_4011E6

然后可以简单看一下这段代码，可以猜测出，7E和D8分别是对name和serial进行处理的函数，因此我们直接从左边点入这两个代码块，首先对7E的伪代码进行阅读

```
int __usercall sub_40137E(eax)(int a1<ebp>, int a2<edi>, unsigned __int8 *a3)
{
    unsigned __int8 *v3; // esi@1
    char v4; // al@2

    v3 = a3;
    while ( 1 )
    {
        v4 = *v3;
        if ( !*v3 )
        {
            sub_4013C2();
            return a2 ^ 0x5678;
        }
        if ( (unsigned __int8)v4 < 0x41u )
            break;
        if ( (unsigned __int8)v4 >= 0x5Au )
            sub_4013D2(v4, (int)v3++);
        else
            ++v3;
    }
    return MessageBoxA(*(HWND *) (a1 + 8), "No luck there, mate!", "No luck!", 0x30u);
}
```

```
void __usercall sub_4013C2(int a1<esi>)
{
    int v1; // edi@1
    int v2; // ebx@1

    v1 = 0;
    v2 = 0;
    while ( *(_BYTE *)a1 )
    {
        LOBYTE(v2) = *(_BYTE *)a1;
        v1 += v2;
        ++a1;
    }
}
```

```
char __usercall sub_4013D2(a1)(char a1<a1>, int a2<esi>)
{
    char result; // al@1

    result = a1 - 32;
    *(_BYTE *)a2 = result;
    return result;
}
```

然后简单阅读后不难发现，对于输入的名字，必须为字母（根据Ascii码判断），如果为小写字母的话，将其转换为大写字母（Ascii码减32），然后将其每一位对应的Ascii码相加，最后得到的和与0x5678进行异或，得到name的处理结束后的参数a

最后我们对serial进行分析，根据伪代码，我们可以自己模拟写出这个Cpp加密算法

```

char __cdecl sub_4013D8(int a1)
{
    int v1; // eax@1
    int v2; // edi@1
    int v3; // ebx@1
    int i; // esi@1

    v1 = 0;
    v2 = 0;
    v3 = 0;
    for ( i = a1; ; ++i )
    {
        LOBYTE(v1) = 10;
        if ( !*( _BYTE *)i )
            break;
        LOBYTE(v3) = *( _BYTE *)i - 48;
        v2 = v3 + v1 * v2;
    }
    return v1;
}

```

即我们可以理解成将一个int型的数进行拆分将其每一位的Ascii码-48（就是本身）进行累加，每次累加，前项乘十。用简单的话去表达，就是把 $1234 = 1000 + 200 + 30 + 4 = 1234$

（然后我就以为上面这个是最终算法了，但是好久没crack出来）

在这个serial的加密算法下面有一个代码块，可以发现，serial最后的结果是要自己和0x1234h异或

```

CODE:004013F5          loc_4013F5:
CODE:004013F5  81 F7 34 12 00 00      xor     edi, 1234h
CODE:004013FB  8B DF                  mov     ebx, edi
CODE:004013FD  C3                     retn
CODE:004013FD          sub_4013D8             endp

```

- 最后判断下这两个处理后的值是否相同，若相同即crack，因此进行总结，假设我们的name为'zhaozijie'，首先需要将'zhaozijie'对应的字母转换为大写字母，随后将其的所有位的Ascii码累加，得到一个数，对0x5678h进行异或，然后再对0x1234h进行异或（因为我们需要得到等式，所以把serial的异或挪到name上），即可得到最终的serial
- 借助C++代码进行crack，得到name = 'zhaozijie'时候对应的serial应该为18145，即可crack成功

```

#include<iostream>
using namespace std;
int main()
{
    char name[100];
    int name2passwd = 0;
    int passwd;
    cin >> name;
    //name = "zhaozijie"
    for (int i = 0; i < 99; i++)
    {
        char temp = name[i];
        if (temp < 'A')
            break;
        if (temp > 'Z')
            name2passwd += temp - 32;
        else {
            name2passwd += temp;
        }
    }
}

```

```
}  
name2passwd = name2passwd ^ 0x5678;  
passwd = name2passwd ^ 0x1234;  
cout << passwd << endl;  
return 0;  
}
```

- 本题的收获：简单进行了汇编代码的审计，感觉并不需要很刨根问底的将一个程序的完全汇编全部都懂，只需要将一些关键步骤或算法的汇编代码理解即可，IDA的伪代码模式相较于汇编语言而言，压力较小，同样开始做题前面较顺，但是后面被异或0x1234卡住了很久，过于依赖伪代码，没有观察到汇编代码的存在额外的部分，总而言之，简单了解了OLD和IDA的使用和简单的Crack操作