

武汉大学国家网络安全学院

课程实验(设计)报告

课程名称：_____软件安全实验_____

实验内容：_____实验五 逆向工程与软件自我保护_____

专业(班)：_____

学 号：_____

姓 名：_____

任课教师：_____

目 录

实验 5 逆向工程与软件自我保护.....	1
5.1 实验名称.....	1
5.2 实验目的.....	1
5.3 实验步骤及内容.....	1
5.4 实验关键过程、数据及其分析.....	1

实验 5 逆向工程与软件自我保护

5.1 实验名称

逆向工程与软件自我保护

5.2 实验目的

熟悉软件保护破解机制、熟悉假壳脱壳

5.3 实验步骤及内容

第一阶段：软件保护破解

方法一：爆破（至少两种方式）

1. 查找显示注册结果相关代码
2. 查找注册码验证相关代码
3. 修改程序跳转

方法二：编写注册机

1. 查找显示注册结果相关代码
2. 查找注册码验证相关代码
3. 根据注册码验证代码编写注册机

第二阶段：软件反动态调试分析

1. 分析 CrackMe1.exe 是如何通过父进程检测实现反 OllyDbg 调试的
2. 分析除父进程检测外，该程序用到的反动态调试技术

第三阶段：加壳脱壳

1. 加壳脱壳深入理解
2. 尝试手动脱壳

5.4 实验关键过程、数据及其分析

第一阶段：软件保护破解

方法一：修改跳转指令

使用 Ollydbg 查找显示注册结果相关汇编代码，查找注册码验证相关代码，修改程序跳转。

测试要破解的 crack.exe 可执行文件，可以发现其需要输入一个用户名和对应序列号，如果随便输入会弹出一个 bad boy 的错误弹窗



使用 Ollydbg 打开可执行文件，由于错误提示是一个弹窗，考虑是一个 MessageBox 函数调用，查看栈搜索 ASCII 字符串，尝试寻找弹窗的提示信息

0040128D	- 68 6C1E4000	push crack.00401E6C	SE 处理程序安
00401292	- 64:A1 000000	mov eax,dword ptr fs:[0]	
00401298	- 50	push eax	
00401299	- 64:8925 0000	mov dword ptr fs:[0],esp	
004012A0	- 83EC 58	sub esp,0x58	
004012A3	- 53	push ebx	
004012A4	- 56	push esi	
004012A5	- 57	push edi	
004012A6	- 8965 E8	mov [local.6],esp	
004012A9	- FF15 3440400	call dword ptr ds:[<&KERNEL32.GetVersion	kernel32.Get
004012AF	- 33D2	xor edx,edx	ntdll.KiFast
004012B1	- 8AD4	mov dl,ah	
004012B3	- 8915 3855400	mov dword ptr ds:[0x405538],edx	ntdll.KiFast
004012B9	- 8BC8	mov ecx,eax	

通过搜索查看字符串，得到引用的字符串

地址	操作数	字符串
004010FB	push crack.00405098	Good Boy!
00401100	push crack.0040507C	Terima kasih kerana mencu
00401107	push crack.00405070	Bad Boy!
0040110C	push crack.0040504C	Tidak tepat, sila cuba se
0040112C	push crack.00405030	masukkan at least 5 huruf
00401210	mov ecx,dword ptr ds:[0x4050AC]	
0040124E	mov eax,dword ptr ds:[0x4050AC]	
00401283	push ebp	(Initial CPU selection)
004013D4	mov ecx,dword ptr ds:[0x4050AC]	
004013E3	mov esi,dword ptr ds:[0x4050AC]	
004016BB	mov ecx,dword ptr ds:[0x405350]	
00401FFA	push crack.004043A4	\n
0040203C	push crack.004043A0	<program name unknown>
00402050	push crack.00404384	...
0040206E	push crack.00404380	Runtime Error!\n\nProgram
00402079	push dword ptr ds:[esi+0x40537C]	\n\n
00402079	push dword ptr ds:[esi+0x40537C]	0C@
00402096	push crack.00404358	R6002\r\n- floating point
004020AA	lea esi,dword ptr ds:[esi+0x40537C]	Microsoft Visual C++ Runt
004020AA	lea esi,dword ptr ds:[esi+0x40537C]	0C@
00402242	movzx eax,word ptr ds:[eax*2+0x4050B6]	R6002\r\n- floating point
0040231B	lea ebx,dword ptr ds:[esi+0x405420]	
004033EB	push crack.00404404	user32.dll
00403402	push crack.004043F8	MessageBoxA
00403413	push crack.004043E8	GetActiveWindow
0040341B	push crack.004043D4	GetLastActivePopup

通过 Follow 跳转到相关的代码，发现这是一个判断跳转的代码逻辑。

004010F7	3BC8	cmp ecx,eax	
004010F9	75 0C	jnz short crack.00401107	
004010FB	68 98504000	push crack.00405098	Good Boy!
00401100	68 7C504000	push crack.0040507C	Terima kasih kerana mencuba
00401105	EB 0A	jmp short crack.00401111	
00401107	68 70504000	push crack.00405070	Bad Boy!
0040110C	68 4C504000	push crack.0040504C	Tidak tepat, sila cuba seka
00401111	FF75 08	push [arg.1]	hOwner = 00401283
00401114	FF15 AC404000	call dword ptr ds:[<&USER32.MessageBoxA	MessageBoxA
0040111A	FF75 10	push [arg.3]	hMem = 78746341
0040111D	8B35 04404000	mov esi,dword ptr ds:[<&KERNEL32.GlobalF	kernel32.GlobalFree
00401123	FFD6	call esi	GlobalFree
00401125	FF75 14	push [arg.4]	hMem = 00000020
00401128	FFD6	call esi	GlobalFree
0040112A	EB 54	jmp short crack.00401180	
0040112C	68 30504000	push crack.00405030	masukkan at least 5 huruf
00401131	57	push edi	ControlID = 636850 (6515536
00401132	FF75 08	push [arg.1]	hWnd = 00401283
00401135	FF15 A4404000	call dword ptr ds:[<&USER32.SetDlgItemT	SetDlgItemTextA
0040113B	EB 43	jmp short crack.00401180	

将 JNZ 修改为 JZ，这样保存的新的 exe 文件当输入的信息是错误的时不会跳转到错误弹窗而是继续执行正确弹窗。

E8 8D010000	call crack.00401278	
59	pop ecx	kernel32.7C817
8B4D		
6A 00		
81F1	000010F9	je short 00001107
3BC8	000010FB	68 98504000 push 0x405098
	00001100	68 7C504000 push 0x40507C
	00001105	EB 0A jmp short 00001111
	00001107	68 70504000 push 0x405070
	0000110C	68 4C504000 push 0x40504C
	00001111	FF75 08 push dword ptr ss:[ebp+0x8]
	00001114	FF15 AC404000 call dword ptr ds:[0x4040AC]
	0000111A	FF75 10 push dword ptr ss:[ebp+0x10]
	0000111D	8B35 04404000 mov esi,dword ptr ds:[0x404004]
	00001123	FFD6 call esi
	00001125	FF75 14 push dword ptr ss:[ebp+0x14]
	00001128	FFD6 call esi
	0000112A	EB 54 jmp short 00001180
	0000112C	68 30504000 push 0x405030
	00001131	57 push edi
	00001132	FF75 08 push dword ptr ss:[ebp+0x8]
	00001135	FF15 A4404000 call dword ptr ds:[0x4040A4]
	0000113B	EB 43 jmp short 00001180
	0000113D	6A 00 push 0x0

功
随便输入一个用户名和序列号，进行测试，可以发现弹出验证正确的弹窗，说明破解成功



方法二：编写注册机

具体思路：

1. 查找显示注册结果相关代码
2. 查找注册码验证相关代码
3. 对相关代码进行分析，还原验证逻辑。
4. 根据注册码验证代码编写注册机

找到弹窗代码前面的汇编代码，可以推测这些应该是进行验证的代码段。

004010E3	> FF75 14	push [arg.4]	
004010E6	. E8 8D010000	call crack.00401278	
004010EB	. 59	pop ecx	kernel32.7C817067
004010EC	. 8B4D 0C	mov ecx,[arg.2]	
004010EF	. 6A 00	push 0x0	
004010F1	. 81F1 FAF9A90	xor ecx,0xA9F9FA	
004010F7	. 3BC8	cmp ecx,eax	
004010F9	75 0C	jnz short crack.00401107	
004010FB	68 98504000	push crack.00405098	Good Boy!
00401100	68 7C504000	push crack.0040507C	Terima kasih kerana mencoba
00401105	EB 0A	jmp short crack.00401111	

此处有一个 cmp 比较语句，比较 ecx 和 eax 的值是否相等。

继续向上查看代码，可以发现 call 语句和 cmp 语句之间应该是程序根据输入产生注册码的过程

004010C6	. 33C0	xor eax,eax	
004010C8	. 85DB	test ebx,ebx	
004010CA	7E 17	jle short crack.004010E3	
004010CC	> 8B4D 10	mov ecx,[arg.3]	
004010CF	. 8B55 0C	mov edx,[arg.2]	
004010D2	. 03D3	add edx,ebx	
004010D4	. 0FBE 0C 08	movsx ecx,byte ptr ds:[eax+ecx]	
004010D8	. 0FAFCA	imul ecx,edx	ntdll.KiFastSystemCall
004010DB	. 40	inc eax	
004010DC	. 894D 0C	mov [arg.2],ecx	
004010DF	. 3BC3	cmp eax,ebx	
004010E1	7C E9	jle short crack.004010CC	
004010E3	> FF75 14	push [arg.4]	
004010E6	. E8 8D010000	call crack.00401278	kernel32.7C817067
004010EB	. 59	pop ecx	
004010EC	. 8B4D 0C	mov ecx,[arg.2]	
004010EF	. 6A 00	push 0x0	
004010F1	. 81F1 FAF9A90	xor ecx,0xA9F9FA	
004010F7	. 3BC8	cmp ecx,eax	

首先通过异或清零 eax，在此处设置断点，进行调试，此时 eax 清零。

继续进行单步执行，发现 arg3 给 ecx 送了数据，取出了用户名的 ASCII 码值。接着相乘后 eax 在 for 循环里每轮+1，用做条件判断

梳理逻辑过程，即每次从输入字符串里取出一个字符，对这个字符做上述的运算。根据分析的运算过程可以写出 C 语言代码的模拟注册机。

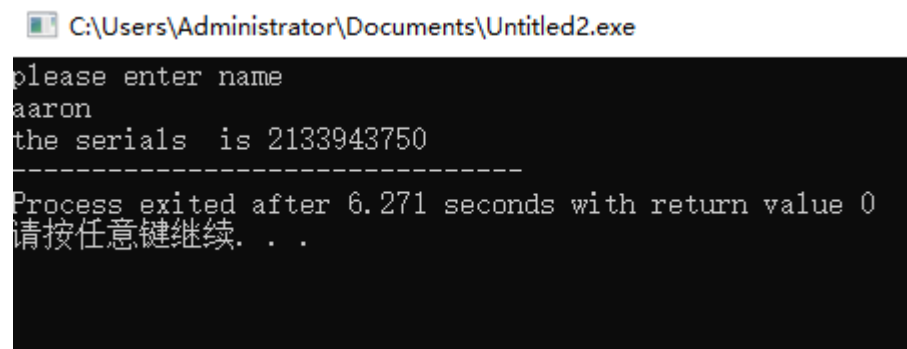
```
#include <stdio.h>
#include <string.h>
int main()
{
    printf("please enter name \n");
```

```

char name[10];
gets(name);
char* arg3 = name;
int ecx;
int arg2 = 6408;
int ebx = strlen(arg3);
for (int eax = 0; eax < ebx; eax++)
{
    arg2 += ebx;
    ecx = arg3[eax];
    ecx *= arg2; //arg2=edx
    arg2 = ecx;
}
printf("the serials is %d",arg2 ^ 0xA9F9FA);
}

```

测试注册机，得到的序列号如下：



```

C:\Users\Administrator\Documents\Untitled2.exe
please enter name
aaron
the serials is 2133943750
-----
Process exited after 6.271 seconds with return value 0
请按任意键继续. . .

```

进行测试，验证成功，说明编写的注册机是正确的。



第二阶段：软件反动态调试分析

使用原版 Ollydbg 打开 CrackMe 执行文件运行，发现文件直接执行到了 return 处，没有进行弹窗。这说明该程序对动态调试进行一定的反制措施。

```
KernelMode - CrackMe1.exe
File View Debug Plugins Options Window Help Tools BreakPoint->
Terminated [Icons] L E M T W H C / K I
C *G.P.U* - main thread, module ntdll
7C92E4F4 C3 ret
7C92E4F5 8DA424 00000000 lea esp,dword ptr ss:[esp]
7C92E4FC 8D6424 00 lea esp,dword ptr ss:[esp]
7C92E500 8D5424 08 lea edx,dword ptr ss:[esp+0x8]
7C92E504 CD 2E int 0x2E
7C92E506 C3 ret
7C92E507 90 nop
7C92E508 55 push ebp
7C92E509 8BEC mov ebp,esp
7C92E50B 9C pushfd
```

使用吾爱破解版进行调试，查看寄存器的变化情况。

```
00502707 55 push ebp
00502708 8BEC mov ebp,esp
00502709 8B55 0C mov esi,[arg.2]
0050270B 56 push esi
0050270E 8B75 08 mov esi,[arg.1]
00502711 57 push edi
00502712 0FB6 06 movzx eax,byte ptr ds:[esi]
00502715 8D48 BF lea ecx,dword ptr ds:[eax-0x41]
00502718 46 inc esi
00502719 83F9 19 cmp ecx,0x19
0050271C 77 03 ja short CrackMe1.00502721
0050271E 83C0 20 add eax,0x20
00502721 8B6A 0A movzx ecx,byte ptr ds:[edx]
00502724 8D79 BF lea edi,dword ptr ds:[ecx-0x41]
00502727 42 inc edi
00502728 83FF 19 cmp edi,0x19
0050272B 77 03 ja short CrackMe1.00502730
0050272D 83C1 20 add ecx,0x20
00502730 85C0 test eax,eax
00502732 74 04 je short CrackMe1.00502738
00502734 3BC1 cmp eax,ecx
00502736 74 DA je short CrackMe1.00502712
00502738 5F pop edi
00502739 2BC1 sub eax,ecx
0050273B 5E pop esi
0050273C 5D pop ebp
0050273D C3 ret
```

可以发现，CrackMe 可执行文件会检测父进程的名字，将当前进程名与 Windows 的资源管理器 explore 比较，如果相同则返回 true，可以进行调试，如果不相同则返回 false，不可进行调试。

```
00502712 > 0FB6 06 movzx eax,byte ptr ds:[esi]
00502715 . 8D48 BF lea ecx,dword ptr ds:[eax-0x41]
00502718 . 46 inc esi
00502719 . 83F9 19 cmp ecx,0x19
0050271C ~ 77 03 ja short CrackMe1.00502721
0050271E . 83C0 20 add eax,0x20
00502721 > 8B6A 0A movzx ecx,byte ptr ds:[edx]
00502724 . 8D79 BF lea edi,dword ptr ds:[ecx-0x41]
00502727 . 42 inc edi
00502728 . 83FF 19 cmp edi,0x19
0050272B ~ 77 03 ja short CrackMe1.00502730
0050272D . 83C1 20 add ecx,0x20
00502730 > 85C0 test eax,eax
00502732 ~ 74 04 je short CrackMe1.00502738
00502734 . 3BC1 cmp eax,ecx
00502736 ^ 74 DA je short CrackMe1.00502712
00502738 > 5F pop edi
00502739 . 2BC1 sub eax,ecx
0050273B . 5E pop esi
0050273C . 5D pop ebp
0050273D . C3 ret
```

分析进程检测代码可知，CrackMe 会判断获取进程名字的大小写，如果为大写加上 0x20 将其转换为 ASCII 小写字母。逐个比较单个字母，相等则跳转到 0x00502712 地址处，进行

下一个字符的比较，不相等则跳出循环返回 false 拒绝进行调试。

第三阶段：加壳脱壳

1. 加壳脱壳深入理解

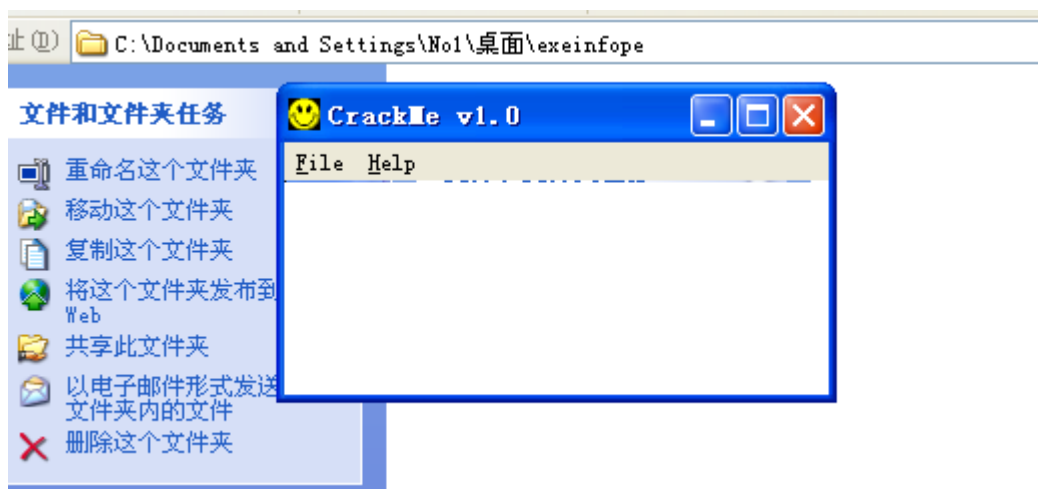
壳是一段执行于原始程序前的代码。一般先于原始程序运行，完成它们保护软件的任务。当加壳后的文件执行时，壳—这段代码先于原始程序运行，他把压缩、加密后的代码还原成原始程序代码，然后再把执行权交还给原始代码。软件的壳分为加密壳、压缩壳、伪装壳、多层壳等类，目的都是为了隐藏程序真正的 OEP 入口地址。

加壳，是一种通过一系列数学运算，将可执行程序文件或动态链接库文件的编码进行改变（目前还有一些加壳软件可以压缩、加密驱动程序），以达到缩小文件体积或加密程序编码的目的。加壳一般是指保护程序资源的方法。

脱壳，一般是指除掉程序的保护，用来修改程序资源。马甲“能穿也能脱。相应的，有加壳也一定会有解壳（也叫脱壳）。脱壳主要有两种方法：硬脱壳和动态脱壳。

2. 尝试手动脱壳

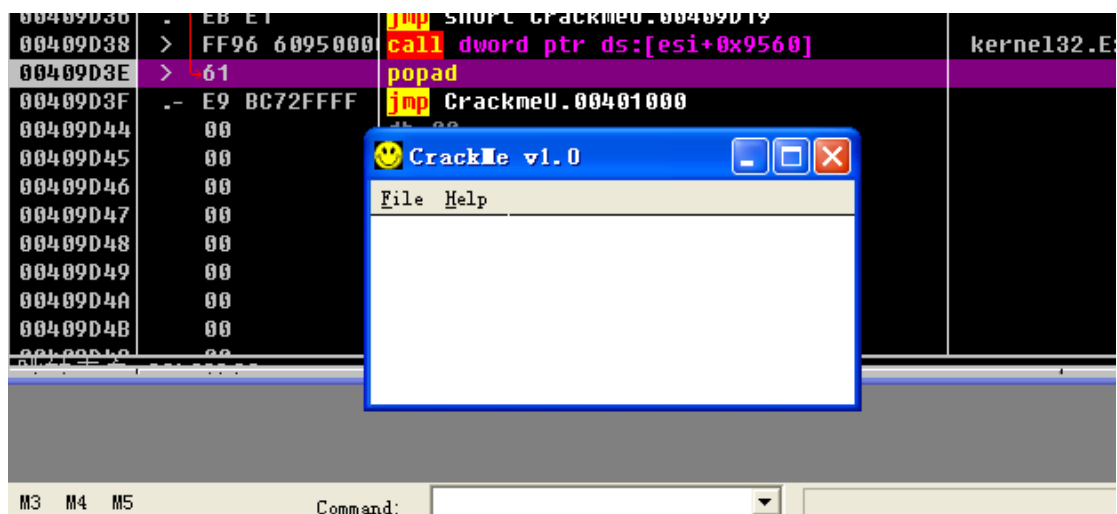
打开测试程序 CrackMeUPX，可以发现其大致的功能是弹出一个框。



使用 OllyDbg 打开程序，查看代码，可以看到程序首先将所有的通用寄存器进行了入栈。



通过打断点调试，找到 popad 指令，通过 F4 进入

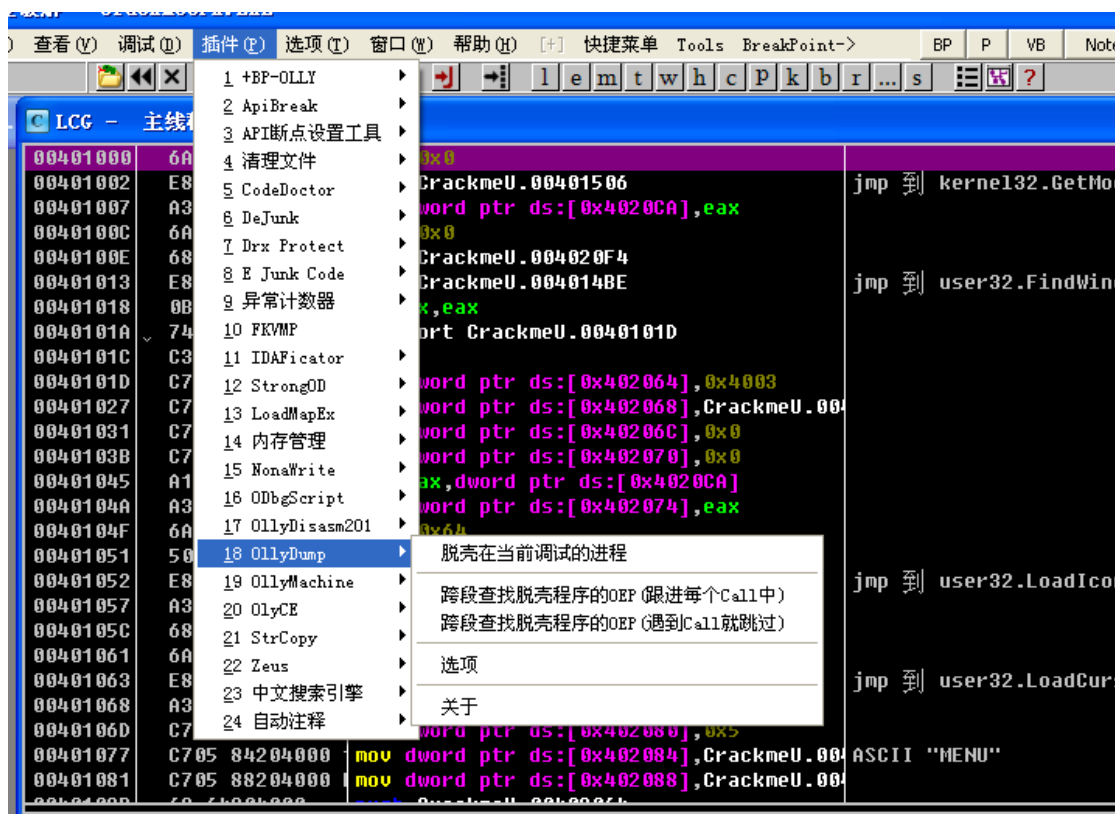


下面的指令跳转到 00401000，很可能就是原始程序的入口点。跟随进入

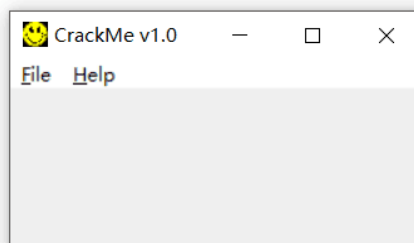


可以看到这一部分的代码就是原始程序的代码，手工脱壳成功。

再使用 Ollydbg 的插件进行脱壳，选择插件，OllyDump，脱壳当前调试的进程



将脱壳后的程序保存，执行，运行正确，脱壳成功



5.5 实验心得与体会

本次实验让我对软件逆向工程有了大致的了解。在爆破试验中，我明白了软件序列号的匹配机制，而通过 Ollydbg 分析验证代码让我明白了如何去逆向破解一个软件的序列号。同时利用高级语言编写注册机让我知道了以前用的各种大型软件注册机的基本实现思想。而此后的反动态调试和加壳脱壳技术则让我感受到了软件逆向的攻击和防御手段。逆向工程与汇编息息相关，这使对汇编接触较少的我分析时有些吃力。但是仍然受益匪浅。这也让我意识到了今后无论是从事安全还是开发的工作，重视逆向防御必不可少。一个大型软件序列号验证可能代码量不及总工作量的千分之一，但是忽略逆向防御就可能会由于遭到逆向而受到严重的损失。而对于汇编语言的阅读分析进一步加深了我对底层的认识以及汇编的重要性。希望将这份热情继续保持下去。