

实验一 Acid burn.exe 程序的逆向分析

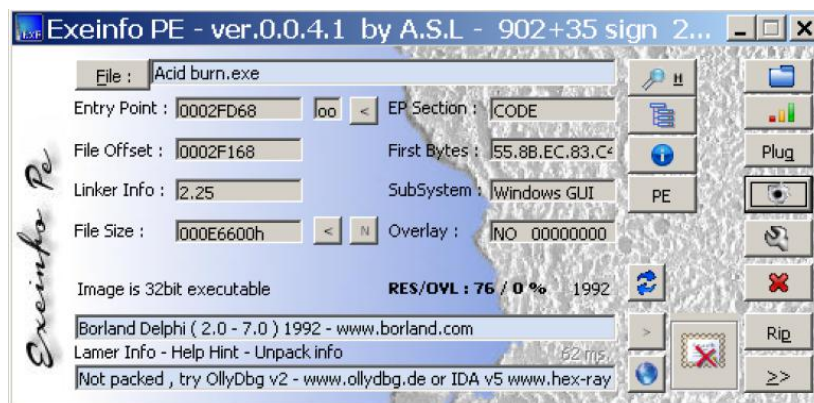
一. 实验目的

1. 掌握逆向分析的一般流程，熟练使用逆向分析的常用工具，并给出实验中相应软件的输出结果；
2. 掌握逆向分析的断点设置方法并对关键程序逻辑进行跟踪和定位；
3. 掌握逆向分析的指令修改方法，对程序的注册验证机制进行爆破；
4. 掌握常用的汇编指令，对程序的注册码算法进行回溯。

二. 实验内容

1. 第一步：屏蔽软件启动的弹出窗口

- (1) 找到合适的查壳工具完成对 Acid burn.exe 程序的查壳，判断程序是否有壳和壳的类型，截图如下（图 1）：



有壳，壳的类型：Borland Delphi

(2) 启动 01lydbg，加载程序；

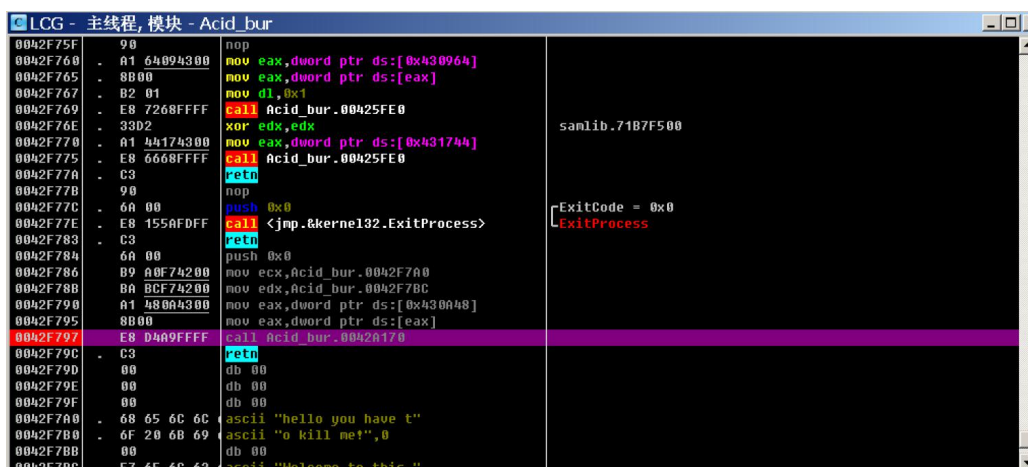
(3) 程序启动时会出现弹窗，截图如下（图 2）：



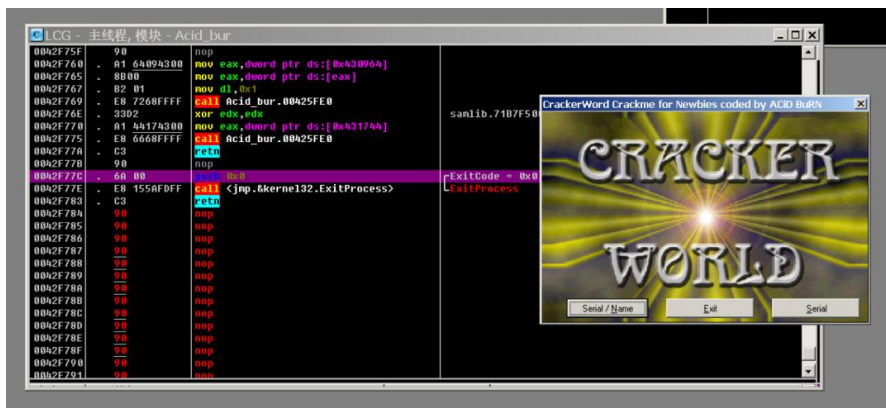
(4) 去除该弹窗，弹窗是通过调用 [MessageBoxA](#) 实现的，请采用 [API 断点设置工具](#) 方式为此函数设置断点，截图如下（图 3）：



(5) 使用 01lydbg 执行程序，程序中断在相关的函数调用处，定位调用这个函数的汇编代码，地址为：[0042F797](#)，截图如下（图 4）：

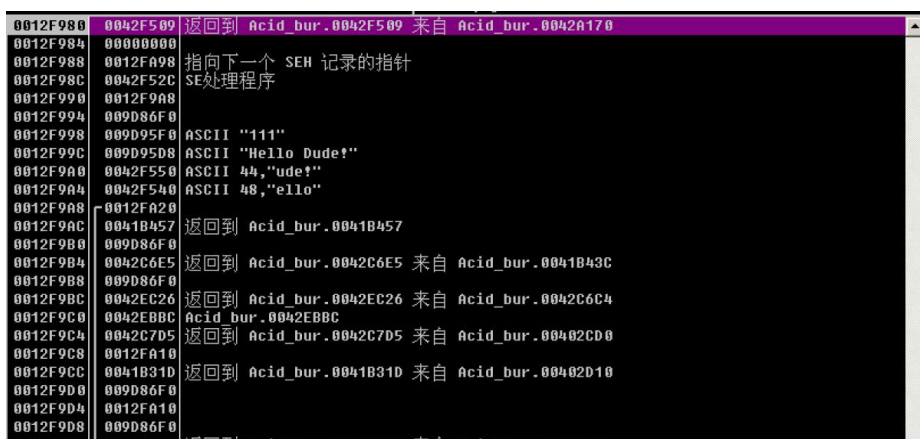


- (6) 选中 call 指令及周围指令,用 nop 指令代替,重新加载程序,然后令程序运行,此时弹窗不再出现。(注:有时重新执行程序后发现修改的指令没有生效,此时按 Ctrl+P 快捷键打开 patches 窗口,发现状态显示“已删除”,即已经修改的指令被禁用,需要选中被删除的 Patch 按空格让状态变为“激活”,然后再运行程序。)



2. 第二步：爆破法 Crack 使用单一序列号的验证

- (1) 在程序主窗口中点击 Serial 按钮,进入只有序列号的验证逻辑,采用上述相似的思路,输入错误的序列号弹出错误对话框,利用此时的堆栈状态找到调用弹出窗口函数 call 地址为 0042F509,截图如下(图 5);



- (2) 向上回溯,找到该代码段的开始位置,对应的汇编指令为: push ebp, 地址为 0042F470。在该位置设置断点,重新运行程序到该代码段开始位置,单步运行,查看可能发生序列号 and 用户输入的序列号比较操作的 call 指令(注:在进入生成序列号的子程序前,需要把用户的序列号字符串和正确的序列号字符串所在的地址送入两个寄存器中,即先执行两个 mov 指令然后再执行 call 指令,可以根据这个特点进行定位)
- (3) call 指令后紧跟一种转移指令,用于判断应该弹出输出正确/错误的对话框,该指令为 jnz 指令, call 指令地址为 004039FC,使用爆破法修改转移指令,令程序弹出正确的对话框,修改指令的截图如下(图6):

0042F4C5	. E8 8EB5FEFF	call Acid_bur.0041AA58
0042F4CA	. 8B45 F0	mov eax,[local.4]
0042F4CD	. 8B55 F4	mov edx,[local.3]
0042F4D0	. E8 2745FDFF	call Acid_bur.004039FC
0042F4D5	90	nop
0042F4D6	90	nop
0042F4D7	. 6A 00	push 0x0
0042F4D9	. B9 64F54200	mov ecx,Acid_bur.0042F564
0042F4DE	. BA 70F54200	mov edx,Acid_bur.0042F570

3. 第三步：爆破法 Crack 使用 Serial/Name 的验证

- (1) 输入自己的学号和任意的序列号，弹出错误提示信息，截图如下（图 7）：



- (2) 错误信息窗口同样使用弹出窗口进行提示，使用相同的方法设置断点并触发，跟踪调用窗口的位置地址为：[0042A170](#)，因此需要查看堆栈找到其返回地址并跟踪至调用该子程序的汇编代码，返回地址为：[0042FB37](#)，堆栈的截图如下（图 8）：

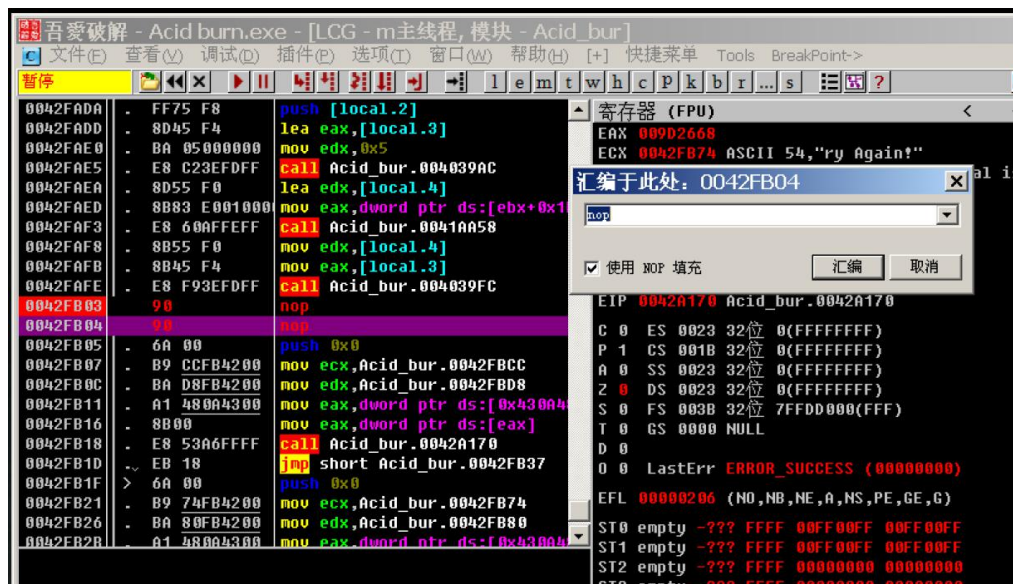


- (3) 向上回溯代码，分析进入错误信息窗口弹出的分支，找到使程序跳入该分支的 jinz 指令，相应的汇编代码地址为：0042FB03，截图如下（图 9）：



```
0042FADA - FF75 F8      push [local.2]
0042FADD - 8D45 F4      lea eax,[local.3]
0042FAE0 - BA 05000000 mov edx,0x5
0042FAE5 - E8 C23EFDFF call Acid_bur.004039AC
0042FAEA - 8D55 F0      lea edx,[local.4]
0042FAED - 8B83 E0010000 mov eax,dword ptr ds:[ebx+0x1]
0042FAF3 - E8 60AFFEFD call Acid_bur.0041AA58
0042FAF8 - 8B55 F0      mov edx,[local.4]
0042FAFB - 8B45 F4      mov eax,[local.3]
0042FAFE - E8 F93EFDFF call Acid_bur.004039FC
0042FB03 - 75 1A      jnz short Acid_bur.0042FB1F
0042FB05 - 6A 00      push 0x0
0042FB07 - B9 CCFB4200 mov ecx,Acid_bur.0042FBCC
0042FB0C - BA D8FB4200 mov edx,Acid_bur.0042FB08
0042FB11 - A1 480A4300 mov eax,dword ptr ds:[0x430A4]
0042FB16 - 8B00      mov eax,dword ptr ds:[eax]
0042FB18 - E8 53A6FFFF call Acid_bur.0042A170
0042FB1D - EB 18      jmp short Acid_bur.0042FB37
0042FB1F - 6A 00      push 0x0
0042FB21 - B9 74FB4200 mov ecx,Acid_bur.0042FB74
0042FB26 - BA 80FB4200 mov edx,Acid_bur.0042FB80
0042FB2B - A1 480A4300 mov eax,dword ptr ds:[0x430A4]
0042FB30 - 8B00      mov eax,dword ptr ds:[eax]
```

- (4) 调试后尝试修改指令，使程序向弹出验证成功的弹窗分支执行，修改指令的截图如下（图 10）：



```
0042FADA - FF75 F8      push [local.2]
0042FADD - 8D45 F4      lea eax,[local.3]
0042FAE0 - BA 05000000 mov edx,0x5
0042FAE5 - E8 C23EFDFF call Acid_bur.004039AC
0042FAEA - 8D55 F0      lea edx,[local.4]
0042FAED - 8B83 E0010000 mov eax,dword ptr ds:[ebx+0x1]
0042FAF3 - E8 60AFFEFD call Acid_bur.0041AA58
0042FAF8 - 8B55 F0      mov edx,[local.4]
0042FAFB - 8B45 F4      mov eax,[local.3]
0042FAFE - E8 F93EFDFF call Acid_bur.004039FC
0042FB03 - 75 1A      jnz short Acid_bur.0042FB1F
0042FB04 - 90      nop
0042FB05 - 6A 00      push 0x0
0042FB07 - B9 CCFB4200 mov ecx,Acid_bur.0042FBCC
0042FB0C - BA D8FB4200 mov edx,Acid_bur.0042FB08
0042FB11 - A1 480A4300 mov eax,dword ptr ds:[0x430A4]
0042FB16 - 8B00      mov eax,dword ptr ds:[eax]
0042FB18 - E8 53A6FFFF call Acid_bur.0042A170
0042FB1D - EB 18      jmp short Acid_bur.0042FB37
0042FB1F - 6A 00      push 0x0
0042FB21 - B9 74FB4200 mov ecx,Acid_bur.0042FB74
0042FB26 - BA 80FB4200 mov edx,Acid_bur.0042FB80
0042FB2B - A1 480A4300 mov eax,dword ptr ds:[0x430A4]
0042FB30 - 8B00      mov eax,dword ptr ds:[eax]
```

寄存器 (FPU)

EAX 00002668

ECX 0042FB74 ASCII 54,"ry Again!"

汇编于此处: 0042FB04

nop

☒ 使用 NOP 填充

汇编 取消

EIP 0042A170 Acid_bur.0042A170

C 0 ES 0023 32位 0(FFFFFFFF)

P 1 CS 001B 32位 0(FFFFFFFF)

A 0 SS 0023 32位 0(FFFFFFFF)

Z 0 DS 0023 32位 0(FFFFFFFF)

S 0 FS 003B 32位 7FDD0000(FFF)

T 0 GS 0000 NULL

D 0

O 0 LastErr ERROR_SUCCESS (00000000)

EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)

ST0 empty -??? FFFF 00FF00FF 00FF00FF

ST1 empty -??? FFFF 00FF00FF 00FF00FF

ST2 empty -??? FFFF 00000000 00000000

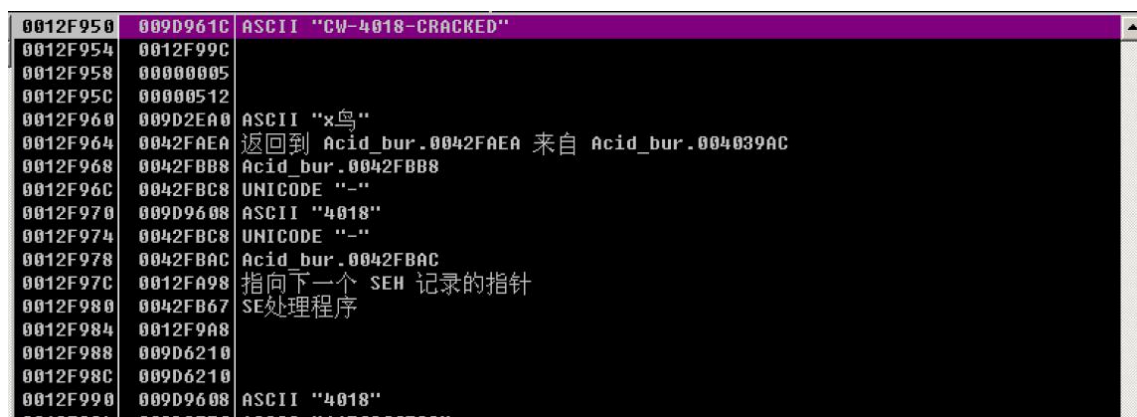
ST3 empty -??? FFFF 00000000 00000000

4. 注册码算法分析

- (1) 输入自己的学号和任意的序列号，弹出错误提示信息。
- (2) 定位注册码生成算法：从 MessageBoxA 的 call 指令向上回溯，指令附近有若干 call 指令，其中某条 call 指令调用的子程序为用户的输入与生成的序列号的比较，根据比较结果决定了程序接下来该弹出何种窗口，分析得到比较用户输入的序列号和正确序列号的 call 指令的地址为 0042FAFE，在此设置断点
- (3) 重新运行程序，触发断点后使用单步步入，分析子程序逻辑，子程序将正确的注册码与用户输入的注册码比较，子程序比较字符串时的汇编代码截图如下（图 11）：

004039FC	\$ 53	push ebx
004039FD	- 56	push esi
004039FE	- 57	push edi
004039FF	- 89C6	mov esi,eax
00403A01	- 89D7	mov edi,edx
00403A03	- 39D0	cmp eax,edx
00403A05	- 0F84 8F000000	je Acid_bur.00403A9A
00403A08	- 85F6	test esi,esi
00403A0D	- 74 68	je short Acid_bur.00403A77
00403A0F	- 85FF	test edi,edi
00403A11	- 74 6B	je short Acid_bur.00403A7E
00403A13	- 8B46 FC	mov eax,dword ptr ds:[esi-0x4]
00403A16	- 8B57 FC	mov edx,dword ptr ds:[edi-0x4]
00403A19	- 29D0	sub eax,edx
00403A1B	- 77 02	ja short Acid_bur.00403A1F
00403A1D	- 81C9	add edx,eax

- (4) 继续向上查找，有一条 `mov edx, 0x5` 指令，它的下一条指令为一个 `call` 指令，地址为 0042FAE5，调用的子程序用于生成正确的注册码，因此在 `call` 指令设置断点，进入子程序，子程序中存在循环，使用“单步步过”方式执行每条汇编语句，同时查看堆栈，发现在某个循环结束，得到正确注册码之后执行 `pop` 指令，该指令将正确序列号字符串的地址送入某个寄存器中，送入的寄存器是 edx，因此用户输入的用户名 1170030728，对应正确的注册码为 CW-4018-CRACKED，`pop` 指令执行前的堆栈截图如下（图 12）：



- (5) 最后给出将程序的正确用户名（自己的学号）和注册码填入检验成功的截图如下（图 13）：



三. 思考题

1. 逆向分析过程中，有着不同的关键逻辑汇编代码定位方法和不同汇编指令方式改变关键逻辑，描述至少一种其他方式。

Call 指令改为 nop，理论上讲还可以使用 jmp

定位时可以看栈中的返回地址找到调用者，或者……

2. 字符串是如何比较的？根据上面定位的 call 指令的子程序，简述序号生成算法。

正确注册码和用户注册码放入两个寄存器中，调用比较函数进行对比。序列号看起来是固定的 CW-4018-CRACKED。好像是先验证学号长度大于等于 4，小于 4 也会报错。