实验一 Acid burn.exe程序的逆向分析

# 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。

# 实验内容

1. **第一步：屏蔽软件启动的弹出窗口**
2. 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）：
3. 启动Ollydbg，加载程序；
4. 程序启动时会出现弹窗，截图如下（图2）：
5. 去除该弹窗，弹窗是通过调用 实现的，请采用 方式为此函数设置断点，截图如下（图3）：
6. 使用Ollydbg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： ，截图如下（图4）：
7. 选中call指令，用 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。）
8. **第二步：爆破法Crack使用单一序列号的验证**
9. 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 ，截图如下（图5）；
10. 向上回溯，找到该代码段的开始位置，对应的汇编指令为： ，地址为 。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和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）
11. 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 指令，call指令地址为 ，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）：
12. **第三步：爆破法Crack使用Serial/Name的验证**
13. 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）：
14. 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： ，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： ，堆栈的截图如下（图8）：
15. 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 指令，相应的汇编代码地址为： ，截图如下（图9）:
16. 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）：

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 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 ，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）：
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 ，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\_ \_，因此用户输入的用户名\_ \_\_，对应正确的注册码为\_ \_，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）：

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