南開大學

汇编语言与逆向技术课程实验报告

实验十： Capture The Flag

学　　院　网络空间安全学院

专　　业　　信息安全

学　　号　　2211044

姓　　名　　陆皓喆

班　　级　　信息安全

一、实验目的

1、熟悉静态反汇编工具 IDA Freeware；

2、掌握对二进制代码内部逻辑关系的分析；

3、掌握对二进制代码的修改和保存。

二、实验原理

1. CTF

CTF 是一种流行的信息安全竞赛形式，可意译为“夺旗赛”。其大致流程是，参赛团队之间通过进行攻防对抗、程序分析等形式，率先从主办方给出的比赛环境中得到一串具有一定格式的字符串或其他内容，并将其提交给主办方，从而夺得分数。

CTF 竞赛模式具体分为以下三类：

一、解题模式（Jeopardy）

在解题模式 CTF 赛制中，参赛队伍可以通过互联网或者现场网络参与，这种模式的 CTF 竞赛与 ACM 编程竞赛、信息学奥赛比较类似，以解决网络安全技术挑战题目的分值和时间来排名，通常用于在线选拔赛。题目主要包含逆向分析、漏洞挖掘与利用、Web 渗透、密码、取证、隐写、安全编程等类别。

二、攻防模式（Attack-Defense）

在攻防模式 CTF 赛制中，参赛队伍在网络空间互相进行攻击和防守，挖掘网络服务漏洞并攻击对手服务来得分，修补自身服务漏洞进行防御来避免丢分。

三、混合模式（Mix）

结合了解题模式与攻防模式的 CTF 赛制，比如参赛队伍通过解题可以获取一些初始分数，然后通过攻防对抗进行得分增减的零和游戏，最终以得分高低分出胜负。

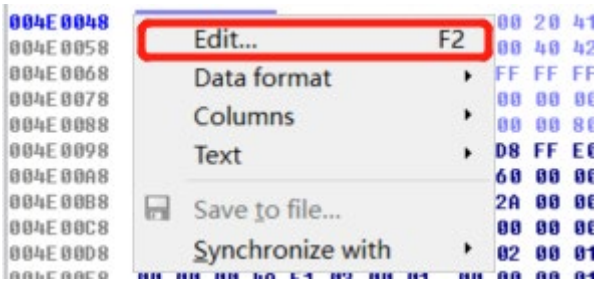
2. 解题

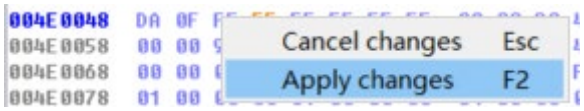
Flag 隐藏在 game.exe 的二进制代码中。通过对 game.exe 的修改，使 game.exe 能够顺利的执行，完成对 Flag 的解密。

1) 技巧 A：利用 IDA Pro 修改静态资源

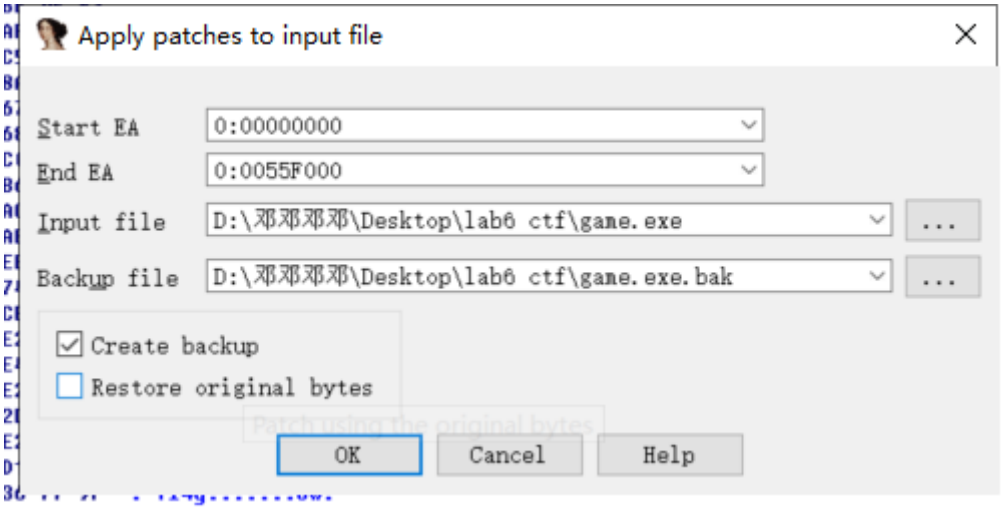
- 第一步，在反汇编代码中（IDA View）找到静态资源。

- 第二步，在十六进制视图中（Hex View）找到指定区域，右键选择 Edit 对资源进行修改。修改完毕后，右键选择 Apply changes 应用修改。





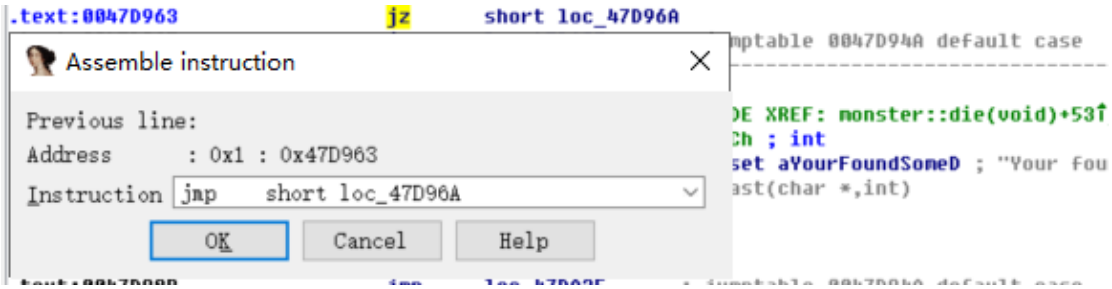
- 第三步，点击 Edit->Patch program->Apply patches to input file，建议选中创建备份的选项，完成修改。



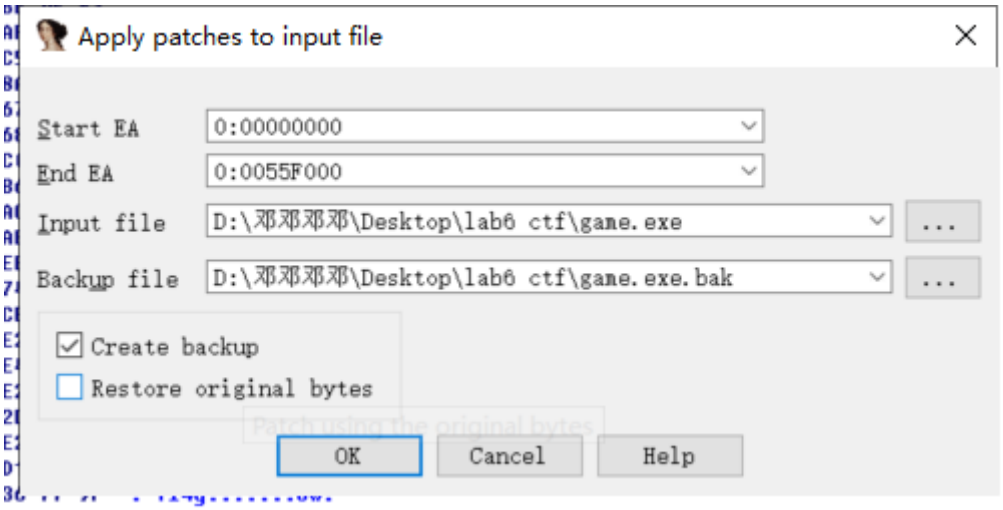
2) 技巧 B：利用 IDA Pro 修改汇编指令

- 第一步，在反汇编代码中（IDA View）找到需要修改的汇编指令。

- 第二步，点击 Edit->Patch program->Assemble，输入新的汇编指令。



- 第三步，点击 Edit->Patch program->Apply patches to input file，建议选中创建备份的选项，完成修改。



三、实验报告

3.1 逆向分析 game.exe 二进制代码的主要逻辑结构和重要数据。

首先，我在没有修改代码的情况下，进行了试玩，发现因为自身血量太低，第一关就被小怪 KO 了。

所以我发现想要通关的话，还是要使用我们的专业知识来对文件进行修改。

我们先打开 game.exe，观察其结构。我们能看到 text 段与 data 段，其中包含了一些重要信息。

```
.data:004E0034                  public __ZN3KEY2v4E
.data:004E0034 ; KEY::v4
.data:004E0034 __ZN3KEY2v4E     db 0E5h                 ; DATA XREF: KEY::writekey(int):loc_403DBA↑o
.data:004E0035                  db 0D3h
.data:004E0036                  db  8Fh
```

在这个版块中，我发现了 KEY4 这个关键信息，我猜测是一共有四个关卡，每通关一关，KEY 的位置就会往后跳一位。

我先想到的是需要修改自身血量与 BOSS 的血量，或者修改子弹伤害来通关。于是，我在板块中寻找到了以下的版块，里面有一些关键信息，比如说移动速度（_MOVE_SPEED）、最大血量(_MAX_HP)、初始血量(_INITIAL_HP)、防御值(_ARMOR)、子弹速度(_FIRE_SPEED)等等。

```
.data:004E0048 _MOVE_SPEED     dd 3.1415925            ; DATA XREF: mainloop(void)+12B7↑r
.data:004E0048                                         ; mainloop(void)+12D6↑r ...
.data:004E004C                 public _MAX_HP
.data:004E004C _MAX_HP         dd 0FFFFFF40h           ; DATA XREF: save(savedata &)+18↑r
.data:004E004C                                         ; apply_save(savedata)+1E↑w ...
.data:004E0050                 public _ARMOR
.data:004E0050 _ARMOR          dd 41200000h            ; DATA XREF: save(savedata &)+4A↑r
.data:004E0050                                         ; apply_save(savedata)+44↑w ...
.data:004E0054                 public _spawnX
.data:004E0054 ; float spawnX
.data:004E0054 _spawnX         dd 41200000h            ; DATA XREF: logic_init(void)+A5↑r
.data:004E0054                                         ; mainloop(void)+5DC↑w
.data:004E0058                 public _spawnY
.data:004E0058 ; float spawnY
.data:004E0058 _spawnY         dd 43960000h            ; DATA XREF: logic_init(void)+9F↑r
.data:004E0058                                         ; mainloop(void)+5FF↑w
.data:004E005C                 public _csize_x
.data:004E005C ; float csize_x
.data:004E005C _csize_x        dd 32.0                 ; DATA XREF: mainloop(void)+1E2↑r
.data:004E005C                                         ; mainloop(void)+271↑r ...
.data:004E0060                 public _csize_y
.data:004E0060 ; float csize_y
.data:004E0060 _csize_y        dd 42400000h            ; DATA XREF: mainloop(void)+1DC↑r
.data:004E0060                                         ; mainloop(void)+26B↑r ...
.data:004E0064                 public _last_combat
.data:004E0064 _last_combat    dd 0FFFFFF9Ch           ; DATA XREF: logic_init(void)+35↑w
.data:004E0064                                         ; mainloop(void)+16FC↑r ...
.data:004E0068                 public _FIRE_SPEED
.data:004E0068 _FIRE_SPEED     dd 8.0                  ; DATA XREF: shot(float,float,float,float,int)+1D6↑r
.data:004E0068                                         ; shot(float,float,float,float,int)+1FE↑r
.data:004E006C                 public _FIRE_SCOPE
.data:004E006C _FIRE_SCOPE     dd 800000.0             ; DATA XREF: mainloop(void)+2A4B↑r
.data:004E0070                 public _INITIAL_HP
.data:004E0070 _INITIAL_HP     dd 0FFFFFF20h           ; DATA XREF: data_init(void)+6↑r
```

我们接着查找有关 died 的信息。

```
.rdata:004EB41C ; const std::string aZKeyDecryptedC
.rdata:004EB41C aZKeyDecryptedC db 'Z KEY DECRYPTED. CONGRATULATIONS.',0Ah,0
.rdata:004EB41C                                        ; DATA XREF: mainloop(void)+E1↑o
.rdata:004EB43F                 align 10h
.rdata:004EB440 ; const char Str[]
.rdata:004EB440 Str             db 'You need to kill enough monsters!',0
.rdata:004EB440                                        ; DATA XREF: mainloop(void)+556↑o
.rdata:004EB462 ; const CHAR aResourceSoundT[]
.rdata:004EB462 aResourceSoundT db 'resource\sound\tp.wav',0
.rdata:004EB462                                        ; DATA XREF: mainloop(void)+593↑o
.rdata:004EB478 ; const char aZKeyDecrypting[]
.rdata:004EB478 aZKeyDecrypting db 'Z KEY DECRYPTING PROGRESS : 0%%',0
.rdata:004EB478                                        ; DATA XREF: mainloop(void)+645↑o
.rdata:004EB498 ; const char aZKeyDecrypting_0[]
.rdata:004EB498 aZKeyDecrypting_0 db 'Z KEY DECRYPTING PROGRESS : 25%%',0
.rdata:004EB498                                        ; DATA XREF: mainloop(void)+668↑o
.rdata:004EB4B9                 align 4
.rdata:004EB4BC ; const char aZKeyDecrypting_1[]
.rdata:004EB4BC aZKeyDecrypting_1 db 'Z KEY DECRYPTING PROGRESS : 50%%',0
.rdata:004EB4BC                                        ; DATA XREF: mainloop(void)+694↑o
.rdata:004EB4DD                 align 10h
.rdata:004EB4E0 ; const char aZKeyDecrypting_2[]
.rdata:004EB4E0 aZKeyDecrypting_2 db 'Z KEY DECRYPTING PROGRESS : 75%%',0
.rdata:004EB4E0                                        ; DATA XREF: mainloop(void)+6C0↑o
```

发现该处有关于游戏进程的提示，我们可以确定下来，该游戏是分为 4 部分，每部分完成后，都会输出进度，比如说在后期我进行游玩的时候的截图：

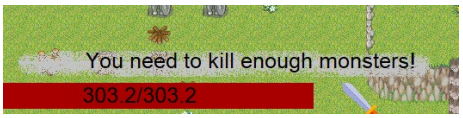
3.2 修改 game.exe 二进制代码， 在实验报告中说明代码修改的具体过程。

第一步，我选择的是修改自身的一些参数，比如说把自身的血量上限与初始血量修改了，这样我们就不会被怪物打死了，可以一直进行闯关。

```
.data:004E004C                 public _MAX_HP
.data:004E004C _MAX_HP         dd 0FFFFFF40h           ; DATA XREF: save(savedata &)+18↑r

.data:004E0070                 public _INITIAL_HP
.data:004E0070 _INITIAL_HP     dd 0FFFFFF20h           ; DATA XREF: data_init(void)+6↑r
```

这样就会发现，我们自身的血量条从原来的红色有数值，变成了现在的无数值灰色（我们无敌了）。



接下来，我发现这游戏的武器的使用十分耗费蓝条（应该是叫游戏中的钻石），而钻石又十分难以获取，所以我想到去 IDA 中进行代码的修改，来帮助我们获得无限的蓝条。

```
.rdata:004EB550 ; const char aThisSkillNeeds[]
.rdata:004EB550 aThisSkillNeeds db 'This skill needs %d MP!',0
```

我们发现这句话应该意思是“这个技能需要蓝条！”，所以 MP 应该是蓝条的意思。对应到游戏中是这样的：

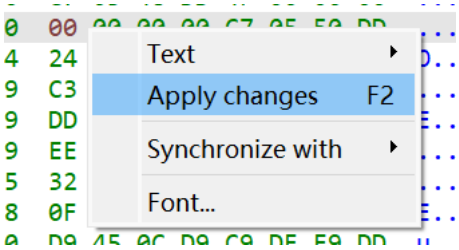
我们继续寻找一下 MP 的位置，发现在 main 函数中有这么一句话，指明了 MP 的位置与变量名。

```
.text:00407EBB                 mov     dword ptr [esp+4], offset aThisSkillNeeds ; "This skill needs %d MP!"
.text:00407EC3                 lea     eax, [ebp+Str]
.text:00407EC9                 mov     [esp], eax      ; Buffer
```

我们进行查找，找到武器的赋值语句：

```
mov     ds:dword_4FDD44, 1
mov     ds:_bullets, 14h
mov     ds:dword_4FDD48, 0
mov     ds:dword_4FDD4C, 5
mov     ds:dword_4FDD50, 0Ah
```

我们发现一技能（用刀砍）是耗费 0，但是伤害太低了，我们将最强的冰冻之术的蓝量的消耗改为 0 即可，这样就可以无限的放大招了！（喜）



上面是调整过程，选中 Apply changes 进行调整。

```
mov     ds:dword_4FDD44, 1
mov     ds:_bullets, 14h
mov     ds:dword_4FDD48, 0
mov     ds:dword_4FDD4C, 0
mov     ds:dword_4FDD50, 0Ah
```

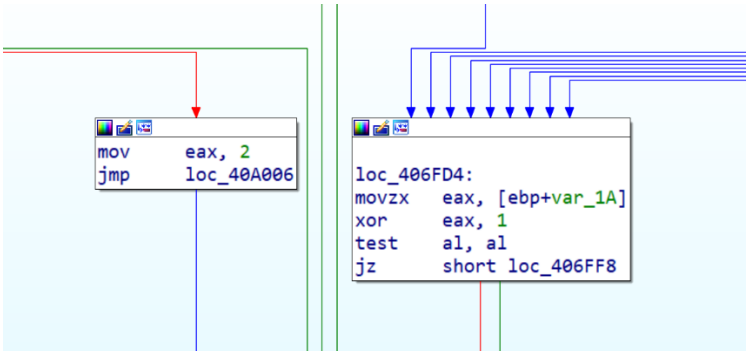
经过调整，我们发现大招的蓝量变成 0 了，这样就可以无限放大招了。

更改完上述属性之后，原则上来说我们已经无敌了，可以随便闯关。前三关靠手打完全可以通过，但是第四关会出现卡关的情况，把仅有的两只怪打死之后，还是会提示你：“You need to kill enough monsters!”。此时我们意识到，必须要修改相应的汇编指令了，来实现跳关功能。

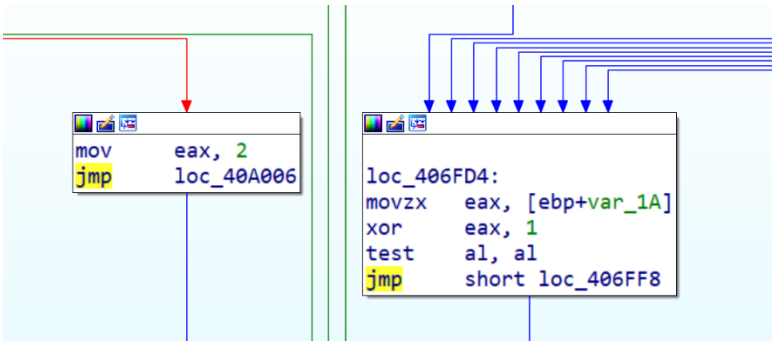
我们搜索“monsters”字符串，查看与 BOSS 有关的代码块。

```
mov     dword ptr [esp+4], 3Ch ; '<' ; int
mov     dword ptr [esp], offset Str ; "You need to kill enough monsters!"
call    __Z5toastPci    ; toast(char *,int)
jmp     loc_407314
```

我们往上寻找，发现上面有相关的代码块。



我们分析一下逻辑关系，发现只需要将上方的右边的代码块的最后一句的 jz 改成 jmp 就可以了，这样就可以实现无条件的跳转，实现了最后一关的直接通过，不需要刷更多的怪才能通关。



修改完毕后，我们重新进行游戏，最后顺利从第四关跳转回第一关的地图（但是没有怪），然后我们往海边走，就会触发最终剧情——跟女主相遇。游戏结束，我们可以获得相对应的 flag 即可。

3.3 提供最后取得 Flag 的截图。

如图所示，我们获得了最后的 flag——{a2fdkd80xo},游戏结束！