# Lower

## 拒绝服务

拒绝服务是攻击者通过极度消耗应用资源，以致程序崩溃或其他合法用户无法进行使用的一种攻击方式。

例如：下面代码片段中，解压文件前，未检查文件大小，攻击者可以通过提供一个超大文件，实施DOS攻击。

public long getWordCount() throws IOException{

FileReader f = FileReader(new InputStreamReader(new FileInputStream(textFile), charsetName));

String line = null;

StringBuffer sb = new StringBuffer();

while( (line = this.readLine()) != null) {//这里如果文件非常大的话，这个位置可能一直在执行，导致问题出现。

sb.append(line);

}

return StringUtils.getWordsCount(sb.toString());

}

## 不安全的随机数

Random类和Math.random()方法提供的随机函数为伪随机。在安全性要求较高的应用中，应使用更安全的随机数生成器，如java.security.SecureRandom类。

例如：下面代码片段中，使用java.security.SecureRandom来生成更安全的随机数。

import java.security.SecureRandom;

import java.security.NoSuchAlgorithmException;

// ...

public static void main (String args[]) {

try {

SecureRandom number = SecureRandom.getInstance("SHA1PRNG");

// Generate 20 integers 0..20

for (int i = 0; i < 20; i++) {

System.out.println(number.nextInt(21));

}

} catch (NoSuchAlgorithmException nsae) {

// Forward to handler

}

}

# Medium

## 系统信息泄露

系统数据或调试信息通过输出流或日志功能输出系统，将有助于攻击者了解系统并制定相应攻击计划。

例：下面代码片段中，通过标准错误流输出异常的堆栈信息，攻击者可能会利用这些堆栈信息制定相应的攻击计划。

try{

...

}catch(Exception e){

e.printStackTrace();

}

## 资源未释放

程序创建或分配流资源后，不进行合理释放，将会降低系统性能。攻击者可能会通过耗尽资源池的方式发起拒绝服务攻击。

例1：在下面Java方法中，创建I/O流对象后未进行合理释放，程序依靠Java虚拟机的垃圾回收机制释放I/O流资源，事实上，程序不能确定何时调用虚拟机的finalize()方法。在繁忙的程序环境下，可能导致Java虚拟机不能有效的使用I/O对象。

FileReader f = FileReader(new InputStreamReader(new FileInputStream(textFile), charsetName));

f.close();//这个close会关闭InputStreamReader 的流 ，但是FileInputStream 可能不会被关闭。

## 配置文件中的明文密码

配置文件中采用明文存储密码，将会降低系统安全性。如下面的配置，一旦站点被攻陷，MySQL数据库就直接被暴露了。应该使用加密码加密后写入配置文件，在程序中加密后使用，比如spring properties的隐式加密。

driverclass=com.mysql.jdbc.Driver

url=jdbc:mysql://localhost:3306/accountbook?&autoReconnect=true&failOverReadOnly=false&useUnicode=true&characterEncoding=utf8

username=root

password=4100107223

hibernate.dialect=org.hibernate.dialect.MySQLDialect

# Height

## 日期格式化异常

SimpleDateFormat类是线程非安全的，当多个线程同时使用getStringDate()方法的时候，可能出现格式化异常的问题。

public class DateHandler extends Date {

private static final long serialVersionUID = 1L;

private static SimpleDateFormat sdf = null;

public static String getStringDate(Date date,String pattern) {

sdf = new SimpleDateFormat(pattern);

return sdf.format(date);

}

## Double解析异常

程序调用Double的解析方法时，可能导致线程被挂起。java.lang.Double.parseDouble()方法解析位于[2^(-1022) - 2^(-1075) :2^(-1022) - 2^(-1076)]范围内的任何数字时可能导致线程被挂起，攻击者可以故意触发该漏洞执行拒绝服务攻击。该漏洞在java6 update24或更高版本中进行了修复。

下面代码片段中，使用了易受攻击的方法。

Double d = Double.parseDouble(request.getParameter("d"));

攻击者可发送 d 参数值位于该范围(例如 "0.0222507385850720119e-00306"）内的请求，致使程序在处理该请求时被挂起。

修复该缺陷的方式如下：

（1）验证传递给parseDouble数据的合法性。

（2）升级JDK版本到6 Update 24或更高版本。

## XML外部实体注入

攻击者通过操纵XML外部实体的URI,使其指向特定的文件，从而造成拒绝服务攻击或者程序崩溃。

例如：下面代码片段尝试对evil.xml文件进行解析

class XXE {

private static void receiveXMLStream(InputStream inStream,

DefaultHandler defaultHandler)

throws ParserConfigurationException, SAXException, IOException {

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser saxParser = factory.newSAXParser();

saxParser.parse(inStream, defaultHandler);

}

public static void main(String[] args)

throws ParserConfigurationException, SAXException, IOException {

receiveXMLStream( new FileInputStream("evil.xml"),

new DefaultHandler());

}

}

如果evil.xml文件中包含以下文本

<?xml version="1.0"?>

<!DOCTYPE foo SYSTEM "file:/dev/tty">

<foo>bar</foo>

...

SAX或者DOM解析器会尝试访问在SYSTEM属性中标识的URL,这意味着它将读取本地/dev/tty文件的内容。在POSIX系统中，读取这个文件会导致程序阻塞，直到可以通过计算机控制台得到输入数据为止。这样，攻击者可以使用这个恶意的XML文件来导致系统挂起，程序会受到XML外部实体注入攻击。