Fastjson\_1.2.24反序列化漏洞分析与复现

参考博客：<https://blog.csdn.net/qq_31615049/article/details/85013129>

环境搭建

编译器：IDEA2021

fastjson版本：1.2.24

导入fastjson的jar包

这里直接导入一个pom.xml文件，然后用maven自动下载对应的包即可

图形用户界面, 文本, 应用程序

描述已自动生成

导入之后是这样的

文本

描述已自动生成

2 创建一个java文件

电脑萤幕的截图

描述已自动生成

点击之后勾选Groovy选项，然后创建src文件夹；

接着再右键src文件夹，创建一个package包，最后右键package创建一个java文件即可。

环境搭建到这里就完成了

IDEA项目中的目录结构如下：  
文本

描述已自动生成

fastjson序列化

这里举一个例子来说明fastjson序列化

1 首先创建一个Student类

文本

描述已自动生成

2 然后再创建一个TestFastJson.java文件来对student类序列化

电脑萤幕的截图

描述已自动生成

3 详细讲解fastjson将对象序列化json字符串的过程

从TestFastJson.java文件一步一步展开分析

TestFastJson类的main函数中首先对Student对象进行了实例化，并为name、age赋值；

接着就是调用fastjson中的序列化相关函数对Student对象进行序列化；

JSON.toJSONString()方法会获取Student对象的变量名、值等内容，并且序列化成json字符串；

JSON.toJSONString()方法中如果使用了SerializerFeature.WriteClassName参数，则会在json字符串前端添加@type来指定具体的类名；

这里需要说明的是，该方法在序列化对象的时候会调用类中的get方法获取类中的变量名和变量值；

JSON.toJSONString()方法是如何获取到get方法的呢？这里用到的应该是字符串匹配的方法来匹配类中存在的所有get方法，从而获取到变量；

为什么能够匹配到get方法，是因为get方法的方法名是类中自动生成的。

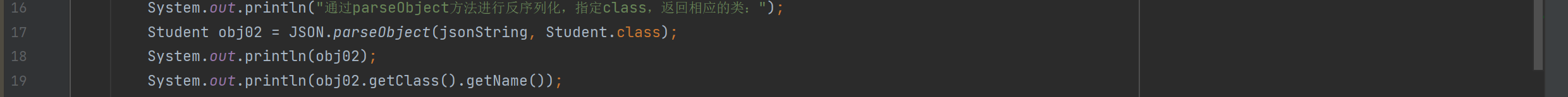
图形用户界面

描述已自动生成从上图红色方框中可以看到，当我们注释掉getAge()方法（或者修改方法名）后，执行后的结果没有getAge()方法的输出，并且在序列化的json字符串中也没有找到age变量和值，由此可以证明上面关于JSON.toJSONString()方法的说明是正确的。

fastjson反序列化

fastjson反序列化是将一个json字符串转换成对象的操作。

这里延续序列化的例子进行解释。



fastjson反序列化的关键方法是fastjson中的JSON.parseObject(jsonString, Student.class)方法，其中第一个参数是json字符串，第二个参数是指定的对象类型（即类名）；

由于实在看不懂fastjson底层代码，所以简单描述一下反序列化的大致操作吧...

首先将一个json字符串传给JSON.parseObject(jsonString, Student.class)方法，其中第一个参数是传入的json字符，第二个参数是指定返回的类（我们序列化的是一个实例化的类的对象），该参数如果省略的话是返回一个实例化的对象；

传入之后，会进行一些操作，这些操作比较复杂，但是总的目的是为了对json字符串中的内容进行各种匹配、编码、转义、过滤等等，以便能够获取到符合自动化匹配的字符，包括类名、变量名、变量值等；

通过对json字符串做上述操作后，通过@type来获取指定的类名，再通过字符串中的引号、冒号等来匹配获取变量名、变量值；

得到了这些信息之后，会调用指定类中的构造函数、get、set、is函数并执行一遍。

下面的图是指定了class参数和没指定class参数时JSON.parseObject()方法的调用：

电脑萤幕的截图

描述已自动生成

可以从下图看一下两者之间反序列化的区别在哪里

图形用户界面, 文本, 应用程序

描述已自动生成

可以看出，指定了class后print()输出的是一个Student类，没有指定时print()返回的是一个Student对象：{"name":"5wimming","age":10}；而且反序列化的过程中确实是将类中的构造函数、get、set、is方法都执行了一遍。

不仅如此，还可以通过.get()方法来获取变量值。

反序列化的过程我感觉就是，先通过@type拿到指定的类，然后再将类中的四大函数存入hashmap，最后将json字符串中的变量名/变量值键值对赋给@type指定的类的实例化对象；在调用反序列化对象obj的时候相当于调用了Student类的实例化对象。

需要特别注意的是，fastjson不能够反序列化私有化的属性，但是可以通过Feature.SupportNonPublicField参数来反序列化私有化属性：

JSON.parseObject(jsonString, Student.class, Feature.SupportNonPublicField)

自动化匹配set方法的条件如下：

1. 方法名长度大于4

2. 非静态方法

3. 返回值为void或当前类

4. 方法名以set开头

5. 参数个数为1

自动化匹配get方法的条件如下：

1. 方法名长度大于等于4

2. 非静态方法

3. 以get开头且第4个字母为大写

4. 无传入参数

5. 返回值类型继承自Collection Map AtomicBoolean AtomicInteger AtomicLong

漏洞分析

经过上面的序列化和反序列化分析，已经可以推断出反序列化漏洞的原因了；

一种是因为反序列化过程中@type指定的类中的四大函数存在危险操作；

另一种是因为自动化匹配四大函数的过程中，有满足自动调用条件的其他函数被误认为是get、set、is、构造方法而被调用，该函数正好可以被利用。

漏洞复现及利用

fastjson1.2.24版本反序列化漏洞利用至少有三种方式。

**1 基于JNDI注入**

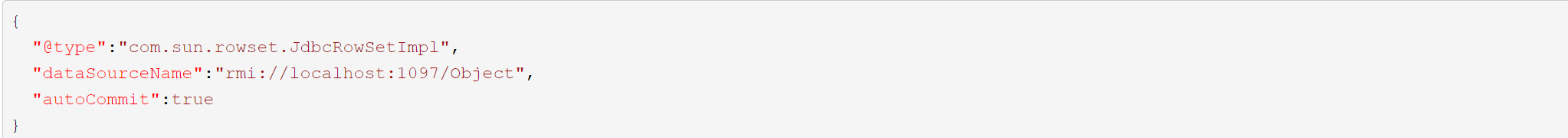
JNDI即Java Naming and Directory Interface，Java命名和目录接口，JNDI提供了很多实现方式，主要有RMI，LDAP，CORBA等。JNDI提供了一个统一的外部接口，底层服务实现是多样的。

以RMI为例，RMI Registry有Name到对象的映射关系，应用通过java.rmi.naming#lookup方法向Registry发出查询请求，得到映射关系，再连接RMI Server实现远程方法调用。

如果说其lookup方法的参数是我们可以控制的，可以将其参数指向我们控制的Registry，那我们可以在Registry绑定一个指向远程类的Reference对象（当对象为Reference类型的时候，应用会加载远程类并实例化），远程类的静态代码块及构造方法均可控，从而导致任意代码执行。

可供利用的类有：com.sun.rowset.JdbcRowSetImpl、org.springframework.jndi.support.SimpleJndiBeanFactory、com.mchange.v2.c3p0.JndiRefForwardingDataSource、org.apache.ibatis.datasource.jndi.JndiDataSourceFactory、org.hibernate.jmx.StatisticsService；

下面通过com.sun.rowset.JdbcRowSetImpl类来构造poc：



fastjson在反序列化上图中的字符串时，会构造一个JdbcRowSetImpl类的对象；接着调用类中的setter方法为类中的变量赋值（变量：dataSourceName、autoCommit）；在调用setDataSourceName()方法时会通过connect方法调用lookup方法来查询dataSourceName是否存在，如果该地址存在则会远程调用该地址指向的远程类，该类是攻击者构造的恶意类。

**2 基于ClassLoader**

这种利用方式是通过com.sun.org.apache.bcel.internal.util.ClassLoader类加载器实现的。

该类加载器的加载机制在遇到一些特殊的类名比如class\_name以$$BCEL$$开头的类，则会调用createClass方法去解析class\_name，在createClass方法中会将$$BCEL$$之后的字符解码成字节数组，并将这个BCEL编码后的类加载到虚拟机中。换言之，我们可以构造className为一个特殊的BCEL编码字符串时，通过这个类加载器来实现对自定义类的加载。

图片包含 图形用户界面

描述已自动生成

driverClassLoader和driverClassName都是json传入，外部可控，那么若将driverClassLoader设置为com.sun.org.apache.bcel.internal.util.ClassLoader，driverClassName设置为经BCEL编码后的自定义类，那么就实现了在反序列化时加载自定义类的目的。于是攻击者可以在static代码块中编写恶意代码，将其进行BCEL编码，在类初始化时实现恶意代码执行。

**3 基于TemplatesImpl**

com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl这个类有个比较特殊的能力，可以解析然后加载实例化\_bytecodes变量中的字符数组，\_bytecodes的值是可控的，所以攻击者只要构造恶意类对应的字符数组，然后通过getOutputProperties方法触发恶意类的加载及实例化，同样实现了远程代码执行的效果。

图形用户界面, 应用程序

描述已自动生成

这种利用方式需要在解析json串时设置Feature.SupportNonPublicField，而业务同学在使用fastjson时往往会直接按照默认参数调用parseObject方法，所以略为鸡肋。