安全小课堂第136期【 浅谈Java原生反序列化漏洞】

JSRC 京东安全应急响应中心 4月15日

上周五在QQ群里
讲师就本期主题和大家进行了**交流**
小妹为大家整理了课堂内容并**沉淀**
如果有所收获的话也可以将它**分享**
让更多的人加入JSRC安全小课堂

136期 浅谈Java原生反序列化漏洞

讲师介绍

Venscor，京东安全工程师，擅长Java代码审计，安全分析与加固，移动安全

背景介绍

Java 序列化是指把 Java 对象转换为字节序列，便于保存在内存、文件、数据库中，而Java 反序列化是指把字节序列恢复为 Java 对象的过程。java的反序列化漏洞波及范围广，危害大，可以执行命令，甚至直接getshell。

课堂内容

京安小妹

什么是Java的序列化和反序列化，以及为什么需要序列化/反序列化?

Venscor

从广义角度，Java的序列化与反序列化是一种编码机制。反序列化就是将Java中的对象按照一定的编码协议编码成字符串的过程。相反，序列化就可以理解成将字符串解码成Java对象的过程。我们知道，对象是Java的一种内存抽象模型，对象只能存在于Java的运行时，换句话说，对象只能够被Java运行中的程序处理。除了本地程序，当需要跨网络传输数据，或者将数据持久化时，就已经脱离了Java的运行环境，只能够以字符串的形式进行传输或者存储。这时候，就需要一种转换机制，实现Java对象和字符串的转换。针对转换的协议差异，有不同的序列化/反序列化解决方案。例如，JDK为我们提供了自带的序列化/反序列化机制，即我们这次讲的Java原生反序列化机制。还有转出成Json的fastjson、Jackson

组件，转换成xml的XStream等。

京安小妹

Java序列化/反序列化过程是怎样的?

Venscor

网上不少文章介绍反序列化漏洞时，基本不讲或很少讲序列化/反序列化过程。从个人学习经验来谈，了解清楚序列化/反序列化过程对理解漏洞是十分有帮助的，尤其是反序列化过程。篇幅原因，笔者这里只介绍反序列化过程，关于序列化过程，大家可以按照思路自行思考研究，相信会有一定的收获。

首先，首先编写一个反序列化的demo：

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

    /*under attacker's control*/
    File f =new File(args[0]);
    try {
        FileInputStream fis = new FileInputStream(f);
        ObjectInputStream ois=new ObjectInputStream(fis);
        /*where vul produce*/
        TestSerial tt = (TestSerial) ois.readObject();
        System.out.println(tt.name);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

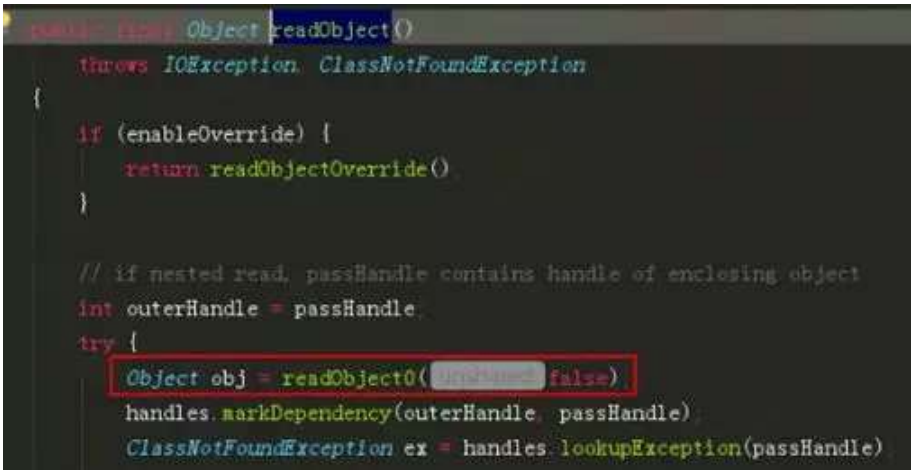
从Developer角度，反序列化过程仅仅是调用了一个API。我们要做的，就是看看调用这个API的背后发生了什么。在跟踪调用过程之前，我们做一个思考：如果让我们自己来实现反序列化，抛开编码规则，我们怎么从包含对象全部信息的字符串来生成一个Java对象？熟悉Java的同学可能会想到两种方法。

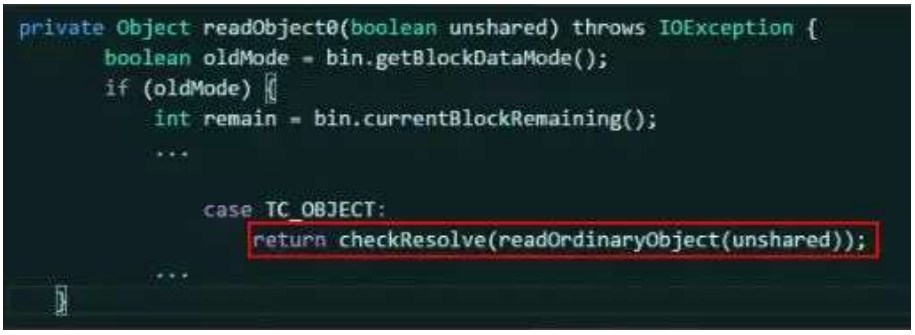
(1)一是从字符串中解析出对象对应的类的信息，然后调用对象的无参构造器，最后调用各种Setter方法来设置对对象的field属性。这种方法有一个壁垒，就是生成对象的field时，如果对应的Setter不是public，Setter方法将不能在package外调用，因此对象的field属性无法设置。

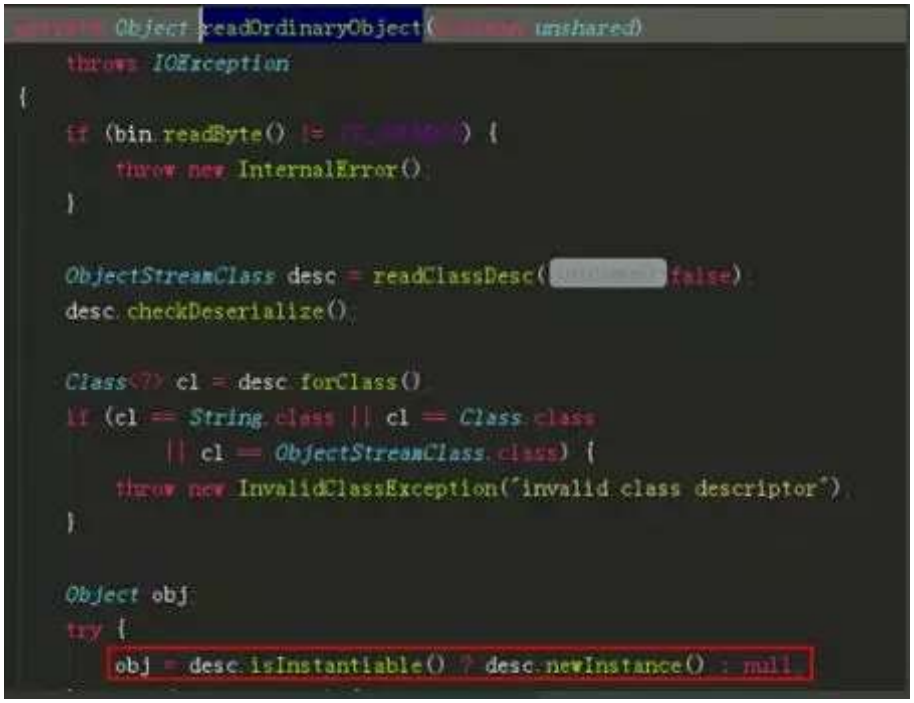
(2)另一种是利用Java的反射机制，这一点也是很容易想到的，Spring

的依赖注入技术中就大量使用了Java的反射机制来从xml文件中生成对象。当然，反射还是调用了构造器和Setter。

那么，我们就看下Java原生的反序列化过程是不是通过上面的两种方法。于是跟踪readObject()的实现过程。（一段枯燥无味的跟踪，需要耐心）







看到这个newInstance()方法，我们第一反应就是Class类的newInstance()方法，也就是通过反射调用构造器的过程。于是心中一乐，感觉找到新大陆，人生似乎到达了高潮。于是编码验证想法，在构造器中加一个弹计算器代码，看计算器是否弹出。

```
public TestSerial() {
    System.out.println("hahahaha");
    try {
        Runtime.getRuntime().exec("calc");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
FileInputStream fis = new FileInputStream(f);
ObjectInputStream ois=new ObjectInputStream(fis);
/*where vul produce*/
TestSerial tt = (TestSerial) ois.readObject();
```

十分不幸的是，计算器没有弹出。于是，怀疑一定有鬼，肯定是鬼。在缺定了不是鬼的情况下，回到上面的newInstance()方法。

```
Object newInstance()
    throws InstantiationException, InvocationTargetException,
           UnsupportedOperationException
{
    if (cons != null) {
        try {
            return cons.newInstance();
        } catch (IllegalAccessException ex) {
            // should not occur, as access checks have been suppressed
            throw new InternalError(ex);
        }
    } else {
        throw new UnsupportedOperationException();
    }
}
```

还在调用newInstance()，那就继续跟进：

```
private static Constructor<?> getSerializableConstructor(Class<?> cl) {
    Class<?> initCl = cl;
    while (Serializable.class.isAssignableFrom(initCl)) {
        if ((initCl = initCl.getSuperclass()) == null) {
            return null;
        }
    }
    try {
        Constructor<?> cons = initCl.getDeclaredConstructor((Class<?>[]) null);
        int mods = cons.getModifiers();
        if ((mods & Modifier.PUBLIC) != 0 ||
            ((mods & (Modifier.PRIVATE | Modifier.PROTECTED)) == 0 &&
             !packageEquals(cl, initCl)))
        {
            return null;
        }
        cons = reflFactory.newConstructorForSerialization(cl, cons);
        cons.setAccessible(true);
        return cons;
    } catch (NoSuchMethodException ex) {
        return null;
    }
```

原来发现，此newInstance()非我们想的彼newInstance()，所以，看代码也不能以貌取人，还是要进去看内在。原来Java反序列化时，JDK提供单独的构造器，方法中默认构造器并不会被调用。利用这个构造器生成对象的过程就没必要再去跟了，再跟下去就是通过Java字节码组建对象模型了。知道这个结论对于我们理解反序列化就足够了。

我们专门提一下结论：

- java反序列化生成对象时，会产生一种新的构造器；
- 利用新的构造器来生成对象，并且不会调用Setter方法，并且生成对象的原始数据都在之前序列化的String里。

京安小妹

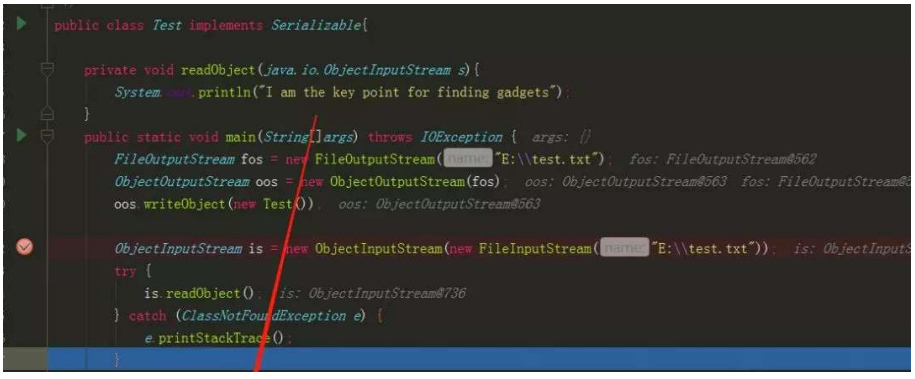
反序列化漏洞原理是怎样的？

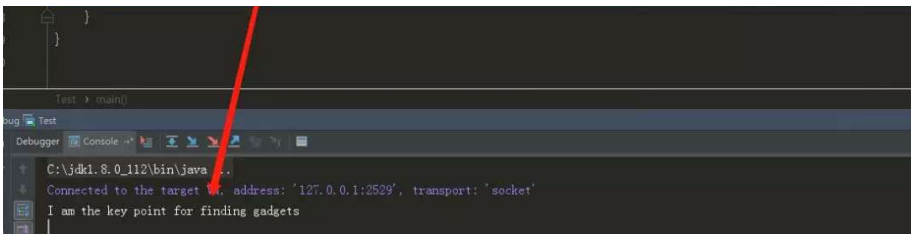
Venscor

反序列化的漏洞原理其实简单的，复杂的是如何找到可以利用的调用链，我习惯用二进制的gadget来描述这个调用链。首先，当业务直接调用了readObject()，且被反序列化的字符串可控并且没有限制黑名单时，就造成了漏洞。当然，漏洞可利用还需要运行环境中有对应的gadget库。一个写出漏洞的Demo如下：

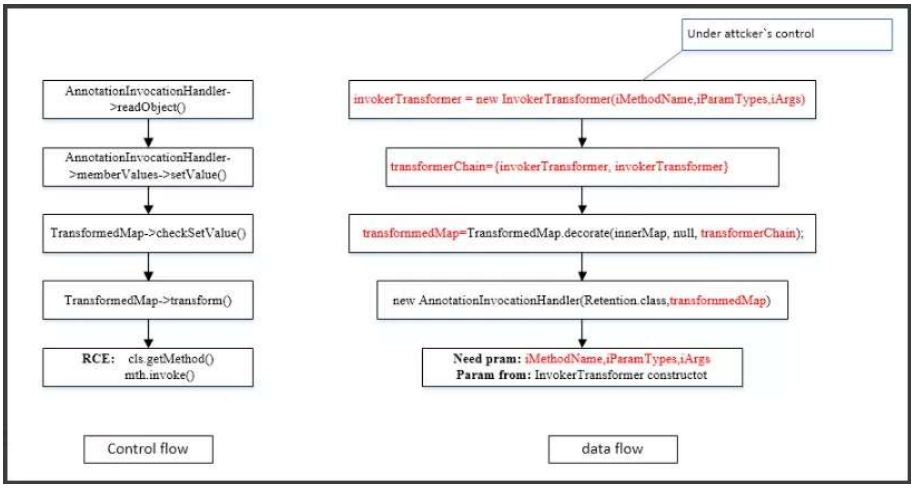
```
public static void main(String[] args) {
    /*under attacker's control*/
    File f =new File(args[0]);
    try {
        FileInputStream fis = new FileInputStream(f);
        ObjectInputStream ois=new ObjectInputStream(fis);
        /*where vul produce*/
        ois.readObject();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
}
```

接下来，就聊一下这样程序，当运行时中存在对应的可利用gadget库时，如何造成RCE。如果真的从刨根问底的角度，讲实话还没有这个实力。有些结论，我们需要站在巨人的肩上：被反序列化的对象，生成对象时，如果该类实现了readObject()方法，该类的这个方法将在反序列化时被调用。





那么剩下的过程，就是寻找gadget，如果有个类实现了readObject()，并且内部调用了什么Runtime.exec(cmd)，并且cmd也是可控的就好了。可惜理想很丰满，是现实很骨感。直接写出这样的类的程序员可以拉出去枪毙10分钟了。真正可用的gadget常常是隐藏很深，很难发现的。我分析过几个现成的gadget过程，这里就不列漫长的调用了，网上也有不少博客文档，有点耐心都能够分出来。这里给大家分享一个当时写的CommonCollections的gadget的数据流和控制流图，可以帮助分析。



京安小妹

黑白盒角度如何挖掘反序列化漏洞?

Venscor

实际上，对于纯业务代码，存在Java原生反序列化的漏洞不会特别多，大家可以思考一下为什么。 对于这类漏洞的挖掘，其实我想分三个角度讲。

(1)白帽子角度：

要么所有参数盲打，所以先从前端流量看有没有Java序列化标致：ac ed 00 05或rO0AB。选哪一种其实是一种博弈。推荐大神的工具给大家https://github.com/mbechler/serianalyzer: 。有些SRC不许用扫描器，不要犯法!

(2) 甲方安全工程师角度：

黑盒：同白帽子

白盒：从readObject()开始审计

入侵检测：匹配流量特征（ac ed 00 05或rO0AB）、Runtime字符串被序列化后的特征

(3)安全研究角度：

寻找新的CVE：对一些开源应用开启审计，从readObject()开启生涯。

寻找新的gadget：需要辅助一些静态代码审计工具，大量跑库，然后去确认。（食物链顶端的人忽略我这种渣渣~）

京安小妹

反序列化漏洞如何防御？

Venscor

懂安全业务：白名单干干干（这种不存在的，大家放心吧）

安全工程师：复写原生Java的反序列化类，然后加黑名单并维护黑名单。（这里其实也有缺陷，大家可以站在甲方角度思考一下）

写在最后：笔者能力有限，如有错误，还请谅解，欢迎指正。

本期小课堂就结束啦

感谢讲师的无私分享

感谢大家的参与互动

交流

开课时间：周五下午15:30

QQ开课群：464465695

留言：针对本期主题内容，你还有什么疑问吗？欢迎留言交流~

这是**JSRC安全小课堂**

持续陪伴你的第

***3*年 *36*天**

交流、沉淀、分享

扫描上方二维码，获取小课堂往期合辑