安全小课堂第103期【web漏洞挖掘之JAVA反序列化漏洞】

京东安全应急响应中心 7月24日

Java反序列化漏洞是近一段时间里一直被重点关注的漏洞，自从 Apache Commons-collections 爆出第一个漏洞开始，围绕着Java反序列化漏洞的事件就层出不穷，为了详细了解Java反序列化漏洞的成因和原理。

JSRC **安全小课堂第103期**，邀请到**九歌**作为讲师就**JAVA反序列化漏洞**为大家进行分享。同时感谢朋友们的精彩讨论。

Java反序列化漏洞的背景？

京安小妹

九歌：

Java 序列化是在 JDK 1.1 中引入的，是 Java 内核的重要特性之一。Java 序列化将java对象转换为二进制流，反序列化则是将二进制流转换为实际程序中使用的 Java 对象的过程。通过系列化将java对象持久化，便于传输和存储。

可序列化的对象需要实现 java.io.Serializable 接口或者 java.io.Externalizable 接口。实现该接口只是为了声明该Java类的对象是可以被序列化的。实际的序列化和反序列化工作是通过ObjectOuputStream和ObjectInputStream来完成的。

ObjectOutputStream 的 writeObject 方法可以把一个Java对象写入到流中，ObjectInputStream 的 readObject 方法可以从流中读取一个 Java 对象。在写入和读取的时候，虽然用的参数或返回值是单个对象，但实际上操纵的是一个对象图，包括该对象所引用的其它对象，以及这些对象所引用的另外的对象。Java会自动帮你遍历对象图并逐个序列化。

举一个序列化的简单例子

```
//定义person1对象
```

```
Person person1 = new Person();
person1.name = "九歌";
// We'll write the serialized data to a file
FileOutputStream fos = new FileOutputStream("object.ser");
ObjectOutputStream os = new ObjectOutputStream(fos);
//writeObject()方法将person1对象写入object.ser文件
os.writeObject(person1);
os.close();
//从object.ser文件中反序列化person1对象
FileInputStream fis = new FileInputStream("object.ser");
ObjectInputStream ois = new ObjectInputStream(fis);
// Read the object from the data stream, and convert it back to a String
Person objectFromDisk = (Person)ois.readObject();
System.out.println(objectFromDisk.name);
ois.close();
```

这是个简单的系列化，反序列化例子。

讲师

Java反序列化漏洞的产生原因及利用场景？

京安小妹

九歌：

九歌：

在读取java反序列化的数据中，没有对数据进行校验。我们可以向这个序列化接口传输任意构造的对象数据；而当运行环境中存在漏洞jar包时，攻击者可以精心构造反序列化对象并执行恶意代码。

第三方库中类的帮忙，使漏洞一下子爆发了。

Apache Commons Collections库是Apache公司开发的公共工具库中的一个，里面封装了一些java集合框架操作，实现了对Map，Set，List等数据集合的扩展。大量地应用在java程序开发过程中。JBoss RMI漏洞因为包含了Apache Commons Collections第三方库。

1、 敏感的InvokerTransformer的transform(Object input)方法，可以通过调用Java的反射机制来调用任意函数。

2、 ConstantTransformer中的transform(Object input)方法只是返回我们传进去的对象，可以通过ConstantTransformer控制InvokerTransformer中transform方法的input参数这个Object类型对象。

3、 ChainedTransformer这个类串联所有Transformer的transform()方法。

这样就可以形成我们一个执行任意代码的执行链了：

类TransformedMap，用来对Map进行某种变换。TransformedMap的setValue()触发Transformer链的transform()方法。流程即为：setValue ==> checkSetValue ==> valueTransformer.transform(value)。

TransformedMap的decorate()函数可以传入Transformer生成变换后的TransformedMap。

AnnotationInvocationHandler类的readObject()方法触发Map的setValue()。

当java每次读取序列化对象时，就会触发命令执行的Transform链。这就找到了一个存在漏洞的jar包。

理清楚这些调用链，就能构造恶意的序列化对象了

主要有这4个利用场景

1、信息泄露:被系列化的对象包含敏感数据的话，存在信息泄露的风险。

2、伪造：因为反序列化过程中没有对数据进行校验,可以篡改被系列化的二进制流，来进行数据伪造。

3、拒绝服务: 当篡改的数据不符合序列化对象的格式要求时候，可能会导致在反序列化对象的过程中抛出异常，从而拒绝服务。

4、命令执行：当反序列化对象时的运行环境中存在有漏洞的 jar 包（比如 commons Collections 低版本），攻击者通过构造恶意数据流可以达到命令执行的效果。

讲师

如何挖掘Java反序列化漏洞?

京安小妹

九歌：

1、确定反序列化输入点：首先应找出readObject方法调用

1>源码中搜索readObject()方法调用的地方

2>对该应用进行网络行为抓包，寻找序列化数据：java序列化的数据一般会以标记（ac ed 00 05）开头，base64编码后的特征为rO0AB。

2、观察反序列化时的readObject()方法是否重写，重写中是否有设计不合理，可以被利用之处。

代码审计时还有：

ObjectInputStream.readObject

ObjectInputStream.readUnshared

XMLDecoder.readObject

Yaml.load

XStream.fromXML

ObjectMapper.readValue

JSON.parseObject

...

3、查看项目工程中是否引入可利用的commons-collections 3.1、commons-fileupload 1.3.1等第三方库。

4、查找Java RMI服务（Java RMI的传输100%基于反序列化），Java RMI的默认端口是1099端口（jboss 1090则是RMI服务端口）。RMI命令执行有两个条件：1)、RMI 对外开放；2)、系统环境中存在有漏洞 Jar 包。

检查代码中是否使用了以下方法：

LocateRegistry.getRegistry （默认端口 1099）

LocateRegistry.createRegistry

Java反序列化漏洞的攻击检测思路?

九歌：

1、大体攻击思路如下：

寻找序列化API接口->构造恶意的系列化对象->向服务端发送系列化对象->查看执行结果

RMI：如使用registry.bind()函数便可将攻击payload发送到RMI服务中。

不过具体攻击方法过程也是有不同的，比如有被动式的：通过JRMP协议达到执行任意反序列化。

2.服务器端使ysoserial的 JRMP对1099 端口进行监听。

ysoserial.exploit.JRMPListener会将含有恶意代码的payload发送回请求方。发送封装JRMPClient生成的payload到受害方，受害方通过JRMPClient(反序列化得到)连接JRMPListener服务获取含有恶意代码的payload进行反序列化时，触发REC。

WebLogic反序列化漏洞CVE-2018-2628利用RMI的缺陷；通过T3协议在Weblogic Server中执行反序列化操作就是此方法。

3、JNDI调用RMI的方式，需在客户端建立RMI服务和供恶意类下载的http服务：

4、 ysoserial是一款非常好用的Java反序列化漏洞检测工具，该工具通过多种机制构造POC；包含(weblogic、jboss、websphere、jenkins)UI版的检测工具网上也很多；网上各种自编写的python poc也很多。

5、 提供2个反序列化漏洞扫描工具：

https://github.com/nccgroup/freddy burpsuite的插件，支持JSON and XML serialisation

https://github.com/andresriancho/w3af/blob/develop/w3af/plugins/audit/deserialization.py w3af插件

讲师

Java反序列化漏洞的防御手段?

京安小妹

九歌:

1、通用措施

(1)、对序列化的流数据进行加密

(2)、在传输过程中使用 TLS 加密传输

(3)、对序列化数据进行完整性校验

2、针对信息泄露

使用 transient 标记敏感字段，这样敏感字段将不进行序列化。

3、针对序列化对象属性的篡改

可以通过实现 validateObject 方法来进行对象属性值的校验。

步骤如下： 实现 ObjectInputValidation 接口并重写 validateObject 方法；实现 readObject 方法，并注册 Validation。

其验证时机是在读取流之后，所以只能够对正常的序列化对象进行验证

4、类白名单校验

通过重写 ObjectInputStream 的 resolveClass() 方法来实现。这样，我们需要自定义一个对象流读取类继承自 ObjectInputStream

然后，在反序列化的时候，使用自定义的 SecObjectInputStream 。

```
public class AntObjectInputStream extends ObjectInputStream{
    public AntObjectInputStream(InputStream inputStream)
            throws IOException {
        super(inputStream);
    }

    /**
     * 只允许反序列化SerialObject class
     */
    @Override
    protected Class<?> resolveClass(ObjectStreamClass desc) throws IOException,
            ClassNotFoundException {
        if (!desc.getName().equals(SerialObject.class.getName())) {
```

```
            throw new InvalidClassException(
                    "Unauthorized deserialization attempt",
                    desc.getName());
        }
        return super.resolveClass(desc);
    }
}
```

5、RASP黑名单检测

Hook java.io.ObjectInputStream.resolveClass(ObjectStreamClass)函数

```
var deserializationInvalidClazz = [
    'org.apache.commons.collections.functors.InvokerTransformer',
    'org.apache.commons.collections.functors.InstantiateTransformer',
    'org.apache.commons.collections4.functors.InvokerTransformer',
    'org.apache.commons.collections4.functors.InstantiateTransformer',
    'org.codehaus.groovy.runtime.ConvertedClosure',
    'org.codehaus.groovy.runtime.MethodClosure',
    'org.springframework.beans.factory.ObjectFactory'
     ...

]
```

黑名单不全存在被绕过风险

6、针对RMI

(1)、使 RMI 只开放在内网

(2)、升级本地的 jar 包

7、升级漏洞版本

讲师

本期JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。

简历请发送：cv-security@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心