JDK解决反序列化的方法

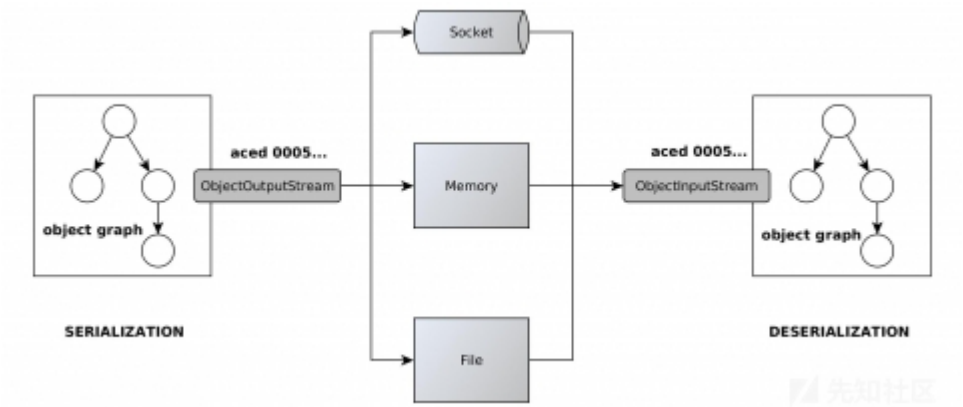
niexinming / 2018-11-09 09:02:00 / 浏览数 3397 技术文章 技术文章 顶(0) 踩(0)

翻译自： https://access.redhat.com/blogs/766093/posts/3135411
翻译： 聂心明

Java反序列化漏洞已经是过去两年安全圈里面最热的流行词了，因为每一个使用原始java序列化的框架都会受到反序列化攻击。一开始，还有很多不同的方法去试图解决这个
https://github.com/kantega/notsoserial ，https://github.com/Contrast-Security-OSS/contrast-rO0 ， https://github.com/mbechler/serianalyzer
）。这篇文章着重讲java反序列化漏洞和解释oracle在最新的jdk中提供了怎样的缓解措施。

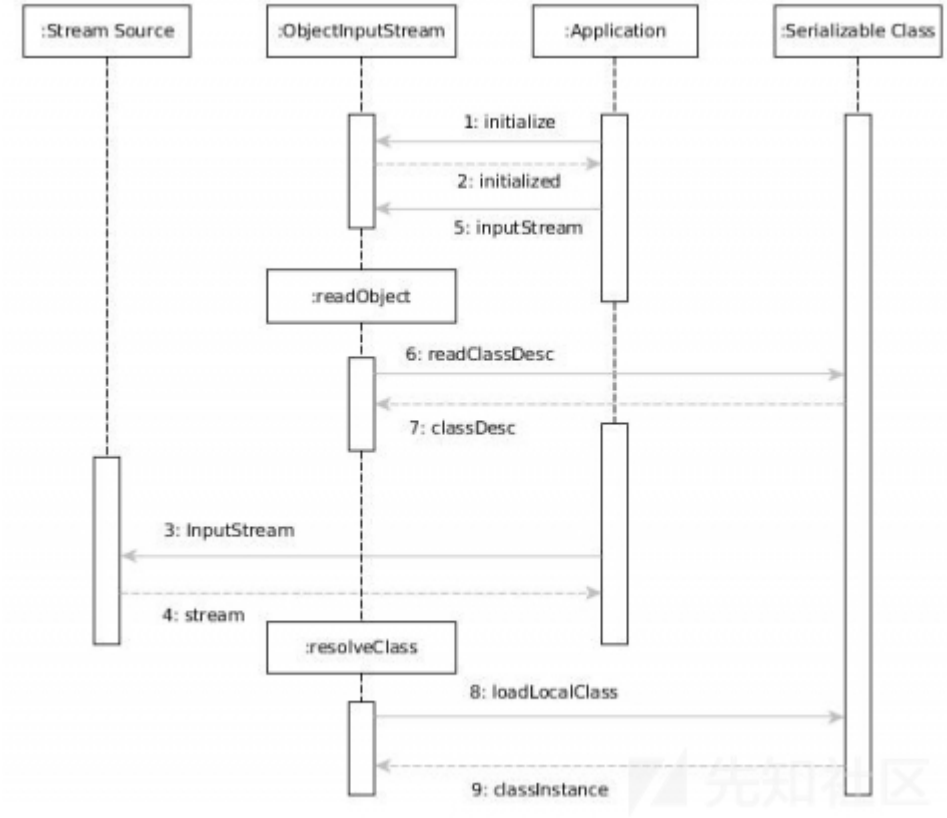
背景

让我们回顾java反序列化的进程。java序列化（ https://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html ）
是Java内置的功能，这个功能可以把java对象转换成二进制数据，也可以把二进制数据转换成对象。通过调用serialization
把对象转换成二进制数据，通过调用deserialization把二进制数据转换成java对象。在企业环境中，能直接存储和恢复对象的状态是构建分布式系统的关键因素。比如，JMS
（ https://en.wikipedia.org/wiki/Java_Message_Service ）通过序列化把流对象数据通过通信线路送到目的地。 RESTful（
https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html ）客户端应用可能通过序列化把 OAuth token（
https://www.oauth.com/oauth2-servers/access-tokens/ ）对象存储在硬盘上，以便做进一步的身份验证。java的远程方法调用JMI ，（
https://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmiTOC.html ）在JVM之间直接使用序列化互相通信。除了这些还有其他使用序列化的例子。



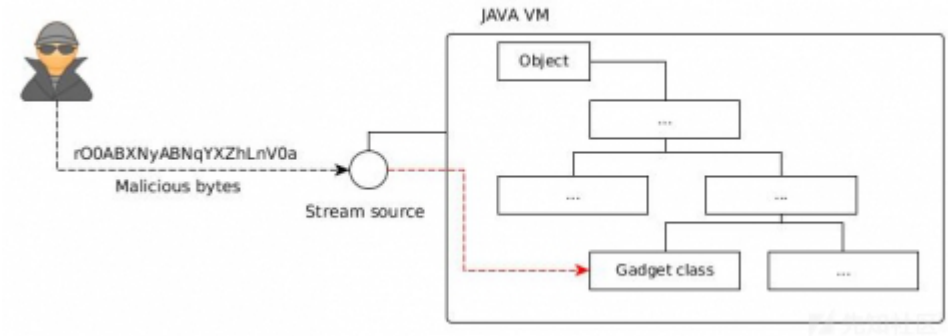
检查流

当应用代码触反序列化的时候， ObjectInputStream （ https://docs.oracle.com/javase/8/docs/api/java/io/ObjectInputStream.html
）将对象流数据初始化为对象。ObjectInputStream 确保恢复已序列化的对象。在这个过程中，ObjectInputStream将字节流与JVM类路径中可用的类进行匹配。



所以发生了什么问题？

在反序列化过程中，当readObject()把二进制数据转换成对象结构的时候，它会寻找序列化流中
与对象类型相关的魔术字节,这些对象类型通常被写入流中，或者是那些已被定义的类型（比如：enum, array,
String，等）。在处理流数据时，上面提到的对象类型需要被解析，如果对象类型无法被解析，这种类型就会被解析成为一般类型TC_OBJECT（
https://docs.oracle.com/javase/7/docs/api/java/io/ObjectStreamConstants.html#TC_OBJECT
），最终，二进制数据流中所携带的对象将从JVM类路径中恢复，如果没有找到相关的类，就会报错。
问题出现的地方是，给readObject()提供一个字节流，此字节流可以被构造成特殊的类，这个类存在于JVM的类路径中，并且可以被使用，这篇文章列举了已知的利用链，这
（ https://github.com/kantega/notsoserial
）被认为有rce漏洞。并且，安全研究员不断发现有此类漏洞的类。现在你可能会问，怎么有这么多类用于rce？依靠这些原始类就可以构造特定恶意的类，从而实现攻击，这



当然攻击者就可以通过输入二进制流来达到攻击的目的，其中的详细信息超出了本文的范围。要想得到更详细的信息可以参考ysoserial（
https://github.com/frohoff/ysoserial
）这个工具，这大概是生成payload最好的工具了吧。

怎样缓解反序列化攻击

轻率的说，通过实现 LookAheadObjectInputStream（
https://www.owasp.org/index.php/Deserialization_Cheat_Sheet#Guidance_on_Deserializing_Objects_Safely
）
策略就可以完全缓解反序列化漏洞。缓解的实现方法是写一个ObjectInputStream 的子类，这个子类要重写 resolveClass()（
https://docs.oracle.com/javase/7/docs/api/java/io/ObjectInputStream.html#resolveClass(java.io.ObjectStreamClass)
），并在这个方法中验证一个类是否能被加载。这个方法看上去能有效地缓解反序列化漏洞，最常见的两种实现方法是白名单和黑名单（
https://www.schneier.com/blog/archives/2011/01/whitelisting_vs.html
）
。在白名单中，只能让可接受的类被反序列化解析，其他的类会被阻止。黑名单则是收集已知会造成问题的类，然后把它们全部阻止。
白名单和黑名单都有自己的优点和缺点，但是我认为基于白名单的实现方法能更好的缓解反序列化漏洞，它能有效的识别那些安全的输入，这种做法也是安全实践的一部分。

```
protected Class<?> resolveClass(ObjectStreamClass desc)
              throws IOException, ClassNotFoundException {
    String name = desc.getName();
```

```
    if(isBlacklisted(name) ) {
            throw new SecurityException("Deserialization is blocked for security reasons");
    }

    if(isWhitelisted(name) ) {
            throw new SecurityException("Deserialization is blocked for security reasons");
    }

    return super.resolveClass(desc);
}
```

jdk中新的反序列化过滤方案

尽管有一些特别的实现来缓解反序列化漏洞带来的影响，但是关于如何解决这样的问题，官方的规范依然很匮乏。为了解决这个问题，Oracle 最近引进 serialization filtering（ http://openjdk.java.net/jeps/290 ）来提高反序列化的安全性，它似乎结合了黑名单和白名单两种方式。新的反序列化过滤器被集成在JDK 9之中，然鹅，这个特性已经被移植到更老的JDK之中了。核心原理是，反序列化过滤基于 ObjectInputFilter（ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.html ）接口，这个接口提供一种配置能力，目的是在反序列化过程中验证输入的数据。通过ObjectInputFilter接口参数： Status.ALLOWED（ http://download.java.net/java/jdk9/docs/api/java/io/ObjectInputFilter.Status.html#ALLOWED ）， Status.REJECTED（ http://download.java.net/java/jdk9/docs/api/java/io/ObjectInputFilter.Status.html#REJECTED ）或者 Status.UNDECIDED（ http://download.java.net/java/jdk9/docs/api/java/io/ObjectInputFilter.Status.html#UNDECIDED ）去检查输入数据的状态。依靠反序列化脚本可以配置这些参数，比如，如果想用黑名单的形式，那么遇到一些特殊的类就要返回Status.REJECTED，并且如果返回Status.UN arrayLength（ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.FilterInfo.html#arrayLength-- ），每一个内置对象的深度 depth（ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.FilterInfo.html#depth-- ），当前对象的引用数量 references（ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.FilterInfo.html#references-- ），当前二进制流占用空间的大小 streamBytes（ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.FilterInfo.html#streamBytes-- ）。这些提供了关于输入流更多的细粒度信息，并且在每一次匹配中都会返回相应的状态。

如何配置过滤器

jdk 9 支持三种方式配置过滤器： custom filter（ http://openjdk.java.net/jeps/290 ）， 也可以使用process-wide filter（ http://openjdk.java.net/jeps/290 ）配置全局的过滤器，built-in filters（ http://www.oracle.com/technetwork/java/javase/8u121-relnotes-3315208.html ）专门用于RMI，现在习惯用 Distributed Garbage Collection (DGC)（ https://docs.oracle.com/javase/8/docs/platform/rmi/spec/rmi-arch4.html ）

基于场景的过滤器

当自己的反序列化的场景和普通场景的反序列化方式不同时，那么自定义过滤器（custom filter）这个方案就非常合适。通常通过实现ObjectInputFilter 接口和重写checkInput函数来创建自定义过滤器。

```
static class VehicleFilter implements ObjectInputFilter {
       final Class<?> clazz = Vehicle.class;
       final long arrayLength = -1L;
       final long totalObjectRefs = 1L;
       final long depth = 1l;
       final long streamBytes = 95L;

       public Status checkInput(FilterInfo filterInfo) {
           if (filterInfo.arrayLength() < this.arrayLength || filterInfo.arrayLength() > this.arrayLength
                   || filterInfo.references() < this.totalObjectRefs || filterInfo.references() > this.totalObjectRefs
                   || filterInfo.depth() < this.depth || filterInfo.depth() > this.depth || filterInfo.streamBytes() < this.st
                   || filterInfo.streamBytes() > this.streamBytes) {
               return Status.REJECTED;
           }

           if (filterInfo.serialClass() == null) {
               return Status.UNDECIDED;
           }

           if (filterInfo.serialClass() != null && filterInfo.serialClass() == this.clazz) {
               return Status.ALLOWED;
           } else {
               return Status.REJECTED;
           }
       }
   }
```

jdk 9 还在ObjectInputStream 类中添加两个函数，目的是让过滤器能set/get当前的数据流。

```
public class ObjectInputStream
   extends InputStream implements ObjectInput, ObjectStreamConstants {

   private ObjectInputFilter serialFilter;
   ...
   public final ObjectInputFilter getObjectInputFilter() {
       return serialFilter;
   }

   public final void setObjectInputFilter(ObjectInputFilter filter) {
       ...
       this.serialFilter = filter;
   }
   ...
}
```

与jdk 9 相反，JDK 8最新的版本（ 1.8.0_144 ）似乎只允许使用ObjectInputFilter.Config.setObjectInputFilter来设置过滤器。

Process-wide （全局）过滤器

通过设置 jdk.serialFilter （ https://docs.oracle.com/javase/9/docs/api/java/io/ObjectInputFilter.Config.html ）
来配置Process-wide过滤器，这样的配置也可以作为系统属性（ https://docs.oracle.com/javase/tutorial/essential/environment/sysprop.html ）或者
安全属性（ http://docs.oracle.com/javase/7/docs/technotes/guides/security/PolicyFiles.html
）。如果系统属性被定义，那么它常常配置的是过滤器；否则，过滤器就要根据安全属性（比如：jdk1.8.0_144/jre/lib/security/java.security
）来配置过滤器了。
jdk.serialFilter的值作为过滤规则，过滤器通过检查类的名字或者限制输入二进制流的内容来达到过滤的目的。可以用逗号和空格来分割过滤规则。数据流在被检查之前会被

```
- maxdepth=value // the maximum depth of a graph
- maxrefs=value // the maximum number of the internal references
- maxbytes=value // the maximum number of bytes in the input stream
- maxarray=value // the maximum array size allowed
```

其他的规律也会匹配由Class.getName()返回的类名和包名。类名和包名的规则也接受星号（*），双星号（ ** ），句号（ . ）和斜杠（ / ）。下面是一些可能发生的场景

```
// this matches a specific class and rejects the rest
"jdk.serialFilter=org.example.Vehicle;!*"

// this matches all classes in the package and all subpackages and rejects the rest
- "jdk.serialFilter=org.example.**;!*"

// this matches all classes in the package and rejects the rest
- "jdk.serialFilter=org.example.*;!*"

// this matches any class with the pattern as a prefix
- "jdk.serialFilter=*;
```

内置过滤器

jdk 9 也引进了一个内置的过滤器，配置这个过滤器主要用于RMI和Distributed Garbage Collection (DGC) 。RMI Registry 和
DGC的内置过滤器是白名单的形式，白名单包含了服务器能够执行的类。下面是 RMIRegistryImpl 和 DGCImp的白名单类

RMIRegistryImpl

```
java.lang.Number
java.rmi.Remote
java.lang.reflect.Proxy
sun.rmi.server.UnicastRef
sun.rmi.server.RMIClientSocketFactory
sun.rmi.server.RMIServerSocketFactory
java.rmi.activation.ActivationID
java.rmi.server.UID
```

DGCImpl

```
java.rmi.server.ObjID
java.rmi.server.UID
java.rmi.dgc.VMID
```

```
java.rmi.dgc.Lease
```

除了这些类，用户也可以用sun.rmi.registry.registryFilter和sun.rmi.transport.dgcFilter
添加自己的过滤器，系统和安全属性的配置和上文所提到的配置是一致的。

结语

然而，java反序列化不是它自己的漏洞，使用序列化框架反序列化不信任的数据才是问题所在。这两点的不同非常重要，因为后者是因为糟糕的程序设计而引入的漏洞，而不
290（ http://openjdk.java.net/jeps/290
）之前的反序列化框架，根本不会验证对象的合法性。而且现在有大量的方法去缓和反序列化漏洞，在JDK本身中没有具体的规范来处理这个缺陷。但是在新版的JEP
290中，Oracle引入了新的过滤机制，这个机制允许开发人员结合自己的应用场景来配置自己的过滤器。新的过滤机制似乎能更容易的缓解反序列化那些不被信任的输入数据

上一篇：WordPress设计漏洞导致Wo... 下一篇：某电商CMS前台getshell分析

1. 0 条回复
   - 动动手指，沙发就是你的了！