Apache Shiro 1.2.4反序列化漏洞分析

转载

架构师成长营 于 2019-08-30 14:57:06 发布 3526 收藏 1
分类专栏： 安全
原文链接：https://www.freebuf.com/vuls/178014.html

***本文中涉及到的相关漏洞已报送厂商并得到修复，本文仅限技术研究与讨论，严禁用于非法用途，否则产生的一切后果自行承担。**

0×00 Apache Shiro

这个组件的漏洞很久之前就爆出来了，但是最近工作中又遇到了，刚好最近也在看Java反序列化的东西，所以决定拿出来再分析一下，期间也遇到了一些奇怪的问题。

网上的分析文章中大部分都是手动添加了commons-collections4-4.0的依赖，目的是为了使用ysoserial生成的CommonsCollections2这个payload，然而我遇到的情况是使用了CommonsBeanutils1就可以直接打成功，所以这里我们不再重复网上对CommonsCollections2的分析了。

0×01 调试分析

调试环境：

> JDK 1.8.0_72
>
> Tomcat 8.0.30

首先我们把shiro的源码clone回来，并切到有问题的分支上去。

```
git clone https://github.com/apache/shiro.git shiro-rootcd shiro-root
git checkout 1.2.0
```

为了能让shiro自带的sample跑起来，要对samples/web/pom.xml文件做一些修改，需要将jstl的版本改为1.2，并且删掉servlet-api的scope字段。同时将jstl-1.2.jar放置在WEB-INF/lib下面。然后应该就可以跑起来并且调试了。

我们将断点打到org.apache.shiro.mgt.DefaultSecurityManager中的resolvePrincipals方法，并且发送一个带有rememberMe Cookie的请求，应该就可以断下来了。

```
484     /unchecked/
485     protected SubjectContext resolvePrincipals(SubjectContext context) {  context:  size = 3
486
487         PrincipalCollection principals = context.resolvePrincipals();  principals: null
488
489         if (CollectionUtils.isEmpty(principals)) {
490             log.trace("No identity (PrincipalCollection) found in the context.  Looking for a remembered identity.");
491
492             principals = getRememberedIdentity(context);  principals: null  context:  size = 3
493
494             if (!CollectionUtils.isEmpty(principals)) {
495                 log.debug("Found remembered PrincipalCollection.  Adding to the context to be used " +
496                         "for subject construction by the SubjectFactory.");
497
498                 context.setPrincipals(principals);
```

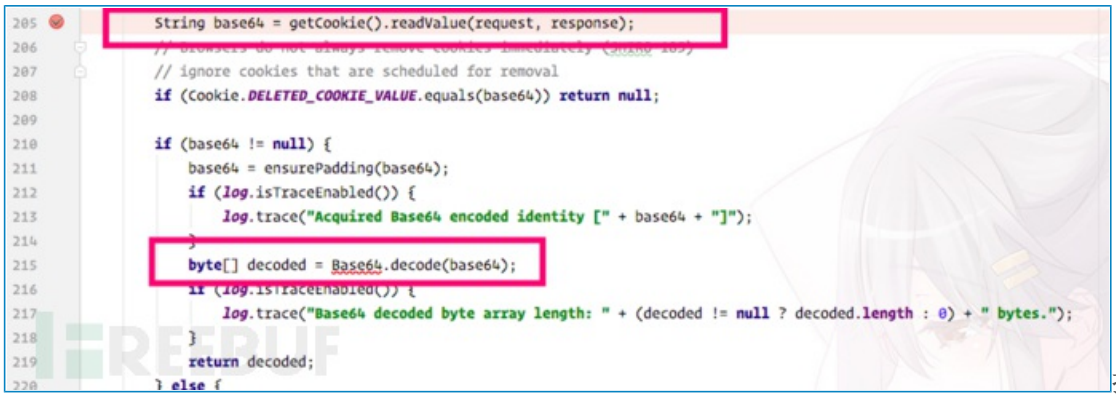
我们继续跟进这个getRememberedIdentity方法：

```
600     protected PrincipalCollection getRememberedIdentity(SubjectContext subjectContext) {
601         RememberMeManager rmm = getRememberMeManager();
602         if (rmm != null) {
603             try {
604                 return rmm.getRememberedPrincipals(subjectContext);
605             } catch (Exception e) {
606                 if (log.isWarnEnabled()) {
607                     String msg = "Delegate RememberMeManager instance of type [" + rmm.getClass().getName() +
608                             "] threw an exception during getRememberedPrincipals().";
609                     log.warn(msg, e);
610                 }
611             }
612         }
613         return null;
614     }
615 }
```

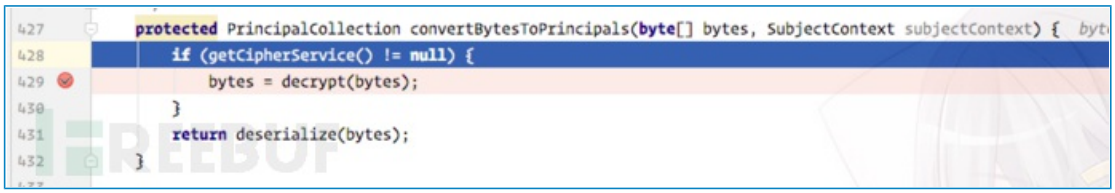
继续一直跟到getRememberedSerializedIdentity方法：

```
185     protected byte[] getRememberedSerializedIdentity(SubjectContext subjectContext) {  subjectContext:  size = 3
186
187         if (!WebUtils.isHttp(subjectContext)) {  subjectContext:  size = 3
188             if (log.isDebugEnabled()) {
189                 String msg = "SubjectContext argument is not an HTTP-aware instance.  This is required to obtain a " +
190                         "servlet request and response in order to retrieve the rememberMe cookie. Returning " +
191                         "immediately and ignoring rememberMe operation.";
192                 log.debug(msg);
193             }
194             return null;
195         }
196
197         WebSubjectContext wsc = (WebSubjectContext) subjectContext;
198         if (isIdentityRemoved(wsc)) {
199             return null;
200         }
201
202         HttpServletRequest request = WebUtils.getHttpRequest(wsc);
203         HttpServletResponse response = WebUtils.getHttpResponse(wsc);
204
205         String base64 = getCookie().readValue(request, response);
206         // Browsers do not always remove cookies immediately (SHIRO-183)
207         // ignore cookies that are scheduled for removal
208         if (Cookie.DELETED_COOKIE_VALUE.equals(base64)) return null;
```

在这个方法中，读出了我们传入的Cookie，并且进行了base64解码：



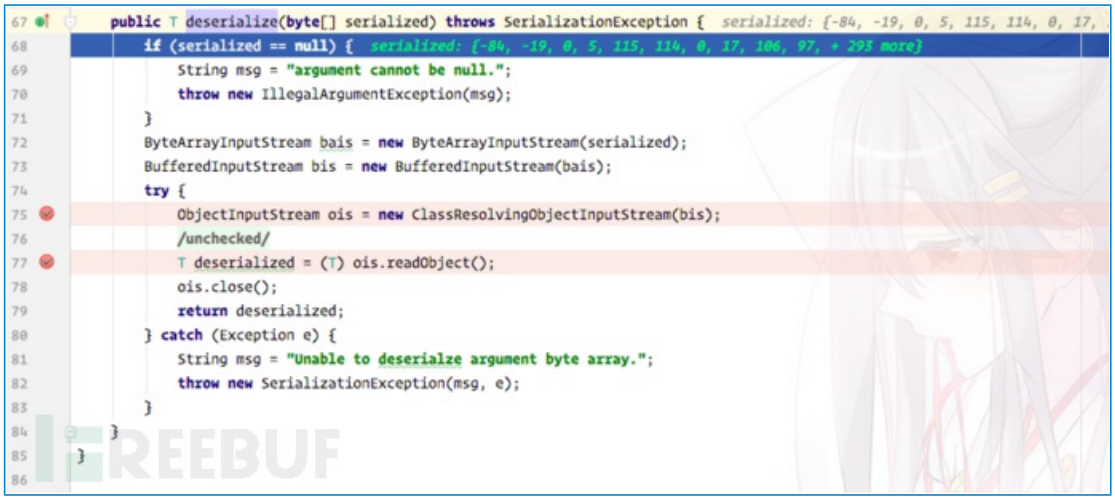
接下来shiro会调用convertBytesToPrincipals并将base64解码后的字节数组作为参数传入：



这里通过函数名也能猜出来，进行了两个操作，分别是解密和反序列化，我们先来看解密部分，经过简单的调试后，发现是一个AES解密，并且存在一个预设秘钥Base64.decode(“kPH+bIxk5D2deZiIxcaaaA==”);，在shiro自带的sample中，并没有通过其他的方式设置这个秘钥，所以这里用的就是这个预设值。

而AES解密中遇到的IV也是从我们传入的Cookie中的前几个字节中获取的，于是我们可以轻易的构造出包含任意内容的Cookie值，解密好的明文就是序列化的内容，调用了deserialize进行反序列化。

最终会调用到org.apache.shiro.io.DefaultSerializer#deserialize方法进行反序列化：



整个流程十分简单，简要概括一下就是：读取cookie -> base64解码 -> AES解密 -> 反序列化

所以我们的payload构造起来也是十分的简单，完整的PoC我会放到我的GitHub上。

0×02 疑点解惑

在调试的过程中，遇到了一些问题，并没有成功的弹出计算器，这里记录一下。

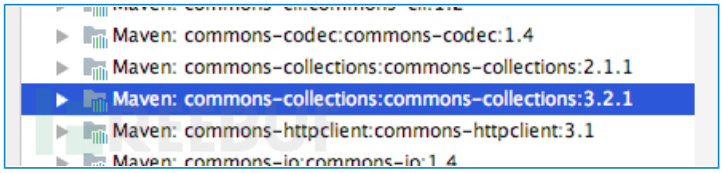
1. 为什么本地搭建环境，使用CommonsBeanutils1打不成功，总是提示ClassNotFound异常？

这个问题我当时调试了好久，一度以为是payload有问题或者shiro的代码因为年代久远有了变化，因为当时遇到的case就是这个payload一发入魂的，表示十分的迷茫。后来发现了关键问题，我们从github上clone到的sample中，commons-beanutils这个依赖的版本是1.8.3，而ysoserial生成的payload的版本是1.9.2，所以在默认的sample中是无法打成功的。于是我修改版本号到1.9.2，一发入魂。

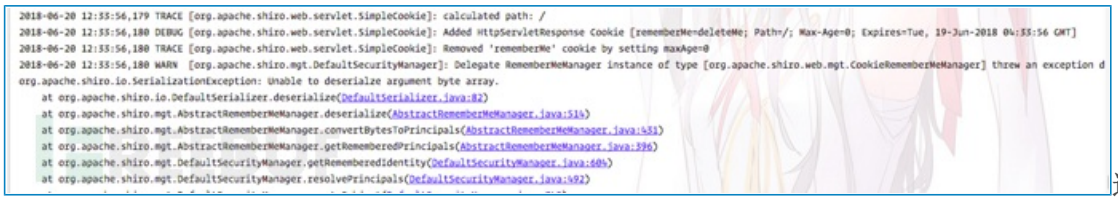
所以遇到的那个案例中，实际的依赖环境版本可能也是这样的吧，所以才能直接打成功。

2. 假如我的依赖中就是没有高版本的commons-beanutils和commons-collections之类的包，怎么利用？

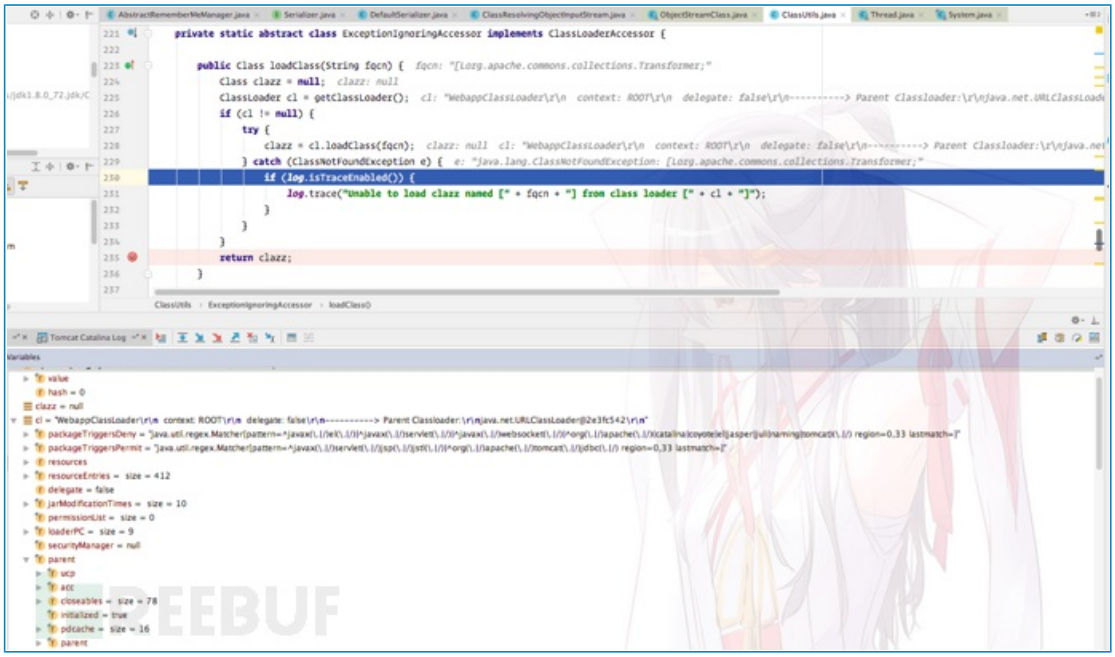
这个项目clone下来之后，可以看到是存在一个commons-collections的包的：



但是使用ysoserial提供的CommonsCollections1是无法成功的，会爆出一个异常：



这就十分的奇怪了，为什么偏偏无法反序列化byte array类型，调试了一下发现是在这里抛出的异常：



查了一下Java中Class.forName()以及ClassLoader.loadClass()的区别，发现forName()总是使用调用者的ClassLoader()，而loadClass()则是可以自己指定一个不同的ClassLoader。那shiro中的ClasssLoader是怎么来的？是通过Thread.currentThread().getContextClassLoader();获取的，也就是WebappClassLoader。但是为啥提示无法加载byte array呢，搜索了一番看到了orange博客下面的讨论，了解到了整个事情的真相：

> Shiro resovleClass使用的是ClassLoader.loadClass()而非Class.forName()，而ClassLoader.loadClass不支持装载数组类型的class。

具体的内容我在这里就不展开讨论了，详情可参考文末原作者的链接。各位同学可以去研究一下，包括在低版本下反弹shell的方法，在orange的博客中也有讲到。

0xFF 参考链接

> 强网杯“彩蛋”——Shiro 1.2.4(SHIRO-550)漏洞之发散性思考
>
> Pwn a CTF Platform with Java JRMP Gadget
>
> Loading an array with a classloader

***本文作者：美丽联合安全MLSRC，转载请注明来自FreeBuf.COM**

欢迎关注技术公众号：架构师成长营