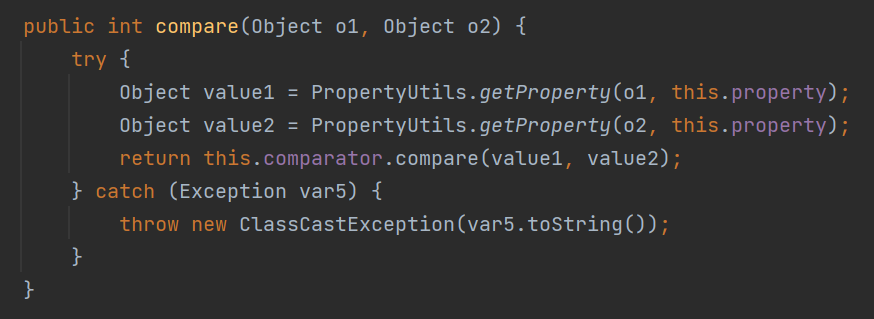
基于一道CTF题的CommonsBeanutils利用分析

wp地址：https://mp.weixin.qq.com/s/hXoUs4ZJgLHHaTvoyhwFxg

### CommonsBeanutils1利用链

##### 核心

org.apache.commons.beanutils.BeanComparator#compare，通过getProperty调用任意指定对象的特定getter方法



##### 入口

ObjectInputStream.readObject()

PriorityQueue.readObject()

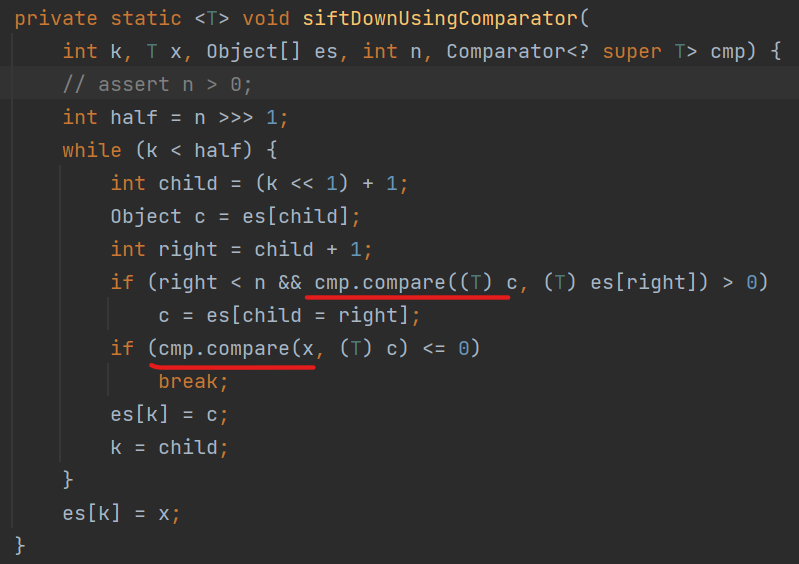
PriorityQueue.heapify()

PriorityQueue.siftDown()

PriorityQueue.siftDownUsingComparator()

BeanComparator.compare()

核心是通过PriorityQueue类反序列化readObject一系列的函数调用，直到siftDownUsingComparator方法调用指定的BeanComparator的compare方法



##### 出口

TemplatesImpl链，不是本文重点，这里不再分析

### Jdk1.4下的利用

Java 1.4 的环境下，没有 PriorityQueue 这个类（入口），TemplatesImpl 和 JdbcRowSetImpl这两个类（出口）也没有，在本题中，作者发现了新的可利用类

##### 入口

java.util.HashMap#readObject

java.util.HashMap#putForCreate

java.util.HashMap#eq

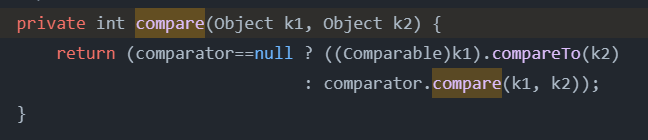
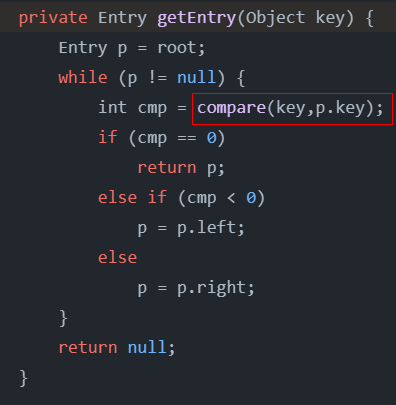
java.util.AbstractMap#equals

java.util.TreeMap#get

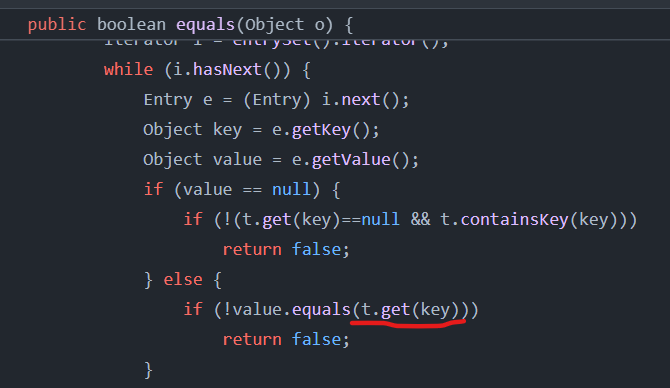
java.util.TreeMap#getEntry

org.apache.commons.beanutils.BeanComparator#compare

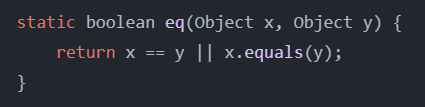
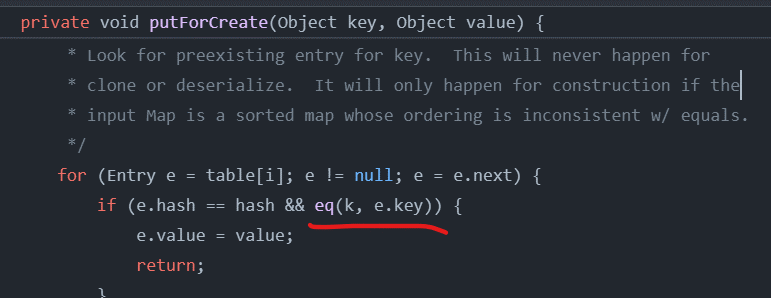
首先是TreeMap#get方法中调用getEntry方法，其中会调用key的compare方法

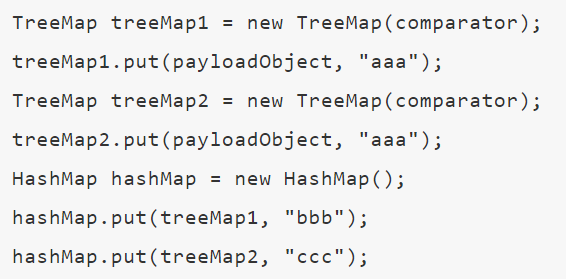


在java.util.AbstractMap#equals方法中又会调用传入参数的get方法，该抽象类提供了Map的基本实现，TreeMap和HashMap都继承了该抽象类



最后就是java.util.HashMap在反序列化时，会通过putForCreate方法，调用key的equals函数，通过上述利用链的调用，就可以进入到cb链，这里还需要先通过key的hash值对比，可以通过创建两个构造起来完全一样、但是引用地址不一样的对象来解决 hash 判断的问题

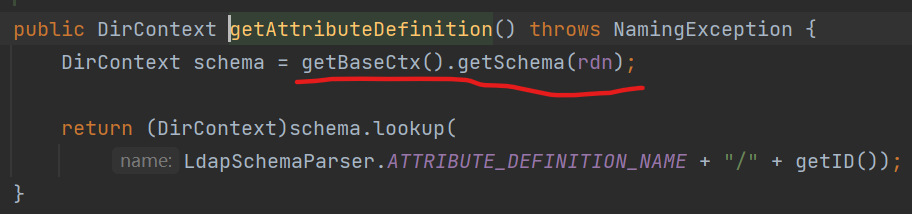




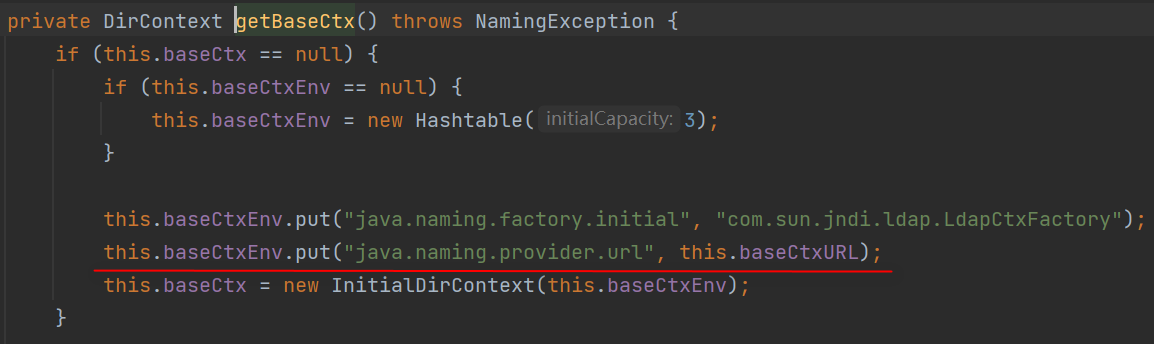
##### 出口

commons.beanutils链是通过调用类的getter方法，出题者在jdk中发现了一条新的实现JNDI注入的链

在com.sun.jndi.ldap.LdapAttribute#getAttributeDefinition方法中，通过getBaseCtx()和getSchema()方法的组合，实现JNDI注入



首先是getBaseCtx()方法，会用 baseCtxURL 属性去填充 java.naming.provider.url，然后创建一个 InitialDirContext 对象并返回，也就是可以控制JNDI注入的url



在一般的JNDI注入中，是通过触发InitialContext.lookup方法，这个方法在造成JNDI注入时，也是有它的调用栈

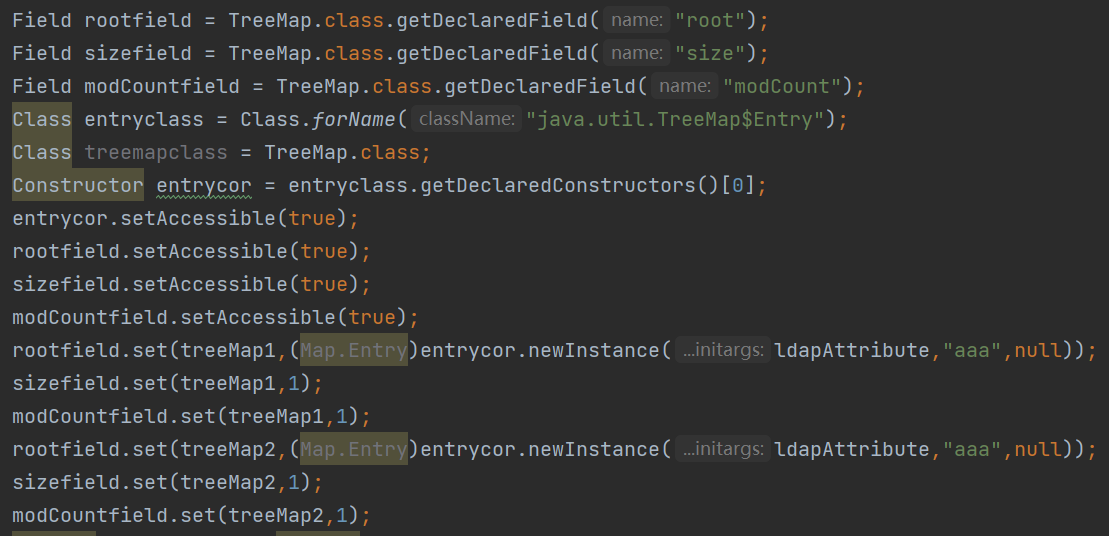
javax.naming.InitialContext#lookup(java.lang.String)-> com.sun.jndi.url.ldap.ldapURLContext#lookup(java.lang.String)-> com.sun.jndi.toolkit.url.GenericURLContext#lookup(java.lang.String)-> com.sun.jndi.toolkit.ctx.PartialCompositeContext#lookup(javax.naming.Name)-> com.sun.jndi.toolkit.ctx.ComponentContext#p\_lookup-> com.sun.jndi.ldap.LdapCtx#c\_lookup-> ......

如果可以直接调用其中任意一个方法，都是可以造成JNDI注入，而在LdapAttribute#getSchema方法中，会通过以下调用栈，最后执行LdapCtx#c\_lookup函数，造成JNDI注入

com.sun.jndi.ldap.LdapAttribute#getAttributeDefinition-> javax.naming.directory.InitialDirContext#getSchema(javax.naming.Name)-> com.sun.jndi.toolkit.ctx.PartialCompositeDirContext#getSchema(javax.naming.Name)-> com.sun.jndi.toolkit.ctx.ComponentDirContext#p\_getSchema-> com.sun.jndi.toolkit.ctx.ComponentContext#p\_resolveIntermediate-> com.sun.jndi.toolkit.ctx.AtomicContext#c\_resolveIntermediate\_nns-> com.sun.jndi.toolkit.ctx.ComponentContext#c\_resolveIntermediate\_nns-> com.sun.jndi.ldap.LdapCtx#c\_lookup-> ......

### Jdk≥1.8下的利用

JDK 1.8 对 HashMap 进行了比较大的优化，底层实现由之前的 “数组+链表” 改为 “数组+链表+红黑树”，具体代码在这条链利用过程中的改变主要是TreeMap#put方法中加入了对key值的检查，如果发现键值对中的key不是comparator类型，就会报错



而该类的root属性和Entry类型分别是私有变量和内部类，所以在构造exp的时候，需要通过反射方法设置root的值



其他部分虽然有代码变动，但不影响入口链的执行，可以得到新的exp：<https://github.com/LL2XX/WriteUps/blob/main/RWCTF%20Old%20System/jdk_1.8_exp.java>