

0x01 起因

关于Java反序列化的文章已经相当的多了，而且大家也对于这个东西说的很清楚了，所以今年我想换个角度来看这个东西。我们都知道Java反序列化漏洞的产生原因在于在 `readObject` 方法的时候，写入了漏洞代码，这个和PHP的反序列化漏洞很像，在反序列化的时候出发了在 `__destruct` 等魔术函数中的漏洞代码。这里就有一个问题了，先看一下我的demo吧， `ObjectCalc.java` 为重写 `readObject` 方法文件。

```
//ObjectCalc.java
import java.io.IOException;
import java.io.Serializable;

class ObjectCalc implements Serializable{
    public String name;
    //重写readObject()方法
    private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException{
        //执行默认的readObject()方法
        //in.defaultReadObject();
        //执行打开计算器程序命令
        Runtime.getRuntime().exec("open /Applications/Calculator.app/");
    }
}
```



那么我通过下面的代码可以触发反序列化漏洞，弹出计算器。

```
1 import java.io.*;
2 public class unSerializableCalc{
3     public static void main(String args[]) throws Exception{
4         //从文件中反序列化obj对象
5         FileInputStream fis = new FileInputStream("/Users/l1nk3r/Desktop/object");
6         ObjectInputStream ois = new ObjectInputStream(fis);
7         //恢复对象
8         ObjectCalc objectFromDisk = (ObjectCalc)ois.readObject();
9         System.out.println(objectFromDisk.name);
10        ois.close();
11    }
12 }
```



那么问题来了我们通过 第8行 `ois.readObject` 获取到的输入流过程中调用了 `readObject` 方法，为什么最后会调用到被反序列化类（ `ObjectCalc` ）中的 `readObject` 方法，这个 `readObject` 调用过程到底是怎样的。

0x02 深入分析

为了弄清楚这个问题，我决定在 ObjectCalc.java 文件中的命令执行位置下一个断点，好的相关调用栈已经出来了，这时候我们跟进一下。

Call Stack (top):

- ObjectCalc.readObject(ObjectInputStream) line: 12
- NativeMethodAccessorImpl.invoke0(Method, Object, Object[]) line: not available [native method]
- NativeMethodAccessorImpl.invoke(Object, Object[]) line: 62
- DelegatingMethodAccessorImpl.invoke(Object, Object[]) line: 43
- Method.invoke(Object, Object...) line: 498
- ObjectStreamClass.invokeReadObject(Object, ObjectInputStream) line: 1158
- ObjectInputStream.readSerialData(Object, ObjectStreamClass) line: 2176
- ObjectInputStream.readOrdinaryObject(boolean) line: 2067
- ObjectInputStream.readObject0(boolean) line: 1571
- ObjectInputStream.readObject() line: 431
- unSerializableCalc.main(String[]) line: 10

Source Code (ObjectCalc.java):

```
1 import java.io.IOException;
2 import java.io.ObjectStreamClass;
3 import java.io.Serializable;
4
5 class ObjectCalc implements Serializable{
6     public String name;
7     //重写readObject()方法
8     private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException{
9         //执行默认的readObject()方法
10        //in.defaultReadObject();
11        //执行打开计算器程序命令
12        Runtime.getRuntime().exec("open /Applications/Calculator.app/");
13    }
14 }
```

先跟进一下 ObjectInputStream.readObject，这里我简化了一下代码，关键位置在第431行调用了 readObject0 方法，并且传入 false。

```
421 public final Object readObject()
422     throws IOException, ClassNotFoundException
423 {
424     if (enableOverride) {
425         return readObjectOverride();
426     }
427
428     // if nested read, passHandle contains handle of enclosing object
429     int outerHandle = passHandle;
430     try {
431         Object obj = readObject0(false);
432         handles.markDependency(outerHandle, passHandle);
433         ClassNotFoundException ex = handles.lookupException(passHandle);
434         if (ex != null) {
435             throw ex;
436         }
437         if (depth == 0) {
438             vlist.doCallbacks();
439         }
440         return obj;
441     } finally {
442         passHandle = outerHandle;
443         if (closed && depth == 0) {
444             clear();
445         }
446     }
447 }
```

继续跟进一下 readObject0 方法，关键在下面这两行，此时的 TC_OBJECT 的值为115，且调用了 readOrdinaryObject 方法。

```
case TC_OBJECT:
    return checkResolve(readOrdinaryObject(unshared));
```

跟进 readOrdinaryObject 方法，调用了 readSerialData 方法。

```
private Object readOrdinaryObject(boolean unshared)
    throws IOException
{
    ...
    if (desc.isExternalizable()) {
        readExternalData((Externalizable) obj, desc);
    } else {
        readSerialData(obj, desc);
    }
}
```

```
}
```

继续跟进一下 readSerialData 方法，该方法的实现如下所示。

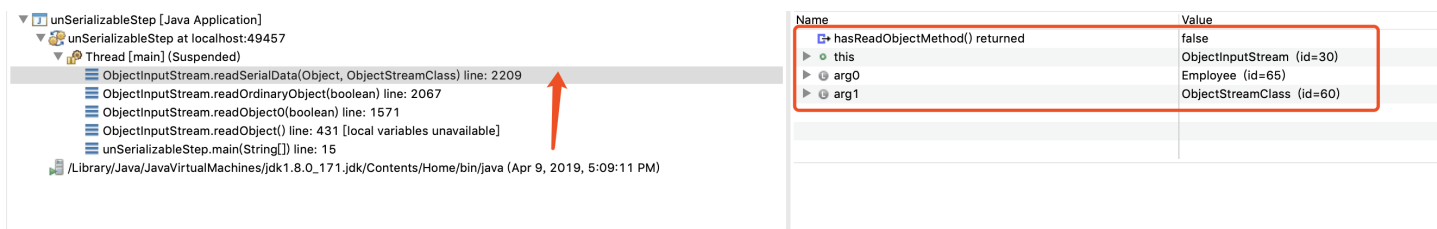
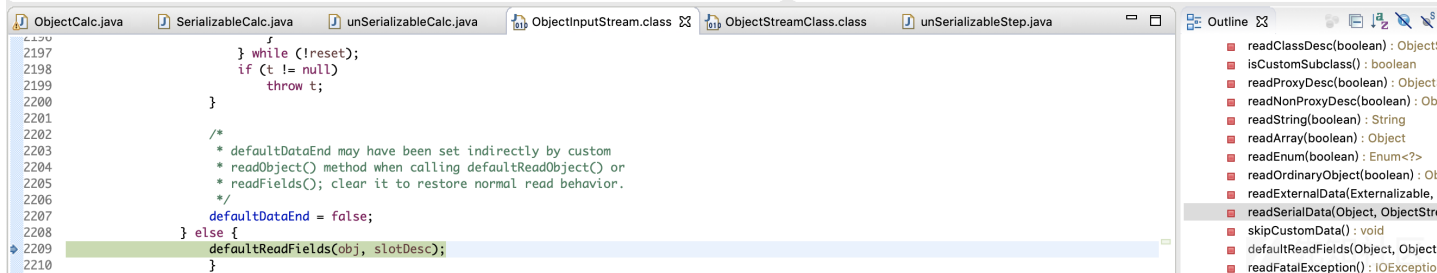
```
1 private void readSerialData(Object obj, ObjectOutputStream desc)
2     throws IOException
3 {
4     ObjectOutputStream.ClassDataSlot[] slots = desc.getClassDataLayout();
5     for (int i = 0; i < slots.length; i++) {
6         ObjectOutputStream slotDesc = slots[i].desc;
7
8         if (slots[i].hasData) {
9             if (obj == null || handles.lookupException(passHandle) != null) {
10                 defaultReadFields(null, slotDesc); // skip field values
11             } else if (slotDesc.hasReadObjectMethod()) {
12                 ...
13                 bin.setBlockDataMode(true);
14                 slotDesc.invokeReadObject(obj, this);
15             } catch (ClassNotFoundException ex) {
16                 ...
17                 handles.markException(passHandle, ex);
18             } finally {
19                 ...
20             }
21             ...
22             defaultDataEnd = false;
23         } else {
24             defaultReadFields(obj, slotDesc);
25         }
26         ...
    }
```



从动态调试结果来看，重写 readObject 会进入第14行的 slotDesc.invokeReadObject 方法中，再跟进一下 slotDesc.invokeReadObject 方法，该方法主要代码如下：

```
void invokeReadObject(Object obj, ObjectInputStream in)
    throws ClassNotFoundException, IOException,
        UnsupportedOperationException
{
    requireInitialized();
    if (readObjectMethod != null) {
        try {
            readObjectMethod.invoke(obj, new Object[] { in });
        }
    }
}
```

其中 readObjectMethod.invoke 这个方法很熟悉了，java的反射机制，也就说通过重写 readObject 的整个调用流程会经过java的反射机制。这里再看一个不通过重写 readObject 反序列化的调用过程，我省略了前面的跟踪调试过程，大家看下图。

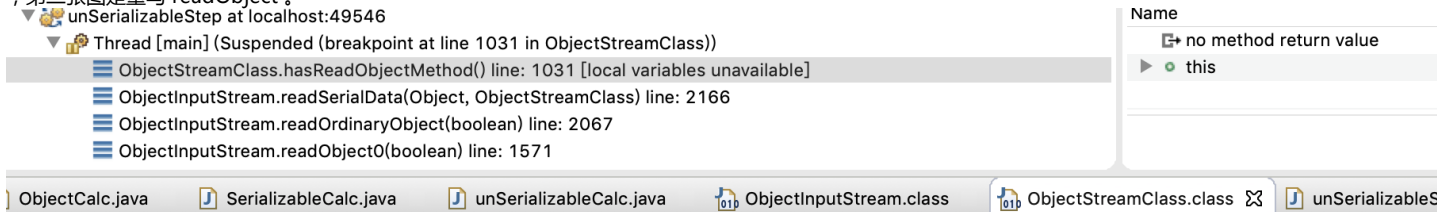
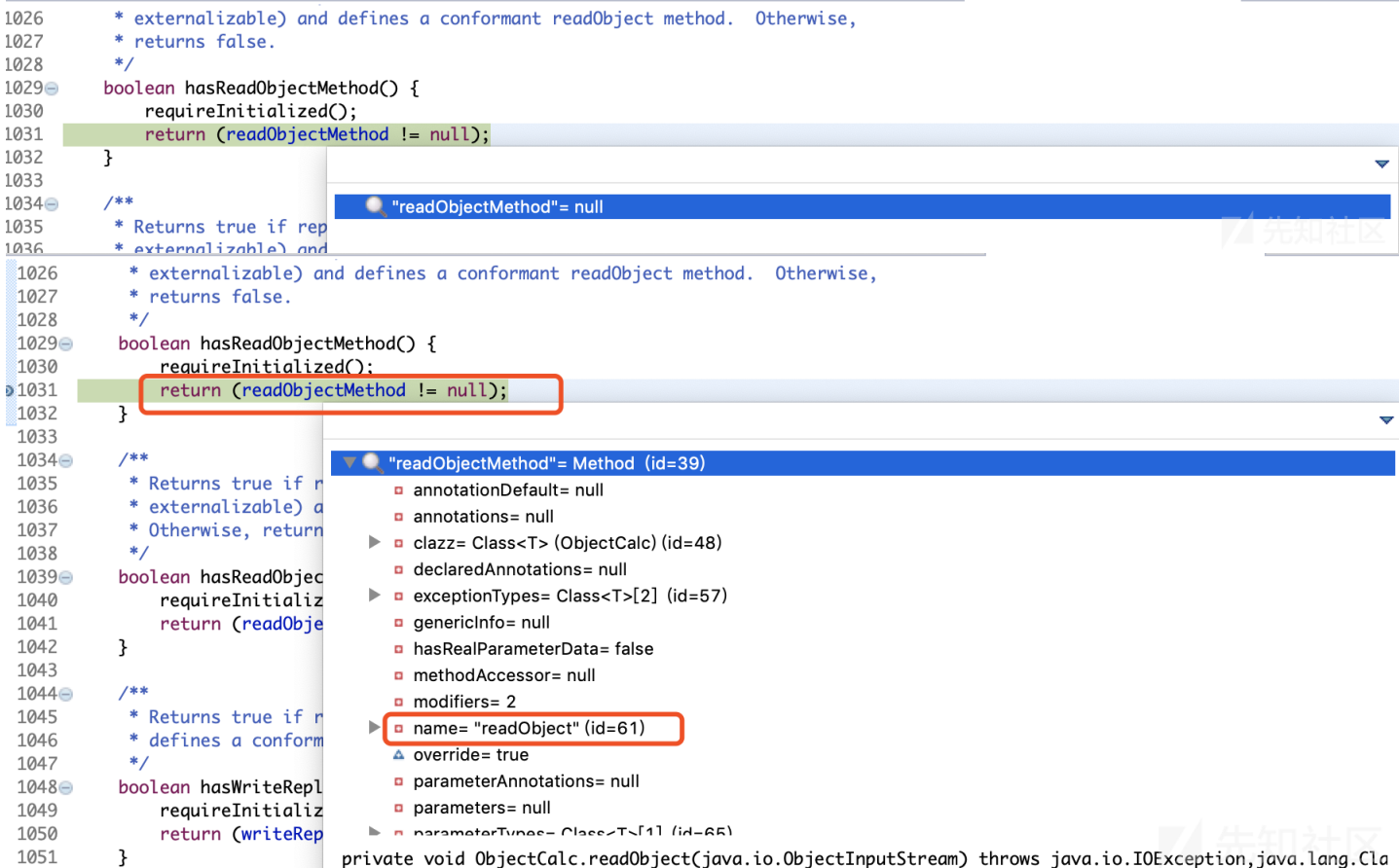



不通过重写 readObject 的反序列化过程一样是进入 readSerialData 中，但是是通过 defaultReadFields 进行处理，这里有个关注点是 slotDesc.hasReadObjectMethod() 返回的结果是false，也就是下面这个if判断的结果，我简化了一下流程。

```
else if (slotDesc.hasReadObjectMethod()) {
    slotDesc.invokeReadObject(obj, this);
    ...
} else {
    defaultReadFields(obj, slotDesc);
}
```

也就是说实际上是否重写了 readObject 影响的是 slotDesc.hasReadObjectMethod() 的结果，那么跟进一下 hasReadObjectMethod 方法，这里我在return (readObjectMethod != null);下了一个断点，对比一下重写 readObject 结果和不重写 readObject 结果的差别，第一张图是不重写 readObject

，第二张图是重写 readObject。

很明显我们发现了返回结果不一样，第一张图的结果自然return为false，第二张图return结果自然为true，也就是说重写 readObject 结果和不重写 readObject 结果的差别本质上在于进入的循环不一样。

0x03 小结

根据上面的动态调试结果，简单做个小结，也就是说如果反序列化的过程中被反序列化类重写了 readObject ，该数据在反序列化的过程中核心流程走到 readSerialData 方法中的 slotDesc.invokeReadObject 方法，通过反射机制触发相关流程，并且调用重写的 readObject 。如果没有重写 readObject ，则调用 ObjectInputStream 类中的 readObject 方法，并且执行反序列化。

点击收藏 | 1 关注 | 1

[上一篇：某开源堡垒机的数据库默认密码及ip...](#) [下一篇：在逗号被禁止的情况下的SQL注入技巧](#)

1. 0 条回复
- 动动手指，沙发就是你的了！

[登录](#) 后跟贴

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](#)