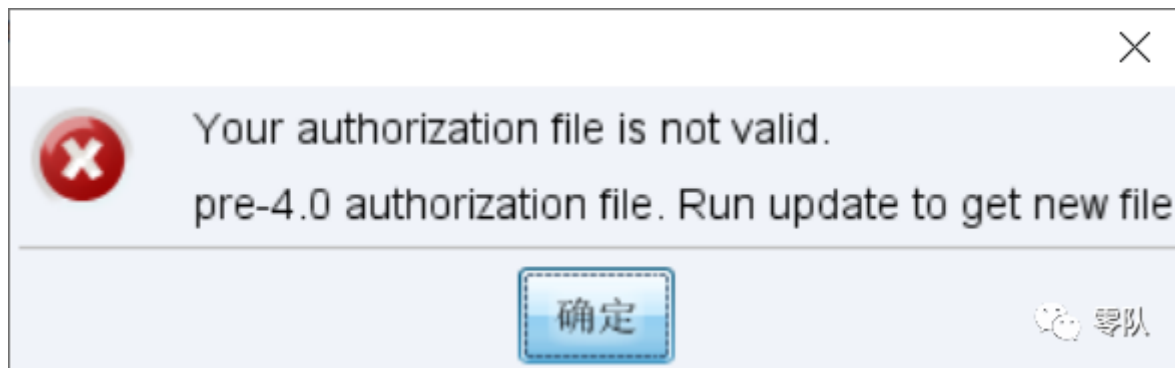


Cobaltstrike 4 破解之 我自己给我自己颁发 license

CobaltStrike4 License 分析



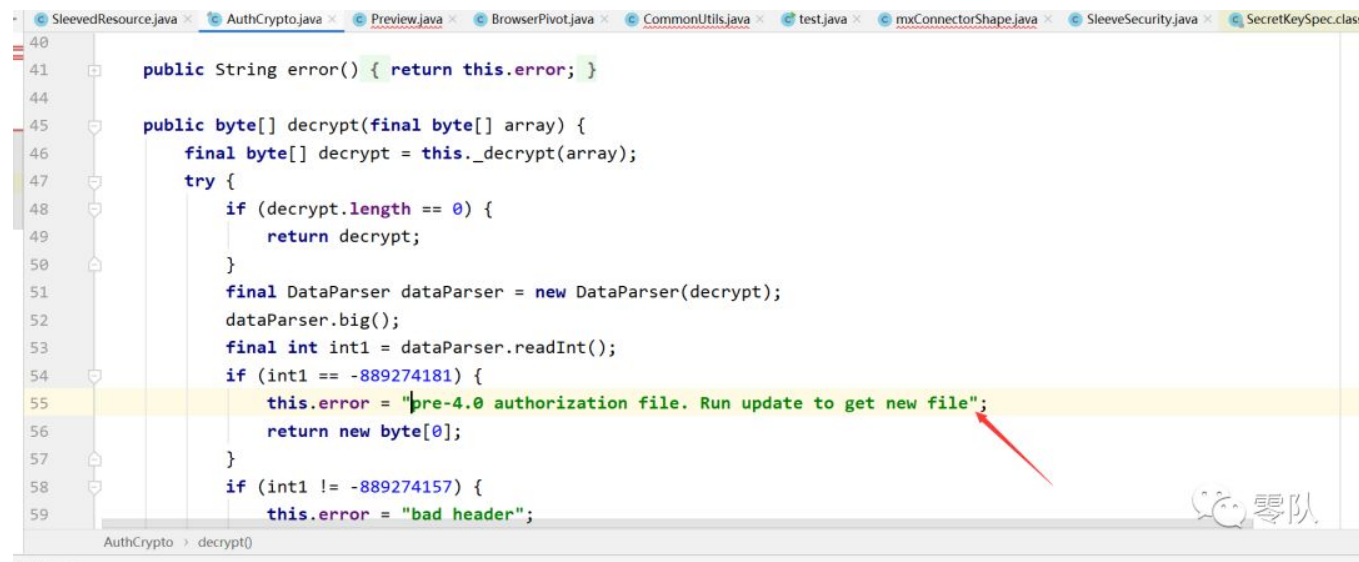
19 号早上一起就有小伙伴发给我新的 cobaltstrike，对了下哈希是官网 19 年 12 月 5 号版。便着手破解了，首先复制 3.14 的 cobaltstrike.auth 进去，发现启动异常。提示让我更新



No.1

基础逻辑梳理

反编译之后直接搜 发现在 AuthCrypto.java 第 55 行 看一下 context



Load() -> 加载公钥, 验证哈希

```
SleevedResource.java AuthCrypto.java Preview.java BrowserPivot.java CommonUtils.java test.java mxConnectorShape.java SleeveSecurity.java SecretKeySpec.class crypto.txt
20 catch (Exception ex) {
21     this.error = "Could not initialize crypto";
22     MudgeSanity.LogException("AuthCrypto init", ex, false);
23 }
24 }
25
26 public void load() {
27     try {
28         final byte[] all = CommonUtils.readAll(CommonUtils.class.getClassLoader().getResourceAsStream(name: "resources/authkey.pub"));
29         if (!"7ef5ed238ec450fd1ae2e935d65ddbcdb".equals(CommonUtils.toHex(CommonUtils.MD5(all)))) {
30             CommonUtils.print_error( S: "Invalid authorization file");
31             System.exit( status: 0);
32         }
33         this.pubkey = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(all));
34     }
35     catch (Exception ex) {
36         this.error = "Could not deserialize authpub.key";
37         MudgeSanity.LogException("authpub.key deserialization", ex, false);
38     }
39 }
40
41 public String error() { return this.error; }
44
```


看一下 decrypt 的调用, 找到 Authorization()

out 39
parser 40
pe 41
phish 44
profiler 44
proxy 45
report

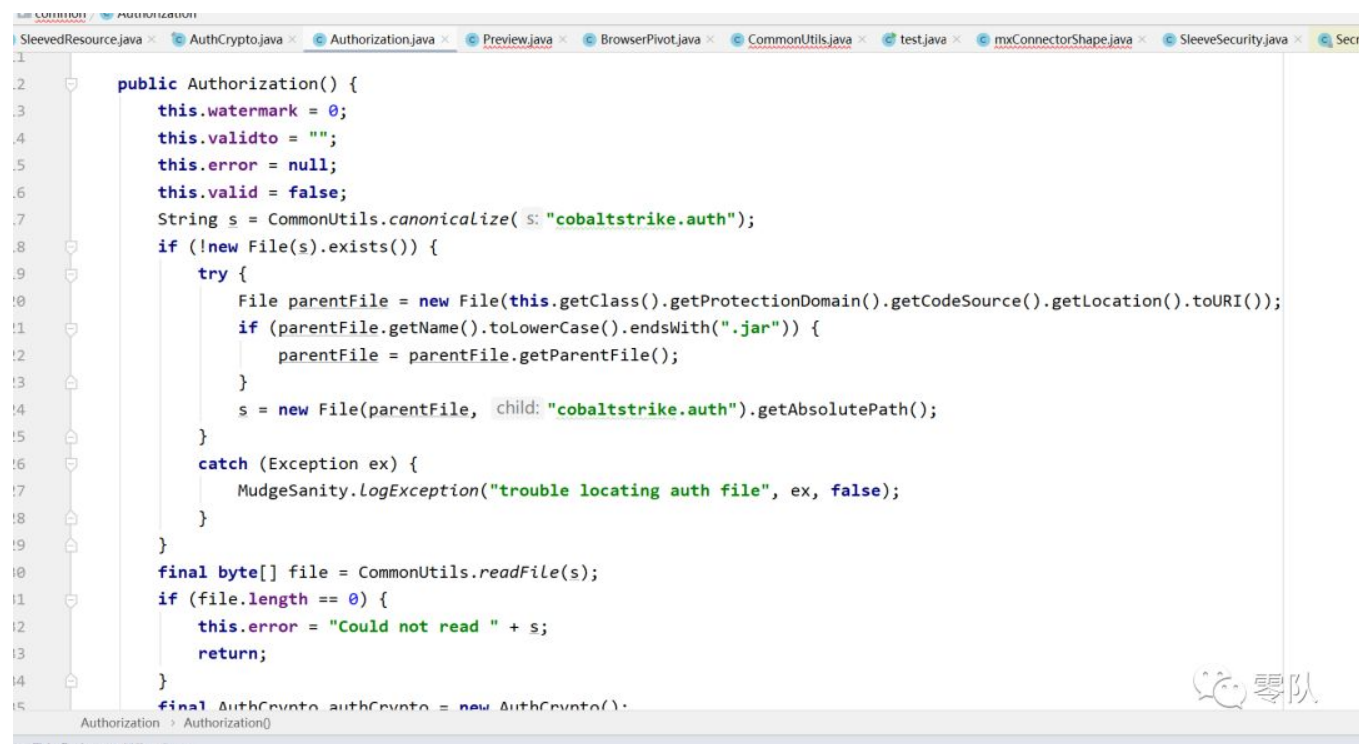
```
}  
  
public String error() { return this.error; }  
  
public byte[] decrypt(final byte[] array) {  
    AuthCrypto > decrypt()  
}
```

Id: Usages of decrypt(byte[]) in Project and Libraries x

- Method
 - decrypt(byte[])
- Found usages 2 usages
 - Unclassified usage 2 usages
 - decompiled-cobaltstrike4 2 usages
 - common 2 usages
 - Authorization 1 usage
 - Authorization() 1 usage
 - 36 final byte[] decrypt = authCrypto.decrypt(file);
 - AuthUtil 1 usage



先看构造方法



```
1 public Authorization() {
2     this.watermark = 0;
3     this.validto = "";
4     this.error = null;
5     this.valid = false;
6     String s = CommonUtils.canonicalize("cobaltstrike.auth");
7     if (!new File(s).exists()) {
8         try {
9             File parentFile = new File(this.getClass().getProtectionDomain().getCodeSource().getLocation().toURI());
10            if (parentFile.getName().toLowerCase().endsWith(".jar")) {
11                parentFile = parentFile.getParentFile();
12            }
13            s = new File(parentFile, "cobaltstrike.auth").getAbsolutePath();
14        }
15        catch (Exception ex) {
16            MudgeSanity.LogException("trouble locating auth file", ex, false);
17        }
18    }
19    final byte[] file = CommonUtils.readFile(s);
20    if (file.length == 0) {
21        this.error = "Could not read " + s;
22        return;
23    }
24    final AuthCrypto authCrypto = new AuthCrypto();
```

读取了 cobaltstrike.auth 文件 很显然这个 class 就是主管 license 认证的 class 了

No.2

Cobaltstrike.auth 解密

文件最后被我们刚才看到的那个 decrypt 函数解密

```

10 File parentFile = new File(this.getClass().getProtectionDomain().getCodeSource().getLocation().toURI());
11 if (parentFile.getName().toLowerCase().endsWith(".jar")) {
12     parentFile = parentFile.getParentFile();
13 }
14 s = new File(parentFile, child: "cobaltstrike.auth").getAbsolutePath();
15 }
16 catch (Exception ex) {
17     MudgeSanity.logException("trouble locating auth file", ex, false);
18 }
19 }
20 final byte[] file = CommonUtils.readFile(s);
21 if (file.length == 0) {
22     this.error = "Could not read " + s;
23     return;
24 }
25 final AuthCrypto authCrypto = new AuthCrypto();
26 final byte[] decrypt = authCrypto.decrypt(file);
27 if (decrypt.length == 0) {
28     this.error = authCrypto.error();
29     return;
30 }
31 try {
32     final DataParser dataParser = new DataParser(decrypt);

```




我们先看一下 35 行的

•

```
final AuthCrypto authCrypto = new AuthCrypto();
```

构造方法中生成了一个 RSA/ECB/PKCS1Padding 的 cipher

```
public AuthCrypto() {
    this.pubkey = null;
    this.error = null;
    try {
        this.cipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
        this.load();
    }
    catch (Exception ex) {
        this.error = "Could not initialize crypto";
        MudgeSanity.LogException("AuthCrypto init", ex, false);
    }
}
```



零队

很明显 authpub.key 是 rsa 加密的, 这样子思路就很清晰了, 我们自己生成一个 RSA 公私钥, 公钥保存成 authpub.key, 私钥拿来给自己签 cobaltstrike.auth, 还不是美滋滋

首先重要的事情我们就是要研究他密钥的组成, 回到刚才 Authorization 的源码

首先是解密

```
45 public byte[] decrypt(final byte[] array) {
46     final byte[] decrypt = this._decrypt(array);
47     try {
48         if (decrypt.length == 0) {
49             return decrypt;
50         }
51         final DataParser dataParser = new DataParser(decrypt);
52         dataParser.big();
53         final int int1 = dataParser.readInt();
54         if (int1 == -889274181) {
55             this.error = "pre-4.0 authorization file. Run update to get new file";
56             return new byte[0];
57         }
58         if (int1 != -889274157) {
59             this.error = "bad header";
60             return new byte[0];
61         }
62         return dataParser.readBytes(dataParser.readShort());
63     }
64     catch (Exception ex) {
65         this.error = ex.getMessage();
66         return new byte[0];
67     }
68 }
69 }
```



绿框框住的是解密的逻辑，解密之后会进行一些判断

他将解密好的数据，交给了 dataParser，跟进一下

readInt

```
public int readInt() throws IOException {
    this.buffer.clear();
    this.content.read(this.bdata, off: 0, len: 4);
    return this.buffer.getInt(index: 0);
}
```



No.3

有符号 Int 转 byte 数组

这个 this.buffer 的数据类型是 protected ByteBuffer buffer;

第一行二话不说先 clear, 可以知道这个肯定是当 temp 用的, this.content 是 DataInputStream content;

Read 方法每次读取, 都会将指针后移, 例如这个函数就是一次读四个 byte,

This.bdata 是一个 byte[8]

我举个例子

例如我们有个 byte:

•

```
byte[] b = {1,2,3,4,5,6,7,8,9,0};
```

第一次读取 4 个 byte, 也就是 1, 2, 3, 4

然后使用 getInt 取成数字, getInt 将 byte 转成数字这块也是比较有意思,

先将 1, 2, 3, 4 转换成四个 16 进制的数字也就是

01, 02, 03, 04

然后直接相加 最后数字就是 01020304 ,

然后将这个 16 进制的数字转换成 10 进制,

最后就成了 16,909,060

16,909,060

HEX	102 0304
DEC	16,909,060
OCT	100 401 404
BIN	0001 0000 0010 0000 0011 0000 0100

QWORD MS

```
7 public class test2 {
8     public static void main(String[] args) throws IOException {
9         byte[] bdata = new byte[8];
10        ByteBuffer buffer = null;
11        byte[] b = {1,2,3,4,5,6,7,8,9,0};
12        (buffer = ByteBuffer.wrap(bdata)).order(ByteOrder.BIG_ENDIAN);
13        DataInputStream content = new DataInputStream(new ByteArrayInputStream(b));
14        content.read(bdata, off: 0, len: 4);
15        int i = buffer.getInt();
16        System.out.println(i);
17    }
18 }
19
```

test2 > main()

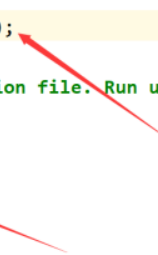
Run: test2 x

"C:\Program Files\Java\jdk1.8.0_231\bin\java.exe" ...

16909060

继续回到代码

```
53     final int int1 = dataParser.readInt();
54     if (int1 == -889274181) {
55         this.error = "pre-4.0 authorization file. Run update to get new file";
56         return new byte[0];
57     }
58     if (int1 != -889274157) {
59         this.error = "bad header";
60         return new byte[0];
61     }
62     return dataParser.readBytes(dataParser.readShort());
63 }
64 catch (Exception ex) {
65     this.error = ex.getMessage();
66     return new byte[0];
67 }
68 }
69 }
```



很显然，我们刚才的提示就是因为 cobaltstrike.auth 解密的 byte[] 前四位取整后得出 -889274181，这个取出的数为什么是负数？

主要是因为这是有符号数，大学上课好好听的同学这里就知道咋搞了，我算了半天，最后在求教了大佬蝎子菜菜 @MarchRain 之后才知道的

<https://zhidao.baidu.com/question/51297200.html>

语言有符号数与无符号数之间的转换：无符号数：不存在正负之分，所有位都用来表示数的本身。

有符号数：最高位用来表示数的正负，最高位为1则表示负数，最高位为0则表示正数。

转换为有符号数：看无符号数的最高位是否为1，如果不为1（即为0），则有符号数就直接等于无符号数；

2.如果无符号数的最高位为1，则将无符号数取补码，得到的数就是有符号数。

3..有符号数转换为无符号数：看有符号数的最高位是否为1，如果不为1（即为0），则无符号数就直接等于有符号数；

4.如果有符号数的最高位为1, 则将有符号数取补码, 得到的数就是无符号数。



<https://baike.baidu.com/item/%E8%A1%A5%E7%A0%81/6854613?fr=aladdin>

求负整数的补码，将其原码除符号位外的所有位取反（0变1，1变0，符号位为1不变）后加1 [4]。

同一个数字在不同的补码表示形式中是不同的。比如-15的补码，在8位二进制中是11110001，然而在16位二进制补码表示中，就是1111111111110001。以下都使用8位2进制来表示。

例：求-5的补码。

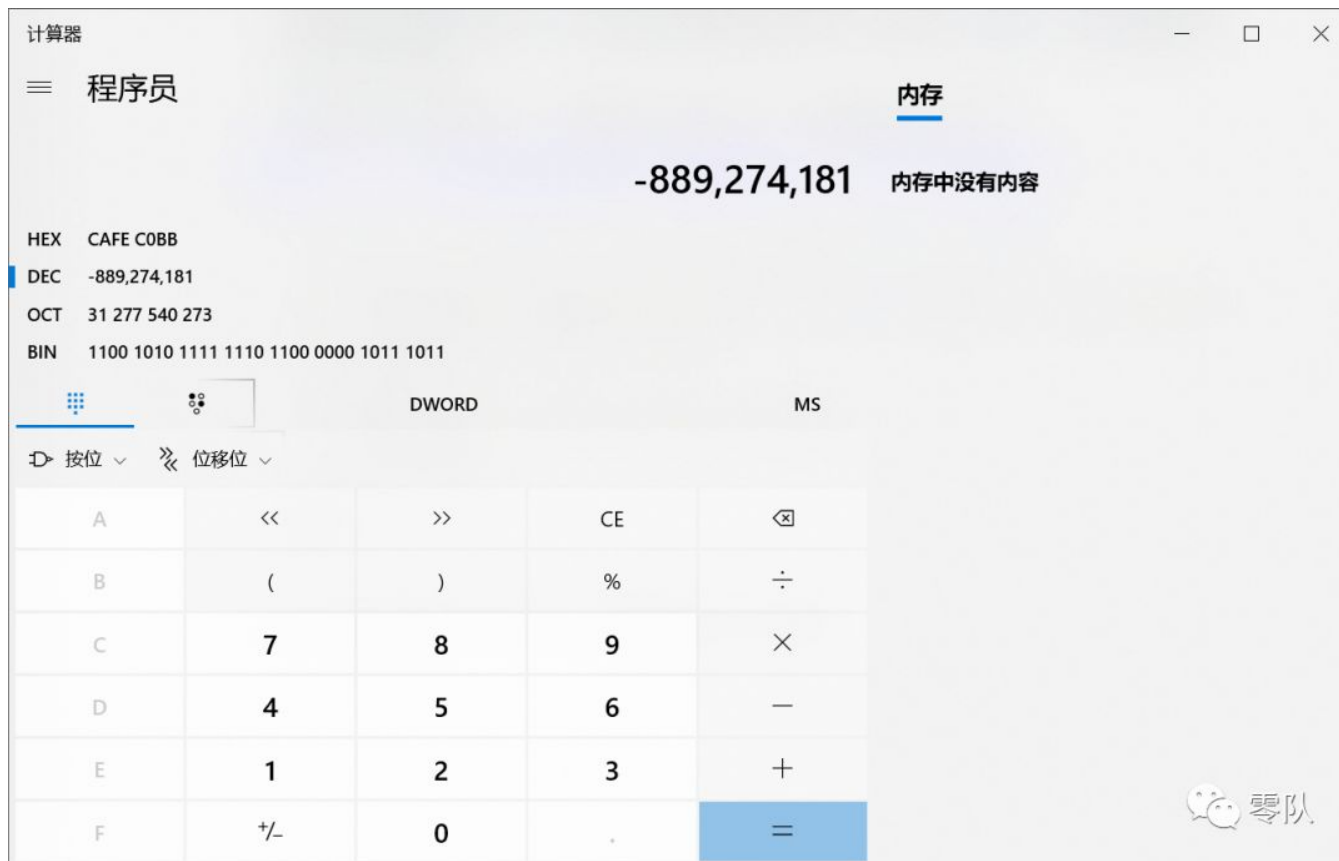
-5对应正数5 (00000101) → 所有位取反 (11111010) → 加1 (11111011)

所以-5的补码是11111011。

0的补码


$$\det A = 1 \text{ and } \det B = -1 \implies \det A \neq \det B. \quad (1)$$

了解了这些 我们 - 889274181 用计算器先取二进制



分成四组

负数运算 需要先看符号位，如果是负数，需要计算反码，在对结果取补码得到数字

1100 1010 -> 反码：0011 0101 -> 补码：0011 0110 -> 十进制 -54

1111 1110 -> 反码：0000 0001 -> 补码：0000 0010 -> 十进制 -2

1100 0000 -> 反码：0011 1111 -> 补码：0100 0000 -> 十进制 -64

1011 1011 -> 反码：0100 0100 -> 补码：0100 0101 -> 十进制 -69

最后就是 byte[] b = {-54,-2,-64,-69}

这个就是 cobaltstrike 3.14 的 auth 头，我们需要改成 - 889274157

具体的计算方式，大家就按照上面的办法自行计算吧

```
40     }
41     try {
42         final DataParser dataParser = new DataParser(decrypt);
43         dataParser.big();
44         final int int1 = dataParser.readInt();
45         this.watermark = dataParser.readInt();
46         final byte byte1 = dataParser.readByte();
47         final byte[] bytes = dataParser.readBytes(dataParser.readByte());
48         if (byte1 < 40) {
49             this.error = "Authorization file is not for Cobalt Strike 4.0+";
50             return;
51         }
52         if (29999999 == int1) {
53             this.validto = "forever";
54             MudgeSanity.systemDetail( s: "valid to", s2: "perpetual");
55         }
56         else {
57             this.validto = "20" + int1;
58             CommonUtils.print_stat( s: "Valid to is: " + this.validto + "");
59             MudgeSanity.systemDetail( s: "valid to", CommonUtils.formatDateAny( s: "MMMM d, YYYY", this.getExpirationDate()));
60         }
61         this.valid = true;
```

这里我方便大家读代码，直接和大家说这些 int byte 的用处

Int1 用于判断 license 失效的时间 29999999 为永久有效。

this.watermark 是水印，我估计是产品控制的一部分

3. Product Control Statement

Functionally, Cobalt Strike aspires to differ little from the advanced threat malware it emulates. As Cobalt Strike makes progress on its defined mission, the dual-use potential of the product becomes a greater challenge. Strategic Cyber LLC's goals are to ensure Cobalt Strike is a force for good that empowers security professionals.

Towards those ends, Strategic Cyber LLC has processes and technology measures to:

- Limit distribution of Cobalt Strike to security professionals who will use the product for ethical penetration testing purposes only
- Make Cobalt Strike less attractive to malicious actors
- Discourage uncontrolled proliferation of the licensed Cobalt Strike product

The Export Compliance Statement documents some of these measures. In addition, Strategic Cyber LLC degrades the trial product's ability to evade defenses and adds a customer identifier to files generated by the licensed product.



byte1 是代表后面的 bytes[] 的长度

这三个 int 都是由 byte[4] getInt 转化而来的，所以这些都可以跳过了，最重要的是

No.4

License 第五部分分析

-
-

```
final byte[] bytes =  
    dataParser.readBytes(dataParser.readByte());
```



```

41     try {
42         final DataParser dataParser = new DataParser(decrypt);
43         dataParser.big();
44         final int int1 = dataParser.readInt();
45         this.watermark = dataParser.readInt();
46         final byte byte1 = dataParser.readByte();
47         final byte[] bytes = dataParser.readBytes(dataParser.readByte());
48         if (byte1 < 40) {
49             this.error = "Authorization file is not for Cobalt Strike 4.0+";
50             return;
51         }
52         if (29999999 == int1) {
53             this.validto = "forever";
54             MudgeSanity.systemDetail( s: "valid to", s2: "perpetual");
55         }
56         else {
57             this.validto = "20" + int1;
58             CommonUtils.print_stat( s: "Valid to is: " + this.validto + "");
59             MudgeSanity.systemDetail( s: "valid to", CommonUtils.formatDateAny( s: "MMMM d, YYYY", this.getExpirationDate()))

```

我们来跟一下 这个 bytes 用来做什么

```

try {
    final DataParser dataParser = new DataParser(decrypt);
    dataParser.big();
    final int int1 = dataParser.readInt();
    this.watermark = dataParser.readInt();
    final byte byte1 = dataParser.readByte();
    final byte[] bytes = dataParser.readBytes(dataParser.readByte());
    if (byte1 < 40) {
        this.error = "Authorization file is not for Cobalt Strike 4.0+";
        return;
    }
    if (29999999 == int1) {
        this.validto = "forever";
        MudgeSanity.systemDetail( s: "valid to", s2: "perpetual");
    }
    else {
        this.validto = "20" + int1;
        CommonUtils.print_stat( s: "Valid to is: " + this.validto + "");
        MudgeSanity.systemDetail( s: "valid to", CommonUtils.formatDateAny( s: "MMMM d, YYYY", this.getExp
    }
    this.valid = true;
    MudgeSanity.systemDetail( s: "id", s2: this.watermark + "");
    SleevedResource.Setup(bytes);
}

```

首先跟进 Setup


```

private static SleevedResource singleton;
private SleeveSecurity data;

public static void setup(final byte[] array) {
    SleevedResource.singleton = new SleevedResource(array);
}

public static byte[] readResource(final String s) {
    return SleevedResource.singleton._readResource(s);
}

private SleevedResource(final byte[] array) {
    (this.data = new SleeveSecurity()).registerKey(array);
}

private byte[] _readResource(final String s) {
    // 将resources/ 替换成 sleeve/
    final byte[] resource = CommonUtils.readResource(CommonUtils.strrep(s, s2: "resources
    if (resource.length > 0) {
        System.currentTimeMillis();

```

先 new 了一个 SleevedResources 对象，这个 array 就是我们传进来的 bytes

```

private SleevedResource(final byte[] array) {
    (this.data = new SleeveSecurity()).registerKey(array);
}

```

跟进，发现是注册 key 用的 跟进 registerKey

```

public void registerKey(final byte[] array) {
    synchronized (this) {
        try {
            final byte[] digest = MessageDigest.getInstance("SHA-256").digest(array);
            final byte[] copyOfRange = Arrays.copyOfRange(digest, from: 0, to: 16);
            final byte[] copyOfRange2 = Arrays.copyOfRange(digest, from: 16, to: 32);
            this.key = new SecretKeySpec(copyOfRange, S: "AES");
            this.hash_key = new SecretKeySpec(copyOfRange2, S: "HmacSHA256");
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}

```



首先他用我们的 array, 获取了一个 digest, 大小是 256

然后分了 0-16 个 byte 作为 AES 加密的 key 16-32 作为 hmac 的 key

✓ Found usages 3 usages

▼ Value read 2 usages

▼ decompiled-cobaltstrike4 2 usages

▼ dns 2 usages

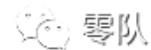
▼ SleeveSecurity 2 usages

▼ encrypt(byte[]) 1 usage

↔ 78 this.mac.init(this.hash_key);

> decrypt(byte[]) 1 usage

> Value write 1 usage



Found usage 发现就加密解密的地方又用到

No.5

解密 key 分析

因为我之前调试时赋值了一串随机 byte，启动 CS 连接 teamserver 的时候发现报了 [Sleeve] Bad HMAC



的错误，这个错误正好在 decrypt 里

```
public byte[] decrypt(final byte[] array) {
    try {
        final byte[] copyOfRange = Arrays.copyOfRange(array, from: 0, to: array.length - 16);
        final byte[] copyOfRange2 = Arrays.copyOfRange(array, from: array.length - 16, array.length);
        byte[] doFinal = null;
        synchronized (this) {
            this.mac.init(this.hash_key);
            doFinal = this.mac.doFinal(copyOfRange);
        }
        if (!MessageDigest.isEqual(copyOfRange2, Arrays.copyOfRange(doFinal, from: 0, to: 16))) {
            CommonUtils.print_error( S: "[Sleeve] Bad HMAC on " + array.length + " byte message from resource");
            return new byte[0];
        }
        byte[] do_decrypt = null;
        synchronized (this) {
            do_decrypt = this.do_decrypt(this.key, copyOfRange);
        }
        final DataInputStream dataInputStream = new DataInputStream(new ByteArrayInputStream(do_decrypt));
        dataInputStream.readInt();
        final int int1 = dataInputStream.readInt();
        if (int1 < 0 || int1 > array.length) {
            CommonUtils.print_error( S: "[Sleeve] Impossible message length: " + int1);
            return new byte[0];
        }
    }
}
```

我们看看这个 decrypt 是用来解密什么的


疯狂 found usage


▼ **Method**


  decrypt(byte[])



▼ **Found usages 1 usage**

▼ Unclassified usage 1 usage

▼  decompiled-cobaltstrike4 1 usage



▼  common 1 usage

▼   SleevedResource 1 usage

▼   _readResource(String) 1 usage


27 return this.data.decrypt(resource);


 零队



  _readResource(String)



▼ **Found usages 1 usage**

▼ Unclassified usage 1 usage

▼  decompiled-cobaltstrike4 1 usage

▼  common 1 usage

▼   SleevedResource 1 usage

▼   readResource(String) 1 usage

15 return SleevedResource.singleton._readResource(s);

 零队

最后在 SleevedResource

中找到了 readResource

```
SleevedResource.java x AuthCrypto.java x Authorization.java x test2.java x DataF
10 public static void Setup(final byte[] array) {
11     SleevedResource.singleton = new SleevedResource
12 }
13
14 public static byte[] readResource(final String s) {
15     return SleevedResource.singleton._readResource(
16 }
17
18 private SleevedResource(final byte[] array) {
```

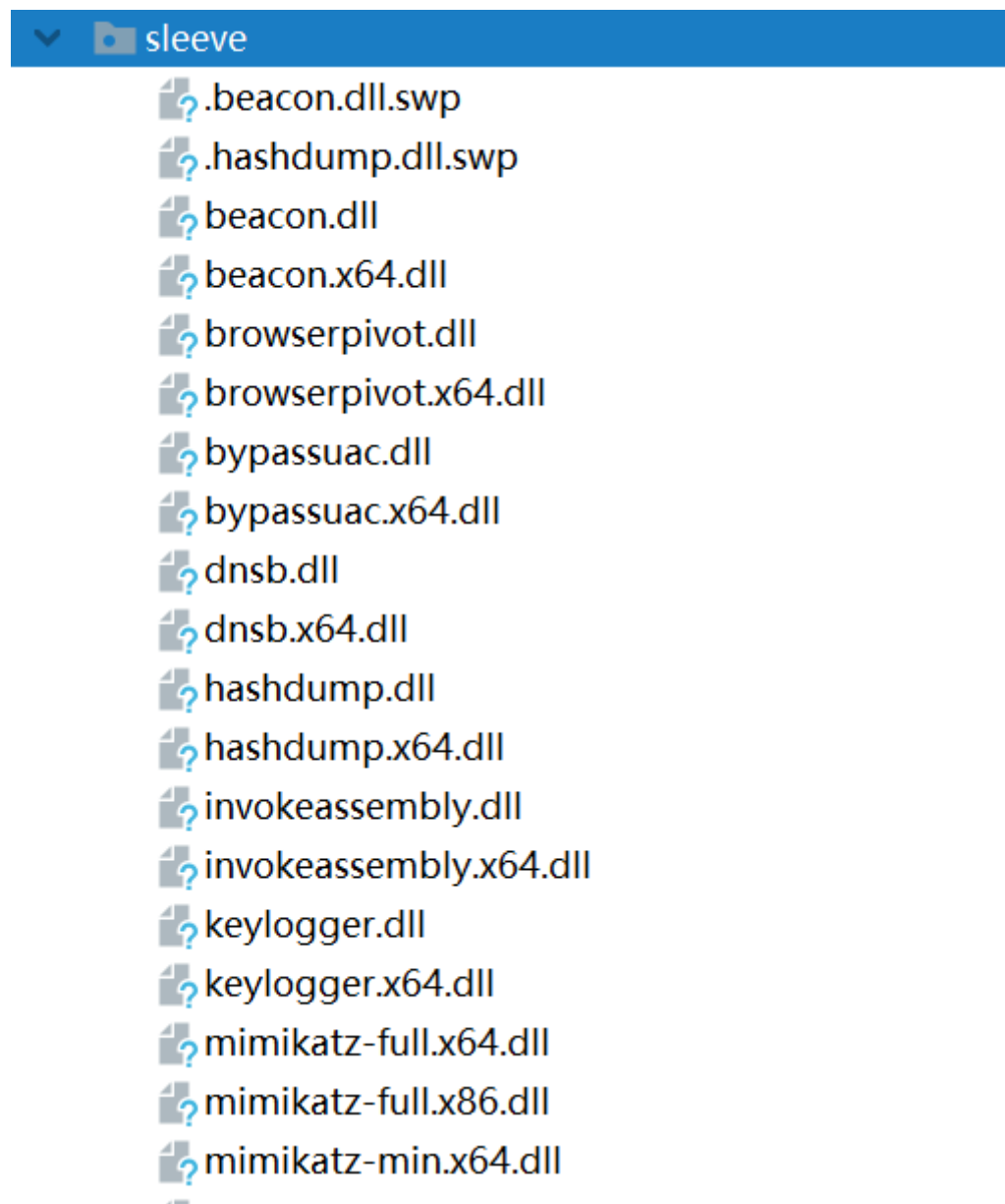
Found 一下发现有很多的调用，找一个比较干净的来分析

```
Unclassified usage 16 usages
  decompiled-cobaltstrike4 16 usages
    beacon 9 usages
      BeaconPayload 4 usages
        setupGargle(Settings, String) 1 usage
          81 final PEParse load = PEParse.load(SleevedResource.readResource(s));
        exportBeaconStage(int, String, boolean, boolean, String) 1 usage
        exportSMBDLL(String) 1 usage
        exportTCPDLL(String, String) 1 usage
      Job 2 usages
      JobSimple 1 usage
        getDLLContent() 1 usage
          39 return SleevedResource.readResource(this.getDLLName());
      PostExInline 2 usages
    beacon.setup 2 usages
      BrowserPivot 1 usage
        export_dll() 1 usage
          45 final String bString = CommonUtils.bString(SleevedResource.readResource(this.x64 ? "resources/browserpivot.x64.dll" : "resources/browserpivot.dll"));
      SSHAgent 1 usage
    c2profile 5 usages
    Lint 4 usages
    Preview 1 usage
```

判断框架位数 之后传入一个文件

```
protected byte[] export_dll() {  
    final String bString = CommonUtils.bString(SleevedResource.readResource(this.x64 ? "resources/browserpivot.x64.dll" : "resources/browserpivot.dll"));  
    final Packer packer = new Packer();  
    packer.little();  
    packer.addShort(this.port);  
    return CommonUtils.toBytes(CommonUtils.replaceAt(bString, CommonUtils.bString(packer.getBytes()), bString.indexOf("COBALTSTRIKE")));  
}
```

进行替换文件夹 读成 byte



? mimikatz-min.x86.dll

? netview.dll

? netview.x64.dll

? pivot.dll

? pivot.x64.dll

? portscan.dll

? portscan.x64.dll

? postex.dll

? postex.x64.dll

🗣️ 零队

```
private byte[] _readResource(final String s) {  
    // 将resources/ 替换成 sleeve/  
    final byte[] resource = CommonUtils.readResource(CommonUtils.strrep(s, s2: "resources/", s3: "sleeve/"));  
    if (resource.length > 0) {  
        System.currentTimeMillis();  
        return this.data.decrypt(resource);  
    }  
    final byte[] resource2 = CommonUtils.readResource(s);  
    if (resource2.length == 0) {  
        CommonUtils.print_error(s: "Could not find sleeved resource: " + s + " [ERROR]");  
    }  
    else {  
        CommonUtils.print_stat(s: "Used internal resource: " + s);  
    }  
    return resource2;  
}
```

🗣️ 零队

然后 decrypt

```

public byte[] decrypt(final byte[] array) {
    try {
        final byte[] copyOfRange = Arrays.copyOfRange(array, from: 0, to: array.length - 16);
        final byte[] copyOfRange2 = Arrays.copyOfRange(array, from: array.length - 16, array.length);
        byte[] doFinal = null;
        synchronized (this) {
            this.mac.init(this.hash_key);
            doFinal = this.mac.doFinal(copyOfRange);
        }
        if (!MessageDigest.isEqual(copyOfRange2, Arrays.copyOfRange(doFinal, from: 0, to: 16))) {
            CommonUtils.print_error(s: "[Sleeve] Bad HMAC on " + array.length + " byte message from resource");
            return new byte[0];
        }
        byte[] do_decrypt = null;
        synchronized (this) {
            do_decrypt = this.do_decrypt(this.key, copyOfRange);
        }
        final DataInputStream dataInputStream = new DataInputStream(new ByteArrayInputStream(do_decrypt));
        dataInputStream.readInt();
    }
}

```

获取前 16 位和后所有位 存成两个 byte

先用我们 license 中的 key 生成的 digest 中的第 16-32 个 byte 作为密钥 加密这前 16 个 byte

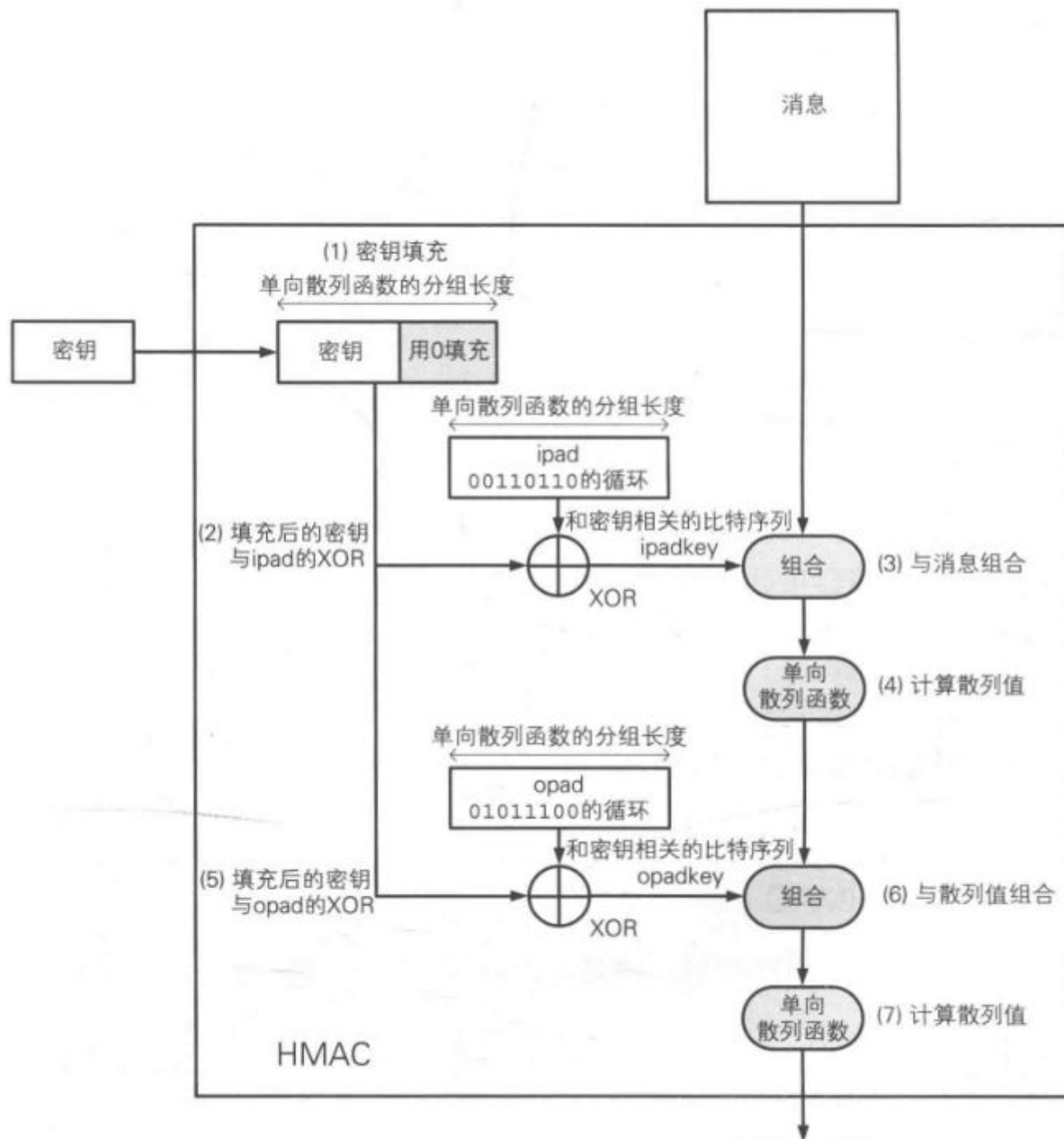
```

...
synchronized (this) {
    this.mac.init(this.hash_key);
    doFinal = this.mac.doFinal(copyOfRange);
}
...
if (!MessageDigest.isEqual(copyOfRange2, Arrays.copyOfRange(doFinal, from: 0, to: 16))) {
    CommonUtils.print_error(s: "[Sleeve] Bad HMAC on " + array.length + " byte message from resource");
    return new byte[0];
}

```

然后和这后 16 位做比较

也就是说 license 剩下的部分，我们知道 Hmac 的明文和密文，然后推导 key。



看图，至少要穷举两次

这还不算完

因为咱们的 license 是通过 sha256 加密得来 我们只知道其中的一小部分 (还得得到 AES 的密钥，和 hmac 的密钥拼接成一个 32 位的 byte)，然后需要穷举来找到一个满足这个 32 位头的字符串

怕了怕了

正当我一筹莫展的时候

灵光一闪，不如从别人的 license 文件里面白嫖一个加进去

直接用他官方的 pubkey 来解密 cobaltstrike.auth 白嫖一个密钥就完事了

No.6

总结

最后给上解密密钥

-

```
{27, -27, -66, 82, -58, 37, 92, 51, 85, -114, -118, 28, -74, 103, -53, 6 };
```

完整的密钥有兴趣的小伙伴请跟着文章自己分析，一起交流成长。

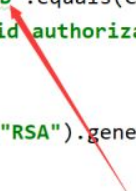
文末也有我生成的现成补丁

后面的事情就很简单了 自己生成 rsa 的公私钥

将前面分析出的 license, 拿我们的私钥签名

将公钥保存到 authpub.key 并计算 md5 值

```
25
26 public void load() {
27     try {
28         final byte[] all = CommonUtils.readAll(CommonUtils.class.getClassLoader().getResourceAs!
29         if (!"7ef5ed238ec450fd1ae2e935d65ddbcb".equals(CommonUtils.toHex(CommonUtils.MD5(all))))
30             CommonUtils.print_error( s: "Invalid authorization file");
31             System.exit( status: 0);
32     }
33     this.pubkey = KeyFactory.getInstance("RSA").generatePublic(new X509EncodedKeySpec(all));
34 }
35 catch (Exception ex) {
36     this.error = "Could not deserialize authpub.key";
37     MudgeSanity.LogException("authpub.key deserialization", ex, false);
38 }
```



所以整个 cobaltstrike 我们只需要改这一行代码就行了

下面附上补丁 包含三个文件

AuthCrypto.class

authpub.key

cobaltstrike.auth

将 AuthCrypto.class

复制到 common/AuthCrypto.class

将 authkey.pub

复制到 resources/authkey.pub

将 cobaltstrike2.auth

放在和 cobaltstrike.jar 同级目录

自给自足，丰衣足食，再也不用因为各种破解版的 cobaltstrike 是否有后门而担心



链接：

<https://pan.baidu.com/s/1yNtuezjZZeQ5KKVHsRnQEw>

提取码：2mur