

当frida来"敲"门

原创 队员编号006 酒仙桥六号部队 5月14日

这是 酒仙桥六号部队 的第 6 篇文章。

全文共计3156个字，预计阅读时长9分钟。

1 渗透测试瓶颈

目前，碰到越来越多的大客户都会将核心资产业务集中在统一的APP上，或者对自己比较重要的APP，如自己的主业务，办公APP进行加壳，流量加密，投入了很多精力在移动端的防护上。

```

POST /login HTTP/1.1
Host:
Content-Type: application/json;charset=UTF-8
Connection: close
Accept: */*
User-Agent: (iPhone; iOS 13.4; Scale/3.00)
Accept-Language: zh-Hans;q=1
Content-Length: 512
Accept-Encoding: gzip, deflate

l4yIYM93e+prMN/iXRCBWS3HuaghYMIHTUvejXE1yHeM4PLb2yGIEckOP8u6c/
kxvKR3Z4Z6dCUF39nldjnQKLWM4qGCzPYvPYEidQX2K7FTjMO3c2i4f2B5OUI
di5TLBVMQkQN49BlzY8k6+LqRa/KoT2/kAid3NuO1z+bkMSCpFroBvKqQ0J7
M0r198ujoBOEOlhxmtzGAXIfDW4mGKxv4eUs3PgQI5QG+iWRWesJ6O6tgnRe
TjZuZjVwCSvO/Khe61h88lytG2gppflkVj7oAY/3zCy1EZ2oKsVxy6N+xzASbZK
S8UtKi5ZdXDJLgOTqCWOrOmCex3RMj7b0xYRvBKIwCDNrm8nodHGK4hG58
3A2s7lQArf2yUPWgUy5uvbN06n6gpQsHG2kQ3BqudZ+ruIkXyYfzdf/7gCEip
eiT0hdnoA8+L1xspCiljmAb1CL05J9n3yskOrnmrlblxQ/bmWKnKxBjFiYf19fu
OvaZsG3Obrezq+ahQKAyjj

HTTP/1.1 200
Server: nginx
Date: Wed, 29 Apr 2020 06:49:27 GMT
Content-Type: application/json
Connection: close
Vary: Accept-Encoding
Set-Cookie: JSESSIONID=36b6dd84-bb9d-47c6-9fbb-2d2503cfce00;
Path=/xweb; HttpOnly
Set-Cookie: rememberMe=deleteMe; Path=/xweb; Max-Age=0;
Expires=Tue, 28-Apr-2020 06:49:27 GMT
Content-Length: 174

"r9J6bAdrAdyR7fbCeCP5eRc5wsaS6sNfielaJ3k8PGVh1XDD1AbIp6ET6Qptad
8NYW1avd45qrPLENO4LufDFdTh/kjdBw9B+bykr/KB1z61xHB6SXVclMffQqs
3Mm9Tkj8jiFOK9du1acAVR1OwCW0z0Oc9Q8P2+ahzT6vRpQ8="

```

而现在挖漏洞除了拿到shell以外，客户又要求可以尽可能的挖到核心业务系统的漏洞，并将漏洞范围订在主域名，核心业务系统现在又基本集中在移动端，移动端现在都会进行APP加壳，流量加密。这就导致无法进行平常渗透测试过程，像老生常谈的中间人攻击，进行拦截，篡改数据包就很难进行。

接下来就尝试解决中间人攻击的问题，目标是

- 1.看到明文的request和response的数据包；
- 2.做到可以拦截，篡改数据包。

2 frida

frida是平台原生app的Greasemonkey，说的专业一点，就是一种动态插桩工具，可以插入一些代码到原生app的内存空间去，（动态地监视和修改其行为），这些原生平台可以是Win、Mac、Linux、Android或者iOS。而且frida还是开源的。

FRIDA

环境需要越狱的IOS或者ROOT的Android。安装的版本需要一致。

MAC:

```
frida --version  
12.7.16
```

越狱Iphone:

```
ios:~ mobile$ frida-server --version  
12.7.22  
ios:~ mobile$ █
```

通过USB链接越狱手机，可以执行frida-ps -aU 就代表环境安装成功。

```
frida-ps -aU
PID   Name          Identifier
-----
3949  Terminal      com.officialscheduler.mterminal
3894  微信          com.tencent.xin
3947  ██████████    ██████████
3957  照片          com.apple.mobileslideshow
3886  设置          com.apple.Preferences
4000  邮件          com.apple.mobilemail
```

3 越狱检测绕过

启动目标APP时，APP自身会进行环境检测，如果处于越狱环境会提示如下：



点击“我知道了”就直接退出APP。

所以先尝试先绕过第一步越狱环境检测。可以先尝试搜索包含“*jail,jeil,jb,break*”关键字的函数。

关于函数追踪可以使用frida-trace，如：

```
1 # Trace recv* and send* APIs in Safari
2 $ frida-trace -i "recv*" -i "send*" Safari
3
4 # Trace ObjC method calls in Safari
5 $ frida-trace -m "-[UIView drawRect:]" Safari
6
7 # Launch SnapChat on your iPhone and trace crypto API calls
8 $ frida-trace -U -f com.toyopagroup.picaboo -I "libcommonCrypto*"
```

burp的插件brida也支持对函数名进行检索hook，和"Jail"相关的越狱检测函数如下：

```

1  **** Result of the search of Jail
2  OBJC: +[BLYDevice isJailBreak]
3  OBJC: +[IFlySystemInfo isJailbroken]
4  OBJC: +[UIScreen _shouldDisableJail]
5  OBJC: +[UIStatusBarWindow isIncludedInClassicJail]
6  OBJC: -[_UIHostedWindow _isConstrainedByScreenJail]
7  OBJC: -[_UIRootWindow _isConstrainedByScreenJail]
8  OBJC: -[_UISnapshotWindow _isConstrainedByScreenJail]
9  OBJC: -[BLYDevice isJailbroken]
10 OBJC: -[BLYDevice setJailbrokenStatus:]
11 OBJC: -[RCCountly isJailbroken]
12 OBJC: -[UIClassicWindow _isConstrainedByScreenJail]
13 OBJC: -[UIDevice isJailbroken]
14 OBJC: -[UIStatusBarWindow _isConstrainedByScreenJail]
15 OBJC: -[UITextEffectsWindowHosted _isConstrainedByScreenJail]
16 OBJC: -[UIWindow _clampPointToScreenJail:]
17 OBJC: -[UIWindow _isConstrainedByScreenJail]

```

想将目标定在“OBJC: +[BLYDevice isJailBreak]”。

frida启动APP，并加载脚本的命令如下：

```
1 frida -U -f com.x.x -l js-scripts
```

js脚本编写可以看官方文档：

<https://frida.re/docs/javascript-api/>

```

1 //hook传入值, ObjC: args[0] = self, args[1] = selector, args[2-n] = argument
2 Interceptor.attach(myFunction.implementation, {
3   onEnter: function(args) {
4     var myString = new ObjC.Object(args[2]);
5     console.log("String argument: " + myString.toString());

```

```

6   }
7   });
8
9   //hook返回值,
10  Interceptor.attach(Module.getExportByName('libc.so', 'read'), {
11    onEnter: function (args) {
12      this.fileDescriptor = args[0].toInt32();
13    },
14    onLeave: function (retval) {
15      if (retval.toInt32() > 0) {
16        /* do something with this.fileDescriptor */
17      }
18    }
19  });

```

定义js脚本后，尝试hook出“OBJC:

+ [BLYDevice isJailBreak]”的传入值和返回值。

```

1  function hook_specific_method_of_class(className, funcName)
2  {
3    var hook = ObjC.classes[className][funcName];
4    Interceptor.attach(hook.implementation, {
5      onEnter: function(args) {
6        // args[0] is self
7        // args[1] is selector (SEL "sendMessageWithText:")
8        // args[2] holds the first function argument, an NSString
9        console.log("\n\t[*] Class Name: " + className);
10       console.log("[*] Method Name: " + funcName);
11       //For viewing and manipulating arguments
12       //console.log("\t[-] Value1: "+ObjC.Object(args[2]));
13       //console.log("\t[-] Value2: "+(ObjC.Object(args[2])).toString());
14       console.log("\t[-] arg value  "+args[2]);
15       Interceptor.attach(hook.implementation,
16         {
17           onLeave: function(retval) {
18             console.log("[*] Class Name: " + className);
19             console.log("[*] Method Name: " + funcName);
20             console.log("\t[-] Return Value: " + retval);
21           }
22         }
23     });

```

```

22         );
23     }
24     });
25 }
26 //Your class name and function name here
27 hook_specific_method_of_class("BLYDevice", "- isJailbroken")

```

```

Spawned [redacted] . Use %resume to let the main thread start executing!
[iPhone: [redacted]]->
[iPhone: [redacted]]-> %resume
[iPhone: [redacted]]->
[*] Class Name: BLYDevice
[*] Method Name: - isJailbroken
[-] arg value 0x8
[*] Class Name: BLYDevice
[*] Method Name: - isJailbroken
[-] Return Value: 0x1
[*] Class Name: BLYDevice
[*] Method Name: - isJailbroken
[-] arg value 0x0
[*] Class Name: BLYDevice
[*] Method Name: - isJailbroken
[-] Return Value: 0x1
[*] Class Name: BLYDevice
[*] Method Name: - isJailbroken
[-] Return Value: 0x1

```

篡改后，发现未能绕过，可能不是这个函数做最终的逻辑判断，想到竟然都弹窗提示了，和UI有关系。

那么可能是“OBJC: -[UIDevice isJailbroken]”这个类，最终构造绕过越狱检测代码如下：

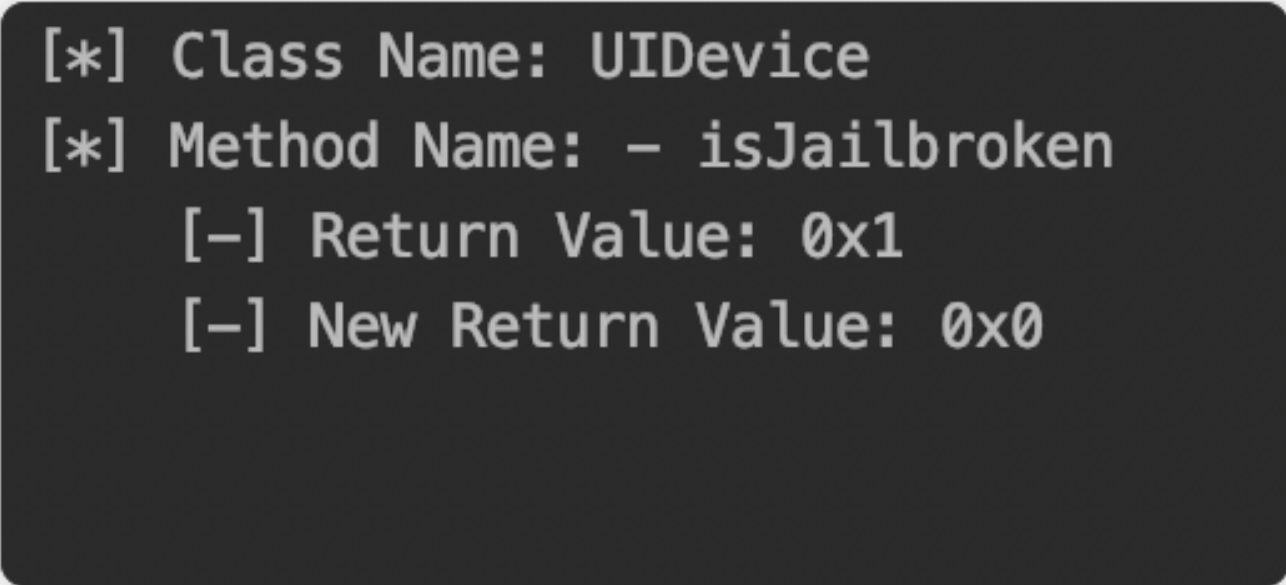
```

1  if (ObjC.available)
2  {
3      try
4      {
5          var className = "UIDevice";
6          var funcName = "- isJailbroken";
7          var hook = eval('ObjC.classes.' + className + '[' + funcName +
8          Interceptor.attach(hook.implementation,
9          {
10             onLeave: function(retval) {
11                 console.log("[*] Class Name: " + className);
12                 console.log("[*] Method Name: " + funcName);
13                 console.log("\t[-] Return Value: " + retval); // 输出原本的返回值
14                 var newretval = ptr("0x0")

```

```
15         retval.replace(newretval)// 替换新的返回值
16         console.log("\t[-] New Return Value: " + newretval)
17     }}
18     );
19 }
20 catch(err)
21 {
22     console.log("[!] Exception2: " + err.message);
23 }
24 }
25 else
26 {
27     console.log("Objective-C Runtime is not available!");
28 }
```

执行结果如下：



```
[*] Class Name: UIDevice
[*] Method Name: - isJailbroken
[-] Return Value: 0x1
[-] New Return Value: 0x0
```

成功绕过。



4 HOOK加解密函数

越狱检测绕过后，进一步开始尝试定位加解密的函数。

关于定位加解密函数这块在Android可以尝试使用traceview去分析追踪函数。

(<https://developer.android.google.cn/studio/profile/traceview>)

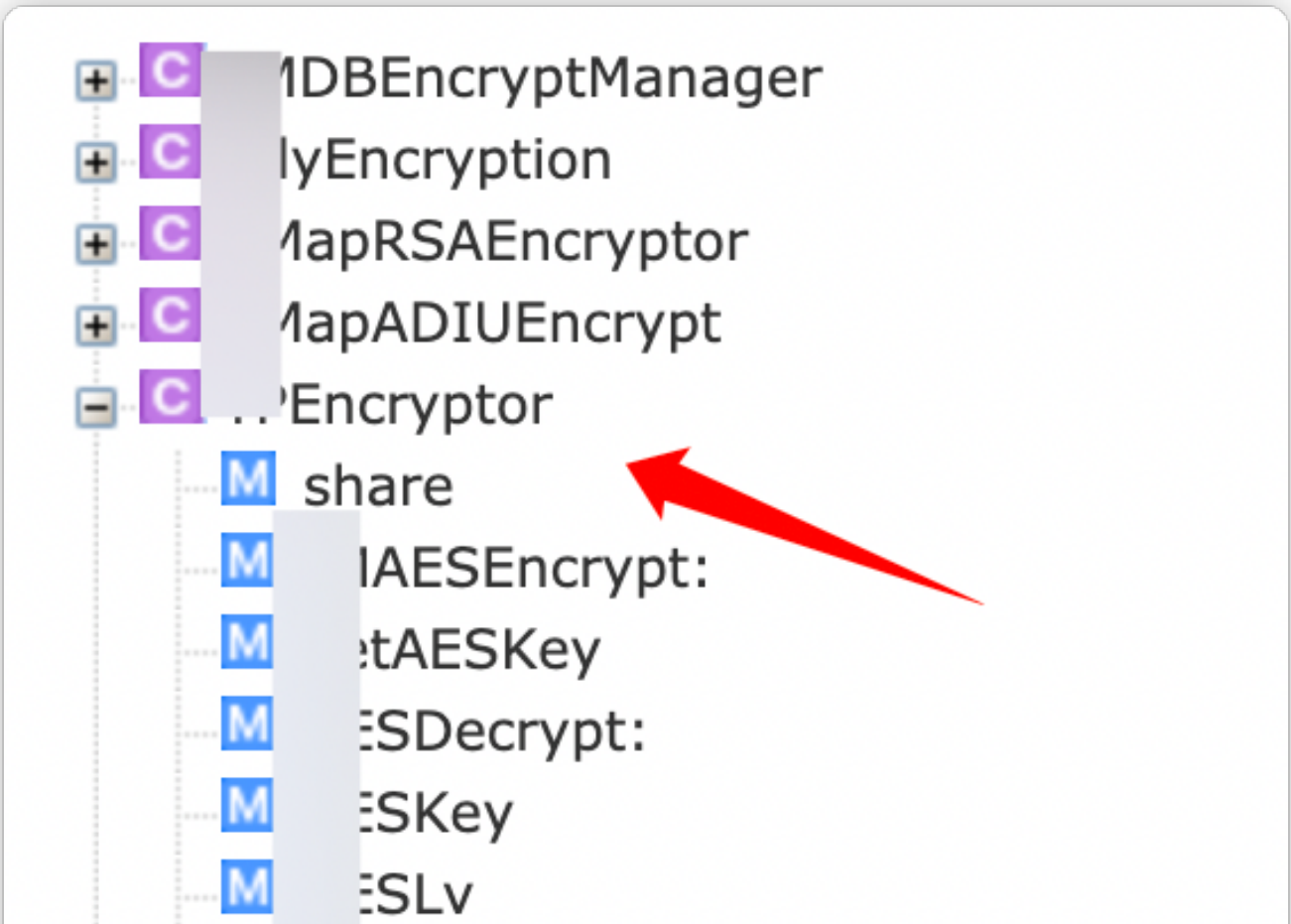
Name	Incl Cpu Tir	Incl Cpu Time	Excl Cpu Time %	Excl Cpu Time	Incl Real Time %	Incl Real Time	Excl Real
16 android.os.Parcel.writeInt (I)V	9.2%	0.562	5.5%	0.335	4.5%	0.558	
▼ Parents							
20 android.os.Parcel.writeValue (Ljava/lang...	53.4%	0.300			53.4%	0.298	
9 android.os.BaseBundle.writeToParcelInn...	15.5%	0.087			15.4%	0.086	
18 android.content.Intent.writeToParcel (L...	14.4%	0.081			14.5%	0.081	
5 android.app.ActivityManagerProxy.start...	9.1%	0.051			9.0%	0.050	
13 android.os.Parcel.writeArrayMapIntern...	4.8%	0.027			4.7%	0.026	
119 android.net.Uri.writeToParcel (Landro...	2.8%	0.016			3.0%	0.017	
▼ Children							
self	59.6%	0.335			59.7%	0.333	
33 android.os.Parcel.nativeWriteInt (J)V	40.4%	0.227			40.3%	0.225	
17 android.os.BaseBundle.putInt (Ljava/lang/Str...	8.1%	0.498	1.2%	0.072	4.2%	0.518	
18 android.content.Intent.writeToParcel (Landro...	8.1%	0.496	0.5%	0.033	4.0%	0.497	
19 android.util.ArrayMap.put (Ljava/lang/Objec...	7.3%	0.447	3.4%	0.210	3.8%	0.468	
20 android.os.Parcel.writeValue (Ljava/lang/Obj...	7.1%	0.437	1.6%	0.098	3.5%	0.434	
21 android.view.IWindow\$Stub.onTransact (ILan...	6.9%	0.420	1.1%	0.065	5.4%	0.668	
22 android.app.Activity.isTopOfTask (I)Z	6.7%	0.410	0.2%	0.012	5.0%	0.619	

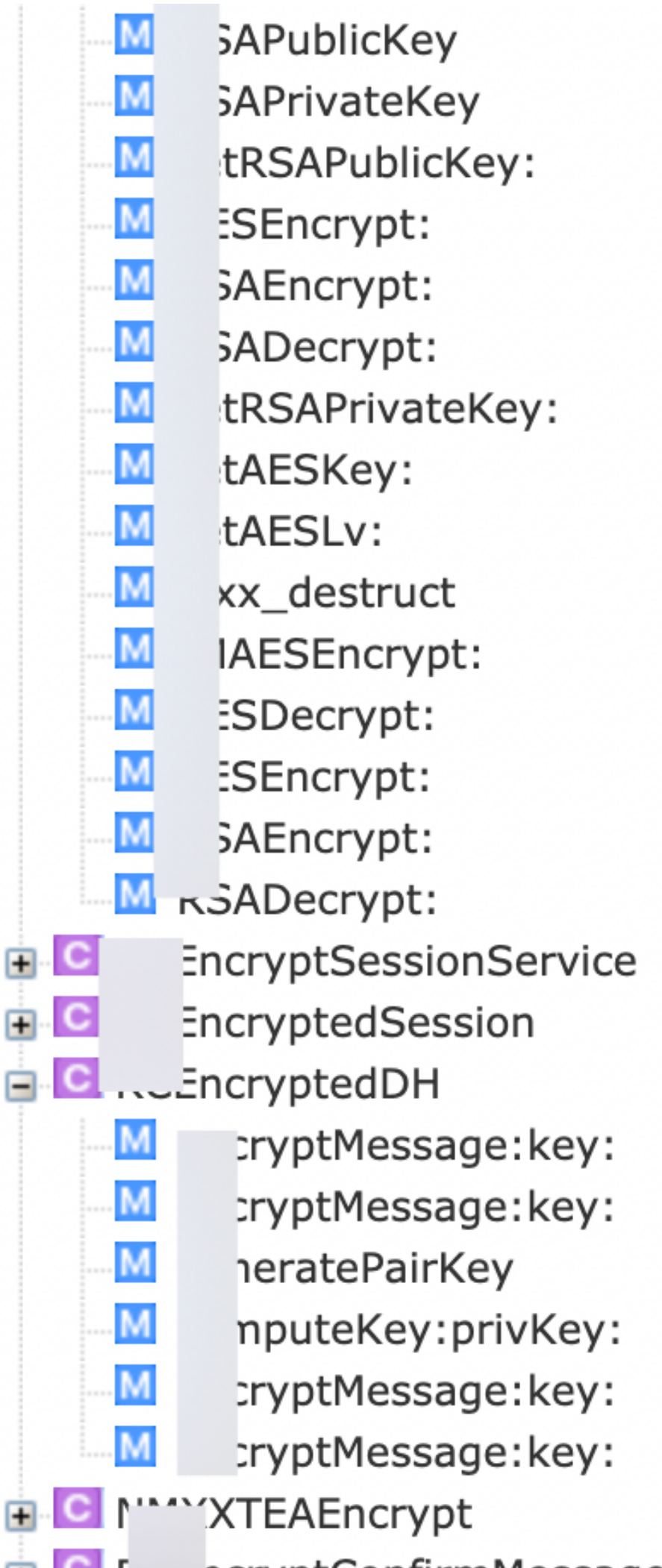
IOS可以尝试使用runtime去追踪函数，uidump从界面按钮入手，Nslog日志等位置入手，或者直接找相关关键字的函数去入手。

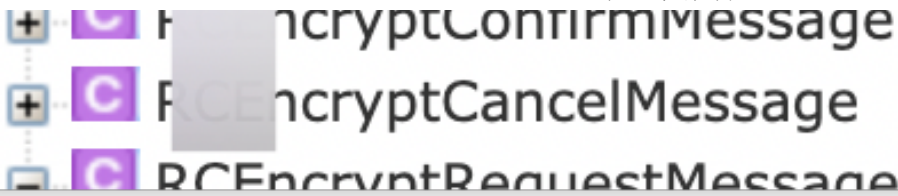
例如crypt(decrypt,encrypt),HTTP,Network,目标厂商的名字简写找不到，可以尝试搜索NSString系统库等。

这边推荐一个大佬的github项目。使用可以参考这个github项目，非常好用，先用之前写好的绕过越狱检测的脚本启动APP，这边通过查找函数名找到对方关键的加解密函数“*encryptor”。

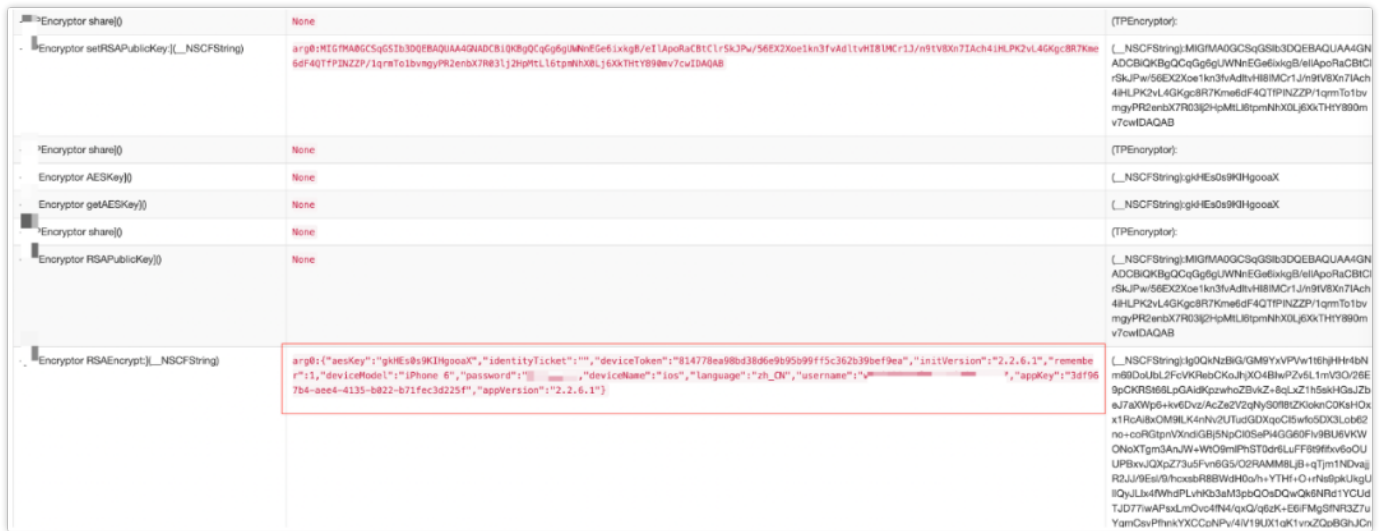
github项目：<https://github.com/lyxhh/lxhToolHTTPDecrypt>



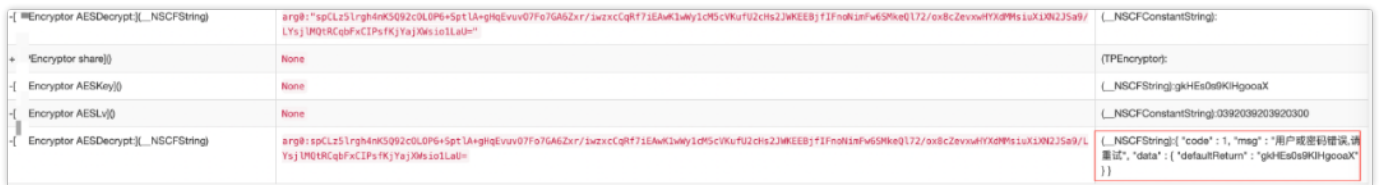




hook此函数的所有方法，在点击登录按钮后，观察到有请求的数据包被当做参数传入到-[XXEncryptor RSAEncrypt:]方法内，并返回了加密后的字符串。-[XXEncryptor setRSAPublicKey:]根据定义的方法名判断应该是RSA公钥信息。

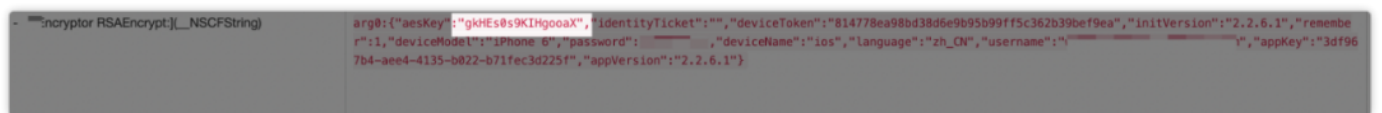


其他方法则去处理了返回包。如-[XXEncryptor AESDecrypt:]方法，将服务端返回的加密字段，使用AES对称解密解密为明文。

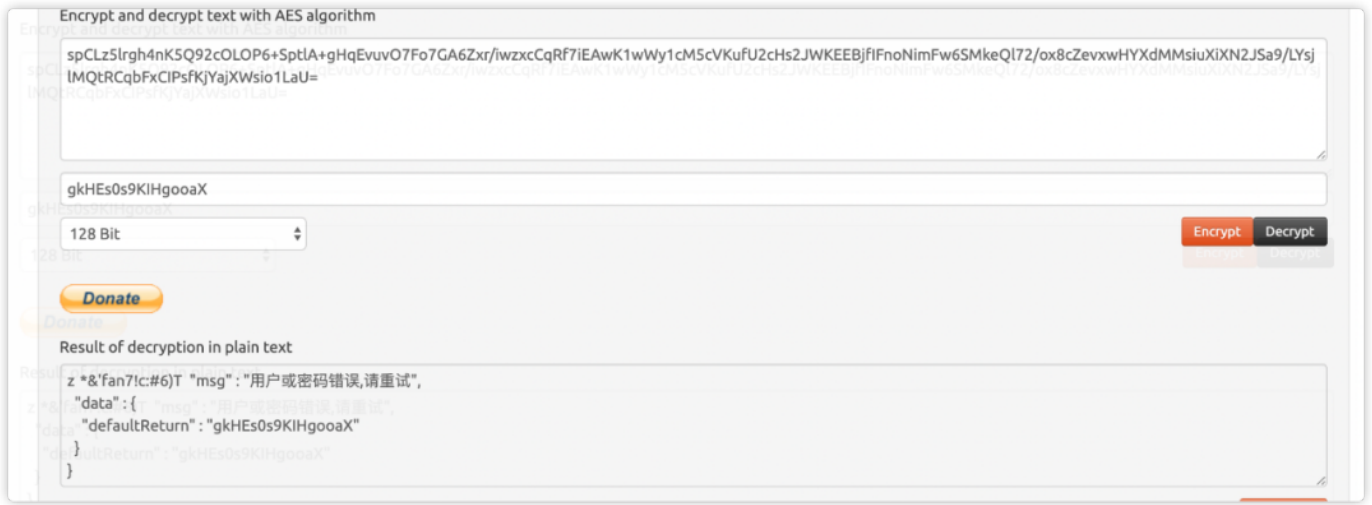


之前我们在Hook请求包函数的时候发现明文的数据包里面带有aeskey，说明此处的逻辑应该是：

本地生成aeskey代入到request包->使用定义的RSA公钥加密request->发送到服务端并解密request后->处理请求包内容，并使用AESKey加密Response返回到客户端->客户端在使用Aeskey解密服务端的Response包。



大概是这么一个流程，事实也证明返回包确实可以使用hook到的aeskey进行解密。



后面的思路是hook[XXEncryptor AESDecrypt:]解密方法去解密请求包和返回包，返回包是可以解，但是突然想到请求包是RSA非对称的，需要私钥。想尝试在客户端找到RSA的私钥或者RSA解密方法，结果也确实有RSADecrypt方法。

```

✓ {"methodInfo": "+ share", "length": 0}
{"methodInfo": "- IMAESEncrypt:", "length": 1}
{"methodInfo": "- getAESKey", "length": 0}
{"methodInfo": "- AESDecrypt:", "length": 1}
{"methodInfo": "- AESKey", "length": 0}
{"methodInfo": "- AESLv", "length": 0}
{"methodInfo": "- RSAPublicKey", "length": 0}
{"methodInfo": "- RSAPrivateKey", "length": 0}
{"methodInfo": "- setRSAPublicKey:", "length": 1}
{"methodInfo": "- AESEncrypt:", "length": 1}
{"methodInfo": "- RSAEncrypt:", "length": 1}
{"methodInfo": "- RSADecrypt:", "length": 1}
{"methodInfo": "- setRSAPrivateKey:", "length": 1}
{"methodInfo": "- setAESKey:", "length": 1}
{"methodInfo": "- setAESLv:", "length": 1}
{"methodInfo": "- .cxx_destruct", "length": 0}

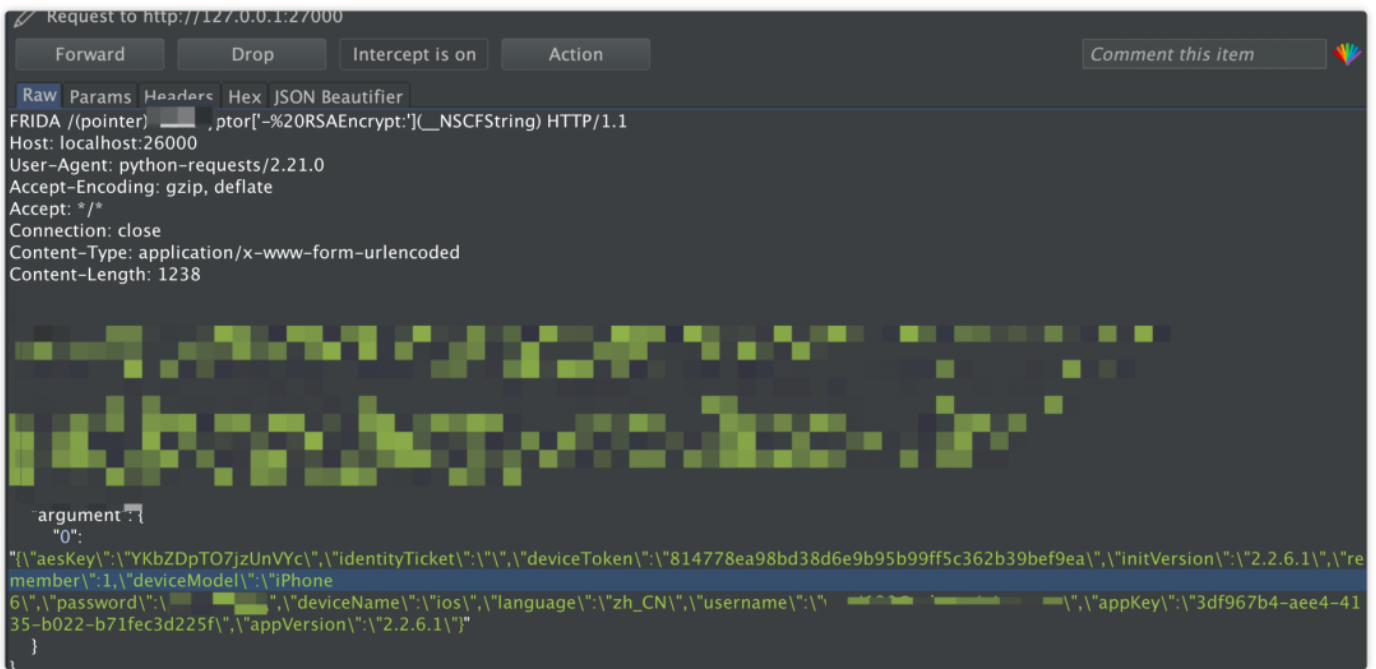
```

但是事实是，从头到尾这个方法都没有被使用过，没有参数被传入，也没有返回值。所以想，可能本地不做请求包的解密。那么调用他的函数解密返回包可行，但解密请求包不行。但是咱们之前是有Hook到明文的request，可以再request被传入到-[XXEncryptor RSAEncrypt]方法前，先去修改arg。



具体操作方法可以参考lyxhh，将加密前的请求包转入Burp后就可以实现篡改数据了。

lyxhh: <https://github.com/lyxhh/lxhToolHTTPDecrypt>



新手的话可以先用la0s的JS，先看看对方是不是使用了IOS统一封装的Crypto库,js脚本如下：

JS:https://la0s.github.io/2018/12/07/iOS_Crypto/

```

1 / Intercept the CCCrypt call.
2 Interceptor.attach(Module.findExportByName('libcommonCrypto.dylib', 'CC
3   onEnter: function (args) {
4     // Save the arguments
5     this.operation    = args[0]
6     this.CCAgorithm  = args[1]
7     this.CCOptions   = args[2]
8     this.keyBytes    = args[3]
9     this.keyLength   = args[4]
10    this.ivBuffer     = args[5]
11    this.inBuffer     = args[6]
12    this.inLength     = args[7]
13    this.outBuffer    = args[8]
14    this.outLength    = args[9]
15    this.outCountPtr  = args[10]
16
17    console.log('CCCrypt(' +
18                'operation: ' + this.operation + ', ' +
19                'CCAgorithm: ' + this.CCAgorithm + ', ' +
20                'CCOptions: ' + this.CCOptions + ', ' +
21                'keyBytes: ' + this.keyBytes + ', ' +

```

```
22     'keyLength: ' + this.keyLength + ', ' +
23     'ivBuffer: ' + this.ivBuffer + ', ' +
24     'inBuffer: ' + this.inBuffer + ', ' +
25     'inLength: ' + this.inLength + ', ' +
26     'outBuffer: ' + this.outBuffer + ', ' +
27     'outLength: ' + this.outLength + ', ' +
28     'outCountPtr: ' + this.outCountPtr + '))'
29
30     if (this.operation == 0) {
31         // Show the buffers here if this an encryption operation
32         console.log("In buffer:")
33         console.log(hexdump(ptr(this.inBuffer), {
34             length: this.inLength.toInt32(),
35             header: true,
36             ansi: true
37         })))
38         console.log("Key: ")
39         console.log(hexdump(ptr(this.keyBytes), {
40             length: this.keyLength.toInt32(),
41             header: true,
42             ansi: true
43         })))
44         console.log("IV: ")
45         console.log(hexdump(ptr(this.ivBuffer), {
46             length: this.keyLength.toInt32(),
47             header: true,
48             ansi: true
49         })))
50     }
51 },
52 onLeave: function (retVal) {
53     if (this.operation == 1) {
54         // Show the buffers here if this a decryption operation
55         console.log("Out buffer:")
56         console.log(hexdump(ptr(this.outBuffer), {
57             length: Memory.readUInt(this.outCountPtr),
58             header: true,
59             ansi: true
60         })))
61         console.log("Key: ")
```

```
62     console.log(hexdump(ptr(this.keyBytes), {
63         length: this.keyLength.toInt32(),
64         header: true,
65         ansi: true
66     })))
67     console.log("IV: ")
68     console.log(hexdump(ptr(this.ivBuffer), {
69         length: this.keyLength.toInt32(),
70         header: true,
71         ansi: true
72     })))
73 }
74 }
75 })
```

如果只能hook到部分明文流量，再考虑去对方定义的函数里去找关键的加密函数，如这个APP的关键的XXEncryptor类。



知其黑 守其白

分享知识盛宴，闲聊大院趣事，备好酒肉等你



长按二维码关注 酒仙桥六号部队