Hybrid Android协议加解密分析

Bojack / 2019-10-15 09:20:41 / 浏览数 4427 安全技术 移动安全 顶(0) 踩(0)

前言

App渗透再次遇到加解密，并且这次具体分析与以往状况有点不太一样，于是记录下来。

抓包分析

```
POST /app/fsOrder/getAllFsOrderListByUser/ HTTP/1.1
accept: application/json
authorization: Bearer
timestamp: 1567065637961
Content-Type: application/json; charset=utf-8
Content-Length: 256
Host: xxx.com
Connection: close
User-Agent: okhttp/3.6.0

{"data":"WmPKAOqVK3nmj2751oQM/1fyZJ/QQIMe2itv4LufWyk87WgwkJScqu68J/IQX1Pr","key":"LywemIhGqlzqDBOXOPxdY+nifhoq2lBILu2N6WUpJ/4H
```

可以看到请求包中数据都做了加密，分别有"data"和"key"两个字段。可以大胆猜测data为请求的数据，而key应该是加密data的密钥。惯用手段是用随机值作为key加密数

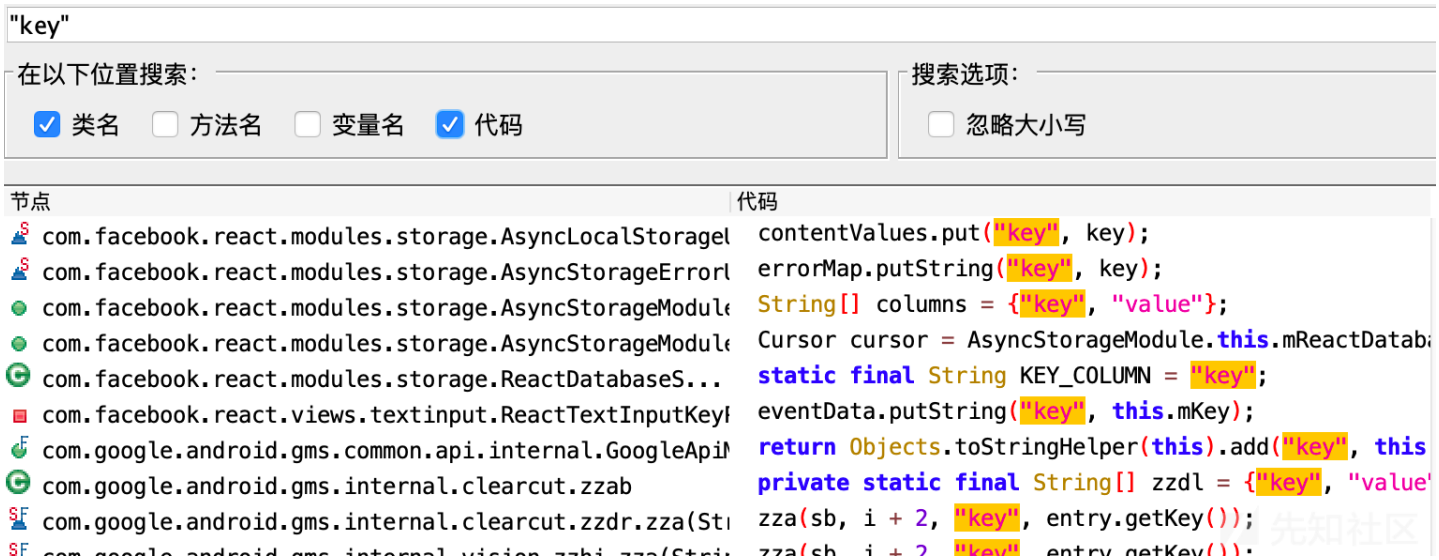
```
HTTP/1.1 200 OK
Date: Thu, 29 Aug 2019 08:00:22 GMT
Content-Type: application/json;charset=UTF-8
Content-Length: 216
Connection: close

{"data":"nt4BMbi2d7oZ/bFg5mwe2w==","key":"o72x8DRL62zUVojRxEsUEP5w2Aa6BXWAZqo8sAFSK9sK47YiGewIIl9LCa1OQ6JgoXn+jVUdENfPSilBBEVw
```

逆向分析Java代码

关键词搜索

因为key和data比较常见搜索结果肯定大多都是第三方SDK的引用，先搜索一下路由`getAllFsOrderListByUser`，奇怪的是并没有任何搜索结果。于是只能搜索"key"和"



多归多正常来说都能在搜索结果找到一些蛛丝马迹，但是这里发现全都是第三方SDK的引用，仔细一看发现主程序的代码少得可怜，并且注意到MainActivity继承了faceboo

```
import com.facebook.react.ReactActivity;

public class MainActivity extends ReactActivity {
   /* access modifiers changed from: protected */
   public String getMainComponentName() {
       return "SafetyMonitoring";
   }
```

}

再结合assets资源目录来看这个app是采用了Hybrid开发模式，部分逻辑写在了javascript上。但这也不意味着加解密就一定在js上面，Google了一下好像还有js调用Andro

但无论如何还是先从Java代码找起，搜索一下常见的加解密函数字段AES、DES、RSA等

```
package com.alipay.security.mobile.module.a.a;

import com.alipay.security.mobile.module.a.a;
import java.lang.reflect.Method;
import java.nio.ByteBuffer;
import java.util.Arrays;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

public final class c {
    private static String a = new String("idnjfhncnsfuobcnt847y929o449u474w7j3h22aoddc98euk#%&&)*&^%#");

    public static String a() {
        String str = new String();
        for (int i = 0; i < a.length() - 1; i += 4) {
            str = str + a.charAt(i);
        }
        return str;
    }

    public static String a(String str, String str2) {
        try {
            PBEKeySpec a2 = a(str);
            byte[] bytes = str2.getBytes();
            SecretKeySpec secretKeySpec = new SecretKeySpec(SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1").
            Cipher instance = Cipher.getInstance("AES/CBC/PKCS5Padding");
            instance.init(1, secretKeySpec, new IvParameterSpec(new byte[instance.getBlockSize()]));
            byte[] salt = a2.getSalt();
            ByteBuffer allocate = ByteBuffer.allocate(salt.length + instance.getOutputSize(bytes.length));
            allocate.put(salt);
            instance.doFinal(ByteBuffer.wrap(bytes), allocate);
```

alipay.security里的Cipher加解密吸引了我，而且这里还有一些硬编码字段，一度让我以为这就是密钥。追踪一下函数的调用关系，似乎没有找到调用的痕迹，继续分析也无

方法剖析Method Profiling

避免其他函数的混淆，我在登陆过程中抓取函数调用栈，看了很久看到com.loc.n.a这里似乎有调用什么方法进行加解密（特别这种命名不清不楚的函数更让人值得关注）

Name	Incl Cpu Time %	Incl Cpu Time
▶ 122 com.facebook.react.uimanager.UIManagerModule.createView (ILjava/lang/String;ILcom/facebook/react/br...	0.6%	8.073
▶ 123 android.os.SystemClock.uptimeMillis ()J	0.6%	7.976
▼ 124 com.loc.n.a (Lcom/loc/n$a;)[B	0.6%	7.672
▼ Parents		
94 com.loc.n.a (Landroid/content/Context;Z)[B	100.0%	7.672
▼ Children		
self	0.0%	0.000
216 com.loc.q.a ([BLjava/security/Key;)[B	49.2%	3.773
484 com.loc.n.a (Ljava/io/ByteArrayOutputStream;Ljava/lang/String;)V	20.3%	1.556
554 com.loc.w.d ()Ljava/security/PublicKey;	17.0%	1.308
753 com.loc.w.b ([B)[B	13.5%	1.035
▶ 125 okhttp3.RealCall.getResponseWithInterceptorChain ()Lokhttp3/Response;	0.6%	7.530

根据Method
Profiling调用栈可以清晰的找到函数调用和被调用关系，这种方式我觉得是定位函数最实在的方法。在分析的时候发现某个函数没法正常反编译成java代码。

```
a(r2, r0)     // Catch:{ Throwable -> 0x00ed }
            byte[] r0 = r2.toByteArray()     // Catch:{ Throwable -> 0x00ed }
            byte[] r3 = com.loc.w.b(r0)     // Catch:{ Throwable -> 0x00ed }
            java.security.PublicKey r0 = com.loc.w.d()     // Catch:{ Throwable -> 0x00ed }
            int r4 = r3.length     // Catch:{ Throwable -> 0x00ed }
            if (r4 <= r5) goto L_0x00bf
            r4 = 117(0x75, float:1.64E-43)
            byte[] r4 = new byte[r4]     // Catch:{ Throwable -> 0x00ed }
            r5 = 0
```

```
r6 = 0
r7 = 117(0x75, float:1.64E-43)
java.lang.System.arraycopy(r3, r5, r4, r6, r7)     // Catch:{ Throwable -> 0x00ed }
byte[] r4 = com.loc.q.a(r4, r0)     // Catch:{ Throwable -> 0x00ed }
int r0 = r3.length     // Catch:{ Throwable -> 0x00ed }
```

动态调试

突然被问到为什么不直接动态调试因为apk没有也加固，一调试就知道是不是在java层做加密了，于是根据Method Profiling定位到的函数下断点调试

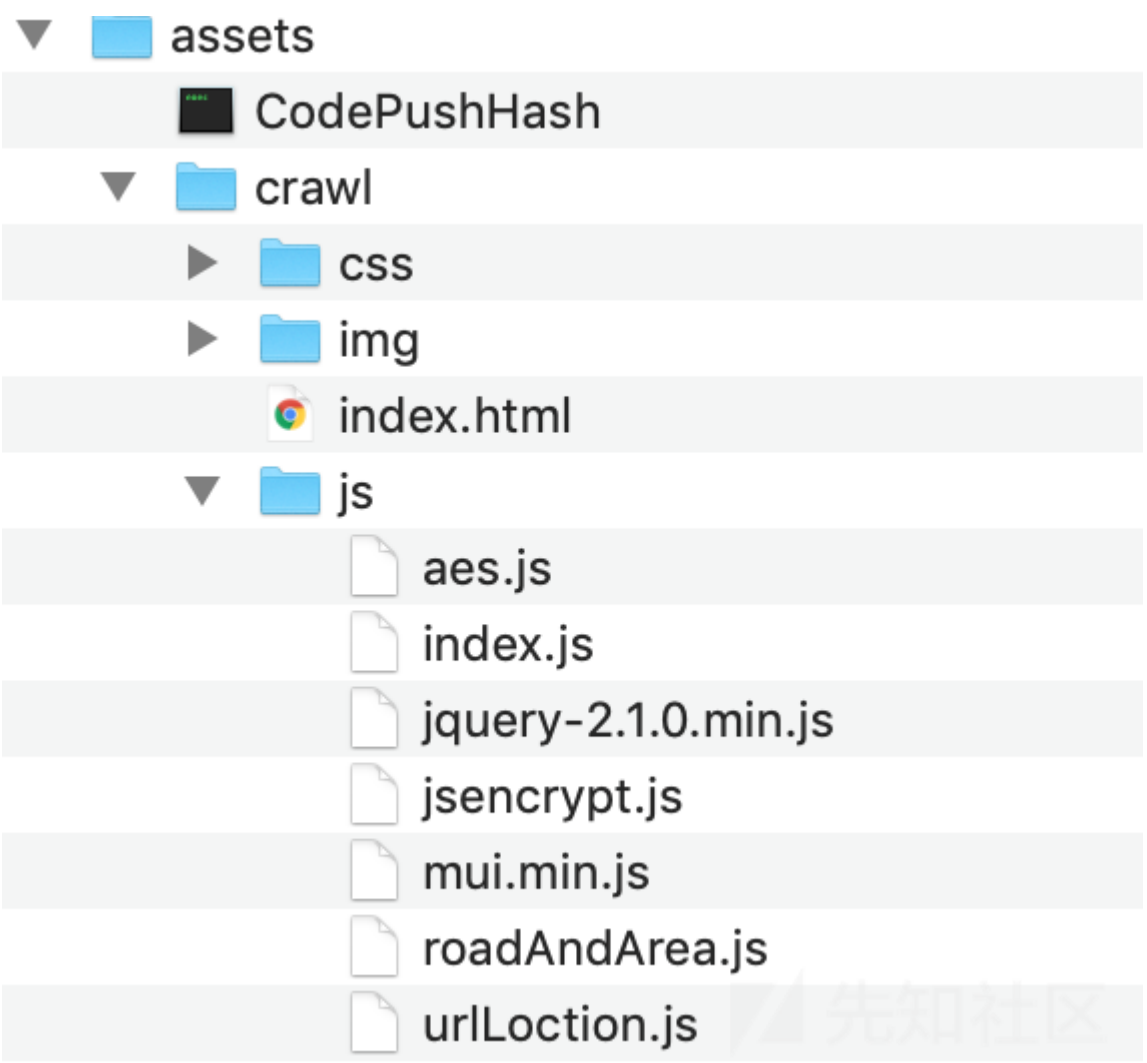
但是奇怪的是再次点击登陆居然没有断下来！Method Profiling抓取到的讲道理应该一定有被调用才对。

到这里大概可以确定加解密是在前端而不是android里面实现了，但是不死心我还是Hook来验证一下，直接hook Javax的Cipher函数，如果有调用的话日志应该就会打印Cipher加密的方式以及密钥key，然而无果。

前端JavaScript分析

搜索

如果真的确定加解密函数在Javascript代码里面的话，事情就变得简单起来了（其实也没有很简单）。对javascript不是特别熟悉，并且也没有办法动态调试Javascript，没有



还好这里JavaScript代码不是特别多，结合grep和find全局搜索一下文件名和函数定位到了`utils.js`

```
.//assets/XXX/js/utils.js:function AESencrypt(text,key) {
.//assets/XXX/js/utils.js:  var value =AESencrypt(body,key);
.//assets/XXXCompany/js/utils.js:function AESencrypt(text,key) {
.//assets/XXXCompany/js/utils.js:   var value =AESencrypt(body,key);
.//assets/XXX/js/urlLoction.js:function AESencrypt(text,key) {
```

并找到了加解密函数以及RSA publicKey和secretKey同时硬编码在js文件里

```
function AESencrypt(text,key) {
   var result = CryptoJS.AES.encrypt(CryptoJS.enc.Utf8.parse(text),CryptoJS.enc.Utf8.parse(key),{
       mode:CryptoJS.mode.ECB,
       padding:CryptoJS.pad.Pkcs7,
   });
   return result.toString();
};
function AESdecrypt(text,key) {
   var result = CryptoJS.AES.decrypt(text,CryptoJS.enc.Utf8.parse(key),{
       mode:CryptoJS.mode.ECB,
       padding:CryptoJS.pad.Pkcs7,
   });
   return result.toString(CryptoJS.enc.Utf8)
};
function RSAencrypt(text) {
   var rsaEn = new JSEncrypt();
   rsaEn.setPrivateKey(rsaPubKey);
   var enc = rsaEn.encrypt(text);
   return enc;
};
function RSAdecrypt(text) {
   var rsaDn = new JSEncrypt();
   rsaDn.setPublicKey(rsaprivateKey2);
```

```
    var dec = rsaDn.decrypt(text);
    return dec;
}
```

加解密逻辑

```
var timeS = new Date().getTime();
var key = '';
key += timeS+'123sadsof313r24rsd';
if (key.length > 16) {
    key = key.substring(0,16);
}
var rsaKey = RSAencrypt(key);
var body = dataConter;
var value =AESencrypt(body,key);
var newBody = JSON.stringify({
    data:value,
    key:rsaKey,
});
```

```
success: function (data, status) {
    if (status == 'success') {
        var aaPwd = RSAdecrypt(data.key);
        var aaData =AESdecrypt(data.data,aaPwd);
        var newRes = JSON.parse(aaData);
    // successCallBack(data);
    // console.log(newRes)
    if(newRes.code==401){
        window.postMessage(['login401']);
    } else{
        successCallBack(newRes);
    }
}},
```

细节分析

1. rsaEn.setPrivateKey(rsaPubKey) RSAEncode这里用公钥作为私钥encode了，一开始没注意还真就用了私钥尝试去解。
2. 这里逻辑也很明了，发包的数据用13位的timestamp+"123"作为密钥key进行AES ECB加密，并且将AES的key进行RSA加密发送到服务器。返回包中，先RSA解密key字段获取AES加密的key，再对data字段进行AES解密。
3. 本地js找加解密的时候也猜想过一种情况，客户端有没有可能先从服务端获取加解密的js文件（这样本地也找不到加解密函数）。为了排除这种想法，要仔细看burp抓取的
4. 又能愉快的对移动app进行渗透测试了。

上一篇：记一次phpstudy后门引发的渗透测试 下一篇：windows样本高级静态分析之识...

1. 1 条回复

jett张建涛 2019-10-15 11:07:44

大佬最近看机会吗？国内知名信息安全公司，移动安全工程师，安全加固和逆向工程都有HC,薪资30K-35k,有意向接触的haul，15376794830手机微信同号，^_^

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板