如何处理移动应用请求的安全性控制

> “移动安全风险大部分在web请求，web端的安全性直接系于源码是否安全。”

前言

对于移动应用的安全性的考究，从App应用在市场上大批量涌现，再到渠道市场出现游戏破解包、非官方二次打包等等的状况后，移动应用的安全性就开始备受关注。一直以来被各家公司所探索，有的直接了当针对源码进行加固，有的则开发起移动应用自动审计系统，试图希望在上线之前做一些风险截断。有的公司像爱加密、梆梆、娜迦等等看中商机，采用比较通用便捷的方式和大企业合作的方式进行加固和审计；有的大企业如阿里、腾讯有这方面的技术资源，则自己就能开发出一套既能自用也能商用的加固、审计系统。

然而实际的效果，毕竟是通用的做法，加固面临的是不断升级的技术对垒；而对外的移动审计系统，扫描的都是本地App的一些安全性，这些审计结果在各大SRC平台的评级都不如web端的级别高，因为本地应用的安全性，最多也只能影响本地数据，而web接口一般都是面临着一个公司的某项核心业务数据。就数据重要性来说，似乎移动安全没那么重要了，但在数据流的传输过程中，本地应用的安全性扮演者十分重要的角色。接下来我将从以下几个方面描述一些实际开发过程中，的一些攻击防御方法：

- web接口的安全性转移。
- 如何信任运行时的系统设备（设备指纹、API Hook）。
- 如何防止外部调用（加解密、lib库防止外部调用）。
- 如何防止调用流程泄漏（堆栈打印、流程混淆）。

一、 web接口的安全性转移

移动应用的请求接口，除了微信、QQ这种IM社交应用会自定义协议，大部分都是遵从HTTP这种应用层协议。对于web渗透者来说，他们对于App的测试，一般都是通过抓包的方式获取请求列表，然后再进行注入、XSS、CSRF、逻辑漏洞等等的渗透攻击。所以防止应用请求被抓包、修改，成为了移动应用的第一道防线。安全的接口定义一般有两种方式：

(一) 关键敏感字段隐藏

这里以Http GET协议为例子，其他请求可以类推：

http://www.eleme.com/request.doipAddress=192.168.0.2&phoneNum=18500010001&userID=zhangsan&pwd=4e4d6c332b6fe62a63afe56171fd3725&sign=0f0a62bf271f450f403f940d4644b

- 这条协议中对于像‘phoneNum’这种敏感字段没有加密，在web越权的测试中，常常通过修改已有权限的账号来查看其他账户的敏感信息。
- ‘pwd’密码字段在传输过程中不应该是明文传输，一方面防止被截获、另一方面防止数据库密码原文存储，一旦泄漏危害很严重。
- ‘sign’字段的作用主要是为了对整个数据(除了sign自身)进行校验，服务器通过sign来校验请求是否被修改，sign算法的安全性依托于源码保护。

这种方式设计的协议请求会暴露我们的字段，像‘usr’、‘pwd’这种字段名，很容易知道是什么意思，所以如果再对这些字段进行混淆处理，那么在协议设计上来说就是一条比较安全的协议了。

(二) 字段全部隐藏

http://www.eleme.com/request.do?data=ABCVAGSDVJ…..AHJDKHASKD

这条协议中作为字段的data包含了请求的所有字段和数据（data的数据格式解密后如第一种方式）并且是加密状态；data算法的安全性依托于源码保护。这种方式，如果对于服务器解析来说，如果系统有多个解析层次，那么这种字段加密的方式，则要求第一层来对data进行解密操作，才能拿到ipAddress、phoneNum、userID等字段。对于并发数据很高的服务系统，这种方式可能会加大负载，但是看起来会更加安全。

(三) 协议、加解密算法的升级

——“哈希算法从来也不是为加密而生的”。

Http传输一直被诟病，这里描述一种场景：app使用了webView这种组件后，应用请求被官方服务器回应后，数据包再经过运营商时被强家了广告弹窗。有的应用渠道商店的Http请求下载过程中，用户需要的是A应用，结果下载下来的却是B应用；使用Https传输则能很好的解决这个问题，传输的数据对于网链上的其他节点来说都是透明的。

通常一个终端用户对于APP的Activity的访问是有层次逻辑的，比如登陆->首页->订单->结算 这种和服务器的会话先后顺序是固定的，如果协议被破解，任何不按照这哥逻辑顺序发送的包都是非法访问的。这里强调的并不是请求到达服务器的时间顺序，因为网络传输的不可避免的会受影响而产生延迟，所以到达时间无法作为检测规则。一般的简单的解决方案是设置类似sessionID的方式进行解决，对于系统性的管理还是推荐HTTP API网关来进行过滤处理。

编码算法base64常常被开发者当作加密算法，Base64编码算法是一种用64个字符（ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/），一般开发者的做法一般都是把编码map表进行替换，然后常规的android.util.Base64类就无法还原明文。但是如同哈希算法一样，因为强度问题，它们生来就不是为加密而生的，所以应该避免使用编码、哈希算法来对数据加密，不要自己设计加密算法和协议，使用业界标准的算法。

二、 如何防止外部调用

通过过web接口我们把关键的敏感字段、hash算法都用算法的方式转移到源码中了，接下来看似就是要保证我们的源码不被轻易分析就好了。但事实上在破解者看来，分析源码可能是他们迫不得已作出的最后选择。聪明的渗透者并不会去分析还原你的算法。

比如，在我们的应用的classes.dex文件中，有一个加密函数声明为：

```
  private String Cryptor.encode(byte[] data);
```

那么破解者可以通过调用这个函数来直接获得这个加密算法：

- dex2jar 工具，讲dex文件转换为jar包。
- 通过ClassLoader加载jar包后，找到Cryptor类。
- 在Cryptor类中找到对应的函数 private String encode(byte[] data)。
- 通过反射的方式设置为encode为public访问权限。
- 传入修改过的data数据，反射调用encode方法。

如果开发人员安全意识很好，这个加密函数被放置在了android native层。那么同样可以加载so文件到我们自己的进程空间后，通过文件偏移来找到该加密函数来进行调用，或者通过IDA进行调试，在调试的过程中，通过内存替换从而达到修改数据。

解决方案

「对于函数的调用要和应用信息进行耦合」比如我们添加的耦合参数为:

A. private String Cryptor.encode(Context context,byte[] data);

我们在encode函数中通过context获取packageName、signature，通过对比包名、签名来区分，当前函数是否运行在非法的进程空间。

B. private String Cryptor.encode(Context context,CryptorContext data);

在A的基础上，通过对参数data进行封装，然后我们可以提高data对象的复杂性，而不是简单的明文数据byte[]类型，同样可以让非法调用的时间复杂度提高。

C. java层的破解难度很低，即便是对dex畸形处理后再动态加载，所以还是建议把核心算法放置在native层。

三、 如何信任运行时的系统设备

前面提倡到大家使用Https来进行服务器通讯，但是Https的通讯同样依赖于源码的完整性。因为不管是作为客户端还是服务器，都无法安全的校验彼此，因为我们的应用运行在应用层，在这层，App只能被运行访问被android framework封装的API、仅App应用用户权限的文件系统。而破解者可以使用Root权限，去访问App的进程沙盒，任意监听App访问的API。这就是一个应用级和一个系统级的对抗，只要我们运行在这个系统上，攻击者永远有权限去监听我们。所以这一部分，我提出的观点就是，不要再信任我们的系统了。不管是那些系统API的返回值，还是我们获取的设备信息。接下来展示两种截获API调用请求的破解案例

(一) 使用**jdb**调试进行**API**的**hook**监听

- 首先我们打开系统调试属性，这里有两种可选方式：
  - 修改ROOM包，将文件系统根目录的文件ro.debuggable属性设置为1。
  - 通过xposed插件BDopener.apk来开启。
- 使用am命令以调试的方式打开app。

```
adb shell "amstart -D <packageName>/<launcherActivityName>"
```

- 查看应用的pid。

```
adb shell "ps |grep <packageName>"
```

- 重定向设备端口到应用的dvm的调试端口。

```
adb forward tcp:8700jdwp:8202
```

- 使用jdb进行附加。

```
jdb -connectcom.sun.jdi.SocketAttach:hostname=127.0.0.1,port=8700
```

- 对我们感兴趣的函数下断点，这里以函数‘private String Cryptor.encode(Contextcontext, byte[] data);’举例：

通常先找到函数所在的类Cryptor所有的函数：

```
jdb> classes
jdb> methodscom.AAA.BBB.Cryptor
```

然后再对目标函数下断点：

```
jdb> stop incom.AAA.BBB.Cryptor.encode(android.content.Context, java.lang.byte[])
jdb> run
```

这些操作设置好后，只要该函数被调用，就可以触发断点，从而打印出堆栈。jdb还能把传进来的参数进行dump，堆栈对于攻击者追踪web请求中的字段又很有帮助。

(二) 使用**Xposed**监听**API**请求

Xposed框架是一款可以在不修改APK的情况下影响程序运行(修改系统)的框架服务，基于它可以制作出许多功能强大的模块，且在功能不冲突的情况下同时运作。以下例子：

```
public class HookUtil implements IXposedHookLoadPackage{
  @Override
```

```
    public void handleLoadPackage(LoadPackageParamlpparam) throws Throwable {

        //** 标记目标app**包名

          if (!lpparam.packageName.equals("com.example.logintest"))

             return;

          XposedBridge.log("Loaded app: " + lpparam.packageName);

          //Hook MainActivity中的isCorrectInfo(String,String)**方法

          //findAndHookMethod(hook方法的类名，classLoader，hook方法名，hook方法参数...，XC_MethodHook)

          XposedHelpers.findAndHookMethod("com.example.logintest.MainActivity", lpparam.classLoader,
"isCorrectInfo", String.class,

                  String.class, new XC_MethodHook() {

                      @Override

                      protected void beforeHookedMethod(MethodHookParamparam) throws Throwable {

                          XposedBridge.log("开始hook");

                          XposedBridge.log("参数1 = " + param.args[0]);

                          XposedBridge.log("参数2 = " + param.args[1]);

                       }

                      @Override

                      protected void afterHookedMethod(MethodHookParamparam) throws Throwable {

                          XposedBridge.log("结束hook");

                          XposedBridge.log("参数1 = " + param.args[0]);

                          XposedBridge.log("参数2 = " + param.args[1]);

                       }

                   });

    }
```


xposed利用在Https上，风险系数立马被放大。https协议验证服务器身份的方式通常有三种，一是根据浏览器或者说操作系统（Android）自带的证书链；二是使用自签名证书；三是自签名证书加上SSL Pinning特性。第一种需要到知名证书机构购买证书，需要一定预算。第二种多见于内网使用。第三种在是安全性最高的，但是需要浏览器插件或客户端使用了SSL Pinning特性。Android应用程序在使用https协议时也使用类似的3种方式验证服务器身份，分别是系统证书库、自带证书库、自带证书库 + SSL Pinning特性。

SSLPinning，即SSL证书绑定，是验证服务器身份的一种方式，是在https协议建立通信时增加的代码逻辑，它通过自己的方式验证服务器身份，然后决定通信是否继续下去。它唯一指定了服务器的身份，所以安全性较高。我们可以使用xposed框架来绕过SSL Pinning。

(三) 设备指纹的可靠性

Android对于系统信息的获取，都已经通过暴露API的方式来取代了直接对底层的信息读取，API获取到的设备信息没有兼容性的问题，但是很容易通过Xposed这种简便的框架进行修改。所以如何保证设备指纹的正确性呢。

系统允许我们读取/proc/cpuinfo、/system/build.prop等类似的状态和配置文件来获取系统信息，防止被hook的思路就是绕过framework 的API读取这些文件。具体就要参考系统底层读的对应的文件。另外5.0版本以上对于这些硬件信息对应的文件的权限进行了提升，所以也并不是所有的信息都能通过读文件的方式获取，这里仅仅提供一个思路。其实我们可以把设备指纹的范畴再进行缩小，搜集应用运行时的信息，尤其是关联上用户信息、操作这些数据，其对于后台大数据风控时的功效和设备指纹基本相同。

四、 如何防止调用流程泄漏

通过上小节，我们看到通过**jdb**调试和**Xposed**的**hook**的方式均可以得到传入参数以及对应的调用的堆栈。这就是站在系统权限之上可以看到的东西。我主要通过在App的java层和native层给出一些解决方法。

- 加强应用自身的AndroidManifest.xml的配置，export、debuggable、权限的签名、组件拒绝服务、provider越权访问、webView的安全配置等基础配置要做好检查，不要出现风险项。虽然这些风险项不一定造成大的影响，但是没有按要求做好，就有会给攻击者的分析过程造成一定的便利。
- 保护好核心的API防止外部调用。
- API可以通过堆栈来检查自己的调用是否正常
- 加解密函数不要在java层使用系统自带的，因该在native层使用标准的加密算法来实现，自定义加密向量表等。
- 对native层可以参考LLVM-Obfuscator进行字符串混淆、流程混淆。

参考

1. Android应用安全开发之浅谈加密算法的坑
2. HTTP API网关选择之一Kong介绍
3. DOpener——开启APK调试与备份选项的Xposed模块
4. 利用xposed绕过安卓SSL证书的强校验
5. Android.Hook框架xposed篇(Http流量监控)