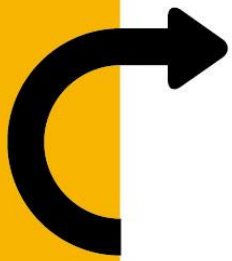
移动端安全测试基础

（PDF测试岗位课程）

（适用于L1，L2）



第一部分 移动端应用安全测试概念

第二部分 Android应用安全测试技术

第三部分 IOS应用安全测试技术

移动端应用安全测试概念

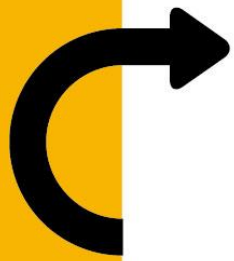
移动app大多通过web api服务的方式跟服务端交互，这种模式与PC端的BS模式类似。移动app以web服务的方式跟服务端交互，服务器端也是一个展示信息的网站，常见的web漏洞在这也存在,比如说SQL注入、文件上传、XSS等等，因此，所有**WEB安全测试相关的内容几乎都适用于移动端应用安全**领域，与此同时，移动端安全领域还有自身的一些特点，移动app运行的环境是移动端，具有一定的封闭性和特殊性，特别地，**移动端app的本地安全**，比如远控、应用破解、信息窃取等等，这些就是本课程重点讨论的内容。

移动端的应用安全测试主要集中在两个系统上，一个是Android系统、一个是IOS系统，在测试过程中，为了逃逸系统的沙盒限制，需要分别对Android系统进行root、IOS系统进行越狱操作，以便获取系统权限用以执行安全测试。

移动端应用安全测试概念

随着智能手机和iPad等移动终端设备的普及，而智能终端的普及不仅推动了移动互联网的发展，也带来了移动应用的安全巨大隐患，归纳起来，主要集中在以下几个方面：

围绕这些隐患，我们需要作出相应的安全测试方案，以便发现此类安全风险，修正并加固app。



第一部分 移动端应用安全测试概念

第二部分 Android应用安全测试技术

第三部分 IOS应用安全测试技术

Android应用安全测试技术

Android应用安全测试测试涵盖多个方面，业内也有很多专业的测试工具和测试平台，测试的方法主要是通过对apk的反编译、源码静态分析以及特征匹配等，一般来说，可以大致分为以下三方面：

- 应用保护与源码安全：
  主要是测试apk是否有签名、完整性的校验，以及遭反编译后源码是否存在泄漏、dex和so文件的隐藏加固，以及硬编码等风险的检测

- 应用组件安全：
  主要是对app本身的组件安全、WebView安全的测试，发现软件在结构上的设计不合理的地方，以及暴露出可能存在的组件漏洞，输入监听、界面劫持和被调试等风险

- 应用数据安全：
  针对应用的输入输出，进行窃取和非法访问的测试，检测协议上会话是否安全加密、证书验证以 及能否被重放攻击，输出信息是否能为渗透破解提供有效数据等方面的检测

应用保护与源码安全

1. Dex保护

原因：

Dex为Android应用的核心，当程序未对dex进行隐藏加密等保护措施时，容易被反编译，暴露程序重要信息

危害：

Dex被反编译将试apk面临被植入广告、恶意代码、病毒等风险，可以获取smali代码，包括应用分析入口Launcher Activity的信息等。

修复建议：

对dex文件进行加密保护，或通过算法进行加固，防止被dex2jar等工具反编译。

应用保护与源码安全

2. so保护

原因：

Android的so文件通过C/C++代码来实现，相对于Java代码来说其反编译难度要大很多，但依然存在反编译风险。

危害：

so被逆向，应用的关键性代码和算法都将会暴露。

修复建议：

对so文件进行加密保护，或通过算法进行加固，防止被工具反编译。

应用保护与源码安全

3. 资源文件保护

原因：

资源文件主要指静态资源文件，例如图片、字符串等，相对dex、so文件重要性要低，但也正因为此，通常apk对资源文件的保护几乎没有。

危害：

一方面，反编译apk获得源码，通过资源文件或者关键字符串的ID定位到关键代码位置，为逆向破解应用程序提供方便；

另一方面，如果资源没有加密，攻击者可以通过反编译apk，获取apk资源文件进行替换，或者盗取资源文件，给应用开发者造成损失。

修复建议：

建议对资源文件做加密处理，防止资源被替换、盗用，以及从资源文件定位到应用关键代码处。

应用保护与源码安全

4. 异常处理

原因：

在开发时偶尔会由于疏忽导致有些异常没有进行处理。如果不提示详细信息又会给用户报告异常带来麻烦，不利于开发人员及时发现并处理异常。但是如果将异常详细信息不经处理直接提示给用户则会带来安全隐患。

危害：

可能导致异常处理的详细信息被窃取导致进一步的靶向入侵。

修复建议：

建议建立apk统一的、规范性的异常处理机制避免导致安全风险。

应用保护与源码安全

5. 权限管理

原因：

冗余权限可导致串谋攻击，串权限攻击的核心思想是程序A有某个特定的执行权限，程序B没有这个权限。但是B可以利用A的权限来执行需要A权限才能完成的功能。

危害：

冗余权限可以被一些恶意程序所利用来进行一些恶意行为，给用户造成权限劫持等损失。

修复建议：

建议去掉多余权限。

应用保护与源码安全

6. 动态调试

原因：

没有在app应用中加入反调试代码，应用如果没有采用反调试技术，攻击者可以很方便地对该应用进行调试，从而为破解、逆向该应用提供了便利。

危害：

动态调试可以获取函数运行时各个变量的取值，程序运行逻辑，加密的数据信息，从而为破解应用提供便利。

修复建议：

建议加入反调试代码。

应用保护与源码安全

7. 代码混淆

原因：

对代码进行混淆，能够提高黑客阅读和理解代码的门槛，在一定程度上增加了黑客破解的难度。

危害：

当代码在编译前未做任何混淆措施，app被反编译后即完全获取源码。

修复建议：

建议在代码中进行必要的代码混淆。

应用组件安全

1. Activity安全

原因：

导出的组件可以被第三方app任意调用，导致敏感信息泄露或者恶意攻击者精心构造攻击载荷达到攻击的目的。

危害：

可能导致Activity被权限攻击，被劫持；亦可导致Activity被绕过本地认证，造成拒绝服务，导致程序崩溃等。

修复建议：

建议如果组件不需要与其它app共享数据或交互，请在AndroidManifest.xml配置文件中将该组件设置为exported = “False”。如果组件需要与其它app共享数据或交互，请对组件进行权限控制和参数校验。

应用组件安全

2. Broadcast Receiver安全

原因：

应用广播接收器默认设置exported='true'，导致应用可能接收到第三方恶意应用伪造的广播，利用这一漏洞，攻击者可以在用户手机通知栏上推送任意消息，通过配合其它漏洞盗取本地隐私文件和执行任意代码。

危害：

可造成信息泄露，拒绝服务攻击等。

修复建议：

建议如果组件不需要与其它app共享数据或交互，请在AndroidManifest.xml配置文件中将该组件设置为exported = “False”。如果组件需要与其它app共享数据或交互，请对组件进行权限控制和参数校验。

应用组件安全

3. Service安全

原因：

设置exported属性为true，导致service组件暴露，第三方程序在没有声明任何权限的情况下，通过发送恶意intent，可以在没有任何提示情况下启动该服务，进行该服务所作的操作，对系统安全性造成极大威胁。

危害：

可造成包括：权限提升，拒绝服务攻击。没有声明任何权限的应用即可在没有任何提示的情况下启动该服务，完成该服务所作操作，对系统安全性产生极大影响。

修复建议：

建议如果组件不需要与其它app共享数据或交互，请在AndroidManifest.xml配置文件中将该组件设置为exported = “False”。如果组件需要与其它app共享数据或交互，请对组件进行权限控制和参数校验。

应用组件安全

4. Content Provider安全

原因：

设置exported属性为true，导致Content Provider组件暴露，可能导致provider提供的数据服务被他人利用。

危害：

会产生sql注入、文件遍历等漏洞，导致用户数据泄露等危害。

修复建议：

建议如果组件不需要与其它app共享数据或交互，请在AndroidManifest.xml配置文件中将该组件设置为exported = “False” 。如果组件需要与其它app共享数据或交互，请对组件进行权限控制和参数校验。

应用组件安全

5. Intent安全

原因：

Intent scheme URLs(意图协议URL)，可以通过解析特定格式的URL直接向系统发送意图，导致自身的未导出的组件可被调用,隐私信息泄露。

危害：

Intent能被第三方劫持，如果含有隐私信息可能导致内部隐私数据泄露。

修复建议：

建议把Intent的隐式调用改成显式调用，或做权限和参数限制。

应用组件安全

6. Webview安全

6.1 Webview远程代码执行

原因：

Webview是Android用于浏览网页的组件，其包含的接口函数addJavascriptInterface可以将Java类或方法导出以供JavaScript调用，实现网页JS与本地JAVA的交互。由于系统没有限制已注册JAVA类的方法调用，因此未注册的其它任何JAVA类也可以被反射机制调用。

危害：

可能导致被篡改的URL中存在的恶意代码被执行，用户手机被安装木马程序，发送扣费短信，通信录或者短信被窃取，甚至手机被远程控制。

修复建议：

取消使用addJavascriptInterface接口，以其他Java与JavaScript互通方案代替；若必须使用，则应对访问的url进行过滤限制或对html页面进行完整性校验，同时显示移除对指定的javascript接口的调用。

应用组件安全

6.2 Webview绕过证书校验

原因：

客户端的Webview组件访问使用HTTPS协议加密的url时，如果服务器证书校验错误，客户端应该拒绝继续加载页面。但如果重载WebView的onReceivedSslError()函数并在其中执行handler.proceed()，客户端可以绕过证书校验错误继续访问此非法URL。

危害：

可能导致“中间人攻击”，攻击者冒充服务器与银行客户端进行交互，同时冒充银行客户端与银行服务器进行交互，在充当中间人转发信息的时候，窃取手机号，账号，密码等敏感信息。

修复建议：

取消在Webveiw组件中对onReceivedSslError()函数的重载。

应用组件安全

6.2 Webview明文存储密码

原因：

Android的Webview组件中默认打开了提示用户是否保存密码的功能，如果用户选择保存，用户名和密码将被明文存储到该应用目录databases/webview.db中。

危害：

本地明文存储的用户名和密码，不仅会被该应用随意浏览，其他恶意程序也可能通过提权或者root的方式访问该应用的webview数据库，从而窃取用户登录过的用户名信息以及密码 。

修复建议：

通过设置WebView.getSettings().setSavePassword(false)关闭webview组件的保存密码功能。

应用数据安全

1. 测试数据移除安全

原因：

如果在AndroidManifest.xml配置文件中设置了application属性为debuggable=“true”，则应用可以被任意调试。如果设置application属性为allowBackup=“true”，则应用在系统没有root的情况下其私有数据也可以通过备份方式进行任意读取和修改。

危害：

应用可以被任意调试，这就为攻击者调试和破解程序提供了极大方便，亦可造成隐私泄露和信息被恶意篡改。

修复建议：

AndroidManifest.xml配置文件中设置了application属性为debuggable=“false”，application属性为allowBackup=“false”，同时移除所有测试时使用的代码和数据 。

应用数据安全

2. 日志信息安全

原因：

调试日志函数可能输出重要的日志文件，其中包含的信息可能导致客户端用户信息泄露，暴露客户端代码逻辑等，为发起攻击提供便利，例如：Activity的组件名，是Activity劫持需要的信息；通信交互的日志，会成为发动服务器攻击的依据；跟踪的变量值，可能泄露一些敏感数据。

危害：

可以跟踪调试信息或者错误异常信息，可以定位应用运行的流程或者关键代码，从而降低黑客破解的难度，还可以窃取敏感信息。

修复建议：

建议关闭调试日志函数调用，或者确保日志的输出使用了正确的级别，涉及敏感数据的日志信息在发布版本中被关闭。

应用数据安全

3．数据访问控制安全

原因：

当应用使用openFileOutput(String name,int mode)方法创建内部文件时，将第二个参数设置为Context.MODE_WORLD_READABLE或Context.MODE_WORLD_WRITEABLE，就会让这个文件变为全局可读或全局可写的。使用getSharedPreferences读取配置信息时一般使用MODE_PRIVATE参数，如果使用WORLD_READABLE和MODE_WORLD_WRITABLE将会导致配置文件被第三方app读写。

危害：

可能导致越权访问文件与数据，或者导致敏感数据泄露。

修复建议：

建议禁止全局文件可读写。 如果开发者要跨应用共享数据，一种较好的方法是实现一个Content Provider组件，提供数据的读写接口，并为读写操作分别设置一个自定义权限。

第一部分 移动端应用安全测试概念

第二部分 Android应用安全测试技术

第三部分 IOS应用安全测试技术

IOS应用安全测试技术

IOS的安全性相比较Android来说要高，原因主要有三方面：IOS是一种闭源的操作系统相对于Android的开源性天生就有安全优势；IOS的沙盒机制更强大，越狱的难度比Android的root难度要高，且IOS版本更新的速度较快；IOS对app的上传发布等有审核机制，同时app的安装和下载均是在IOS的环境中，形成了一个相对比较安全的闭环。和Android系统类似，同样除去适用的WEB安全测试相关内容外，主要有以下几两个方面：

- 应用保护与源码安全：
  主要是测试ipa是否有签名、完整性的校验，以及反编译的安全风险

- 应用数据安全：
  针对应用的输入输出，进行窃取和非法访问的测试，检测协议上会话是否安全加密、证书验证以及能否被重放攻击，输出信息是否能为渗透破解提供有效数据等方面的检测

应用保护与源码安全

1. 反编译保护

原因：

ipa文件一般只能在苹果自带的应用商店下载安装，被认为很安全，普遍未对其做任何防护措施，因此极易被反编译

危害：

ipa被反编译可能导致利用FLEX补丁软件通过派遣返回值来对应用进行patch破解，或者通过使用ida等反汇编工具对ipa进行逆向汇编代码，导致核心代码逻辑泄漏与被修改，影响应用安全。

修复建议：

对程序中出现的URL进行编码加密，防止URL被静态分析，对应用程序的方法名和方法体进行混淆，保证源码被逆向后无法解析代码，对应用程序逻辑结构进行打乱混排，保证源码可读性降到最低。

应用保护与源码安全

2. 资源文件保护

原因：

应用产生的plist、db、xml、log等后缀的文件如果未进行加密保护并存储敏感信息，可能会导致用户敏感信息丢失。

危害：

例如手机支付APP将用户名和密码居然以明文的方式存储在了unionpay.db的文件中；也有一些APP将手机号码和密码存储在了以plist为后缀的文件中等等直接导致用户的敏感信息泄露。

修复建议：

数据请勿本地存储，用后请删或添加访问限制；若必须存储在本地，须对数据进行加密存储。

应用数据安全

1. 证书有效性安全

原因：

若应用与服务端的通讯未对客户端、服务端的证书进行强校验，可能会导致SSL通信被非法利用

危害：

可能会导致用户用户受到中间人攻击，或用户访问钓鱼网站后，泄露密码等敏感信息。

修复建议：

检查服务端证书是否是CA签发，客户端程序和服务器端的SSL通信是否严格检查服务器端证书的有效性。同时客户端程序是否严格核对服务器端证书信息。

应用数据安全

2. 密钥链数据安全

原因：

iOS系统及第三方应用都会使用Keychain来作为数据持久化存储媒介，或者应用间数据共享的渠道，所以Keychain数据库是hacker们最关注的数据源头之一。使用Snoop-it工具结合密钥链查看工具Keychaindumper可以查看到所有密钥链中的数据

危害：

可能会导致应用的密码明文泄露。

修复建议：

规范Keychain的处理机制，严谨密码明文存储。

应用数据安全

3．动态调试安全

原因：

当应用未加入反调试的代码防御时，可通过搭建动态调试平台，例如使用GNU Debugger、openSSH、adv-cmds等工具对应用进行动态调试

危害：

动态调试可以获取函数运行时各个变量的取值，程序运行逻辑，加密的数据信息，从而为破解应用提供便利。

修复建议：

在实际应用中可以在得知自己的程序被调试器附加后自动退出，当然这也是可以通过下断点的方式绕过，但是可以通过多处调用该方法来增加攻击的难度。同时上面的代码采用声明内联函数的方法，使编译器将函数功能插到每处代码被调用的地方，而不至于某个特定的功能函数被劫持。

应用数据安全

4. 动态注入安全

原因：

hook技术起源于Windows操作系统API hook，即改变API执行结果的寄生代码执行技术，当应用对关键的业务功能的方法实现未做必要的防注入措施时可能会导致hook，例如针对IOS系统的动态测试工具Snoop-it、基于MobileSubstrate的脚本工具cycript，可通过脚本动态注入应用并修改应用

危害：

利用hook技术，可以在iOS系统上实现各种hook功能，比如微信自动抢红包，自动聊天机器人，游戏外挂等。

修复建议：

当hook的时候会在app中注入dylib，基于这个明显的特征，建议在关键业务和功能的方法上需要防止和阻止dylib注入，例如在Other Linker Flags中添加-Wl,-sectcreate,__RESTRICT,__restrict,/dev/null等。

Thanks!