

信息泄漏安全场景和危害

Android系统的安全性一直为人诟病，恶意软件侵犯用户隐私的例子层出不穷，即使在Google Play这个权威的应用商店中排名很高的应用，也存在盗取用户密钥隐私的问题。安卓进一步提高审核力度迫在眉睫。并且APP在设计开发时需要考虑的就是在手机被root后，私有数据被暴露的情况下，仍然能保证数据安全，不会泄露用户重要的数据及隐私信息，而对于手机厂商修复系统漏洞速度慢的问题，开发者不能依赖系统更新，更多时候需要自己想办法避免漏洞对用户造成损失。本PPT着力从Android平台的应用层的角度去讲述android应用在隐私方面存在安全隐患的几个方面。

敏感信息可分为产品敏感信息和用户敏感信息两类，对**产品敏感信息**可以这样界定：

泄露后直接对企业安全造成重大损失或有助于帮助攻击者获取企业内部信息，并可能帮助攻击者尝试

更多的攻击路径的信息。

以下这些信息都属于产品敏感信息：登录密码、后台登录及数据库地址、服务器部署的绝对路径、内

部IP、地址分配规则、网络拓扑、页面注释信息（开发者姓名或工号、程序源代码）。

而**用户敏感信息**有两个界定原则：

用户隐私保护主要考虑直接通过该数据或者结合该数据与其它的信息，可以识别出自然人的信息。

一旦发生数据泄露事件，可以被恶意人员利用并获取不当利润。

敏感信息硬编码

而这些敏感信息泄露的原因大多数情况下是因为信息未加密或储存位置不当造成的：

代码中明文使用敏感信息，比如：服务器地址、数据库信息等

数据库中明文保存敏感信息，比如：账号、密码、银行卡等

SD卡中保存敏感信息或隐私数据，比如：聊天记录、通讯录等

日志中打印敏感信息：比如：账号、密码

通信过程中明文传输敏感信息

以上这些做法都会使APP中的敏感信息暴露在黑客的眼皮底下，只要黑客认为该信息有价值，他就会

轻而易举的获取这些敏感信息，最直接的损失可能就是用户的账号被盗、网银被盗刷等。

而黑客如果知道了登录密码、后台登录及数据库地址、服务器部署的绝对路径、内部IP、地址分配规

则、网络拓扑等产品敏感信息，会极大节省攻击时间，获取该应用的绝对控制权，其损失不是能够用金钱

来衡量的。另外，这些敏感信息泄露，也会招来其他试探性的黑客去尝试攻击，导致服务器处于危险境地。

□LogCat输出敏感信息

□敏感数据明文存储于Sdcard

❑数据库敏感数据明文存储

❑Shared Preference全局可读写

❑敏感信息硬编码

❑HTTP敏感信息明文传输

LogCat敏感信息输出

开发者在调试排查问题的过程中，会输出大量的log信息，这些log里会包含一些用户的敏感信息。

比如安全通讯录会输出用户的联系人、安全键盘会输出用户敲击的按键等，这些都是曾经出现的已知安

全案例。在安卓系统4.1版本之前是允许其他应用申请如下读写所有应用日志的权限的，这样就会导致存

在恶意应用窃取用户敏感信息的安全隐患。

```
<uses-permission android:name="android.permission.READ_LOGS" />
```

Ø 应用层Log敏感信息输出

Ø 应用层System.out.println敏感信息输出

Ø 系统bug异常导致Log输出

Ø Native层敏感Log输出

某金融 App 日志中明文打印用户名、密码，如果用户手机被安装了恶意软件，恶意软件可以获取到这些信息，导致账号被盗、被转账等。；

某金融 App 日志中明文打印用户名、密码，如果用户手机被安装了恶意软件，恶意软件可以获取到这些信息，导致账号被盗、被转账等。

Application	Type	Text
com.xxx.xxx	mobile	Request_Certify_5--->getCertifyData --- user: 161020 password: 161020
com.xxx.xxx	mobile	logindata.phone: 161020, logindata.password: 161020
com.xxx.xxx	mobile	Request_Certify_7---> user: 161020 password: 161020
com.xxx.xxx	mobile	logindata.phone: 161020, logindata.password: 161020

LogCat敏感信息输出：防护

防护建议：

- 1、eclipse中配置ProGuard实现release版apk实现自动删除Log.d()/v()等代码。
- 2、使用自定义LogCat类，上线前关闭Logcat开关。

```
public class LG {  
    /**
```

```
* 是否开启debug
*/
public static boolean isDebug = true;
/**
 * 错误
 * @param msg
 */
public static void e(Class<?> clazz, String msg)
{
    if (isDebug) {
        Log.e(clazz.getSimpleName(), msg + "");
    }
}
/**
 * 信息
 * @param msg
 */
public static void i(Class<?> clazz, String msg)
{
    if (isDebug) {
        Log.i(clazz.getSimpleName(), msg + "");
    }
}
/**
 * 警告
 * @param msg
 */
public static void w(Class<?> clazz, String msg) {
    if (isDebug) {
        Log.w(clazz.getSimpleName(), msg + "");
    }
}
/**
 * 调试
 * @param msg
 */
public static void d(Class<?> clazz, String msg) {
    if (isDebug) {
        Log.d(clazz.getSimpleName(), msg + "");
    }
}
```

```
}
```

敏感数据明文存储于Sdcard

Android系统的文件一般存储在sdCard和应用的私有目录下面。在Android系统中sdCard存储大多使用的FAT文件系统格式，不同于linux文件系统基于uid、gid的权限机制，任何在Manifest中声明了如下读写sdcard权限的应用都可以对sdcard进行读写。对于应用敏感数据严谨存储在sdcard，避免被污染或替换。

```
<!-- 在SDCard中创建与删除文件权限 -->
```

```
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
```

```
<!-- 往SDCard写入数据权限 -->
```

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

```
String data=et.getText().toString();
data = "Card Number:"+data+"\n";
FileOutputStream fos;

try {
    File myFile = new File("/sdcard/" + "secret.txt");
    myFile.createNewFile();
    fos = new FileOutputStream(myFile);
    fos.write(data.getBytes());
    fos.close();
    Toast.makeText(getApplicationContext(),
        "Data Saved in secret.txt", Toast.LENGTH_LONG).show();
} catch (FileNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

File Location

数据库敏感数据明文存储

Database配置模式安全风险源于:1)开发者在创建数据库(Database)时没有正确的选取合适的创建模式(MODE_PRIVATE、MODE_WORLD_READABLE以及MODE_WORLD_WRITEABLE)进行权限控制，从而导致数据库(Database)内容被恶意读写，造成账户密码、身份信息、以及其他敏感信息的泄露，甚至攻击者进一步实施恶意攻击。

如果在开发中没有使用正确的创建模式数据库(Database)文件，将会导致敏感信息泄露危害，如个人账户密码、身份信息以及金融账户等重要敏感信息。

openOrCreateDatabase(String dbName , int mode , CursorFactory factory)

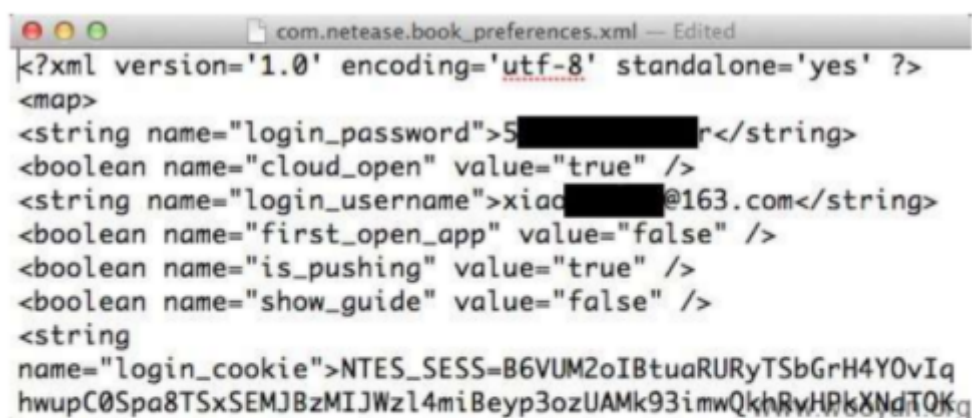
数据库敏感数据明文存储：防护

防护建议：

- 1、敏感信息在进行数据库存储时，对数据进行加密存储，并且应该避免弱加密或者是不安全的加密方式。
- 2、对于敏感的数据库文件，不得使用MODE_WORLD_READABLE或者是MODE_WORLD_WRITEABLE进行创建。

SharedPreferences全局可读写

对于应用的私有目录存储，一般有Shared Preferences、SQLite Databases两种存储方式。Shared Preferences存储安全风险源于：开发者在创建文件时没有正确的选取合适的创建模式(MODE_PRIVATE、MODE_WORLD_READABLE以及MODE_WORLD_WRITEABLE)进行权限控制；开发者过度依赖Android系统内部存储安全机制，将用户信息、密码等敏感重要的信息明文存储在Shared Preferences文件中，导致攻击者可通过root手机来查看敏感信息。图2-1所示是某app在preference中明文存储了用户的用户名和密码。



```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="login_password">5[REDACTED]r</string>
  <boolean name="cloud_open" value="true" />
  <string name="login_username">xia[REDACTED]@163.com</string>
  <boolean name="first_open_app" value="false" />
  <boolean name="is_pushing" value="true" />
  <boolean name="show_guide" value="false" />
  <string
name="login_cookie">NTES_SESS=B6VUM2oIBtuaRURyTSbGrH4Y0vIq
hwupC0Spa8TSxSEMJBzMIJWzl4miBeyp3ozUAMk93imwQkhRvHPkXNdTQKg
```

Shared Preference明文存储密码

SharedPreferences全局可读写：防护

防护建议：

- 避免使用MODE_WORLD_WRITEABLE和MODE_WORLD_READABLE模式创建进程间通信的文件，此处即为Shared Preferences。
- 不要将密码等敏感信息存储在Shared Preferences等内部存储中，即使Android系统内部存储安全机制，

使得内部存储文件可不让其他应用读写，但是在Android系统root之后，该安全机制将失效而导致信息

泄露。因此应该将敏感信息进行加密存储在Shared Preferences等内部存储文件中。

- ❑避免滥用 “Android:sharedUserId” 属性。

- ❑不要在使用 “android:sharedUserId” 属性的同时，对应用使用测试签名，否则其他应用拥有

“android:sharedUserId” 属性值和测试签名时，将会访问到内部存储文件数据。

- ❑使用Secure Preferences第三方加固库进行存储。

使用一个名为“ Secure Preferences” 第三方库保护保存在Shared Preferences中的数据。尽管

在设备被root的情况下，这个库也没法百分之百的保证应用的安全，但相比于普通的Shared

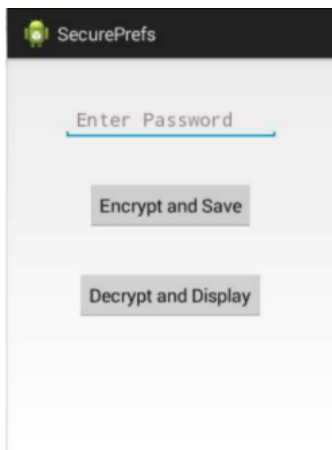
Preferences敏感信息被能得到更好的保护。 “Secure Preferences” 是一个Shared preferences加密

包装库，可以加密保存在XML文件里面的Shared Prefereces数据，并且是开源的。

ferences敏感信息

被能得到更好的保护。

```
SharedPreferences shared;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    tv = (TextView) findViewById(R.id.tvOut);
    et = (EditText) findViewById(R.id.editText1);
    btn1 = (Button) findViewById(R.id.btn1);
    btn2 = (Button) findViewById(R.id.btn2);
    shared = new SecurePreferences(this);
    btn1.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View arg0) {
            String input = et.getText().toString();
            Editor editor = shared.edit();
            editor.putString("PASSWORD", input);
            editor.commit();
            Toast.makeText(getApplicationContext(), "Success", Toast.LENGTH_LONG).show();
            et.setText("");
        }
    });
    btn2.setOnClickListener(new OnClickListener() {
        public void onClick(View arg0) {
            String out = shared.getString("PASSWORD", null);
            tv.setText(out);
        }
    });
}
```

```
root@hammerhead:/data/data/com.isi.secureprefs/shared_prefs # ls
ls
com.isi.secureprefs_preferences.xml
root@hammerhead:/data/data/com.isi.secureprefs/shared_prefs # cat com.isi.secureprefs_
preferences.xml
isi.secureprefs_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="C+ZK6J3dJOI1Q03pXVAXET0bru4Y8Am6m0NpryfTDWA=">0KV2JU9Ma15qauNdL7puLQ
==:TARcXCDKkMF/1rTag63Sxjr7kEHYPihWrSCb+S4DvSw=:cA1Jk7e4yZDcf61lQHsxew==</string>
  <string name="4oRpYmRImVULyMPRdfhKbYS8QANMB5pTE3KmnDtp10s=">noApvWdxvUPKUFJYwp8wIA
==:bRaBUKKeLN9tj06v2xy+ZGEC8WACgxVG2wGB29XGvcI=</string>
</map>
root@hammerhead:/data/data/com.isi.secureprefs/shared_prefs # a|
```

```
private static String generateAesKeyValue() throws NoSuchAlgorithmException {
    // Do *not* seed secureRandom! Automatically seeded from system entropy
    final SecureRandom random = new SecureRandom();

    // Use the Largest AES key length which is supported by the OS
    final KeyGenerator generator = KeyGenerator.getInstance("AES");
    try {
        generator.init(KEY_SIZE, random);
    } catch (Exception e) {
        try {
            generator.init(192, random);
        } catch (Exception e1) {
            generator.init(128, random);
        }
    }
    return SecurePreferences.encode(generator.generateKey().getEncoded());
}
```

敏感信息硬编码

在Android APP开发过程中，开发人员有可能因为对业务不熟悉、架构了解的不彻底或者是方便起见，把企业的一些敏感信息硬编码在APP代码中，导致黑客可以逆向程序APK，拿到这些敏感信息，从而对企业造成危害，并且有的危害是长时间的，比如内网的服务器地址、APK的签名私钥、地址分配规则、网络拓扑、页面注释信息（开发者姓名或工号、程序源代码）、后台登录及数据库地址以及接入支付宝支付时候使用的支付私钥。这些信息一旦泄露，对于企业来讲如果更换这样的信息，付出的代价尤为大，甚至是为了业务，不得不选择让泄露事件继续，使得企业长时间暴露在黑客的攻击之下。



HTTP敏感信息明文传输

某 App 登录时使用 HTTP 明文传输：

HTTP	alog.umeng.com	/app_logs	body	
HTTP	api. [redacted].com	/api/v1/user/phonelogin	Name	Value
HTTP	api. [redacted].com	/api/v1/user/[redacted]/295106/null/Android	celphone	1800925 [redacted]0
			code	484282

修复建议：

- 1、敏感信息如果使用HTTP传输，那么对敏感信息进行加密，并且尽量使用非对称加密，或者是公认的强加密算法。
- 2、使用HTTPS传输敏感信息。
- 3、以下字段在数据库的存储以及传输过程中，我们建议加密处理：密码、手机号、快捷支付手机号、Email、身份证、银行卡、CVV 码、有效期。

边信道信息泄露

Content Provider 信息泄露

Copy/paste buffer缓存

日志记录

URL缓存

浏览器Cookie对象

第三方统计数据

1、Content Provider 信息泄露

即使数据不会储存在设备上，任然可以通过一个恶意应用来提取漏洞Content Provider的数据。

2、Copy/Paste buffer缓存

Android中的Copy/Paste buffer缓存也是一个存在安全问题的地方，由于移动设备拼命的限制，用户更倾向于使用复制粘贴。如果用户把信用卡号这样的敏感信息复制到了剪贴板，攻击者通过一小段代码就能轻轻松松读取到数据

```
ClipboardManager clipboard = (ClipboardManager) getSystemService(Context.CLIPBOARD_SERVICE);  
ClipData.Item data = clipboard.getPrimaryClip().getItemAt(0);  
data.getText();
```

攻击者通过在受害者设备中植入恶意应用，就能随时随地读取受害者的敏感信息，给受害者造成非常严重的损失。

我们用上面的代码开发了一个示例应用，来演示恶意应用如何从剪贴板读取敏感信息。

假设用户在使用合法应用的时候已经将敏感信息复制到了剪贴板，现在，我们使用恶意程序来从剪贴板中读出信息：



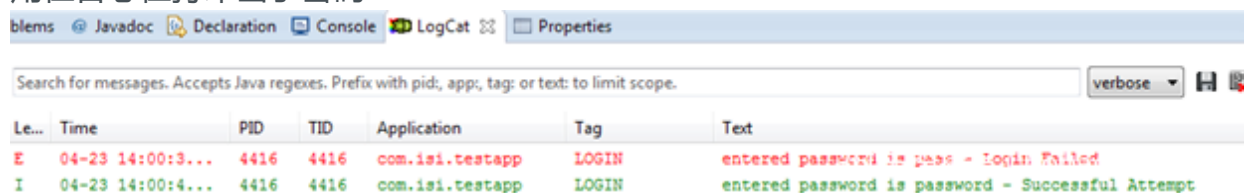
我们的示例只是读出数据并显示在屏幕上，但攻击者能远程控制恶意应用将读取到的信息发送到他自己的服务器上

3、日志

Android提供的日志功能也是一个会造成信息泄露的地方，日志一般是开发者在开发期间调试使用的，在本节中，我们将会看到攻击者如何通过日志信息发现信息泄露。在测试时，我有多种方式使用logcat工具来读取日志信息。

3.1 使用Eclipse

如果使用Eclipse IDE，只需要把你的设备连接到电脑上，我们就能在Logcat选项卡中看到应用运行期间输出的所有日志信息，其中可能就会有一些敏感信息，下面的截图就是测试应用在日志打印出了密码



3.2 使用adb

我们使用adb来查看日志

首先连接设备到电脑，并使用以下命令：

```
# adb logcat
```

该命令在终端中打印出所有的日志，如下图。图中我们感兴趣的应用日志分散在大量的系统事件日志中，我们可以通过logcat的选项来过滤出我们感兴趣的東西：

-v verbose 打印详细

-d debug 打印调试级日志

-l information 打印提示级日志

-e error 打印错误级日志

-w warning 打印警告级日志

[illegible]

我们也能通过以下命令将logcat的输出保存到文件中：

```
#adb logcat > output.txt
```

将日志保存到电脑本地，一遍后期进一步分析。

3.3使用恶意应用

我们也能开发一个恶意应用来读取设备的日志信息，关键代码如下：

```
Process process = Runtime.getRuntime().exec("su -c logcat -d");
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(process.getInputStream()));
StringBuilder log=new StringBuilder();
String line = "";
while ((line = bufferedReader.readLine()) != null)
{
    log.append(line);
    log.toString();
}
```

提示：READ_LOGS权限在Jellybean(Android 4.1 API level 16)后不再对第三方应用开放，但这段代码任然可以在root过的设备上运行。

4.URL缓存和浏览器cookie对象

已经有大量基于web view的应用程序造成URL，cookie和缓存等泄露的问题，这允许攻击者劫持用户的会话。这样的缓存可能存在日志，流量历史，浏览缓存等多种形式。

我们能用grep从logcat的输出中过滤出诸如cookies等敏感信息，命令如下：

```
#adb logcat|grep "cookie"
```

许多的应用都没有禁用掉缓存.可以简单的通过" no-cache" ," no-store" 等HTTP头来避免造成信息泄露。

这方面有许多的漏洞报告。

5.分析发送到第三方的信息

某些情况下，有的应用会使用第三方API。在使用这类应用时，第三方的API可能会读取诸如设备ID及位置信息等敏感信息。

总结：

按照2014年发布的OWASP 移动安全漏洞 TOP 10，非预期的信息泄露排名第四，尽管这看起来是一个非常简单的漏洞，但其泄露的关键信息可能会造成非常严重的安全风险。攻击者检查信息泄露的方法相对非常简单，所以强烈建议开发者在开发应用过程中注意防范信息泄露。