Lesson 2 – Insecure Data Storage

Introduction

This lesson focuses on finding and exploiting security issues in Android applications that are caused by insecure storage of sensitive data in the device memory.

Android applications can store data in:

- Internal storage (shared preferences, databases, cache, raw files).

  Access is protected by Unix permissions mechanism - each application has its own user in Android system, which isolates the applications from one another. This is part of the mechanism called Android Application Sandbox[1]. The files are located in /data/data/package.name/ path on the device.
- External storage (raw files)

  Shared storage. Can be in the internal device memory (usually sdcard0) or an external SD card (usually sdcard1). There is no file access permission mechanism. Every application with permissions READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE can access external storage.

Getting started

You will need:

- a device or emulator with Android, enabled USB debugging and root access (with sqlite3 tool installed),
- computer (Windows or Linux) with Android SDK installed.

Connect the device to the computer using USB cable and check that it is properly discovered by typing:

```
user@goatdroid:~$:adb devices
List of devices attached
000ea1a82cebaf    device
```

If you cannot see the device check that drivers are installed[2] and the device is connected. In case of `unauthorized` text displayed next to device id click on the box displayed on the device to grant the computer access to the device.

Now you can start the device shell:

```
user@goatdroid:~$:adb shell
shell@GT-I9100:/ $
```

And get root access:

```
shell@GT-I9100:/ $ su
root@GT-I9100:/ #
```

1 https://developer.android.com/training/articles/security-tips.html

2 You can find drivers at https://developer.android.com/studio/run/oem-usb.html.

Finding vulnerabilities

Let's start with checking the internal storage. It is stored on the device in path `/data/data/package.name/`.

List the app directory:

```
root@GT-I9100:/ # ls -la /data/data/org.owasp.goatdroid.fourgoats
drwxrwx--x u0_a80   u0_a80            2016-12-28 13:24 app_webview
drwxrwx--x u0_a80   u0_a80            2016-12-28 13:54 cache
drwxrwx--x u0_a80   u0_a80            2016-12-28 13:41 databases
drwx------ u0_a80   u0_a80            2016-12-27 16:28 files
lrwxrwxrwx install   install               2016-12-27 16:28 lib -> /data/app-
lib/org.owasp.goatdroid.fourgoats
drwxrwx--x u0_a80   u0_a80            2016-12-28 13:24 shared_prefs
```

List shared preferences folder:

```
root@GT-I9100:/ # cd /data/data/org.owasp.goatdroid.fourgoats
root@GT-I9100:/data/data/org.owasp.goatdroid.fourgoats/ # cd shared_prefs
root@GT-I9100:/data/data/org.owasp.goatdroid.fourgoats/shared_prefs/ # ls -la
-rw-rw---- u0_a80   u0_a80        124 2016-12-28 13:24 WebViewChromiumPrefs.xml
-rw-rw-r-- u0_a80   u0_a80        207 2016-12-27 17:05 credentials.xml
-rw-rw-r-- u0_a80   u0_a80        156 2016-12-28 12:20 destination_info.xml
```

As you can see, files are owned by u0_a80 – the OS user created by Android to protect the access to application resources. The files credentials.xml and destination_info.xml have read and write permissions for user (u0_a80) and group (u0_a80) and read permission for others.
It means that every application can get access to these files (you need to know the full path to the file, since without root or u0_a80 access, you cannot list the directories).

Now let's check the databases:

```
root@GT-I9100:/data/data/org.owasp.goatdroid.fourgoats/ # cd databases
root@GT-I9100:/data/data/org.owasp.goatdroid.fourgoats/databases/ # ls -la
-rw-rw---- u0_a80   u0_a80      20480 2016-12-28 13:41 checkins.db
-rw------- u0_a80   u0_a80       8720 2016-12-28 13:41 checkins.db-journal
-rw-rw---- u0_a80   u0_a80      16384 2016-12-28 13:25 userinfo.db
-rw------- u0_a80   u0_a80       8720 2016-12-28 13:25 userinfo.db-journal
```

We can see that none of the databases files have any permission for other users. But it does not mean that the data is safe. The data still can be accessed by malicious entity with root access.

Exploiting the vulnerability

Let's open credentials.xml file (notice the dollar sign – no root access is needed):

```
shell@GT-I9100:/ $ cat
/data/data/org.owasp.goatdroid.fourgoats/shared_prefs/credentials.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="username">joegoat</string>
    <string name="password">goatdroid</string>
    <boolean name="remember" value="true" />
</map>
```

We found the username and password strings for the remembered application user. Malicious application or malicious user with access to the device shell (without root access) could access this file and steal the credentials.

Now, we can try to access one of the databases with root access. To do this we will use sqlite3 – sqlite database tool used by Android developers and available on emulators and some of the rooted devices.

```
root@GT-I9100:/data/data/org.owasp.goatdroid.fourgoats/databases/ # sqlite3
checkins.db
SQLite version 3.8.6.1 2015-05-21 17:24:32
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  autocheckin       checkins
sqlite> .headers ON
sqlite> select * from checkins;
id|checkinID|venueName|dateTime|latitude|longitude
1|2d9cd359019f0ebd4f30a6f55823d359c0f44c97d9188baaf604e43e6a412523|BUW|2016-12-
28 13:41:28|52.2427936|21.0252238
```

We used `.tables` command to list the tables in this database file and `.headers ON` command to turn on displaying headers in query statements.
By querying the `checkins.db` database we gained knowledge about device users checkins, their time and locations.

Fixing the vulnerability

The vulnerability is caused by onCreate() method in Login Activity:

```
SharedPreferences prefs = getSharedPreferences("credentials",
MODE_WORLD_READABLE);
```

To fix the permission it is enough to remove the MODE_WORLD_READABLE constant, since by default Shared Preferences have MODE_PRIVATE parameter:

```
SharedPreferences prefs = getSharedPreferences("credentials");
```

The extra security measures have been taken to protect Android developers from committing insecure data storage mistakes. The constants MODE_WORLD_READABLE and MODE_WORLD_WRITABLE are deprecated since Android API level 17 (Android 4.2). Starting from API level 24 (Android 7.0) their use will cause a Security Exception[3].

It is more difficult to protect data against malicious entities with root access. Since root user can access any files on the device, the only way to protect data is to implement encryption. The common choice for databases is SQLCipher[4]. It still leaves the problem of encryption key protection. Some poor encryption implementations use hardcoded encryption keys in the source code, others put it somewhere on the device.

But the only safe way to store private keys on the device is not to store it at all! One of the ways to implement safe encryption of sensitive data on the device is to use cryptographic algorithm that derives the encryption key from password provided by user every time the application is opened. Common choice for key derivation algorithm in Android environment is PBKDFv2 (Password-Based Key Derivation Function 2).

3 https://developer.android.com/guide/topics/data/data-storage.html
4 https://www.zetetic.net/sqlcipher/sqlcipher-for-android/