Saarland University
Information Security & Cryptography Group
Prof. Dr. Michael Backes

Android Security Lab WS 2014/15

Supervised Project: Domain Isolation based on Access Control

M.Sc. Sven Bugiel

Version 1.0 (October 12, 2014)

Login on Lab Machines

Username:	smartseclab
Password:	android

Introduction

In this supervised project you will implement a security extension to AOSP that enables a simple domain isolation between applications from a *work* domain and applications from a *private* domain. Private apps can be, for instance, apps downloaded from Google Play or other source, while work apps are apps specifically endorsed by the employer of the end-user to handle security-sensitive, confidential work tasks. For simplicity we assume that endorsed work apps have been signed with one of a number of known developer keys. Isolation means that there should be no information flow from a work app to a private app. In light of the limited time, we consider only a few selected channels for information flows:

- ContentProvider access
- Broadcasts
- Clipboard
- Package management

As Figure 1 illustrates, different approaches to establish domain isolation exist, but in this exercise you will focus alone on the Android middleware layer. Goal of this exercise is that you learn to understand the implementation of the middleware security architecture and how it can be navigated and modified. Thus, hints are given as to where to put your security extension, but you are required to analyze the related middleware components by yourself.

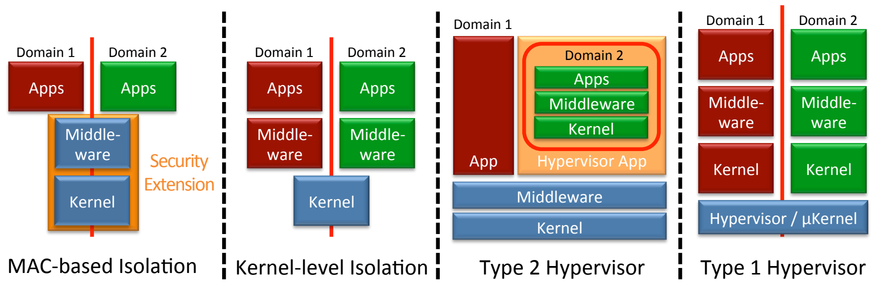


Figure 1: Possible approaches to domain isolation. Figure adapted from [1].

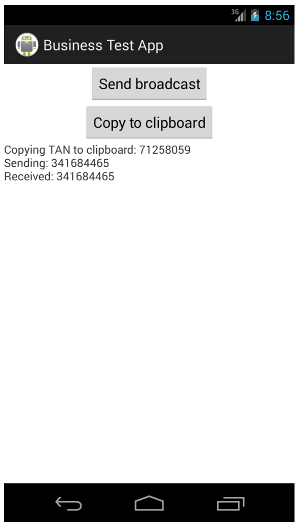
1 Test applications

To test your security extension, you will be provided with some simple test apps that implement the selected communication channels. The work apps are signed with the same developer key (cf. Appendix A).

You can download the apps from the following URLs:

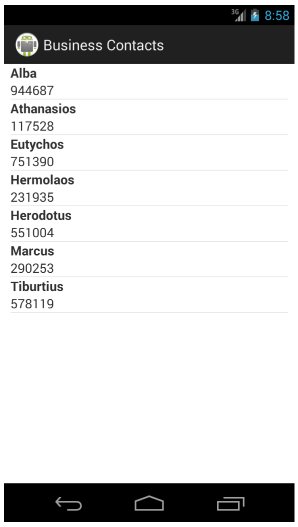
URL	Checksum (SHA-1)
`http://infsec.cs.uni-saarland.de/teaching/14WS/AndroidSecurityLab/files/BadApp.apk`	337b6848424f8698a1144660003206d8a28252ed
`http://infsec.cs.uni-saarland.de/teaching/14WS/AndroidSecurityLab/files/BusinessContacts.apk`	dd0fbb2fcd2e3dd587572e0544633e47e040c394
`http://infsec.cs.uni-saarland.de/teaching/14WS/AndroidSecurityLab/files/BusinessTestApp.apk`	8c779a48a90d4e5a9eeb418a07d30dead4a9addd

Table 1: Test apps download location and checksums



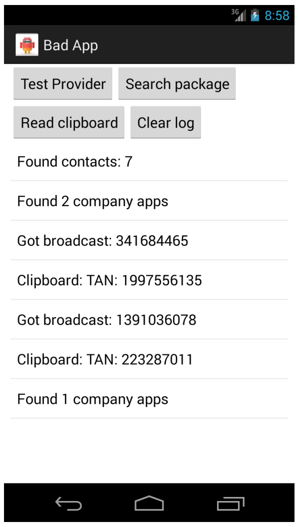
```
Copying TAN to clipboard: 71258059
Sending: 341684465
Received: 341684465
```

(a) Interface of Business Test App (work domain).



```
Alba
944687
Athanasios
117528
Eutychos
751390
Hermolaos
231935
Herodotus
551004
Marcus
290253
Tiburtius
578119
```

(b) Interface of Business Contacts App (work domain).



```
Found contacts: 7
Found 2 company apps
Got broadcast: 341684465
Clipboard: TAN: 1997556135
Got broadcast: 1391036078
Clipboard: TAN: 223287011
Found 1 company apps
```

(c) Interface of Bad App (private).

Figure 2: Provided test apps.

1.1 Business Test App

The business test app (cf. Figure 2a) sends out a broadcast with a randomly generated integer. The app also acts as a receiver of this broadcast. Additionally, it copies a randomly generated integer to the clipboard's primary clip. All events, i.e., sending broadcasts, receiving broadcasts, and copying to the clipboard are logged in the apps activity. Since there might be multiple developer keys for endorsed work apps, the permission system is too inflexible to protect the broadcast (in this scenario) and it relies on the security extension to prevent unauthorized apps from receiving the broadcast or reading the clipboard data.

1.2 Copmany Contacts Provider

The business contacts app (cf. Figure 2b) implements a simple content provider with some pre-installed contacts data. The content provider is exported and can be reached by other applications. Since there might be multiple developer keys for endorsed work apps, the permission system is too inflexible to protect the provider (in this scenario) and it relies on the security extension to protect it from being accessed by unauthorized apps.

1.3 Bad App

The bad app (cf. Figure 2c) represents the private apps. It will try to read the content of the primary clip in the clipboard service and to query the business contacts content provider. Additionally, it has a broadcast receiver with an Intent filter for the broadcast sent by the business test app. Moreover, it will go through the list of all installed packages and check whether it finds any work application (based on package name and signature).

All events, e.g., receiving a broadcast, are logged to the screen and allow you to verify whether it received the correct business data (e.g., same random integer generated by the business app).

2 Requirements

The following is the list of security requirements for your security extension:

- Bad app should not receive the broadcast by the business test app anymore, while the business test app still receives it.

- Bad app should not be able to query the business contacts provider anymore or only receive empty data.
- Bad app should not be able to retrieve any data from the clipboard that the business test app previously copied to the clipboard.
- Bad app should not be able to find any work app anymore by iterating the list of installed applications as provided by the package manager.

3 Implementation

The following steps and hints are recommended for your implementation:

1. Extend the installation routine of the PackageManagerService with additional logic to label applications as *work* or *private* at installation time. For simplicity, labeling should be decided based on the developer key (cf. 1). The SELinux MMAC in `SELinuxMMAC.java` can provide an inspiration on how this can be accomplished.
2. Extend middleware services and core applications with policy enforcing logic (i.e., reference monitoring for communication between work and private domain). The AppOps and Intent firewall of AOSP can give an idea of how policy enforcement for services like clipboard and for broadcasts can be accomplished. See `AppOpsManager.java` and `AppOpsService.java`.
3. Check how an app can use the PackageManagerService to find all installed applications and modify the logic of PackageManagerService to filter the results of this functions so that a private app cannot find a work app.

References

[1] S. Bugiel, L. Davi, A. Dmitrienko, S. Heuser, A.-R. Sadeghi, and B. Shastry, “Practical and lightweight domain isolation on Android,” in *Proc. 1st ACM CCS Workshop on Security and Privacy in Mobile Devices (SPSM’11)*, ACM, 2011.

A Work App Developer Keys

Listing 1: ”Full signature in ASCII format of the work apps’ developer key”

```
308202c3308201aba003020102020204589646b9300d06092a864886f70d01010b050030123110300e
06035504031307436f6d70616e79301e170d3134313031313130323032375a170d34393130303231
30323032375a30123110300e06035504031307436f6d70616e7930820122300d06092a864886f70d
01010105000382010f003082010a028201010086238e5ab6efbb2d56c393ccc29c9af2f59869b628
f22966d5d9a3855a41e32afcc04a9db345db7d5123a4ad36e803cf8c1ca73b27b63cce9c40d37926
3bd87665722501fe8f041bd770ca59d1dacc8cfe0eba1ae40f309c8a4354b1382349cd1c3edc8437
bfe07e5c2f827e6624d8eedab1d150e83b868cf4521ac73849c28f0bf0bb418e93a7242cc3beef08
0d30e56b32f3b36c0796d207dabaef6b0b4381433e3127c9a7cf59c757f7033426df626b700525c2
987ec5fda80117043793bfb161b99c57bfccadbd8677454fa861cb7743da000b81651be66bd30d4c
b579f7e241ff294407797aac7e6f3131481e13f9400809fd6de683475c5fce7e72f7970203010001
a321301f301d0603551d0e04160414990e738f83a0623b56722d4c3951cd7c33a01b70300d06092a
864886f70d01010b050003820101007f7b77583524fdecf929a7836dae87ccdeaf0cb13781d0ec21
929b7b06cb373370b098f26d49939db3edb8e51f81cc7df9f2ef639662a77ef6ce409c785a3018fa
518fc3cef0e29a0a087efe8b018afa167ba93d081bf2f6849fdfea35b256088a92798a5966c73479
05dcde392d019fa7f968ce07b2fde4c2198e46ec1832a9ccc88163a08b8e9f6b8bb33f151beb80b8
574c70b696a75e04159614a90e1751adec5c001b0955b6c06aba33e56e6c9128d61262009d03f8d4
a9151ce42ec38c2c04c4efa9a173b33bded714eed9ecf648cd667ca711ce9605d0d5826dab62e093
38303ad1e025ce8799c4de01bfc5b784ce0fdf0f51fa0300470965b2ac9a1b
```

Listing 2: ”SHA-1 of business developer signature in Listing 1 encoded in ASCII”

```
5BFFBA68E53EDE1557F6D95C0BD834FCD6C8BCA9
```