Attacking Smartphone Privacy Using Local Covert Channels

Jean-François Lalande      Steffen Wendzel

Ensi de Bourges, France

Augsburg University of Applied Sciences, Germany

8th of March 2013
Introduction

Privacy protection is one of the hot topics for smartphones:

- Private data comprises:
  - phone identifiers (IMEI)
  - contacts, phone numbers (MSISDN)
  - sms content
  - files, passwords, ...

- Data leakages enable to:
  - Sell collected information
  - Attack other targets using the collected information

- Malware can use the phone’s capabilities (e.g., send SMS)

[4] Morrow reports: 64% of the enterprises surveyed by Infonetics had data lost or stolen due to the use of mobile devices...
Outline

1. Introduction
   - Privacy Threats in Android
   - Covert channels

2. Malware based on Covert Channels
   - Malware design
   - Demonstration

3. Conclusion and Future Work
Examples of typical malware

Gone in 60 seconds [1]:

- the user launches the application
- **backups user’s data** (contacts, messages, history)
- launches the uninstall process
Examples of typical malware

Gone in 60 seconds [1]:
- the user launches the application
- backups user’s data (contacts, messages, history)
- launches the uninstall process

Walkinwat (fake version of Walk and Text):
- the user launches the application
- displays a “processing” screen
- complains with a license error
- indeed, sent SMS to all your contacts!
Examples of typical malware

Gone in 60 seconds [1]:
- the user launches the application
- backups user’s data (contacts, messages, history)
- launches the uninstall process

Walkinwat (fake version of Walk and Text):
- the user launches the application
- displays a “processing” screen
- **complains with a license error**
- indeed, sent SMS to all your contacts!
Malware countermeasures

A lot of efforts to defend private data:

- classical virus signature detection
- introduction of fine grained security policies
- dynamic tainting propagation mechanisms
- static analysis of the source/bytecode of applications
- collaborative constraint generation at execution time
- ...

For example Taindroid [2]:

- applies taints on resources
- taints variable of a program when accessing the resource
- propagates the taint over the program
- notifies the user if the taint leaks, e.g. via internet or SMS
Malware countermeasures

A lot of efforts to defend private data:

- classical virus signature detection
- introduction of fine grained security policies
- dynamic tainting propagation mechanisms
- static analysis of the source/bytecode of applications
- collaborative constraint generation at execution time
- ...

For example Taindroid [2]:

- applies taints on resources
- taints variable of a program when accessing the resource
- propagates the taint over the program
- notifies the user if the taint leaks, e.g. via internet or SMS
Covert channels

What about security if the malware exploits covert channels?
Covert channels

What about security if the malware exploits covert channels?

Covert channels are channels that:
- unforeseen by a system’s design
- exploit application/OS/hardware capabilities
- escape classical detection solutions

Our goal is to show that:
- covert channels can help to build a unnoticeable malware
- defeats security tainting solutions
Our proposal, similar to Marforio et al. [3]:

- Application 1: Data collector (CC sender)
- Application 2: Data submitter (CC listener)

- Data collector: gets private data
- Data submitter: leaks collected data
- Covert channel: local hidden communication path
Our proposal, similar to Marforio et al. [3]:

- **Data collector**: gets private data
- **Data submitter**: leaks collected data
- **covert channel**: local hidden communication path
Our proposal, similar to Marforio et al. [3]:

- **Data collector**: gets private data
- **Data submitter**: leaks collected data
- **covert channel**: local hidden communication path

---

**J.-F. Lalande - S. Wendzel**

*Attacking Smartphone Privacy Using Local Covert Channels*
Our proposal, similar to Marforio et al. [3]:

- **Data collector**: gets private data
- **Data submitter**: leaks collected data
- **covert channel**: local hidden communication path

Diagram:

- Application 1: Data collector (CC sender)
- Application 2: Data submitter (CC listener)
- Contacts -> Covert channel -> Internet
Required permissions

- The user will not suspect each app independently
- Automatic tools will miss the information flow
- How works the CC?
The covert channel is based on observable events:

- The screen turns off \(\Rightarrow\) starting transmission
- CC sender is killed: \(\Rightarrow\) ending transmission (GET_TASKS)
Why GET_TASKS permission is needed?

The covert channel is based on observable events:

- The screen `turns off` ⇒ starting transmission
- CC sender `is killed`: ⇒ ending transmission (GET_TASKS)
Why GET_TASKS permission is needed?

The covert channel is based on observable events:
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)

![Diagram showing screen state, CC sender state, and self kill](image)
Why GET\_TASKS permission is needed?

The covert channel is based on observable events:

- The screen turns off ⇒ starting transmission
- CC sender is killed: ⇒ ending transmission (GET\_TASKS)
Why GET_TASKS permission is needed?

The covert channel is based on observable events:

- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)
Why GET_TASKS permission is needed?

The covert channel is based on observable events:

- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)
Why GET_TASKS permission is needed?

The covert channel is based on observable events:

- The screen turns off ⇒ starting transmission
- CC sender is killed: ⇒ ending transmission (GET_TASKS)
Demonstration

http://www.dailymotion.com/video/xy02g8
Conclusion and Future Work

The designed covert channel enables:

- to leak private data
- to minimize and separate required permissions
- to leak bytes, correlated with the user action
Conclusion and Future Work

The designed covert channel enables:

- to leak private data
- to minimize and separate required permissions
- to leak bytes, correlated with the user action

We are currently working on:

- Throughput measurements
- Energy consumption of our CC
- Creation of a CC without any required permission
- Evaluation of TaintDroid’s capability to detect it
Questions
References

L. Botezatu.  
All data stored on your smartphone ..... gone in 60 seconds.  
MalwareCity, 2011.

W. Enck, P Gilbert, et al.  
TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones.  

C. Marforio, H. Ritzdorf, et al.  
Analysis of the communication between colluding applications on modern smartphones.  

B. Morrow.  
BYOD security challenges: control and protect your most sensitive data.  