Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels

Jean-François Lalande **Steffen Wendzel**

Ensi de Bourges, France

Fraunhofer FKIE, Germany

ECTCM'13 Workshop @ARES, Sep-04-2013

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
  - Blackmail a user
  - Attack other targets using the collected information
- Malware can use the phone's capabilities (e.g., send SMS)

Outline

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:

- are unforeseen by a system's design
- exploit application/OS/hardware capabilities
- ... in order to break a security policy
- escape classical detection solutions
- can be optimized to keep a low profile

... and can be found in local systems, networks, automation environments, business processes, smart cards, ...

Covert Channels

What about security if the malware exploits covert channels?

Covert channels are channels that:

- are unforeseen by a system's design
- exploit application/OS/hardware capabilities
- ... in order to break a security policy
- escape classical detection solutions
- can be optimized to keep a low profile

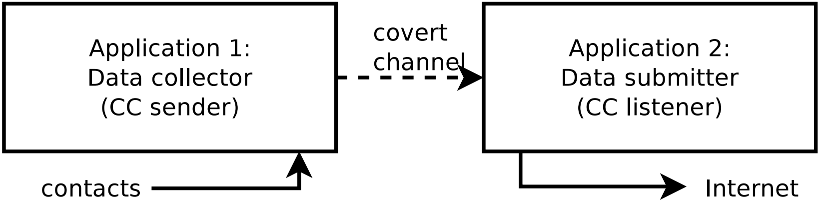
... and can be found in local systems, networks, automation environments, business processes, smart cards, ...

Our goal is to show that:

- covert channels can help to build an unnoticeable malware
- therefore, we improve the covert channel stealthiness

Malware Design

Our proposal, similar to Marforio et al. [?]:

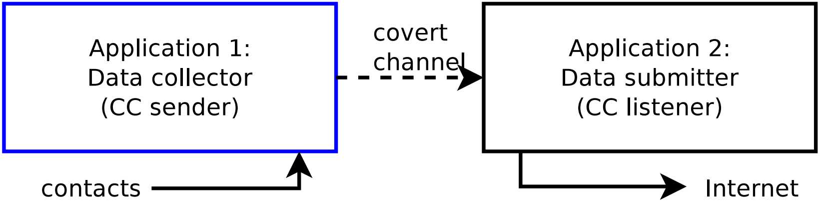


- Data collector: accesses private data
- Data submitter: leaks collected data
- covert channel: local hidden communication path

Scenario: Multiple smart home apps, workout apps, ...

Malware Design

Our proposal, similar to Marforio et al. [**?**]:

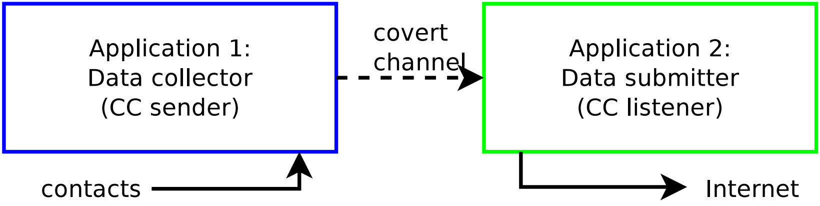


- Data collector: accesses private data
- Data submitter: leaks collected data
- covert channel: local hidden communication path

Scenario: Multiple smart home apps, workout apps, ...

Malware Design

Our proposal, similar to Marforio et al. [?]:

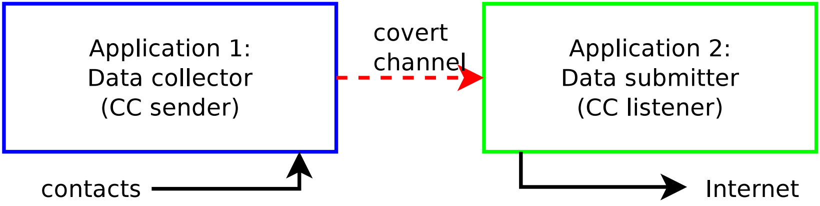


- Data collector: accesses private data
- Data submitter: leaks collected data
- covert channel: local hidden communication path

Scenario: Multiple smart home apps, workout apps, ...

Malware Design

Our proposal, similar to Marforio et al. [?]:



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

Scenario: Multiple smart home apps, workout apps, ...

Advanced Hiding Techniques for Covert Channels

The designed covert channel enables to leak private data and keeps a low profile by:

- minimizing and separating the required permissions
- leaking data correlated with the user interaction
- comprising a low energy footprint

Goal: User should not suspect presence of a malware.

Micro Protocols

We adapted a feature of network covert channel research:
Micro Protocols (MP).

Micro Protocols

We adapted a feature of network covert channel research: Micro Protocols (MP).

- MPs enable reliable covert channels
- MPs enable adaptive covert channels

We split the covert channel into a separate control channel (simple MP) and a data channel.

Four Different Covert Channels

We developed four covert channels linked to different required permissions:

Covert channel type	Control channel	Data channel	Required permission
CC#1: Task list / screen-based	screen state	task list	GET_TASK
CC#2: Process priorities / screen-based	screen state	process priority	
CC#3: Process priorities		process priority	
CC#4: Pure screen-based		screen based	WAKE_LOCK

Control and data channels of our covert channel techniques.

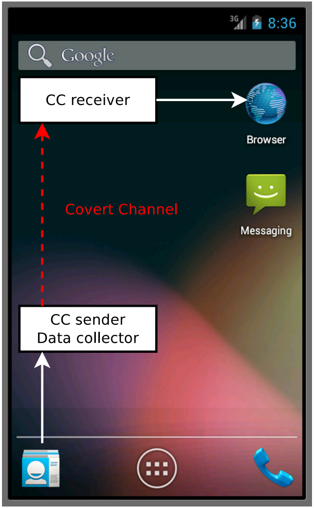
Required permissions (CC#1)

Required permissions

INTERNET

GET_TASKS

READ_CONTACTS

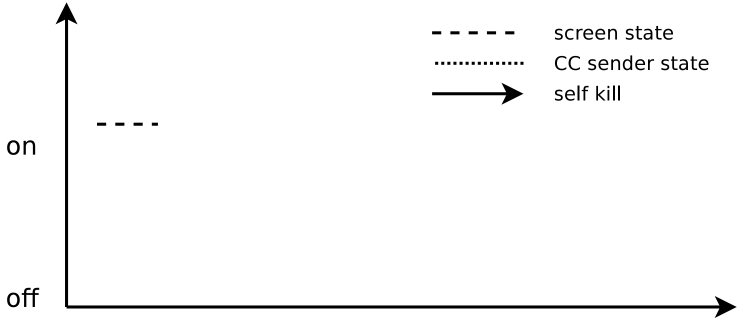


- The user will not suspect each app independently
- Automatic tools will miss the information flow
- How works the CC?

Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

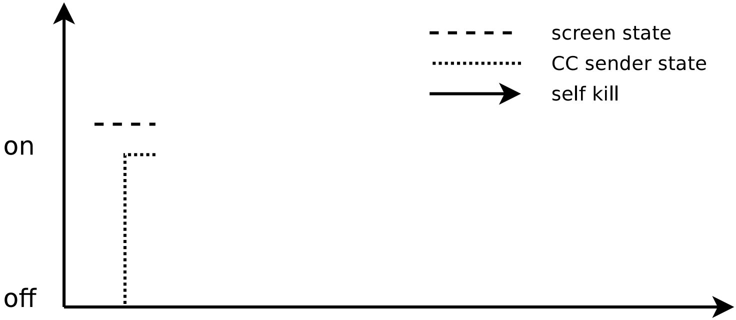
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

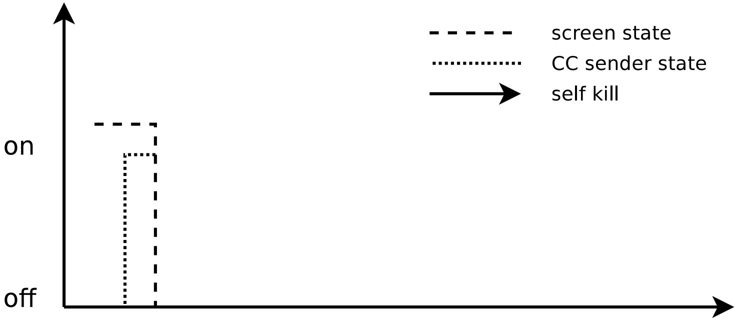
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

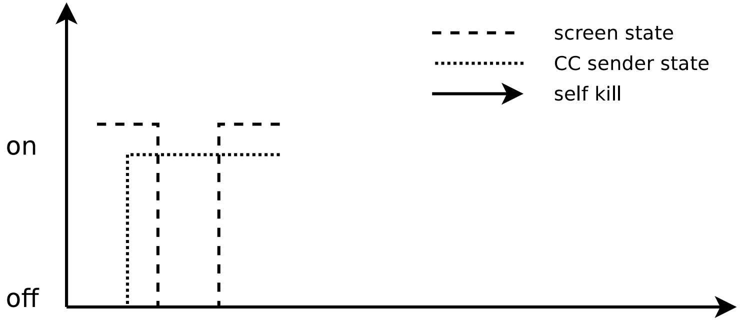
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

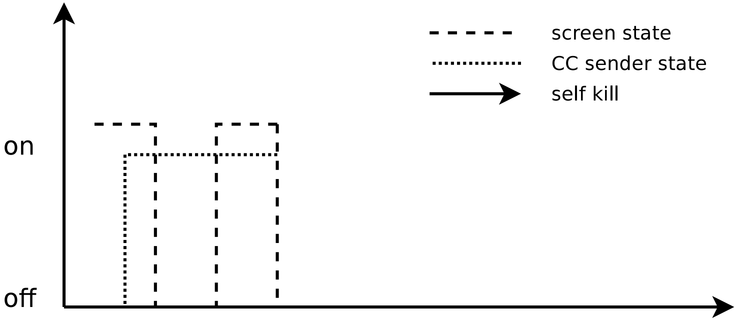
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

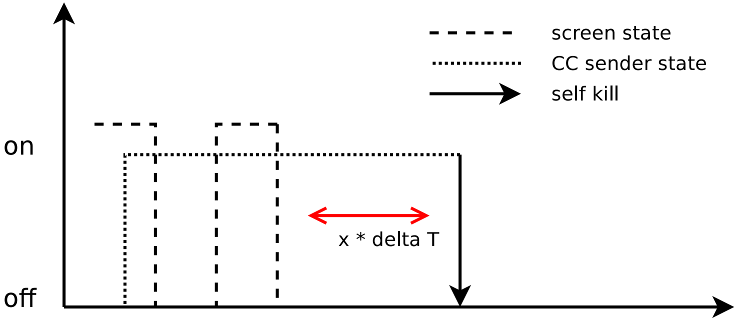
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

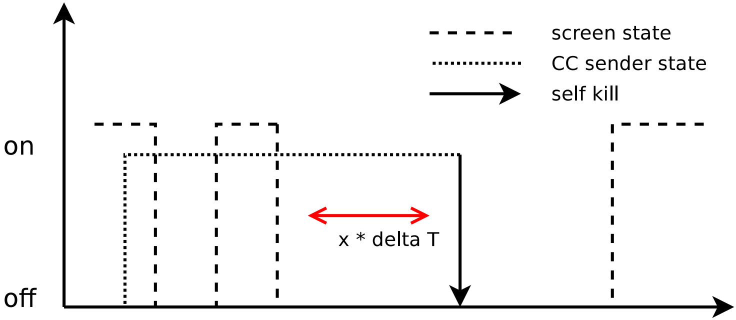
- The screen **turns off** ⇒ starting transmission
- CC sender **is killed**: ⇒ ending transmission (GET_TASKS)



Why GET_TASKS permission is needed?

Example: CC#1 is based on observable screen and task events:

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



Eliminating the requirement for GET_TASKS

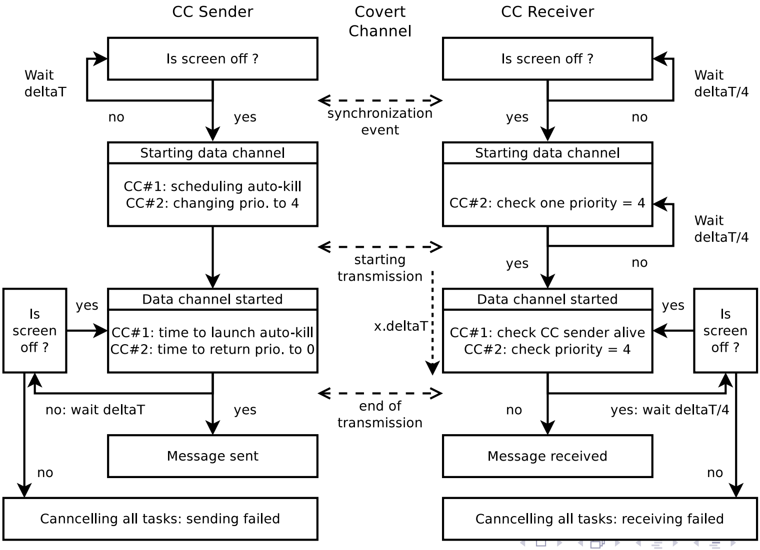
Synchronization realized like in case of CC#1 (screen state)

Data transfer based on process priority (receiver needs to scan for the priority) instead of process existence:

- Sender changes its priority to p known by sender and receiver
- Receiver iterates over all process IDs to determine the presence of a process with priority p (error-prone!)

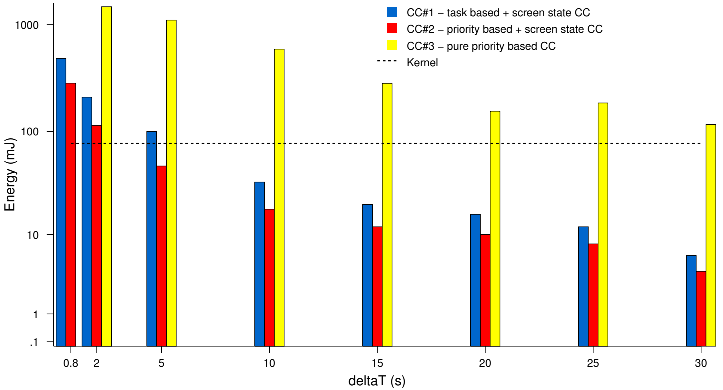
Two variants: With a data channel using the screen state (CC#2) and without data channel (CC#3)

Architecture of CC#1 and CC#2



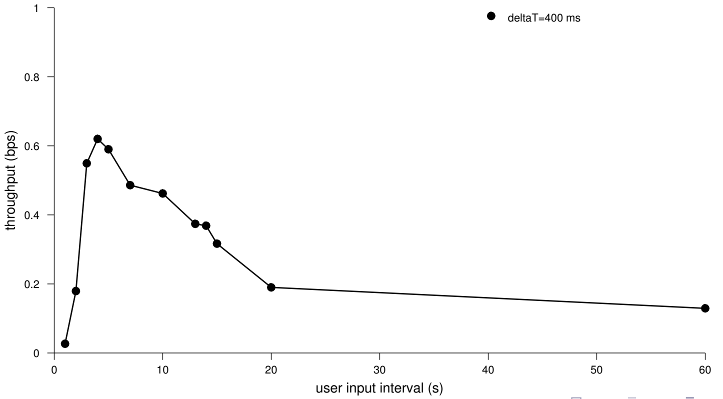
Energy Consumption

Energy consumption of CC#1 and CC#2 during 1 min of transmission (automatic fake user interactions) and of CC#3 during 1 min of runtime (measured using Power Tutor)



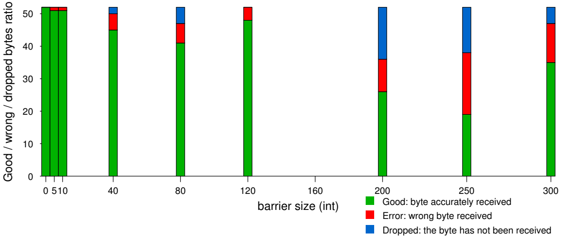
Throughput (Example CC#2)

Interruptions (screen interaction by the user after *n* sec). Low *n* interrupts CC transmission (the screen state changes), high *n* leads to long pause intervals between byte transmissions.



Countermeasures

- Applying the fuzzy time approach
- Applying machine learning
- Introducing barrier values, e.g. $\geq n$ process priority requests/sec
- Introduce errors for process priority requests (CC#3); Fig. shows results for introduced errors (barrier size) per 1000 API requests using solely the process priority:



Conclusion

- Introduced covert channels with a low throughput
- ... but with a high data transmission quality (control channel),
- ... the need for only few privileges,
- ... and a low energy footprint and a behavior correlated to the user's interaction (keeping a low profile)

Future Work

- Utilize multiple covert channels simultaneously
- Therefore: Introduce reliable covert channel protocols (sequence numbers and ARQ)
- Introduce adaptive covert channels (notice blocked communication means dynamically and switch to alternative channels on demand)

Questions

Demonstration Video: `http://www.dailymotion.com/video/x10lcyq_ectcm-2013-hiding-privacy-leaks-in-android-applications_tech`