

Self-Protecting Mobile Software Systems

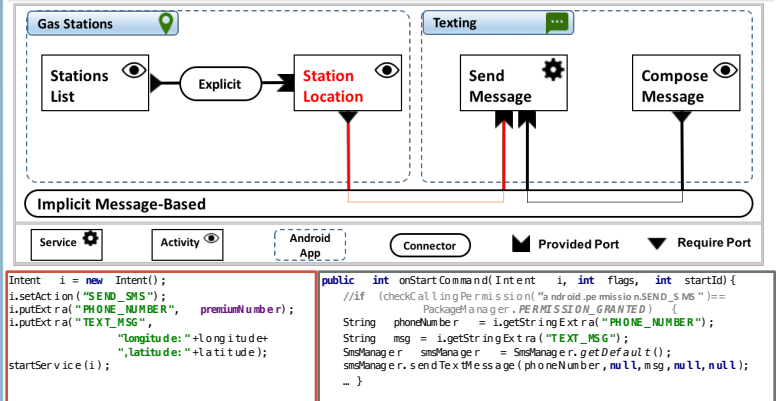
Mahmoud Hammad and Sam Malek
{hammadm, malek}@uci.edu



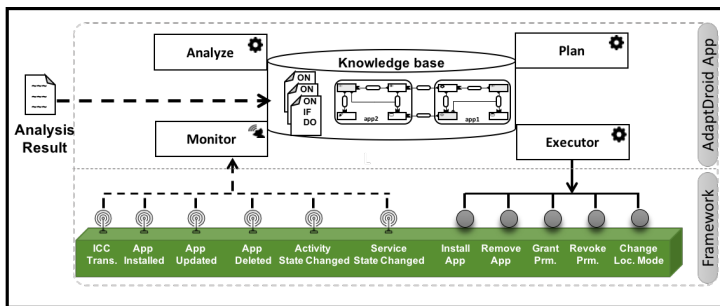
Research Overview

- Two general approaches for detecting security issues in mobile apps
 - Static analysis approaches produce false positives
 - Dynamic analysis approaches produce false negatives
- **Self-protecting software** can be used to mitigate their shortcomings
 - Statically determine the security vulnerabilities in apps
 - Dynamically monitor the execution of program for manifestation of security attacks
 - Prevent exploitation through runtime adaptation
- Self-protecting software relies on an **architectural model** of the system
 - In mobile software, such as Android, architecture of the system is not known ahead of time
 - Apps are installed, removed, and updated continuously

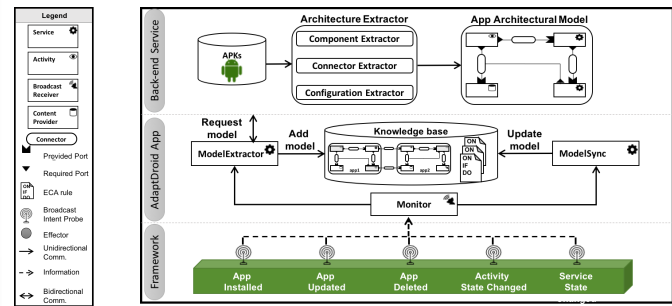
Motivating Example



Self-Protecting Mobile Software



Extraction and Synchronization of Architecture



Architectural Model

- System Model (M_S): Represents the architectural model of a software system
 - $M_S = \langle A, C, N, I, F, P \rangle$, where
 - A : Set of apps that are installed on the device
 - C : Set of Components (Activity, Service, Broadcast Receiver, or Content Provider)
 - N : Set of Connectors (Explicit-Msg based, Implicit-Msg based, RPC, or Data Access)
 - I : Set of *Intents*, that are event messages used in Android to facilitate ICC
 - F : Set of *Intent Filters*, that define ports provided by a component
 - P : Set of permissions either **requested**, to access a resource, or **defined**, to protect a component, by the app
- Environmental Model (M_E): Defines the environment in which apps run
 - $M_E = \langle E, T \rangle$, where
 - E : Set of environmental properties, examples include battery status, network connection, location, and moving speed
 - T : Set of device-specific properties, examples include the device unique ID, device manufacturer, carrier name

Security Goal Model

- Explicitly captures when adaptations are needed
- Adaptation rules are enforced by the self-protection layer
- Uses a rule-based adaptation policy following the ECA paradigm
- ECA rules can be derived from the output of static analysis tools
 - COVERT: a tool for compositional analysis of Android inter-apps vulnerabilities [1]
 - DidFail: a taint static analysis tool for detecting potential information leakage among a set of apps [2]
 - Amondroid: a taint static analysis tool for detecting data leak and data injection[3]
 - IccTA: a static taint analyzer to detect privacy leaks among components in Android applications [4]

ECA Rule – Fine-grain
 ON: $i \in ICC$ occurs
 IF : $i.sender = "StationLocation"$
 and $i.receiver = "SendMessage"$
 DO: $Stop(i)$

ECA Rule – Coarse-grain
 ON: $i \in ICC$ occurs
 IF : $i.senderPkg = \text{"GasStations"}$
 and $i.receiverPkg = \text{"Texting"}$
 DO: $Stop(i)$

Performance: Architecture Extraction

- MacBook Pro with 2.2 GHz Intel Core i7 processor and 16 GB DDR3 RAM
- Installed our modified Android version on an Android Virtual Device (AVD)
- 120 randomly selected real-world apps from IccRE repository
 - Average number of components: 13
 - Average number of Intents: 112

Criteria	Average Time (Sec)
Architecture Extraction	6.7
Add app's architecture to the model	0.8
Remove app's architecture from the model	0.03

Performance: Synchronization

- Android Virtual Device with 1 GB RAM
- 10,000 Monkey events changed the mode of operation for 623 components
- On average, it took 0.04 second to reflect the state of a component into the dynamic model

Attack Detection and Prevention

- ICC_BENCH contains 19 vulnerable apps
- Static analysis tools generated 84 security warnings
- Added an ECA rule for each warning
- 19 rules were matched at runtime, representing the true positives from the static analysis tools
- Our approach **detected** and **prevented** all of the security attacks in the ICC_BENCH

References

- References
- [1] Bagheri, Hamid, et al. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *IEEE Transactions on Software Engineering*, Vol.41, No. 9, September 2015.
 - [2] Klierer, William, et al. Android taint flow analysis for app sets. *International Workshop on the State of the Art in Java Program Analysis*, Edinburgh, United Kingdom, June 2014.
 - [3] Wei, Fengguo, et al. A precise and general inter-component dataflow analysis framework for security vetting of android apps. *ACM SIGSAC Conference on Computer and Communications Security*, Scottsdale, AZ, November 2014.
 - [4] Li, Li, et al. lcta: Detecting inter-component privacy leaks in android apps. *International Conference on Software Engineering*, Florence, Italy, May 2015.