Improving the Security of the Android Ecosystem

Yury Zhauniarovich

Advisor:
Bruno Crispo

University of Trento

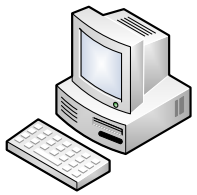
Agenda

- Introduction
- Providing Software and Data Isolation on Android
- Enabling Attestation Service for the Android platform
- Detecting Repackaged Android Applications
- Analyzing Android Apps in the Presence of Dynamic Class Loading and Reflection
- Summary

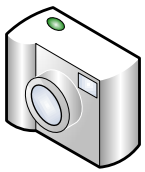
What is Smartphone?

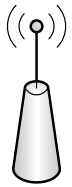
- Have “phone” capabilities
- Equipped with different sensors
- Can run third-party applications
- Controlled by a special mobile operating system

Smartphone is a source of very sensitive user information









Why Android?

- On about 82% of all new mobile devices
- 1+ billion devices activated
- 1+ million apps on Google Play
- Open source
- Open ecosystem
- Numerous third-party markets of different flavors (F-Droid, Yandex.Store, Amazon, etc.)

Agenda

- Introduction
- **Providing Software and Data Isolation on Android**
- Enabling Attestation Service for the Android platform
- Detecting Repackaged Android Applications
- Analyzing Android Apps in the Presence of Dynamic Class Loading and Reflection
- Summary

MOSES: Motivation

- Same device multiple virtual environment (e.g., in BYOD scenarios)
- Demand to increase the control over the capabilities of third-party apps, e.g., prohibit access to location
- Lack for context-based enforcement of security policies on Android
- Absence of remote control over virtual environments by the owners

MOSES: State Of the Art

- Secure containers
  - e.g., Aurasium by R.Xu et al. (USENIX Security ’12)
  - usually, 2 virtual environment (private and work)
  - app rewriting usage
- Mobile paravirtualization
  - e.g., L4Android by M.Lange et al. (ACM SPSM ’11)
  - predefined number of operation modes
  - battery-consuming solutions
- Linux containers
  - e.g., Cells by J.Andrus et al. (ACM SOSP ’11)
  - switching requires user interaction
  - virtual environments are hard-coded

MOSES: Problem

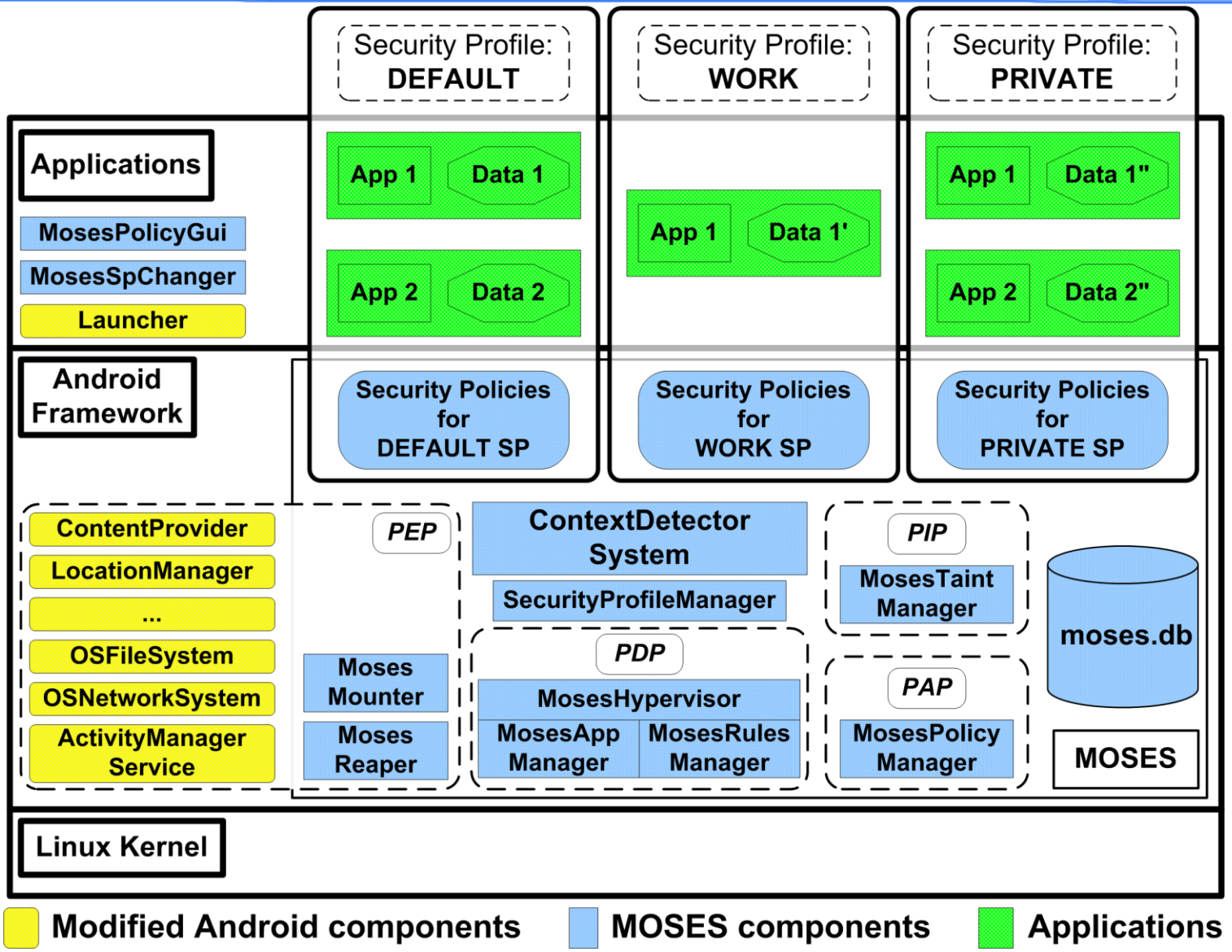
Issue 1: *How to provide several virtual environments*

- on the same device
  - users are not willing to carry several devices
- that separate data and apps belonging to different usage contexts
  - app developers should not rewrite their apps according to new rules
- managed by different owners
  - e.g., working environment is controlled by company administrators
- avoiding energy demanding (para)virualization solutions?
  - smartphones require long working time without recharging

MOSES: Idea

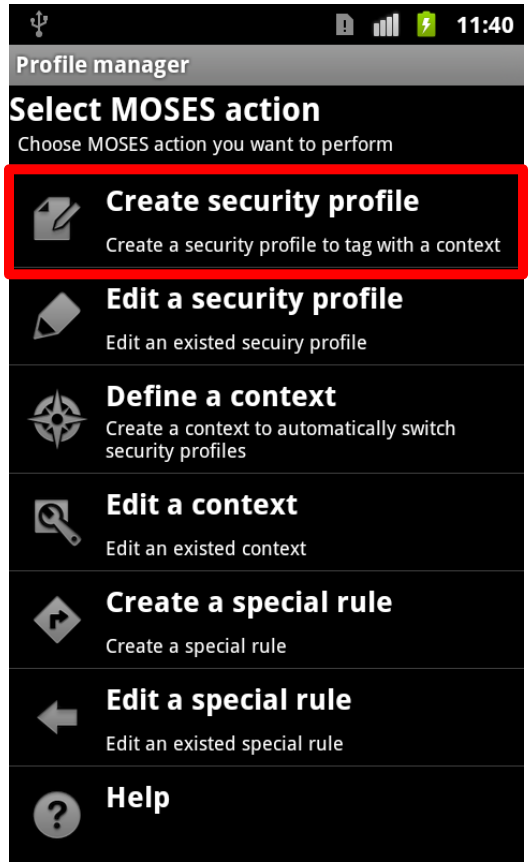
- **IDEA:** Provide a possibility to create virtual environments (or Security Profiles (SP)) through policy-based framework so that applications in one SP cannot access the data of the same app in another SP. Ensure the control over Security Profiles to the owners. Equip SPs with an ability to enforce fine-grained policies.

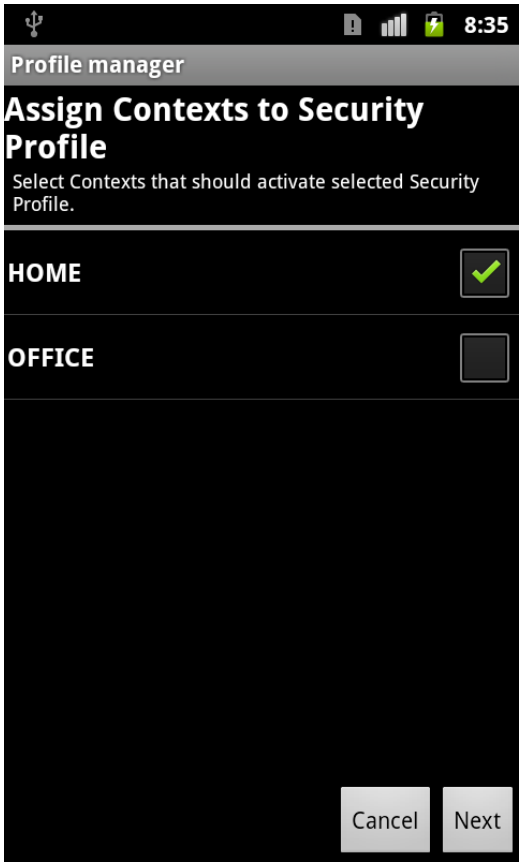
MOSES: Architecture



MOSES Configuration: Security Profile Creation

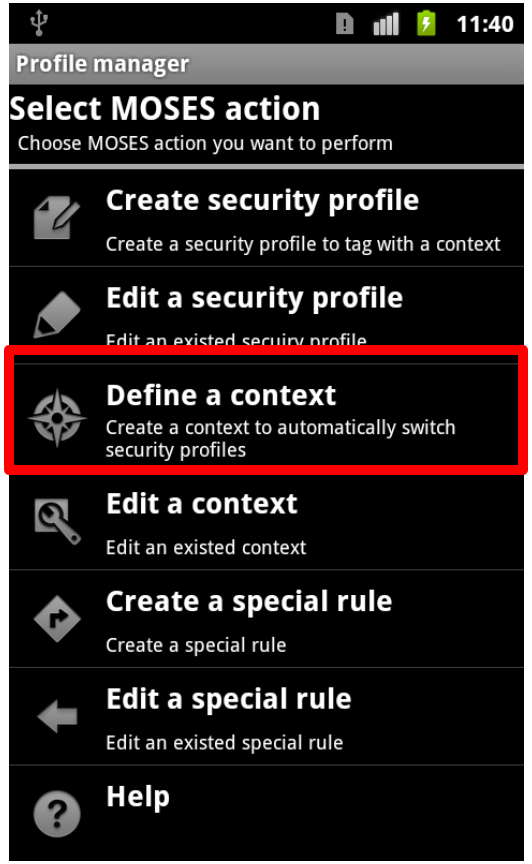
create_profile “private” **in_mode** “permissive” **with_priority** “50”;
allow_apps “*”;
add_sr “browser_deny_receive_from_google” **on_position** “10”;
activate_in_context “home”;

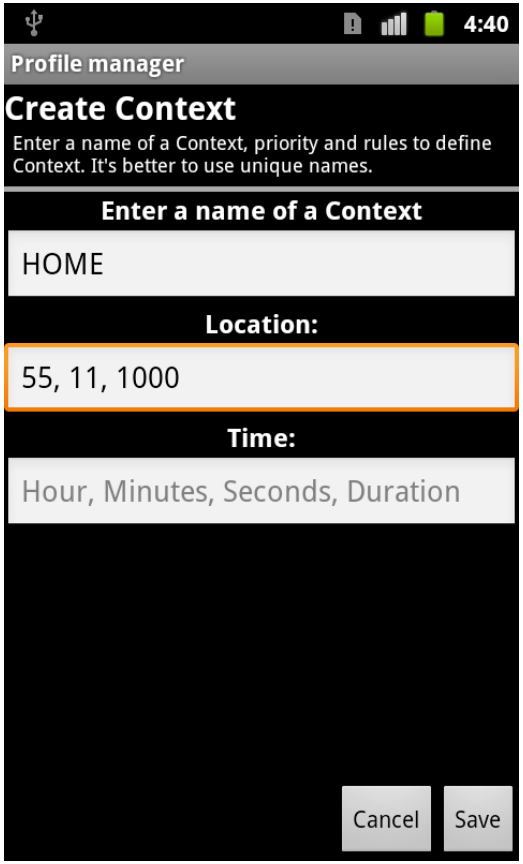




MOSES Configuration: Context Definition

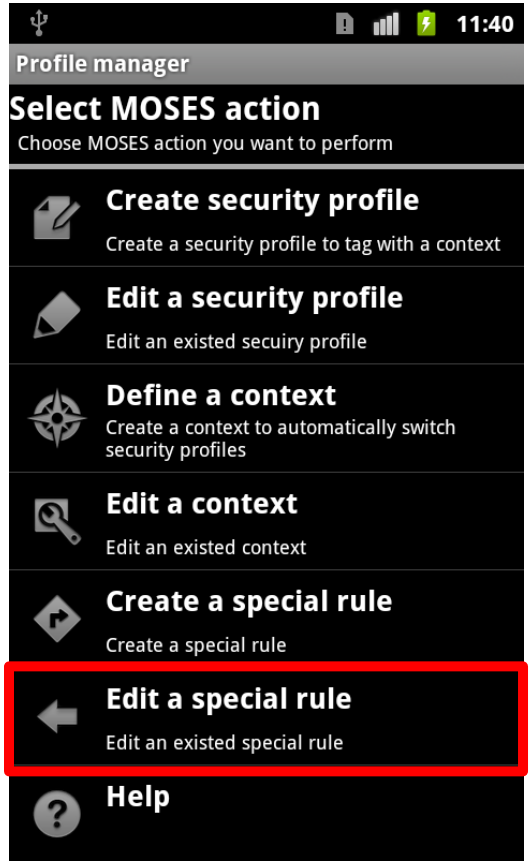
create_context “home”;
condition [(latitude=“55”) AND (longitude=“11”) AND (radius=“1000m”)];

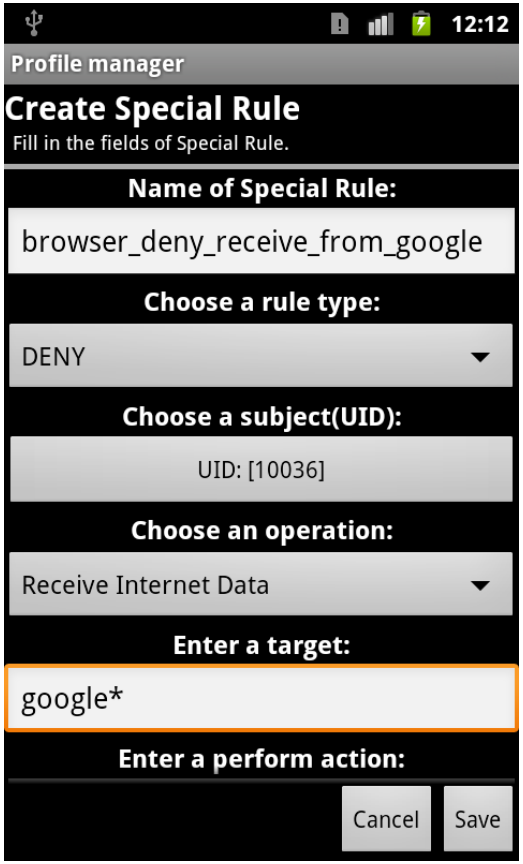




MOSES Configuration: Special Rule Creation

create_sr “browser_deny_receive_from_google”;
action “deny”;
package “com.google.android.browser”;
operation “receive internet data”;
target “google*”;
perform [];





MOSES: Contributions

- First policy-based solution for virtual environments on Android
- Manual and context-based Security Profiles activation
- Security Profiles and Contexts are not predefined, users can configure them dynamically
- Possibility to confine applications using fine-grained security policies
- Compatible with existing applications

Y. Zhauniarovich, G. Russello, M. Conti, B. Crispo, E. Fernandes. "MOSES: Supporting and Enforcing Security Profiles on Smartphones". In IEEE TDSC, to appear in 2014.

G. Russello, M. Conti, B. Crispo, E. Fernandes , Y. Zhauniarovich. "Demonstrating the Effectiveness of MOSES for Separation of Execution Modes". In Proc. of CCS'12, 2012.

Agenda

- Introduction
- Providing Software and Data Isolation on Android
- **Enabling Attestation Service for the Android platform**
- Detecting Repackaged Android Applications
- Analyzing Android Apps in the Presence of Dynamic Class Loading and Reflection
- Summary

TruStore: Motivation

- No possibility to prohibit the installation of uncertified applications in BYOD scenarios
- Large number of third-party markets (Google Play, Yandex.Store, F-Droid, etc.)
- Users trust more to the markets that perform application vetting

TruStore: Problem

Issue 2: *How to support an attestation service on the Android platform maintaining*

- the openness of the ecosystem,
  - all markets should have the same possibility to distribute their apps
  - a user decides to which markets she trusts more
- backward compatibility with already developed apps?
  - app developers should not rewrite their apps according to new rules

TruStore: Idea

- Apple centralized architecture

- **IDEA:** If an application has passed the vetting process of a market, sign it with the market certificate. Ensure on the client-side that only applications signed with the approved certificates can be installed on the device.

- **PROBLEM:** Android has open ecosystem

TruStore: Approach

TruStore: Contributions

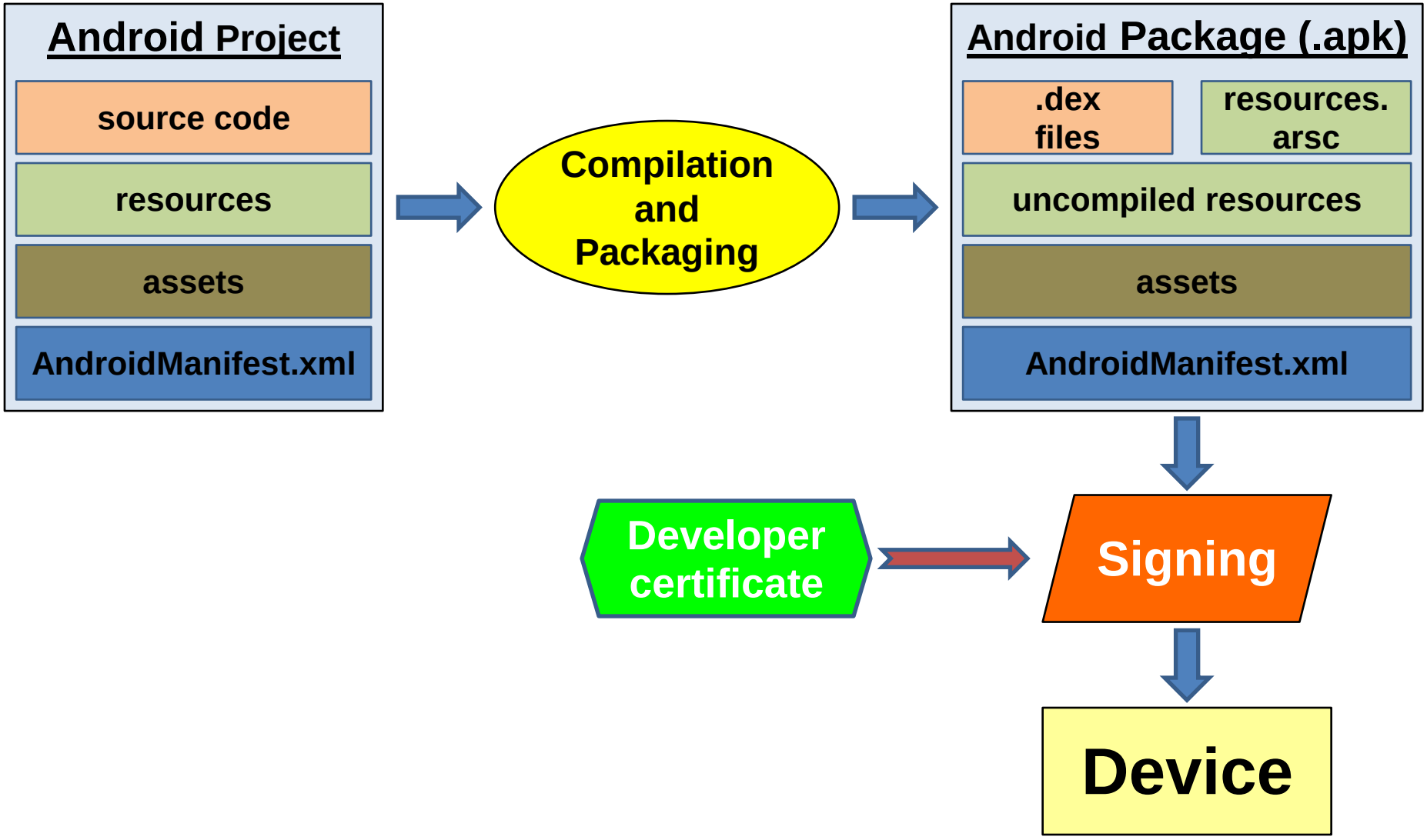
- We proposed an approach to support attestation services for the Android platform:
  - supports the open nature of the Android ecosystem
  - does not change current development, signing and publishing workflow
  - can be applied to already developed applications
  - allows to prohibit installation of uncertified apps in BYOD scenarios

Y. Zhauniarovich, O. Gadyatskaya, B. Crispo. "DEMO: Enabling Trusted Stores for Android". In Proc. of CCS'13, 2013.

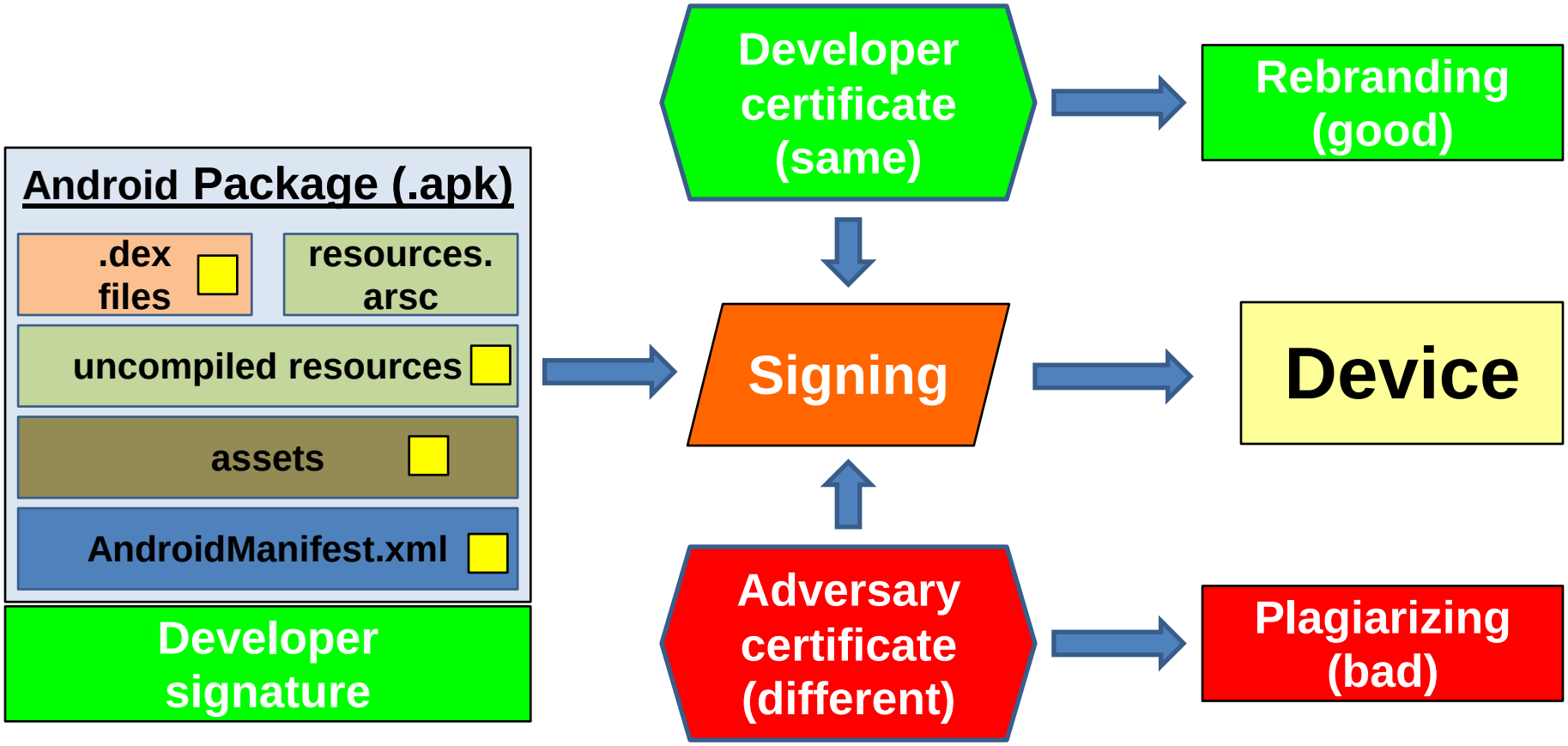
Agenda

- Introduction
- Providing Software and Data Isolation on Android
- Enabling Attestation Service for the Android platform
- **Detecting Repackaged Android Applications**
- Analyzing Android Apps in the Presence of Dynamic Class Loading and Reflection
- Summary

Application Build Process



Repackaging



Motivation

- App repackaging is very easy on Android:
  - Fetch an app → Disassemble → Change → Assemble → Sign with own certificate → Publish
- The code of the application can be easily changed
  - smali/backsmali, AndroGuard, dex2jar, etc.
- **Plagiarizing** is used to:
  - steal advertising revenues (14% of ad revenues)*
  - collect user database (10% of user base)*
  - malware distribution (86% of Android malware samples use this distribution channel)**

*** C.Gibler et al. "Adrob: examining the landscape and impact of Android application plagiarism". In *Proc. of MobiSys '13***

**** Y. Zhou, X. Jiang. "Dissecting Android malware: Characterization and Evolution". In *Proc. of S&P '12***

Problem: Repackaging

Issue 3: *How to detect repackaged Android applications*

- fast
  - 1+ million apps only on Google Play
  - 100+ third-party markets
  - pair-wise comparison
- in effective way?
  - need for a similarity metric to what extent one app is similar to another

FSquaDRA: Idea

- Repackaged apps want to maintain the “look and feel” of the originals
  - Opera Mini fake app: 230 of 234 files are the same

- **IDEA:** compare apps based on the included resource files (**same files → same apps**)

FSquaDRA: Approach

- Obtain hashes of all files inside two apps
- Calculate Jaccard index for the extracted hashes:

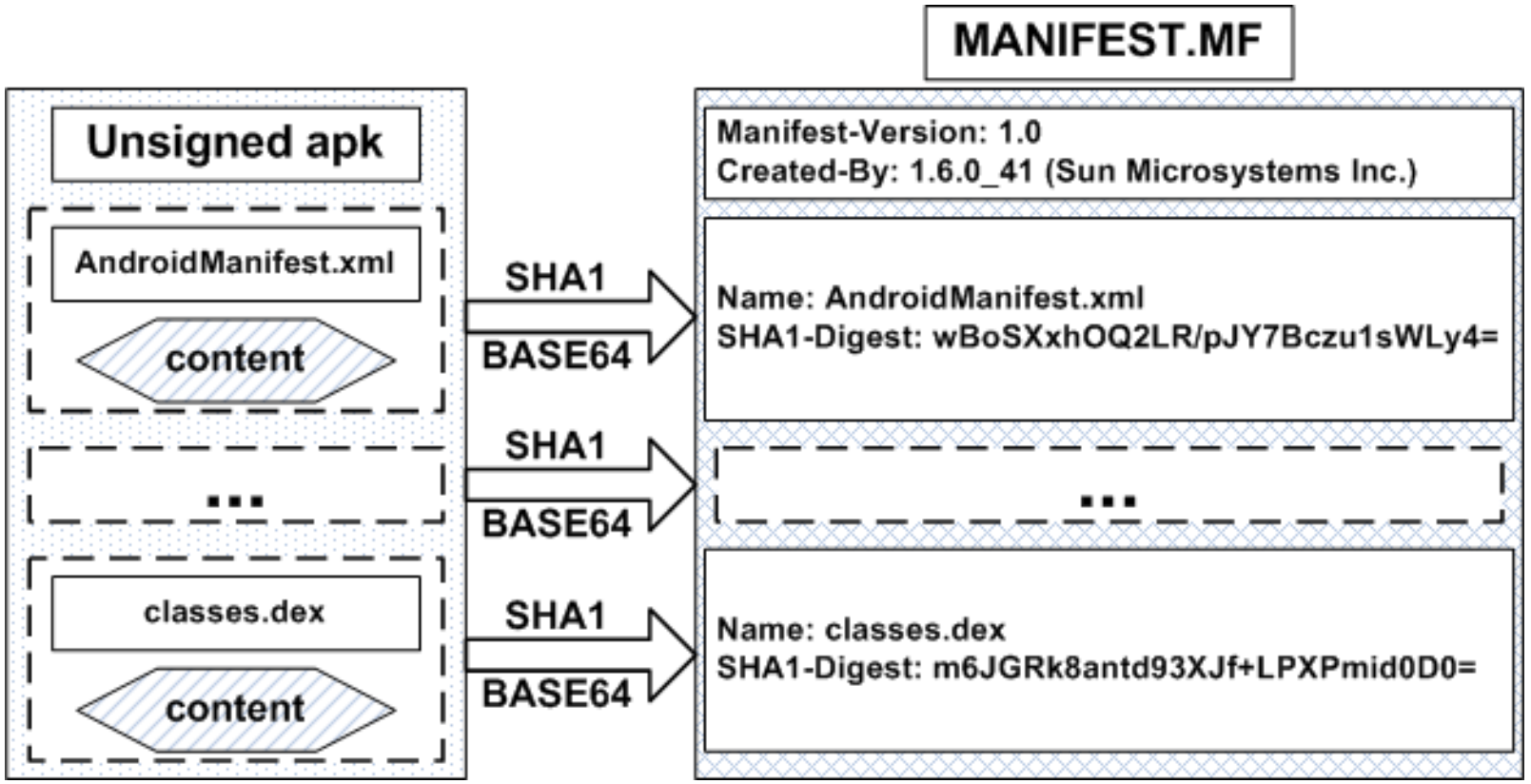
$$jSim(H_k, H_l) = \frac{|H_k \cap H_l|}{|H_k \cup H_l|}$$

H_i – set of hashes of files in apk *i*

- Compare the obtained value with a threshold
- **PROBLEM:** How to compute hashes efficiently?

App Signing Internals

As a part of application signing process SHA1 digest of each file inside apk is calculated



FSquaDRA: Contributions

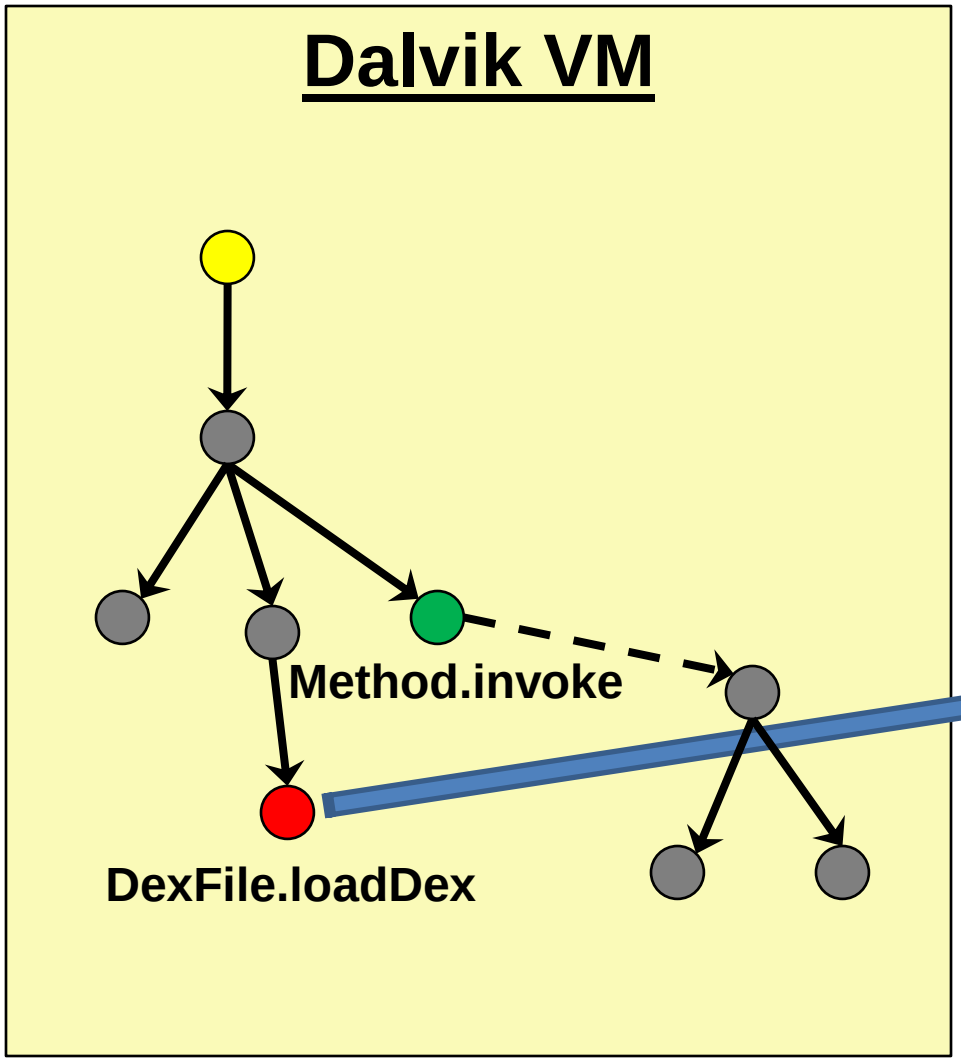
- We are the first who detect repackaged apps based on resource files
- Dataset: 55779 apps collected from 8 markets
- Faster than any known competitor
  - DNADroid by J. Crussell et al. (ESORICS 2012) - **0.012 app pair/sec**
    - PDG subgraph isomorphism
    - Hadoop MapReduce framework with a server and 3 desktops
  - Juxtapp by S. Hanna et al. (DIMVA 2012) - **49.4 app pair/sec**
    - *k*-grams of opcodes → hashing → feature vector → Jaccard distance
    - Intel Xeon CPU (8 cores) , 8GB of RAM
  - Our approach - **6700 app pair/sec**
- Our resource-based similarity score is highly correlated with the code-based similarity score of AndroGuard (**0.79** for plagiarizing, **0.58** for rebranding)

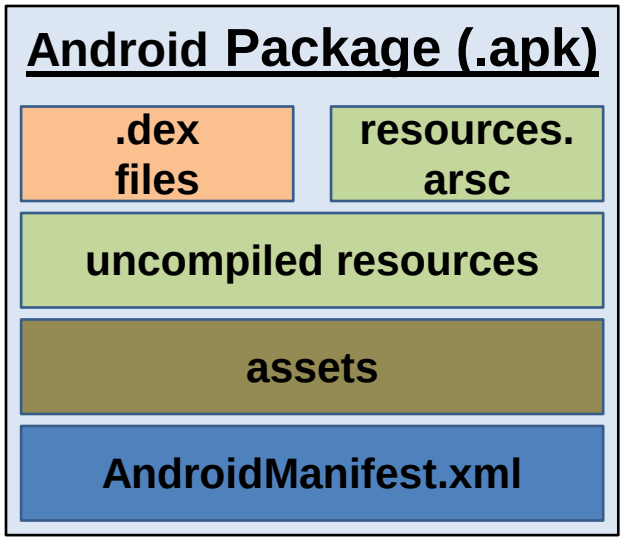
Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, E. Moser. "FSquaDRA: Fast Detection of Repackaged Applications". In Proc. of DBSec'14, to appear in 2014.

Agenda

- Introduction
- Providing Software and Data Isolation on Android
- Enabling Attestation Service for the Android platform
- Detecting Repackaged Android Applications
- **Analyzing Android Apps in the Presence of Dynamic Class Loading and Reflection**
- Summary

Dynamic Code Updates





code files (jar, dex,…)

1. Dynamic Class Loading (DCL)
2. Reflection

Motivation

- In Android, code loaded dynamically has the same privileges as original
- Static analyzers cannot fully inspect an app in the presence of dynamic code update features (AndroGuard, Stowaway, PScout etc.)
- Heavily used by malware to conceal malicious behavior
- Dynamic code update features are used:
  - In legitimate applications
    - Google Play: 19% - DCL, 88% - reflection
    - Third-party markets: 6% - DCL, 74% - reflection
  - In malicious applications
    - Malware dataset: 20% - DCL, 81% - reflection

Problem: Dynamic Code Updates

Issue 4: *How to analyze Android apps in the presence of*

- reflection,
  - detect the name of the called function/class
- dynamic class loading?
  - download and analyze the loaded code

StaDynA: Idea

- Apps with Dynamic Code Update features expose their dynamic behavior **at runtime**

- **IDEA:** combine static and dynamic analysis techniques to detect and explore dynamic code update features

- **Method Call Graph (MCG)** is a directed graph showing the calling relationships between methods in a computer program

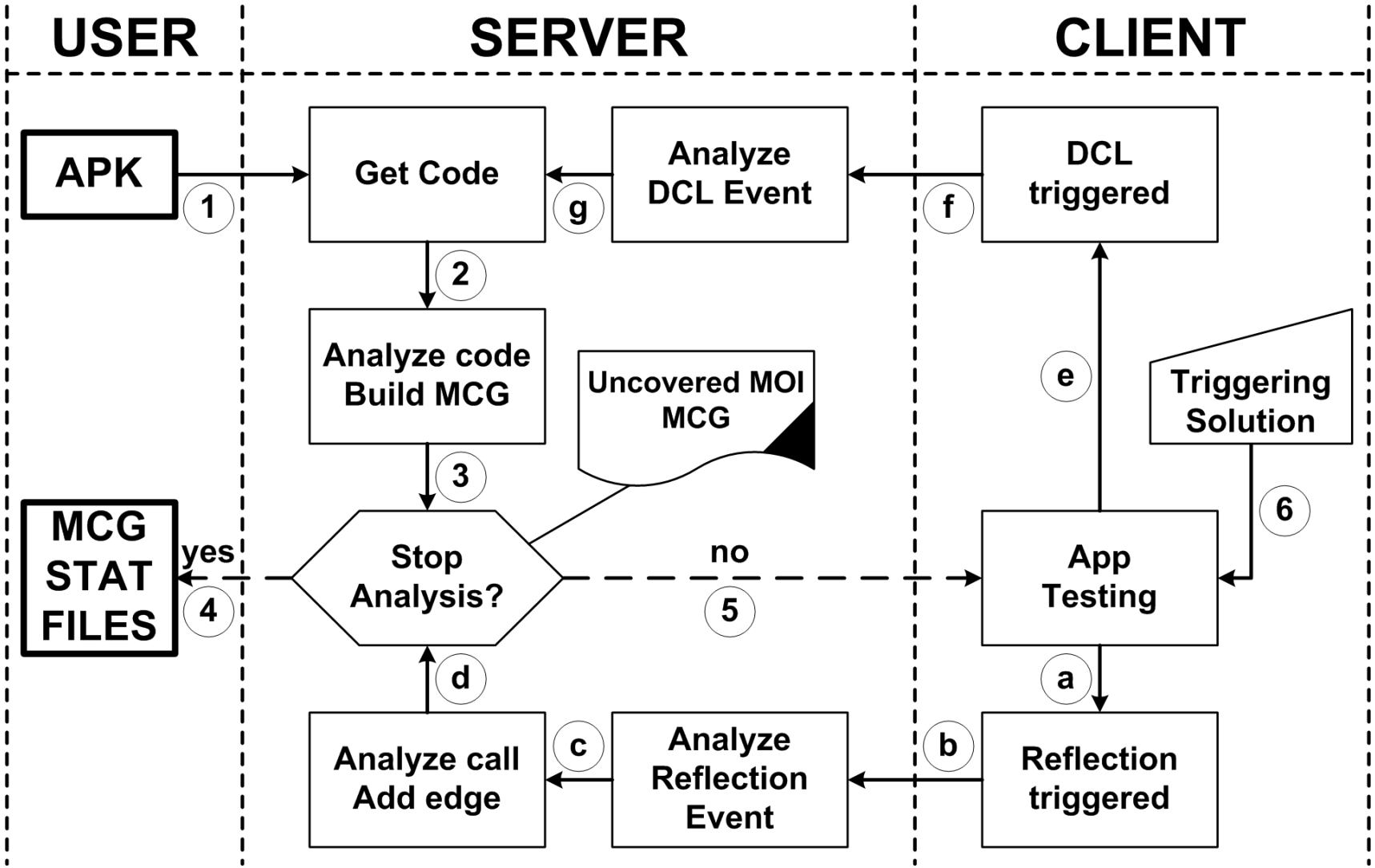
StaDynA: Approach

- Find API calls responsible for reflection and DCL at static time (we call the methods calling these API functions as **Methods of Interest (MOI)**)

Class	Method	Prototype
Dynamic class loading		
$Ldalvik/system/PathClassLoader;$	$< init >$	.
$Ldalvik/system/DexClassLoader;$	$< init >$	.
$Ldalvik/system/DexFile;$	$< init >$	.
$Ldalvik/system/DexFile;$	$loadDex$	.
Class instance creation through reflection		
$Ljava/lang/Class;$	$newInstance$	.
$Ljava/lang/reflect/Constructor;$	$newInstance$	.
Method invocation through reflection		
$Ljava/lang/reflect/Method;$	$invoke$	.

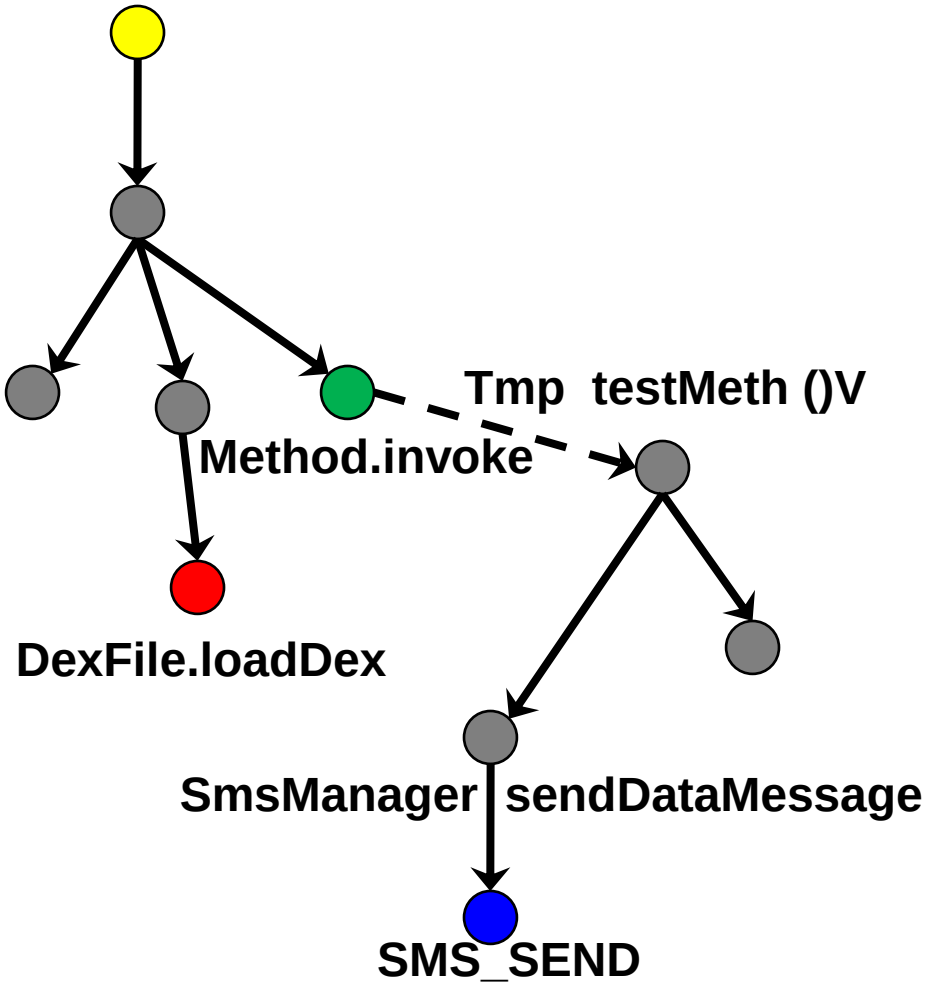
- Analyze their behavior at runtime

StaDynA: Overview



StaDynA: Features

- Stores and analyzes the code loaded dynamically
- Builds MCG of the app including the information obtained at runtime
- Discovers at runtime the qualifiers of the methods/constructors called through reflection
- Discovers suspicious behavior patterns



StaDynA: Contributions

- Dynamic code updates is a serious problem for Android
  - the code loaded dynamically has the same privileges as the original application
- We proposed an approach that facilitates the analysis of apps in the presence of reflection and DCL
  - discovers at runtime the qualifiers of the methods/constructors called through reflection
  - stores and analyzes code loaded dynamically
  - builds MCG of the app including the information obtained at runtime
  - discovers suspicious behavior patterns

Summary

- A policy-based framework for enforcing software isolation of applications and data on the Android platform
- A mechanism to enable attestation services in the Android ecosystem respecting its openness
- An approach to detect repackaged Android applications
- A tool facilitating the analysis of Android applications in the presence of dynamic code update features
- All proposed solutions have implementations

Papers

1. Y. Zhauniarovich, O. Gadyatskaya, B. Crispo, F. La Spina, E. Moser. *"FSquaDRA: Fast Detection of Repackaged Applications"*. In Proc. of DBSec'14, to appear in 2014.
2. Y. Zhauniarovich, G. Russello, M. Conti, B. Crispo, E. Fernandes. *"MOSES: Supporting and Enforcing Security Profiles on Smartphones"*. In IEEE TDSC, to appear in 2014.
3. O. Gadyatskaya, F. Massacci, Y. Zhauniarovich. *"Emerging Mobile Platforms: Firefox OS and Tizen"*. In IEEE Computer, to appear in 2014.
4. Y. Zhauniarovich, O. Gadyatskaya, B. Crispo. *"DEMO: Enabling Trusted Stores for Android"*. In Proc. of CCS'13, 2013.
5. G. Russello, M. Conti, B. Crispo, E. Fernandes , Y. Zhauniarovich. *"Demonstrating the Effectiveness of MOSES for Separation of Execution Modes"*. In Proc. of CCS'12, 2012.
6. M. Conti, B. Crispo, E. Fernandes, Y. Zhauniarovich. *"CRePE: A System for Enforcing Fine-Grained Context-Related Policies on Android"*. In IEEE TIFS, 2012.
7. G. Russello, B. Crispo, E. Fernandes and Y. Zhauniarovich. *"YAASE: Yet another Android security extension"*. In Proc. PASSAT/SocialCom, 2011.

THANK YOU!

Questions…