Mandatory Access Control for the Android Dalvik VM

ESOS'13

Aline Bousquet, Jérémy Briffaut, Laurent Clevy,
Christian Toinard, Benjamin Venelle

June 25, 2013

Android

Android:

- 96 new threats detected in Q4 2012 (F-Secure)
- 238 new threats detected in 2012 (F-Secure), i.e. 79% of the threats detected on mobiles.

SEAndroid: existing protection at the system level (processes, files, but not an application)

Java:

- The JVM provides new vectors of attacks (Kaspersky Labs)
- Vulnerabilities of the JVM affecting Facebook and Twitter

SEDalvik: our protection for the VM level, using the same concepts as SEAndroid

SEDalvik: Mandatory Access Control system

MAC = Mandatory Access Control

- Can guarantee security properties, as opposed to Discretionary Access Control

SEDalvik = Security Enhanced Dalvik

- Is a MAC model for Android's applications
- Is also a MAC implementation for Dalvik
- Controls interactions inside Dalvik

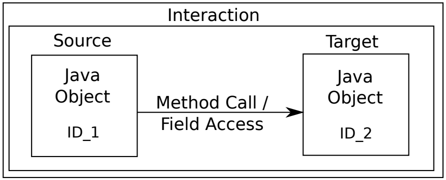
Why use MAC inside the Dalvik VM?

- Control of the interactions between all the Java objects
- No modifications of the applications are needed: self-organization

Requirements

SEDalvik proposes the following MAC approach:

- Type Enforcement to control interactions between source and target objects:

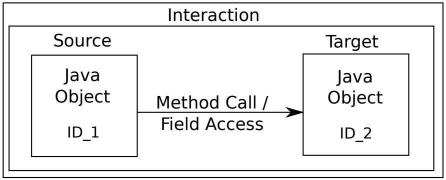


- A reference monitor to intercept *all* interactions
- To be able to label each object: class signature and instance ID
- Fine grain access control:
  - (ObjectA, MethodA) –{is allowed to call}→ (ObjectB, MethodB)
  - (ObjectA, MethodA) –{is allowed to access}→ (ObjectB, FieldB)

Requirements

SEDalvik proposes the following MAC approach:

- Type Enforcement to control interactions between source and target objects:

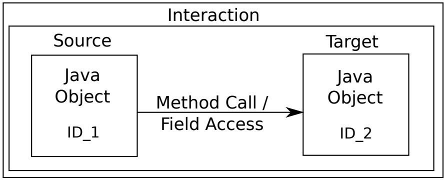


- A reference monitor to intercept *all* interactions
- To be able to label each object: class signature and instance ID
- Fine grain access control:
    - (ObjectA, MethodA) –{is allowed to call}→ (ObjectB, MethodB)
    - (ObjectA, MethodA) –{is allowed to access}→ (ObjectB, FieldB)

Requirements

SEDalvik proposes the following MAC approach:

- Type Enforcement to control interactions between source and target objects:

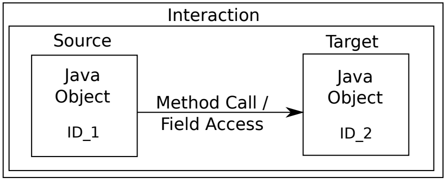


- A reference monitor to intercept *all* interactions
- To be able to label each object: class signature and instance ID
- Fine grain access control:
  - (ObjectA, MethodA) $-\{$is allowed to call$\}\rightarrow$ (ObjectB, MethodB)
  - (ObjectA, MethodA) $-\{$is allowed to access$\}\rightarrow$ (ObjectB, FieldB)

Requirements

SEDalvik proposes the following MAC approach:

- Type Enforcement to control interactions between source and target objects:



- A reference monitor to intercept *all* interactions
- To be able to label each object: class signature and instance ID
- Fine grain access control:
    - (ObjectA, MethodA) $-\{$is allowed to call$\}\rightarrow$ (ObjectB, MethodB)
    - (ObjectA, MethodA) $-\{$is allowed to access$\}\rightarrow$ (ObjectB, FieldB)

Policy

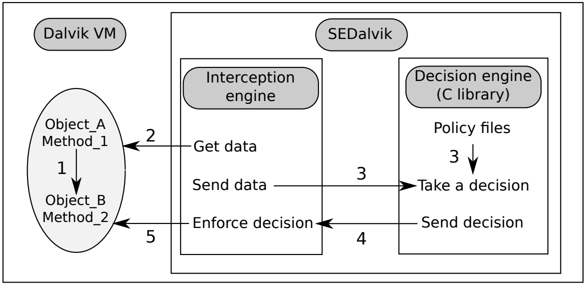
SEDalvik's MAC policy is composed of:

1. Files for labeling the source and target objects
2. Files including MAC rules : Source $\rightarrow$ Permission $\rightarrow$ Target

Policy generation:

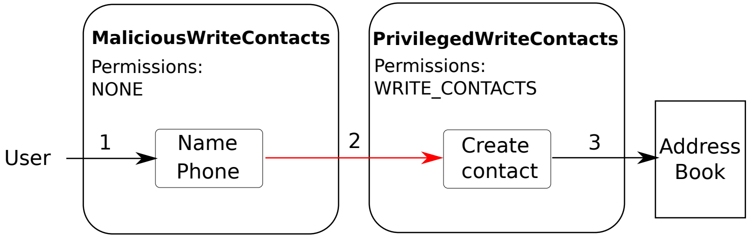
- Policy can be generated when an application is installed
- Policy can be provided with an application

Architecture



1. A method is intercepted
2. Necessary data (about objects and methods) is retrieved
3. Decision engine computes a decision based on data & policy
   - If a matching rule is found, the interaction can continue
   - Else the interaction is stopped
4. Decision is sent back to the interception engine
5. Decision is enforced

A privilege escalation attack



1. The unprivileged malicious app wants to create a new contact
2. The malicious app asks the vulnerable app to create the contact
3. The contact is created by the privileged app

$\longrightarrow$ A privileged escalation has been performed

Policy extract

Security labels:

```
Lpkg/privileged/PrivilegedWriteContactActivity;
   privilegedwritecontactactivity_j
Lpkg/malicious/MaliciousWriteContactActivity;
   maliciouswritecontactactivity_j
```

Rules allowing interactions:

```
allow android_widget_button_j android_content_intent_j
     from onClick invoke (init)
# allow android_widget_button_j maliciouswritecontactactivity_j
     from onClick invoke startService
allow object_j android_content_intent_j
     from createFromParcel invoke (init)
```

The policy for this usecase has about 600 labels and 10200 rules.

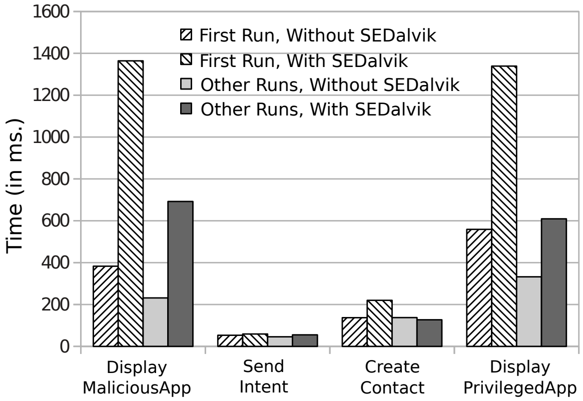
Results

```
[traceid=62472;stamp=27241435;pid=592] type=allow
 scontext=object_j tcontext=android_content_intent_j
 { onClick invoke (init) }
 sinstance=5328 tinstance=7472

[traceid=62507;stamp=27251525;pid=592] type=deny
 scontext=object_j tcontext=maliciouswritecontactactivity_j
 { onClick invoke startService }
 sinstance=5328 tinstance=1736
```

- This solution blocks all messages sent by the malicious application $\Rightarrow$ Too general
- Need a more precise way to stop messages:
  - Block the message during the transmission
  - Use a reference monitor that can detect sequences of interactions

Benchmark



- Important overhead for the graphical part ⇒ possible improvement
- Small overhead for critical parts (sending intent, actions on personal data)

Conclusion

SEDalvik

- A MAC implementation for Dalvik
- Tested on Android emulator and device

Self-organizing:

- Self-configuration
- No modification of the applications needed

Future works:

- SEDalvik+SEAndroid for in-depth control of the interactions (OS, Java application),
- SEDalvik+SEAndroid+PIGA to guarantee advanced security properties by controlling direct/indirect flows

Questions?