6.858 Quiz 2 Review

Android Security

Haogang Chen

Nov 24, 2014

Security layers

Layer	Role
Reference Monitor	**Mandatory Access Control (MAC) for RPC:** enforce access control policy for **shared resources**
Java VM	**Memory safety:** (neither required nor trusted)
Linux Kernel	**Isolation:** apps run with different UIDs. (principals are apps, as opposed to users)

Basic architecture

- Apps are composed of **components**
- 4 types of components
  - **Activity:** UI, only one active at a time
  - **Service:** background processing, RPC server
  - **Content provider:** provides read/write RPC
  - **Broadcast receiver:** listen for notifications

Intent: RPC primitive

- Has 4 fields
  - **Component:** target
  - **Action:** opcode
  - **Data:** arguments
  - **Category:** for filtering
- The **reference monitor** checks sender's permission labels upon message delivery.

Permission labels

- **Application** defines permissions as string labels
  - `<permission name="com.android.phone.DIALPERM"></…>`
- Application asks for permissions in its **manifest**
  - `<use-permission name="com.android.phone.DIALPERM"></…>`
- Application assigns a **type** for each permission

Permission types

Type	Reference Monitor's grant policy
Normal	Silent check, no user interaction required. (no security guarantee for any serious use…)
Dangerous	Ask the user upon app installation. (useful when you want to interact with others' apps)
Signature	Silently grant to apps signed by the same developer. (useful when you only talk to your own apps)

Implicit and broadcast intent

- Implicit intent
  - Omit the "*target*" field; let Android figure out the receiver
  - Receivers declare interested *actions* and *categories* using **intent filters**
- Broadcast intent
  - Problem: how to ensure only *someone* gets the broadcast?
  - Solution: protected broadcast (not MAC)
    - Request for a permission when broadcasting `sendBroadcast(intent, "perm.FRIEND_NEAR")`

Summary

- **Permissions:** "*Who are allowed talk to me?*"
- **Permission types:** "*How to grant permissions to an app?*"
- **Intent filters:** "*What (implicit intent) do I want to see?*"
- **Protected broadcast:** "*Who are allowed to see my (broadcast) intent?*"

6.858 Quiz 2 Review

TaintDroid

Haogang Chen

Nov 24, 2014

Motivation

- Limitation of the reference manager
  - "*What resource can I access?*"
    - No guarantee on ***how the data is being used***.
      - E.g., a photo editor can silently upload your photo stream to its server
- TaintDroid: track information flow for sensitive data

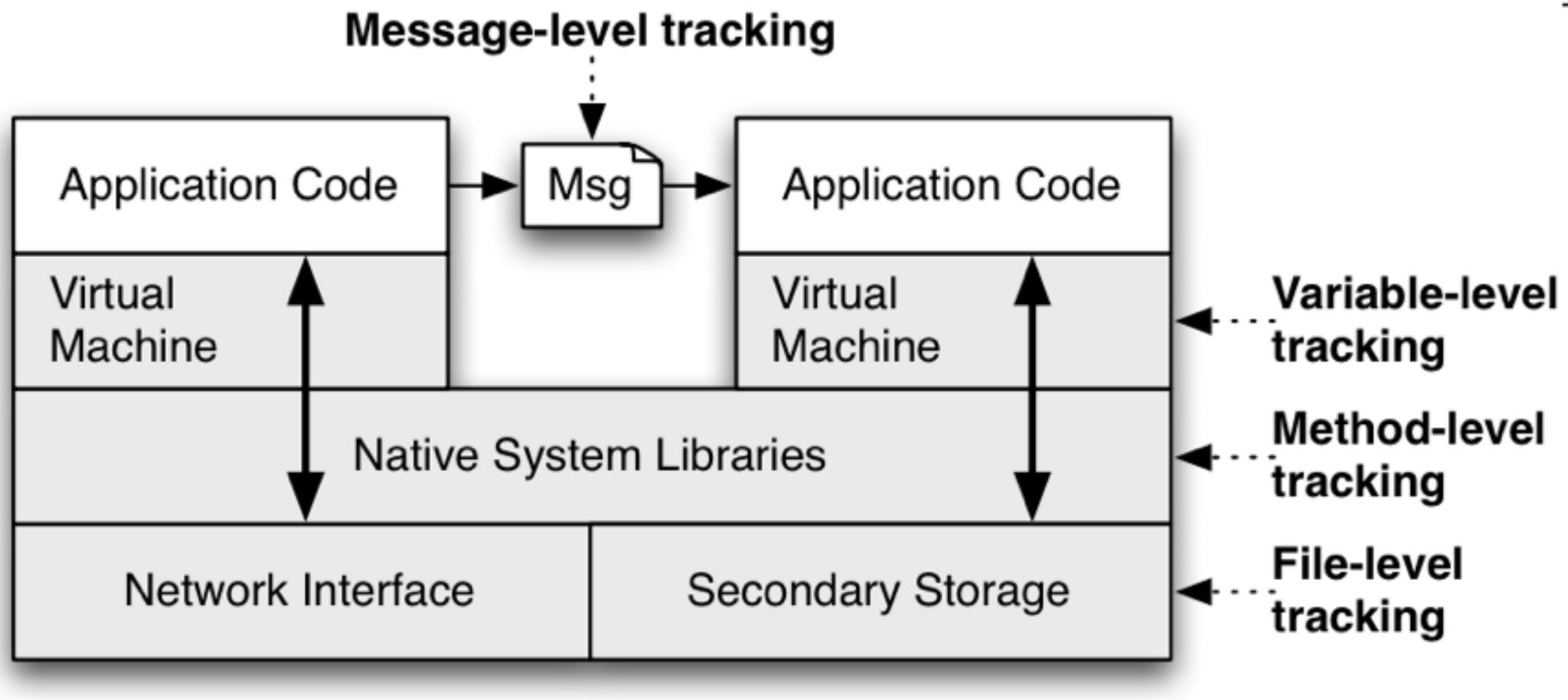
Taint tracking basics

- **Source:** origin of sensitive data
  - E.g., photos, contacts, GPS coordinates
- **Sink:** undesired destination
  - E.g., network interface, TCB boundary
- **Propagation**: how information flows from source to sink
  - E.g., variable copy, arithmetic operations, indexing, message passing, system calls, file read/write.

Approach

- Attach a “**tag**” for each piece of sensitive data
- Propagate the tag together with the data
- Challenges
  - Fine-grained tracking can be extremely slow
  - Coarse-grained tracking introduces false positives
  - Key contribution: trade-offs between performance and accuracy

TaintDroid: multi-level tracking



Component	Trusted?	Action
System app.	Y	**Taint source:** annotate data from sensitive content provider (e.g. camera app)
User app.	N	User apps runs inside Java VMs. They are untrusted and unmodified
Java VM	Y	**Variable-level tracking:** store and propagate taint tags in shadow memory for every variable
RPC library	Y	**Message-level tracking:** propagate taint tags when serializing/deserializing messages
System library	Y	**Method-level tracking:** annotate how taints propagate among arguments and return values
Storage library	Y	**File-level tracking:** attach and propagate taint tags in file's extended attribute.
Network library	Y	**Taint Sink:** annotate the interface, and report any tagged data that reaches the sink

Limitation of taint tracking

- Cannot capture control-flow dependencies

```
// “dirty” is tainted
int laundry(int dirty) {
    int clean;
    if (dirty == 0)
        clean = 0
    else if (dirty == 1)
        clean = 1
    else if (dirty == 2)
        clean = 2
    else …
    return clean;
}
```

6.858 Computer Systems Security

Fall 2014