PINPOINT: Efficient & Effective Resource Isolation for Mobile Security & Privacy

Paul Ratazzi, Ashok Bommisetti, Nian Ji, and Prof. Wenliang (Kevin) Du

Department of Electrical Engineering & Computer Science
Syracuse University, Syracuse, New York
Motivating Examples

- User likes 3rd party keyboard, but wants to ensure it will not leak sensitive information from certain apps
  - Currently, there is no way to list only trusted input methods for certain sensitive apps
- User wants some apps to use accurate sensor data, others to have less accurate data, and the rest to have no access
  - Currently, sensor access does not require permission, and all apps have same access
- User wants location-enabled coupon app to know regional location to get relevant coupons, but not coarse (10s of meters) or fine (~1 meter) locations
  - Currently, only options are no location, coarse location, or fine location
- User wants some location-enabled apps to access location data, and others to have no access
  - On/off setting is currently platform-wide
- User wants to play game, but does not want it to leak sensitive info. via requested READ_PHONE_STATE permission
  - Although need for permission may be legitimate, there is currently no way to allow legitimate use while making leakage impossible
Existing Isolation Approaches

• Cells
  • Leverages Linux Namespaces to allow multiple Android Virtual Phones (VP) on a single kernel
  • Hardware and kernel are shared among independent VPs

• AirBag
  • Leverages Linux Namespaces to allow multiple decoupled app runtimes
  • Hardware, kernel, and native userspace are shared among independent runtimes
  • Condroid improved by restoring binder communications and increasing efficiency

Advantage: powerful general-purpose solution with many applications
Existing Isolation Approaches

• Kernel-level isolation breaks many assumptions of Android’s open platform design

• Significant effort is required to fix things → 2nd order complexity

• Overhead and inconvenience to end-users

Disadvantage: cost and inconvenience may be too high for many simple security and privacy scenarios
## Some Key Namespace Traits

<table>
<thead>
<tr>
<th>Namespace Trait</th>
<th>Value to Android Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fine-grained isolation of specific resources</td>
<td>Tailored isolation environment for each application; few side effects</td>
</tr>
<tr>
<td>High efficiency</td>
<td>Negligible performance impact; design simplicity</td>
</tr>
<tr>
<td>Share-by-default</td>
<td>Preserve open system design; avoid breaking things unrelated to the isolated resource</td>
</tr>
<tr>
<td>Small footprint (files, memory)</td>
<td>Little impact on performance &amp; resources; OTA updates</td>
</tr>
</tbody>
</table>
Our Idea: PINPOINT

→ Employ a Linux Namespace-like approach to Android Framework resources

• Virtualize and isolate only what’s necessary to meet stated security goal(s)
  → Everything else is shared as Android intended
  → Minimize or eliminate side-effects

• Provide isolation “building blocks” that can be used to create containers
About “-visors”

• **Hypervisor (type I native)**
  • Runs on “bare metal”
  • Authority over guest OS(s)

• **Supervisor (a/k/a kernel)**
  • Inside OS
  • Authority over userspace(s)

• **NEW: Hypovisor**
  • Inside userspace
  • Authority over resource(s)
## PINPOINT Methodology

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Define/collect security goal(s)</td>
<td>Protect IMEI from app A</td>
</tr>
<tr>
<td>2</td>
<td>Identify relevant resource(s)</td>
<td>iphonesubinfo and phone system services (5.1)</td>
</tr>
<tr>
<td>3</td>
<td>Identify point(s) of resource access / capability dispatch -&gt; implement hypervisor(s) here</td>
<td>servicemanager</td>
</tr>
<tr>
<td>3a</td>
<td>Security analysis</td>
<td>Prevent inter-app passing of system service binder tokens (modified SEAndroid hook)</td>
</tr>
<tr>
<td>4</td>
<td>Identify and address dependency(ies)</td>
<td>com.android.phone and ProxyController (service startup)</td>
</tr>
</tbody>
</table>
Android System Service Basics

App Runtime
getSystemService()

ServiceManager (a/k/a ContextManager)

System Server
- Service A
- Service B
...
Case Study: System Services

Policy
/etc/ns/nspolicy
<uid, service, namespace>

System Service Hypervisor
servicemanager

Application
getSystemService()

Request (service, uid)

Service A
Service A'
Service A"
Service B
Service B'
Service B"

Handle returned
System Service Hypovisor: servicemanager

```c
uint32_t do_find_service(struct binder_state *bs, const uint16_t *s, size_t len, uid_t uid, pid_t spid)
```

1. Check nspolicy for entry matching caller’s uid and service requested
2. On match, modify incoming request per nspolicy
3. Pass modified request to find_svc() for handle lookup

Example: iphonesubinfo → iphonesubinfo_1 for uid 0010068
Hypovisor Security Analysis

• Fundamental question: “can the hypovisor be: 1) tricked or 2) bypassed?”

  1) Our modifications do not change how service capabilities are dispatched, so any problems here are also a problem with stock Android

  • Subject identified by uid from binder driver (trusted)
  • Policy file restricted
  • Service name values validated

→ servicemanager cannot be tricked
Hypervisor Security Analysis

• Fundamental question: “can the hypervisor be: 1) tricked or 2) bypassed?”

2) For most normal services, servicemanager acts as an open capability dispatch service
   • Once obtained, apps are free to pass capabilities held to other apps
   • App-to-app transfer of system service capabilities bypasses the hypervisor

→ Blocked via modified security_binder_transfer_binder() SEAndroid hook to disallow transfer of u:r:system_server:s0 binders among u:r:untrusted_app:s0
→ task_struct of binder_ref/binder_node contains owner’s SELinux security identifier (SID)
Four Sample Applications

• Security goal: prevent untrusted apps from obtaining accurate location information
  • **LocationManagerService**

• Security goal: prevent critical apps from leaking information through untrusted input methods
  • **InputMethodManagerService**

• Security goal: prevent untrusted apps from obtaining sensitive subscriber information
  • **IPhoneSubInfo**

• Security goal: prevent untrusted apps from obtaining accurate sensor data to steal data, eavesdrop, or track movement/location
  • **SensorService**
InputMethodManagerService

- Goal: protect app from untrusted input method
- One additional namespace
  - Stock input methods only
- Dependency: WindowManagerService
  - Modified to update all `input_method` services with current activity
InputMethodManagerService

Global IME

- ns_policy: <no entry>
- Requests: input_service; receives input_service

Alternate IME

- ns_policy: 10084 input_service_1
- Requests: input_service; receives input_service_1

Unmodified non-critical app (uid 10083) with 3rd party IME available

Unmodified banking app (uid 10084) with only stock IMEs available
Performance Impacts

Quadrant 2.1.1 File I/O score vs. 
# namespaces

~1.6% loss per namespace

system_server process
memory footprint vs. # 
namespaces

~0.6% increase per namespace
Limitations

• Our approach does not provide security domain isolation
  • Apps can pass high level information among namespaces

• Alternate services must be configured and running even if not used
  • Additional system_server memory footprint

• Alternate services must be defined at build time
Future Directions

• Formalize methodology (esp. security analysis)
• Implement other hypovisors

• Provide sample device images
• Open source
Thank You

Questions?

ratazzi@ieee.org