I-ARM-Droid

A Rewriting Framework for In-App Reference Monitors for Android Applications

Benjamin Davis, Ben Sanders, Armen Khodaverdian, Hao Chen
University of California, Davis

Mobile Security Technologies 2012

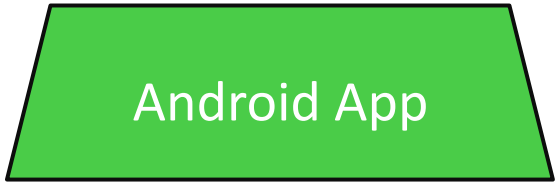
Why In-App Reference Monitors?

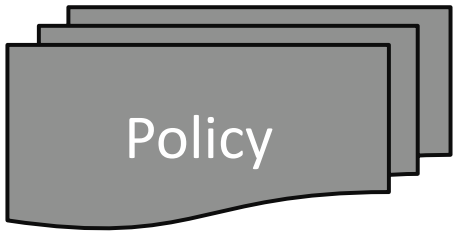
- Current Android limitations
  - Users have limited insight into app behavior
  - Platform provides very limited control over apps
- I-ARM-Droid: reference monitors for Android apps
  - Fine-grained control over app behavior
  - Practical and flexible for a variety of policies

Why Not Platform Modifications?

- Deployment challenges
  - Proprietary binaries for device hardware
  - Requires rooting phone, voiding warranty, etc.
- Inflexible
  - One reference monitor applied to all apps
  - Reference monitor capabilities are pre-defined by platform

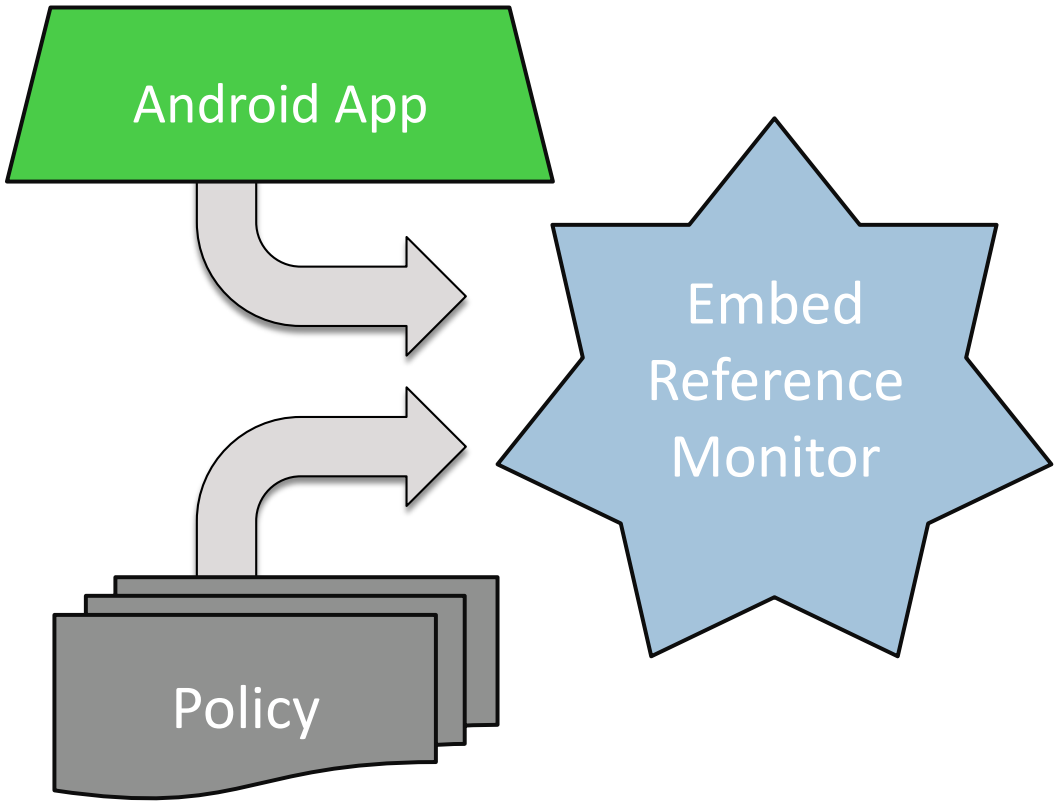
I-ARM: In-App Reference Monitors





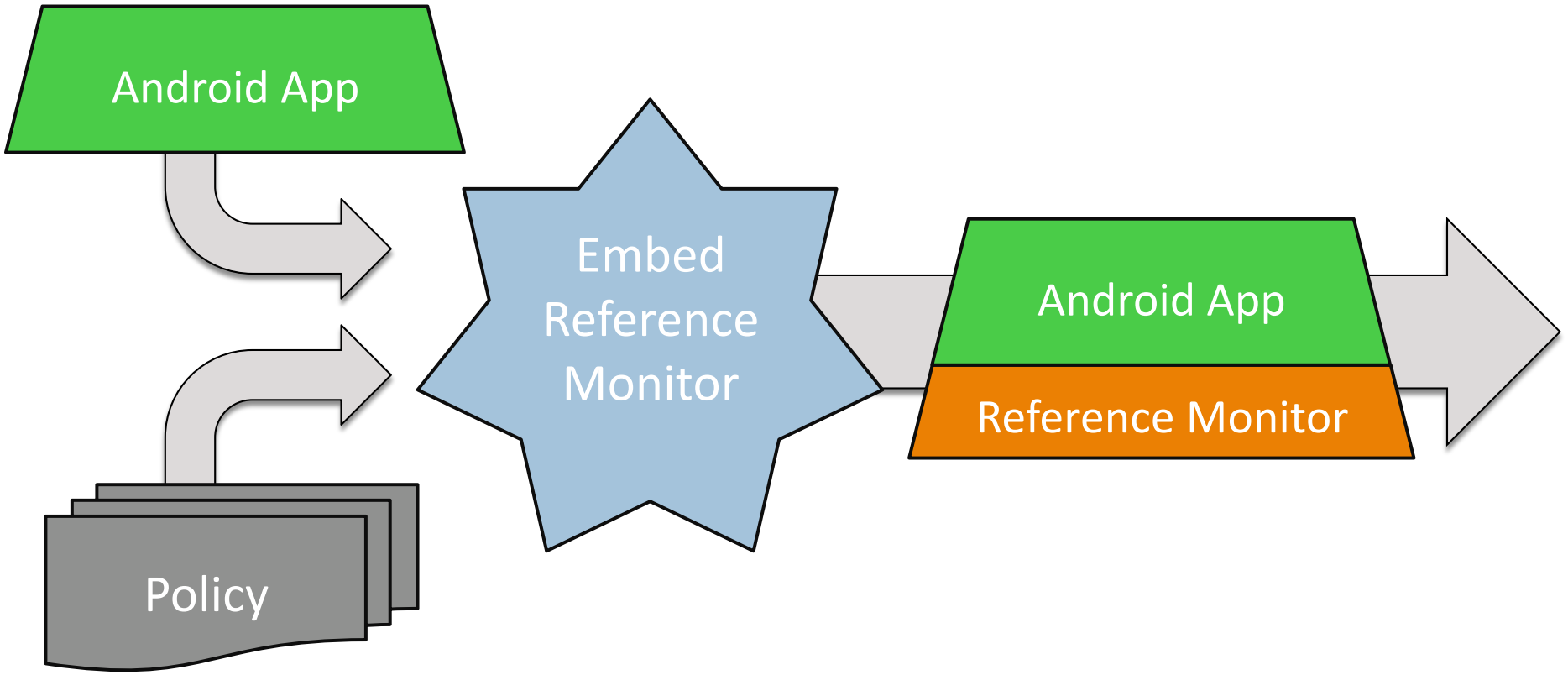
User Input

I-ARM: In-App Reference Monitors



User Input **I-ARM**

I-ARM: In-App Reference Monitors



User Input **I-ARM** **Deploy to unmodified Android Platform**

I-ARM Policies

- Design: method call interposition
- Policies include
  - Target method signatures

    java.io.URL.openStream()
  - Custom handler behavior

```
iarm.URL.openStream(URL obj) {
    if (call site in ad library) { return obj.openStream(); }
    else { Log.d("blocked openStream"); throw IOException(); }
}
```

Rewriting Android Apps

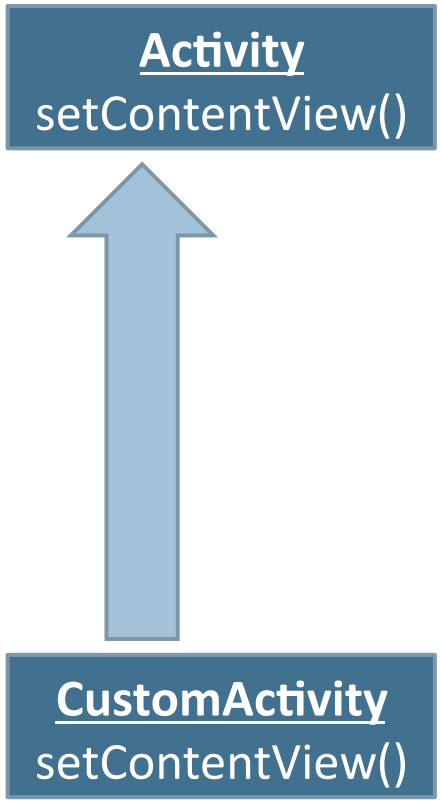
- Leverage structure of Dalvik VM bytecode
- Insert custom handlers for each target method
- Identify target method invocations
- Rewrite app to invoke custom handlers instead

Custom Method Handlers

- Handlers: a static method for each target method
- Rewrite instructions based on method type
  - Static methods
  - Instance methods
  - Constructors

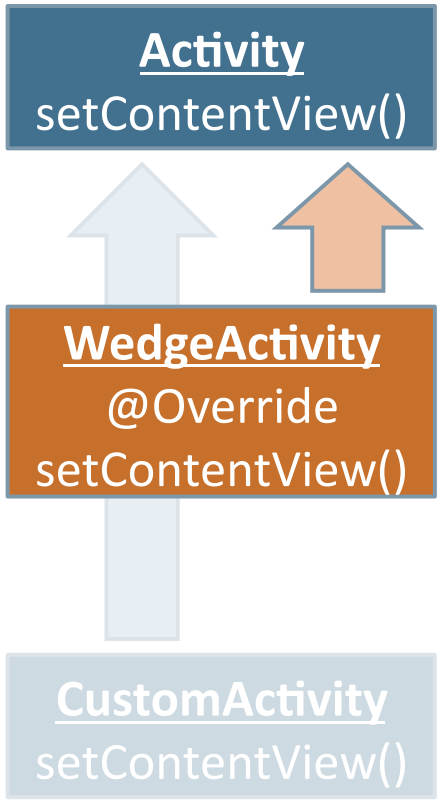
Handling Virtual Method Invocations

- Identify classes with non-final target methods



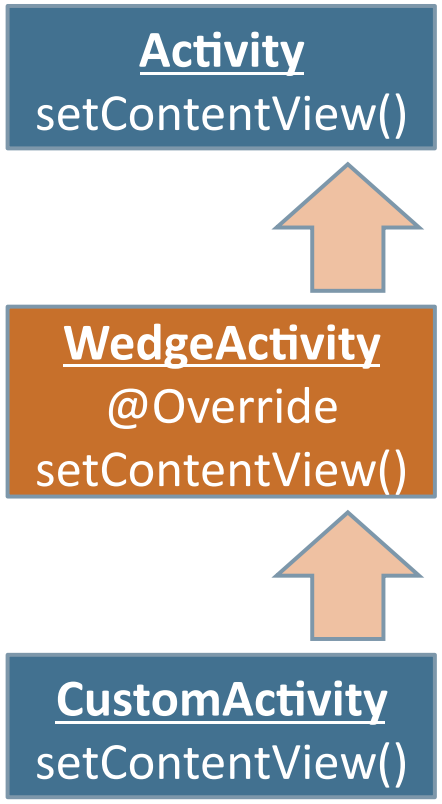
Handling Virtual Method Invocations

- Identify classes with non-final target methods
- Create "wedge" class for each:
  - Extend target method's class
  - Handlers: override all target methods



Handling Virtual Method Invocations

- Identify classes with non-final target methods
- Create "wedge" class for each:
  - Extend target method's class
  - Handlers: override all target methods
- Inject wedge in app class hierarchy
  - Developer class now extends wedge class
- Intercept all virtual method invocations



Discussion: Completeness

- Policy completeness
  - Rely on other tools (e.g. Stowaway, CCS '11)
- Rewriting completeness
  - Reflection
    - We detect *calls* to reflection API statically – insert handler to perform dynamic inspection
  - Native code
    - Requires platform-dependent rewriting techniques
    - Uncommon (< 10% of apps, [Zhou et al. NDSS 2012])
    - We detect existence and invocation

Implementation and Evaluation

- Compatibility & Functionality
  - Applied policies to 30 top apps from Android Market
- No per-app manual effort required for rewriting
- Performance: handlers have low overhead
  - Less than 0.2 microseconds on HTC Thunderbolt
- Size: minimal impact
  - 90 target methods increase code size by <2% (median)

Conclusion

- In-app reference monitors for Android
- Identify and interpose on target method calls
- Flexible, practical and efficient design