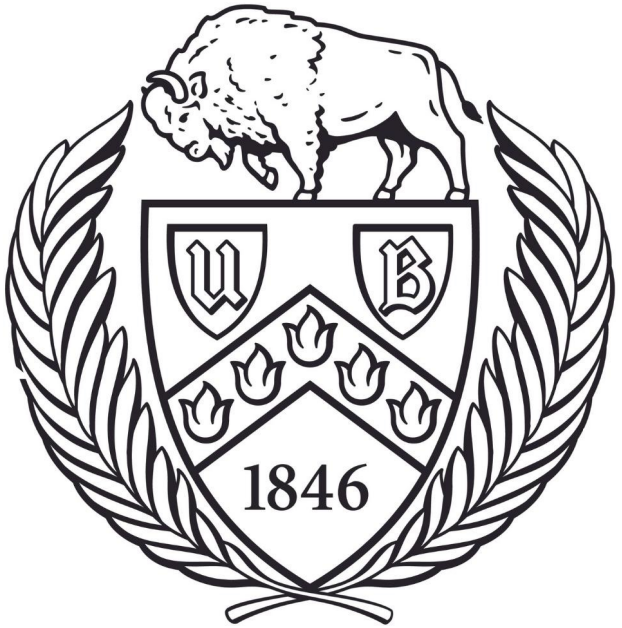
TAINTDROID

Information Flow Tracking System for Realtime Privacy Monitoring



CSE 711: Malware Detection for Android: Static and Dynamic Analysis
Instructor: Prof. Lucasz Ziarek
Presented by Jayesh Suryavanshi

Where does the problem lie?

- Today's smartphone operating systems frequently fail to provide users with visibility into how third-party applications collect and share their private data.
- Apps perform data exfiltration!
- Monitoring the flow of privacy-sensitive data provides valuable input for smartphone users and security service firms seeking to identify misbehaving applications.
- Applications combine data from remote cloud services with information from local sensors. Might have legitimate reasons for doing so but users don't have an idea how their data is being used.
- Sensitive data like ***IMEI, Location, Contacts, Phone number, SIM card serial number*** is being relayed somewhere without users' consent.
- Mobile-phone operating systems currently provide only **coarse-grained controls** for regulating whether an application can access private information but provide little insight into how private information is actually used.

Taint (verb) /tānt/ : contaminate

- In context of this paper, a ‘Taint’ is privacy sensitive data like Location, IMEI, Phone Number and other personally identifiable information (PII) of a user at risk

TaintDroid

- **TaintDroid** is an extension to the Android mobile-phone platform that tracks the flow of privacy-sensitive data through third-party applications. TaintDroid assumes that downloaded, third-party applications are not trusted, and monitors in **real time** how these applications access and manipulate users' personal data.

- **Efficient**, **system-wide dynamic** taint tracking and analysis system capable of simultaneously tracking **multiple sources** of sensitive data.

- Leverages Android's **virtualized execution environment**.

- Primary goals are to detect when sensitive data leaves the system via untrusted applications and to facilitate analysis of applications by phone users or external security services.

- Unlike other solutions, TaintDroid doesn't send data anywhere. Safe to install.

Challenges for TaintDroid

- Many smartphone applications are **closed source** and **obfuscated**, making **static analysis infeasible**. Even if source code is available (via decompilation), **runtime events** and **configuration** often dictate information use.
- Monitoring network disclosure of privacy sensitive information on smartphones presents a lot of challenges like,

1. *Smartphones have **resource constraints***
2. *Third party applications are entrusted with several types of **privacy sensitive information***
3. *Context-based privacy-sensitive information is **dynamic** and can be difficult*
4. *Applications can **share information** amongst themselves – i.e. IPC*

- However, **real time monitoring** can handle these environment-specific dependencies.

Dynamic Taint Analysis

- **Taint Source** is the sensitive Information identified
- **Taint Marking** indicates the information type i.e. GPS, IMEI and other PII
- **Taint Sink** is where the affected data is identified

Similar to ‘Source and Sink’ concept in **directed weighted graphs**

How does the labeled data impact other data?

How is the tracking done?

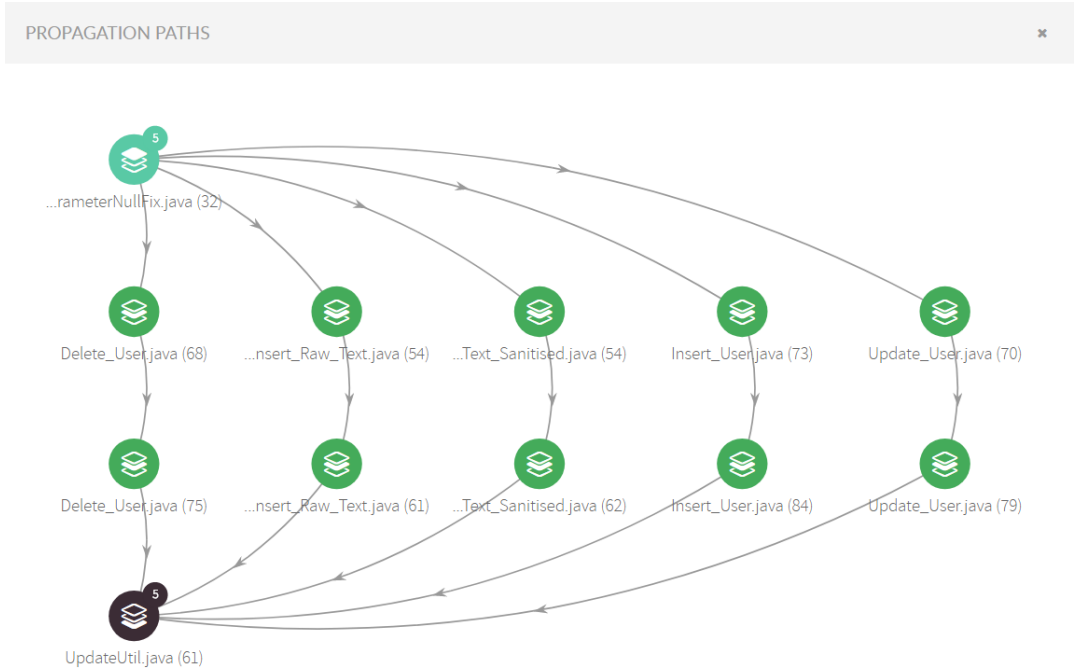


Image reference: https://www.kiuwan.com/docs/display/K5/Understanding+Data-Flow+Vulnerabilities#UnderstandingData-FlowVulnerabilities-TaintedFlowAnalysis

Multilevel approach for Taint Tracking

- Existing approaches rely on **instruction-level dynamic taint analysis** using **whole-system emulation**

  - **Drawback 1 : High performance overhead**

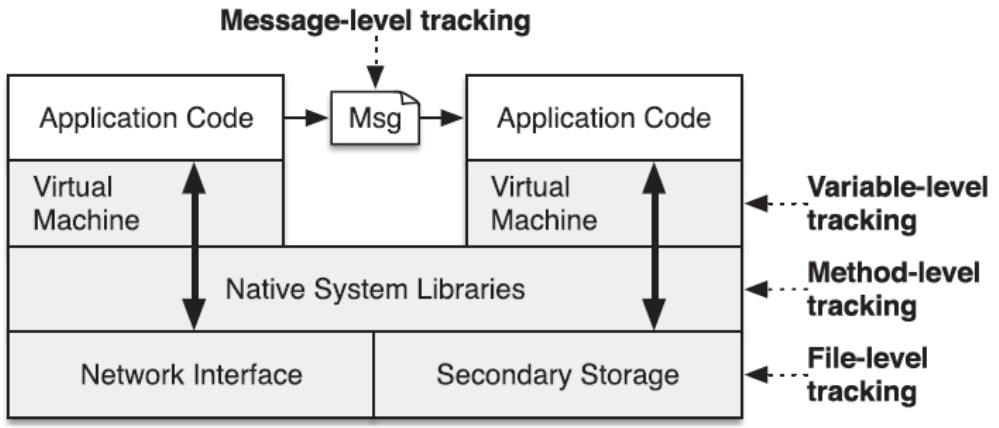
  → 2-20 times slowdown + emulation slowdown

  → Hence, not suitable for real-time analysis

  - **Drawback 2 : Taint Explosion**

  → Stack pointer, base pointer falsely tainted.

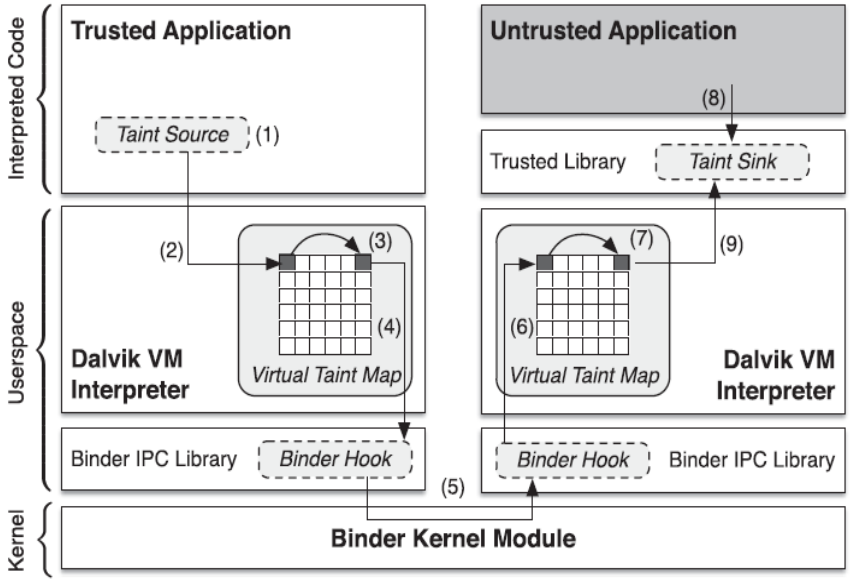
  → Occurs due to unnecessary changes in the taint marking process



Granularity achieved using ...

- **Variable** level tracking - maintain taint markings only for data and not code
- **Message** level tracking - Tracking taint on messages instead of data within messages reduces IPC overhead, extends analysis system-wide
- **Method** level tracking – we run native code without instrumentation & patch the taint propagation on return.
- **File** level tracking - ensure persistent information conservatively retains its taint markings.
- Novel, Efficient, System-wide , multiple marking, multiple granularities

TaintDroid Architecture



- **Dalvik VM Interpreter** - DEX is register based unlike Java bytecode which is stack based; responsible for handling computation; contains JIT from Android Éclair (v2.3)
- **Native Methods-** for performance optimization; third party library support; written in C/C++
  - Internal VM methods: interpreter specific instructions and API's
  - JNI methods – managed code -> bytecode –interacts→ native code (C/C++)
- **Binder IPC**– serialize active and standard data objects using ***Parcels.*** Binder Kernel model passes parcels

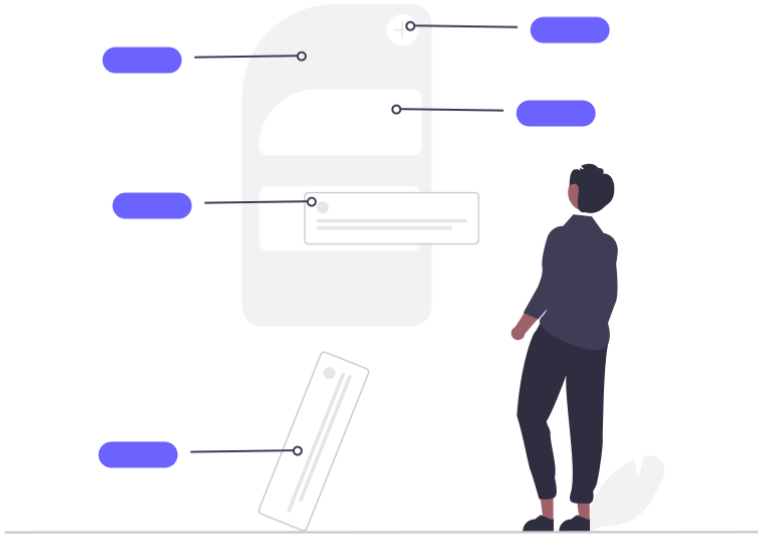
This architecture requires:

- **Taint tag** storage
  - Method Local Variables.
  - Method Arguments.
  - Class Fields
  - Arrays

```
out0
out0 taint tag
out1
out1 taint tag
```

- **Interpreted code** taint propagation

move-op v_A v_B	$v_A \leftarrow v_B$	$\tau(v_A) \leftarrow \tau(v_B)$	Set v_A taint to v_B taint
move-op-R v_A	$v_A \leftarrow R$	$\tau(v_A) \leftarrow \tau(R)$	Set v_A taint to return taint
return-op v_A	$R \leftarrow v_A$	$\tau(R) \leftarrow \tau(v_A)$	Set return taint ($\emptyset$ if void)
move-op-E v_A	$v_A \leftarrow E$	$\tau(v_A) \leftarrow \tau(E)$	Set v_A taint to exception taint
throw-op v_A	$E \leftarrow v_A$	$\tau(E) \leftarrow \tau(v_A)$	Set exception taint



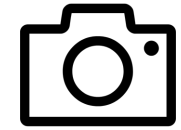
- **Native code** taint propagation
  - Internal VM methods: interpreter specific instructions and API's
  - JNI methods – managed code -> bytecode –interacts→ native code (C/C++)

- **IPC taint** propagation, and
  - Message-level IPC – Parcels
  - Fine-grained IPC tracking – eliminate False positives – optimisations performed to reduce performance overhead
- **Secondary storage** taint propagation
  - One taint tag per file to avoid tag loss – tracking taint tags at finer granularity

Privacy Hook Placement:

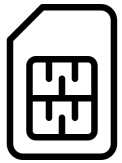
- **Low Bandwidth Sensors**
  - A variety of privacy-sensitive information types are acquired through low-bandwidth sensors.
  - Such information often changes frequently and is simultaneously used by multiple applications.
  - Ex. **Location** (GPS), **accelerometer**, **gyroscope** etc.

- **High Bandwidth Sensors**
  - Smartphone OS may share sensor information via large data buffers, files, or both.
  - Due to flexible APIs, we placed hooks for both data buffer and file tainting for tracking microphone and camera information.
  - Ex. **Camera**, **Microphone**



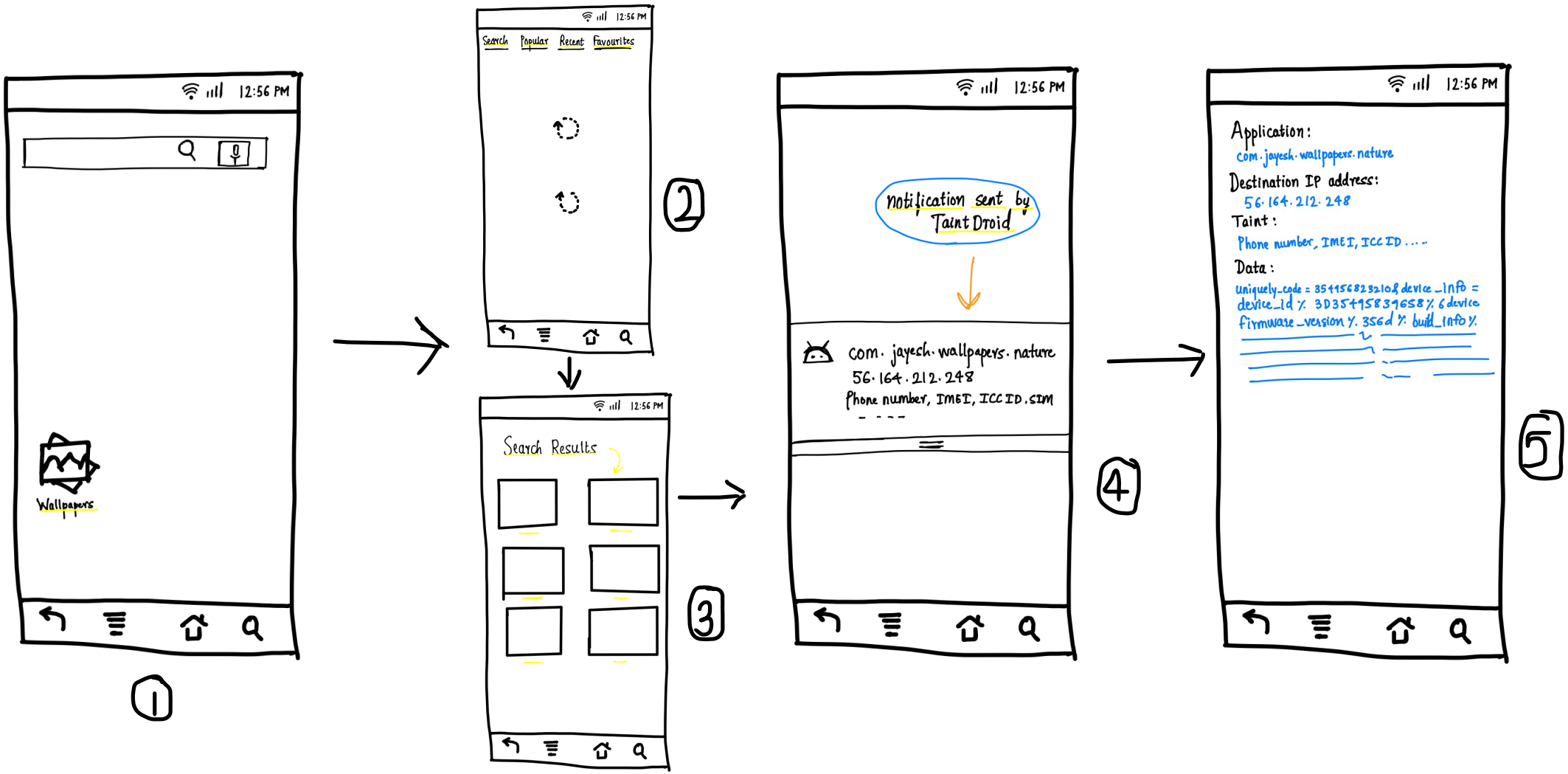
- **Information Databases**
  - **Programmatically taint query response** strings based on the **content provider** authority.
  - Rules are added for bookmarks, SMS, and MMS.

- **Device Identifiers**
  - Information that **uniquely identifies** the phone/user
  - Ex. **IMEI**, IMSI, **Phone Numbers**, **SIM** card identifiers



- **Network Taint Sink**
  - Identify tainted information over network interface

How TaintDroid works?



Performance Evaluation

	Android	TaintDroid
App Load Time	79 ms	91 ms
Address Book (create)	275 ms	309 ms
Address Book (read)	89 ms	98 ms
Phone Call	171 ms	197 ms
Take Picture	2913 ms	2938 ms

Fig. Macrobenchmarks

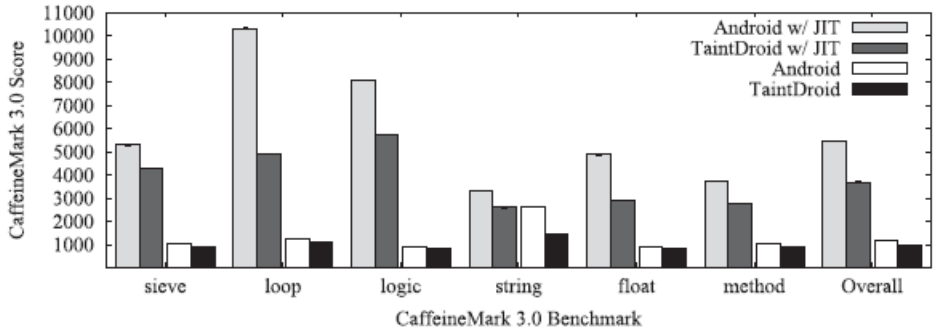


Fig. Java overhead Microbenchmarks

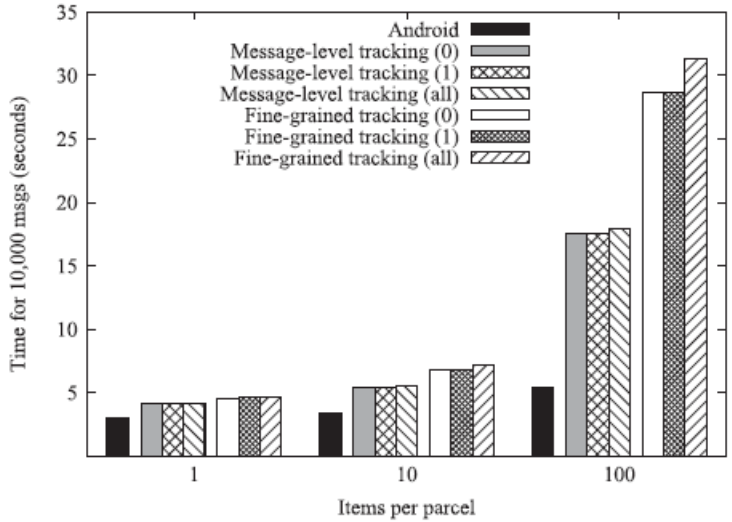
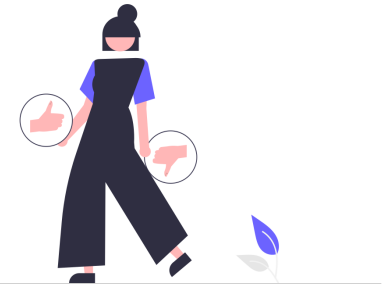


Fig. IPC overhead Microbenchmarks

Conclusion

- Taintdroid provides an **efficient**, **system-wide** information flow tracking tool that can simultaneously track **multiple sources** of sensitive data
- Achieves efficiency, 32% overhead on CPU-bound microbenchmark
- Integrates **four granularities** of taint propagation (**variable-level**, **message-level**, **method-level**, and **file-level**)
- **2010** - Used for the analysis of 30 popular apps, around 20 apps exhibit suspicious handling of sensitive data, around 15/30 expose user location to remote advertising servers.
- **2012** – Similar fraction of apps were doing the same.

Pros



- **Efficient**, **system-wide** information flow tracking
- Can simultaneously track **multiple sources** of sensitive data
- Achieves efficiency, **32% overhead** on CPU-bound microbenchmark
- **Fine-grain control** over taint propagation
- Notifies user in **real-time** about any malicious activity
- Although tested on popular apps, testing **can be scaled** to large number of apps
- Actively used by the systems security community
- Does **not collect data** unlike other approaches; **Safe** to install.

Cons

- **Approach Limitations** – tracks only **explicit** dataflows; not implicit where apps control flow indirectly transfers sensitive information. Implicit dataflow tracking a long standing problem.

- **Implementation Limitations** – cannot track taint tags on ***DirectBuffer*** – data stored in opaque native data structures.

- **Taint Source limitations** – **False positives** when there are configuration identifiers in tracked information. Ex. IMSI numeric string consists MCC, MNC and MSIN all tainted together.

- **Security Evaluation limitations** – trades **precision** for **performance** in array, IPC, file taint tracking due to **coarse grained tracking** in these areas. But even for fine-grained tracking systems ground truth is absent for applications studied.