Where Does My Sensitive Data Go?

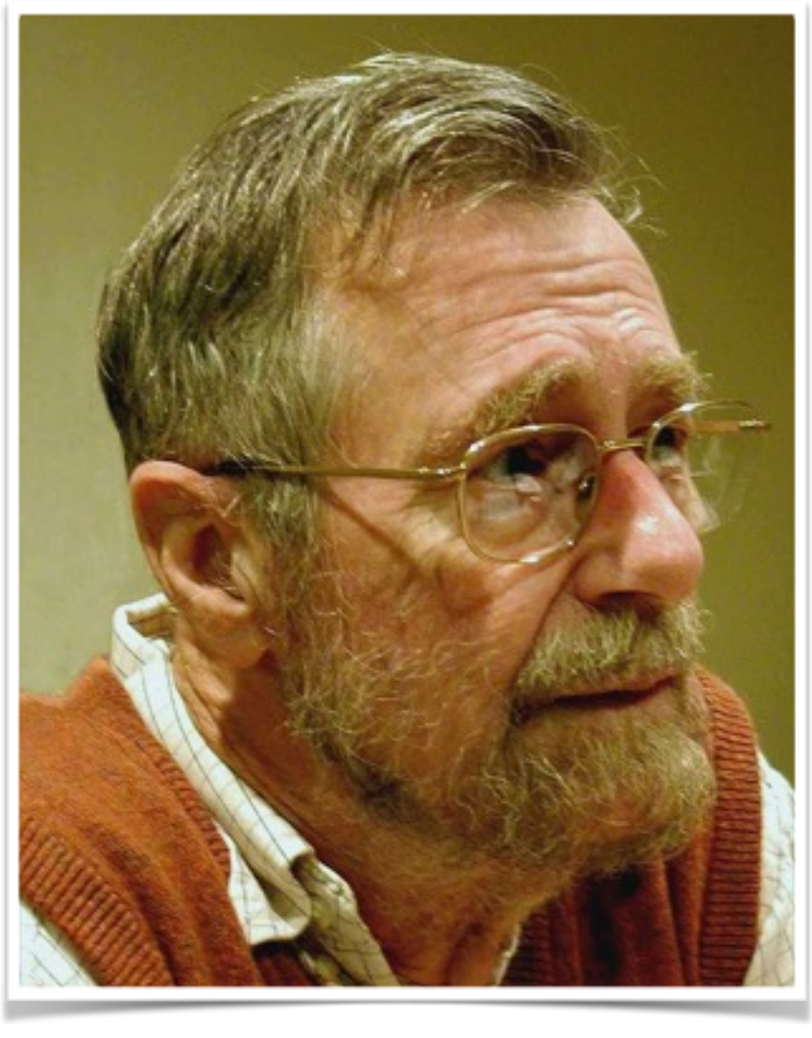
Mining Apps for Abnormal Information Flow

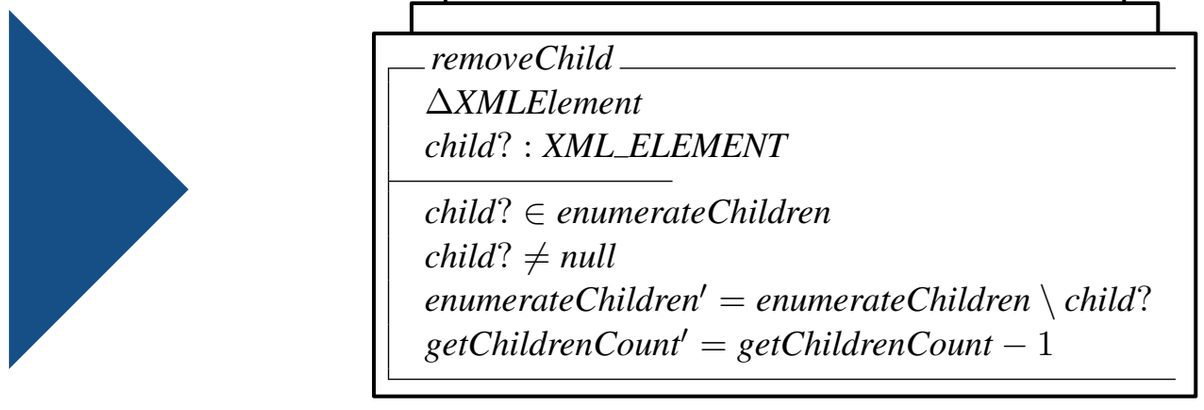
Andreas Zeller
Saarland University, Saarbrücken, Germany

Joint work with Vitalii Avdiienko, Konstantin Kuznetsov, Alessandra Gorla, Steven Arzt, Siegfried Rasthofer, and Eric Bodden

erc Google

Specifying Correctness





Normality

Mining Normality

removeChild

$\Delta XMLElement$

$child? : XML\_ELEMENT$

$child? \in enumerateChildren$

$child? \neq null$

$enumerateChildren' = enumerateChildren \setminus child?$

$getChildrenCount' = getChildrenCount - 1$

Outliers

App Mining

- For 100,000s of apps:
- Gather *descriptions*
- Gather *metadata*
- Gather *execution features*
- Find what is *common* and what is *uncommon*

App Features

App Binary	App Metadata	Store Metadata
APIs	Permissions	Description
Code	Resources	Reviews
Executions	Frameworks	Downloads

Analyzing App Code

APIs

- Easy to extract (grep)
- Easy to process
- Initial classification of program behavior

Code

- Comes in binary form (Dalvik / ARM / both)
- Hard to analyze statically (scale, components)
- Code may be *adverse* (malware)

Executions

- Need *test generators* (on binaries) to assess
- Instrument binary and/or environment
- Code may be *adverse* (malware)

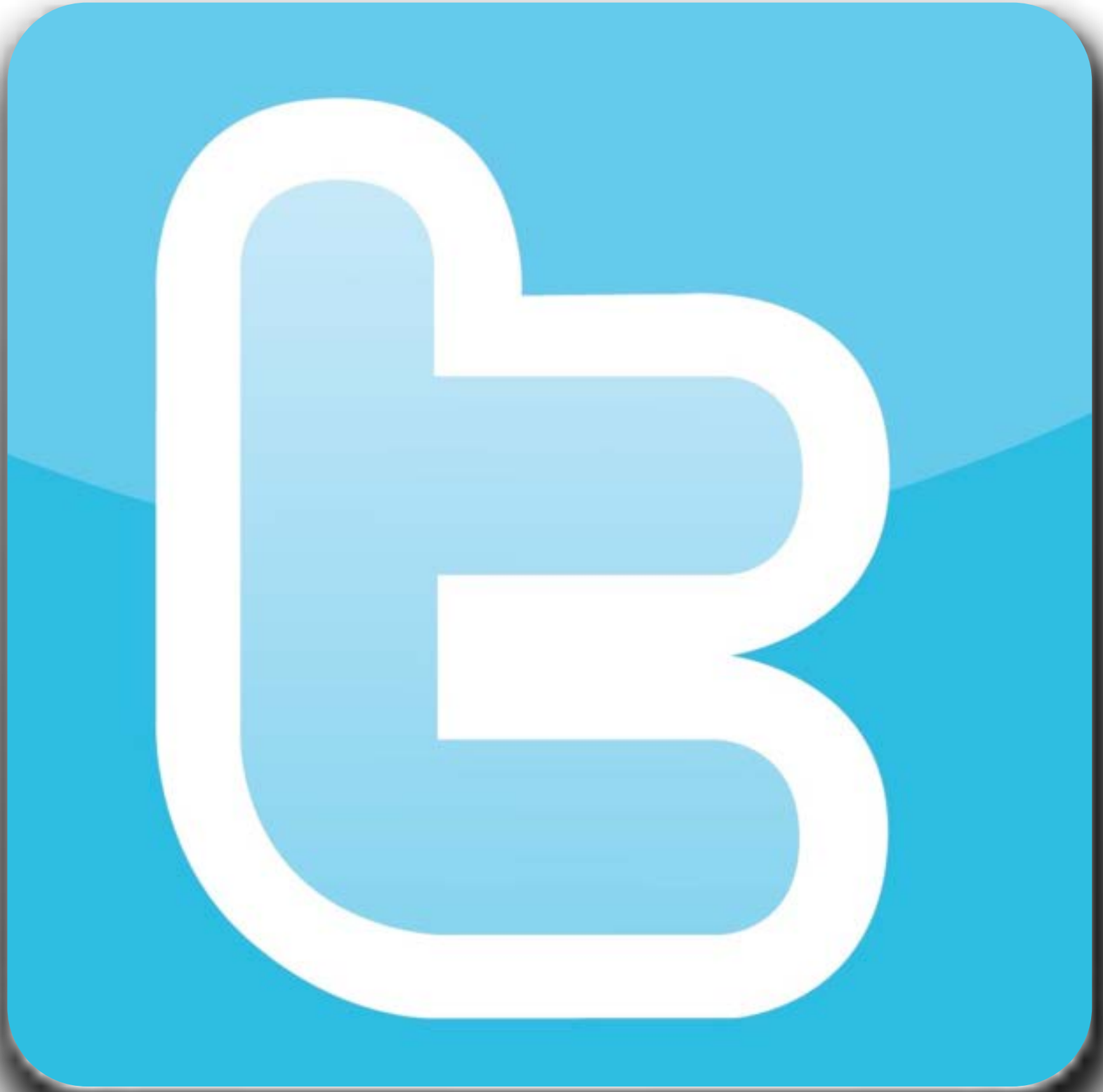
Static Taint Analysis

```
void onCreate() {
  TelephonyManager mgr = (TelephonyManager)
    this.getSystemService(TELEPHONY_SERVICE);
  String devId = mgr.getDeviceId();
  String a = devId;
  String str = prefix(a);
  SmsManager sms = SmsManager.getDefault();
  sms.sendTextMessage("+1 234", null, str, null, null);
}
String prefix(String s) {
  return "DeviceId: " + s;
}
```

- Use **FlowDroid** for analysis (object-, flow-, and context-sensitive)
- Use **SuSI** list of sensitive sources and sinks

Sensitive Data Flow

- Downloaded *2,940 apps* (top 100 per store category)
- Extracted *sensitive data flows*
- Two months of server time



Twitter
Sensitive Data Flow

AccountManager.get() → ContentResolver.setSyncAutomatically()

AccountManager.get() → AccountManager.addOnAccountsUpdatedListener()

AccountManager.get() → Activity.setResult()

AccountManager.get() → Log.w()

AccountManager.getAccountsByType() → ContentResolver.setSyncAutomatically()

AccountManager.getAccountsByType() → Activity.setResult()

AccountManager.getAccountsByType() → Log.w()

Uri.getQueryParameter() → Activity.startActivity()

Uri.getQueryParameter() → Activity.setResult()

Uri.getQueryParameter() → Activity.startActivityForResult()

Uri.getQueryParameter() → Log.d()

Uri.getQueryParameter() → Log.v()

Uri.getQueryParameter() → Log.w()

SQLiteDatabase.query() → Log.d()

SQLiteOpenHelper.getReadableDatabase() → Log.d()

24 hours
64 cores
768 GB RAM

Danti604
Sensitive Data Flow

TelephonyManager.getSubscriberId() → URL.openConnection()

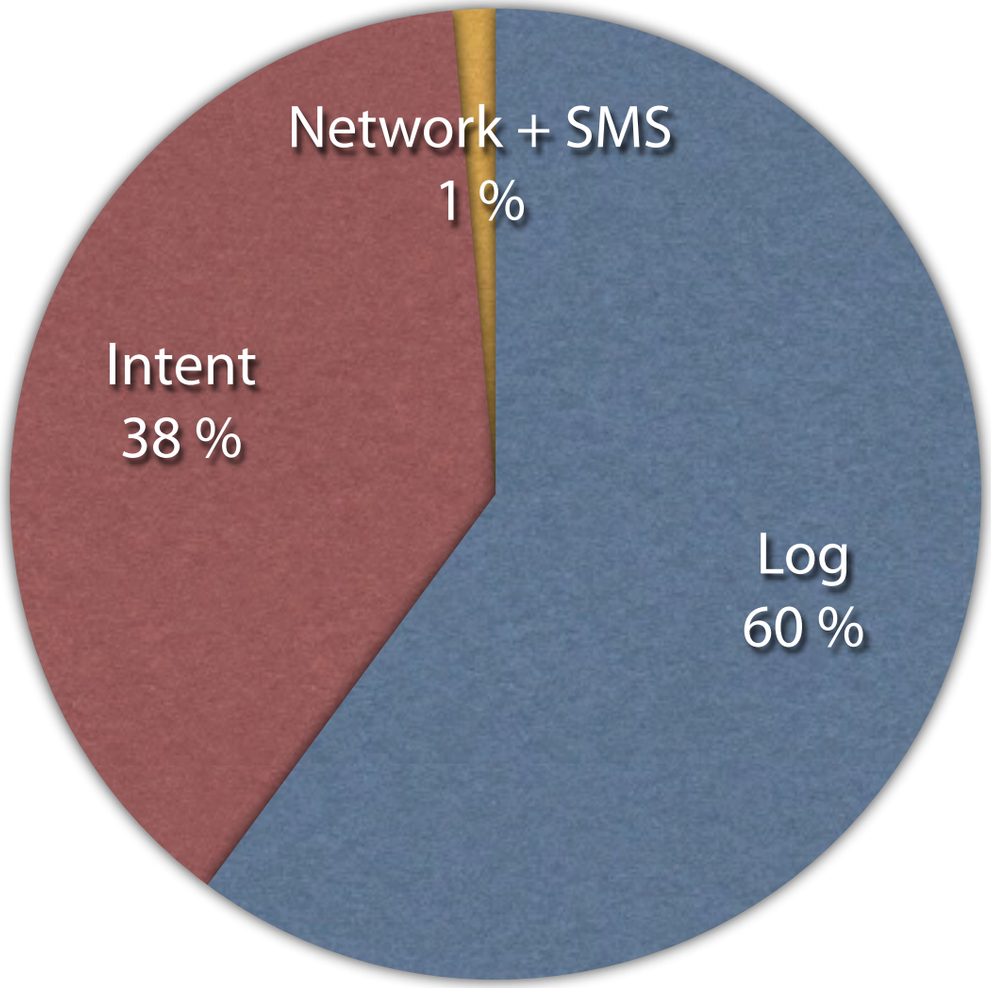
TelephonyManager.getDeviceId() → URL.openConnection()

1 minute
1 core
1 GB RAM

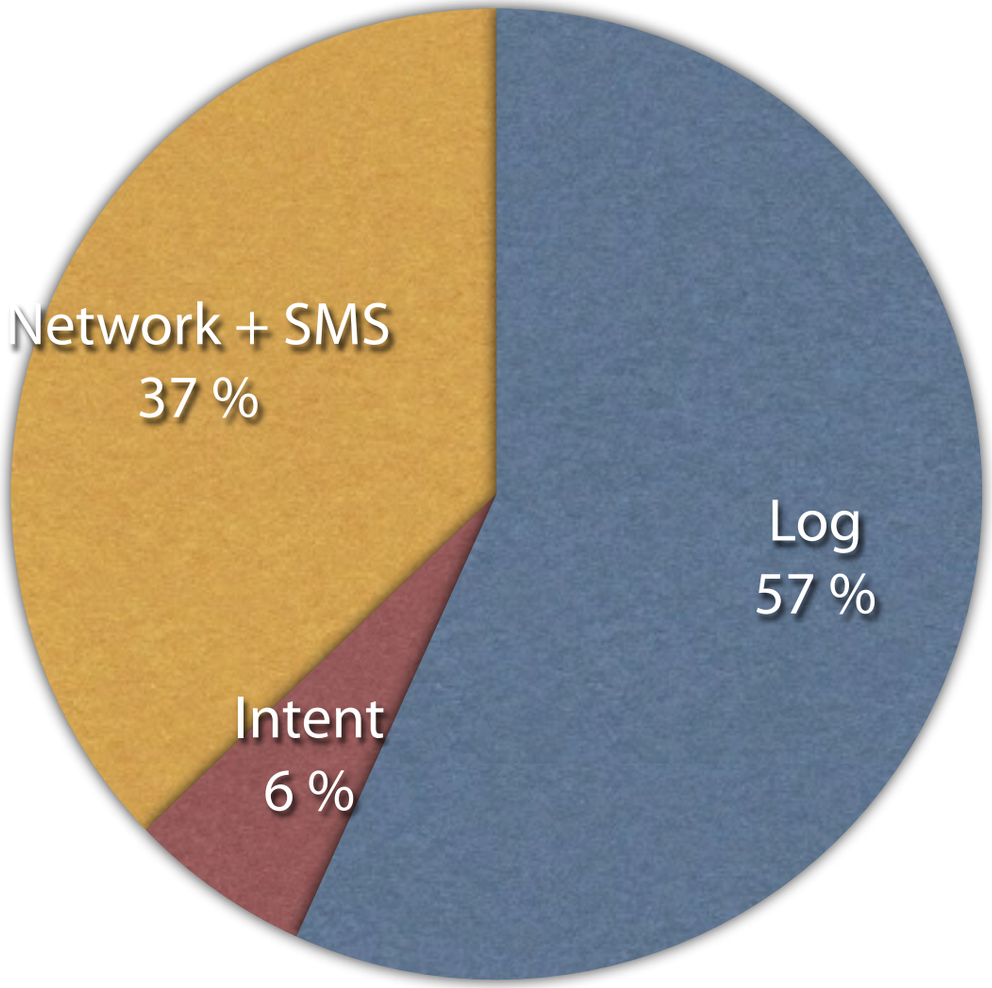
Sensitive Data Flow

- Which sensitive APIs does the *device ID* flow to?

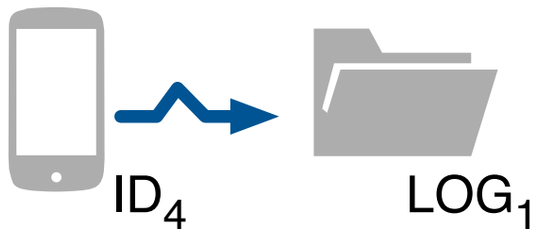
Benign Apps

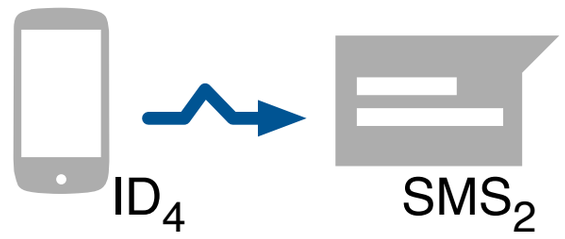


Malicious Apps

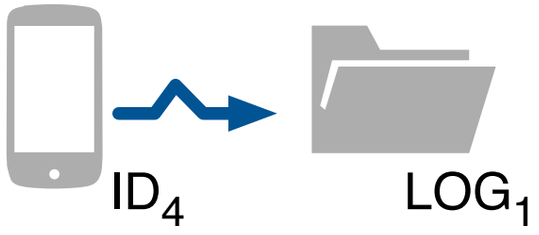


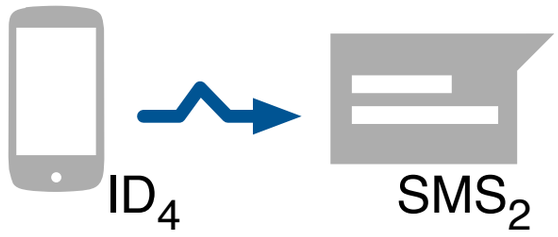
MUDFLOW



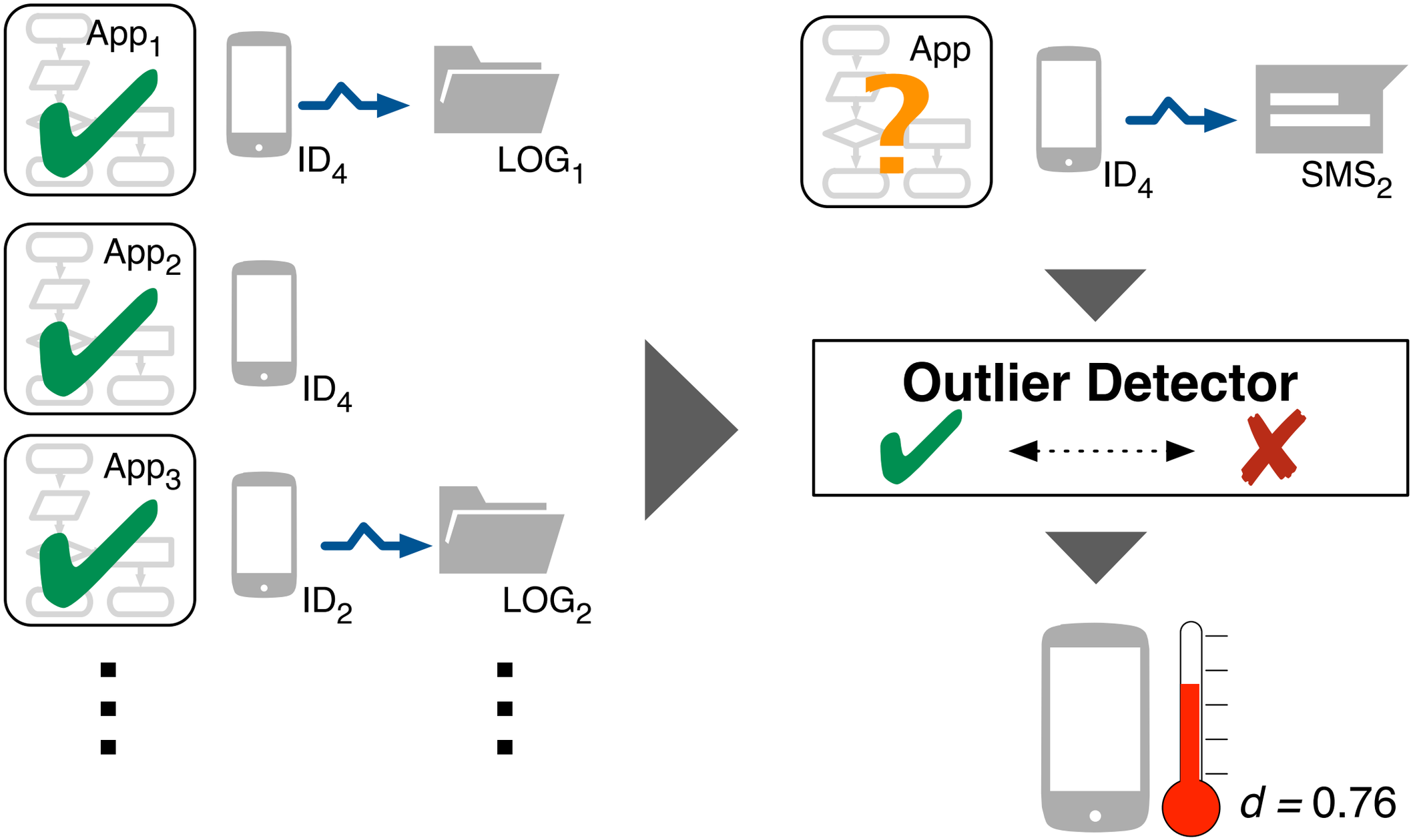


Mining Unusual Data Flow





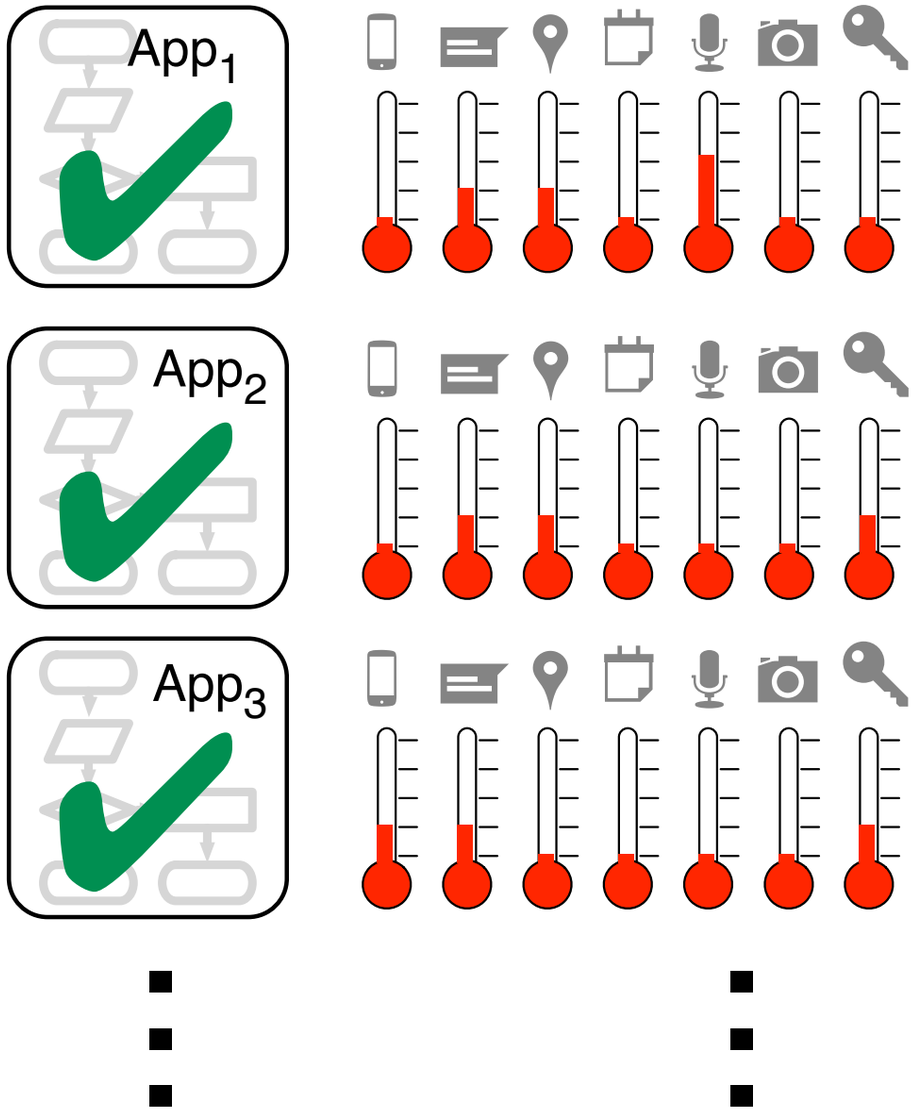
Outlier Detection

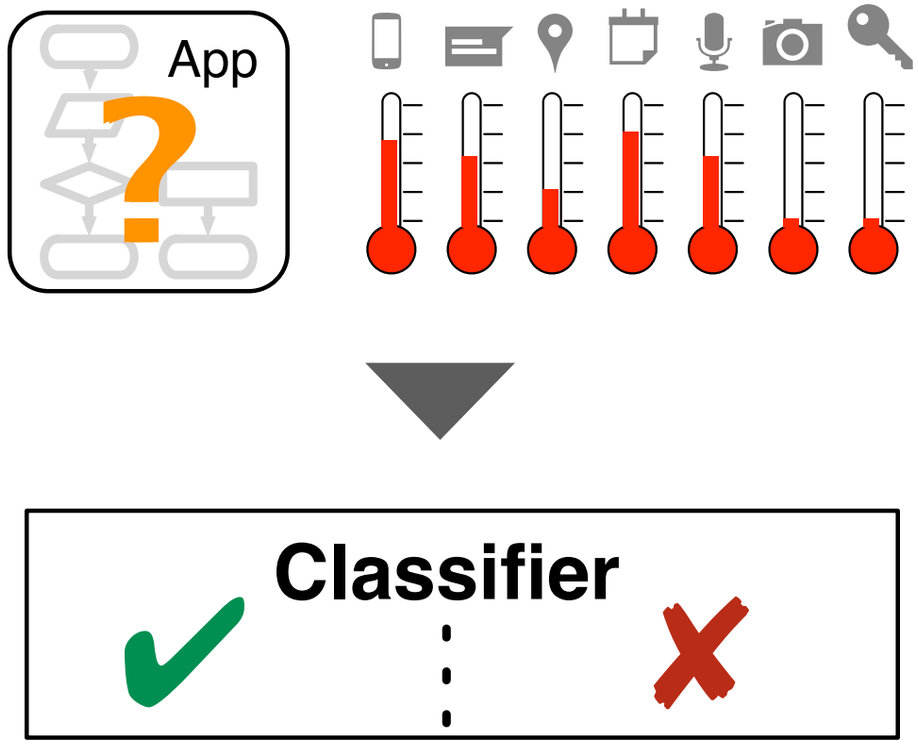


Training

Outlier Detection

Malware Classification



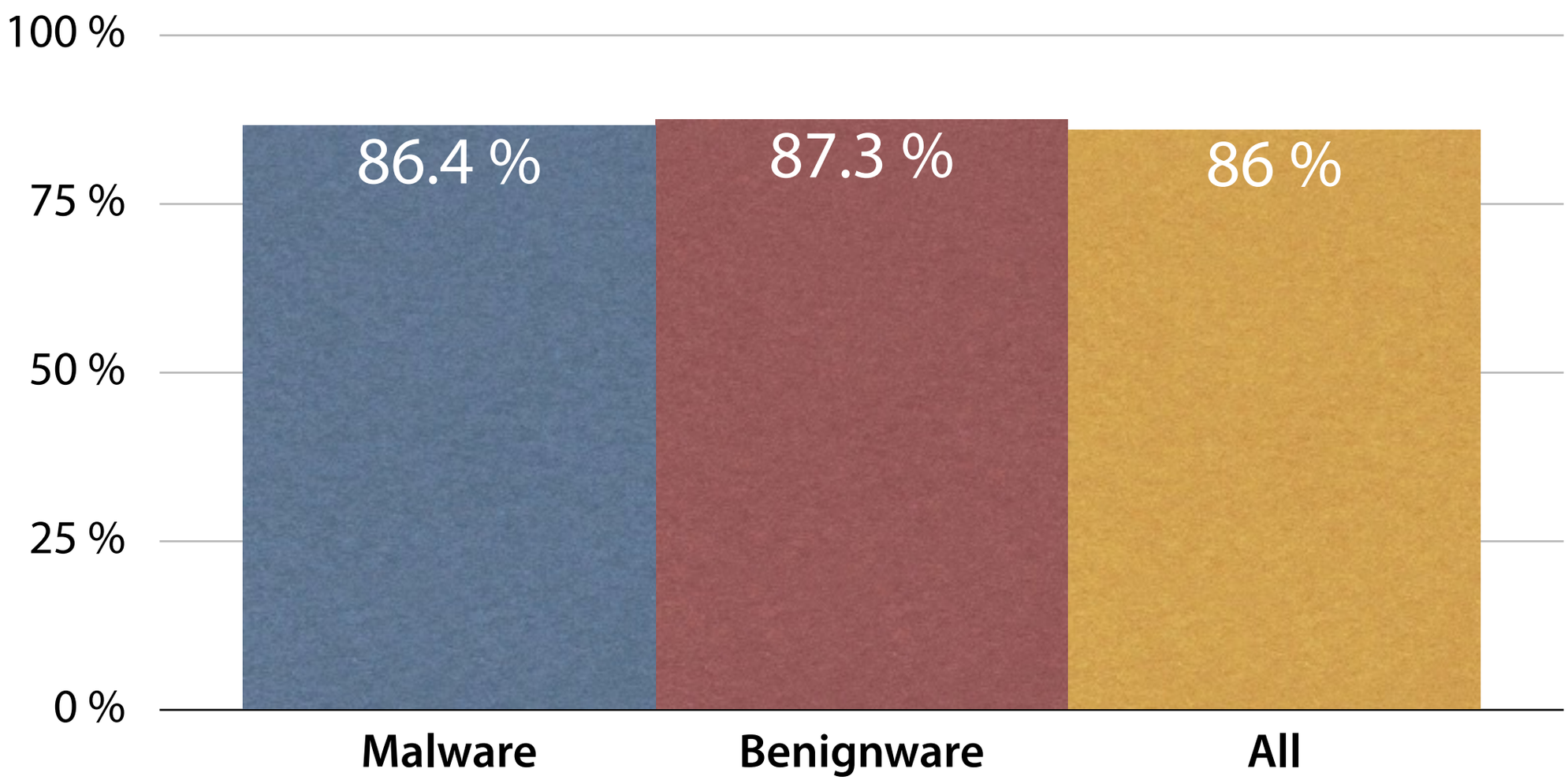


Training

Classifying

Correctly Classified

10,552 malicious apps with at least one sensitive leak



Accurate malware recognition without needing malware samples

M RAD

Error.

DRG▸ RCL STO M−

SHIFT DRG FSE MR MS M+

hyp⁻¹ sin⁻¹ cos⁻¹ tan⁻¹ e^x 10^x

hyp sin cos tan ln log

$\sqrt[x]{y}$ $\sqrt[3]{\ }$ x^3 $1/x$

y^x $\sqrt{\ }$ x^2 % ( )

7 9 DEL AC

nPr

4 ×

Food for Thoughts

Check your legal situation

- Apps may not be reverse engineered
- Apps are copyrighted; cannot be “shared”
- App behavior must not be changed

Industry is not dumb at all

- Vendors *do* monitor their stores
- Vendors *do* analyze apps, usage, sales
- Vendors *want* control over security and privacy

App Mining

- For 100,000s of apps:
- Gather *descriptions*
- Gather *metadata*
- Gather *execution features*
- Find what is *common* and what is *uncommon*

Analyzing App Code

APIs

- Easy to extract (grep)
- Easy to process
- Initial classification of program behavior

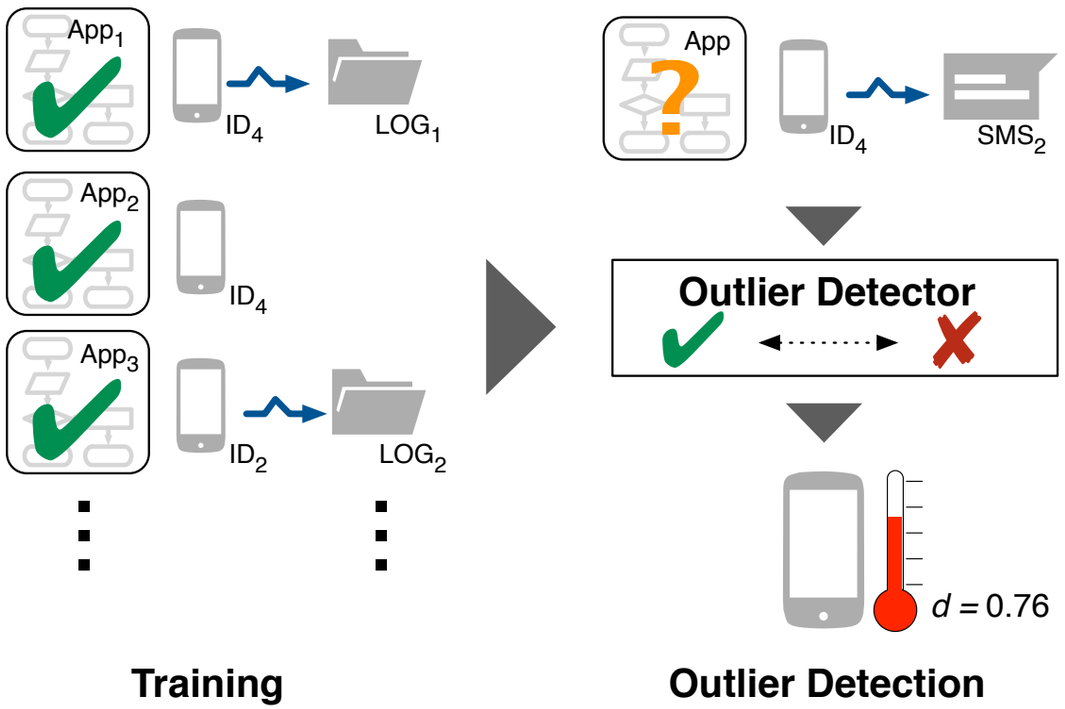
Code

- Comes in binary form (Dalvik / ARM)
- Hard to analyze statically (scale, components)
- Code may be *adverse* (malware)

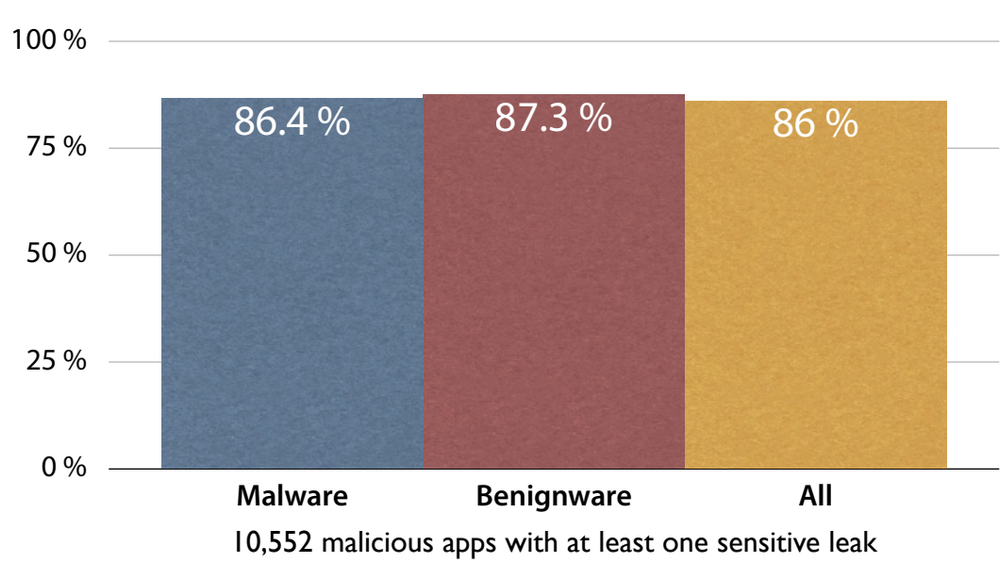
Executions

- Need *test generators* to assess
- Instrument binary and/or environment
- Code may be *adverse* (malware)

Outlier Detection



Correctly Classified



10,552 malicious apps with at least one sensitive leak

Accurate malware recognition without needing malware samples

http://www.st.cs.uni-saarland.de/appmining/

erc Google