# TriFlow: Triaging Android Applications using Speculative Information Flows

Omid Mirzaei, Guillermo Suarez-Tangil, Juan Tapiador, Jose M. de Fuentes

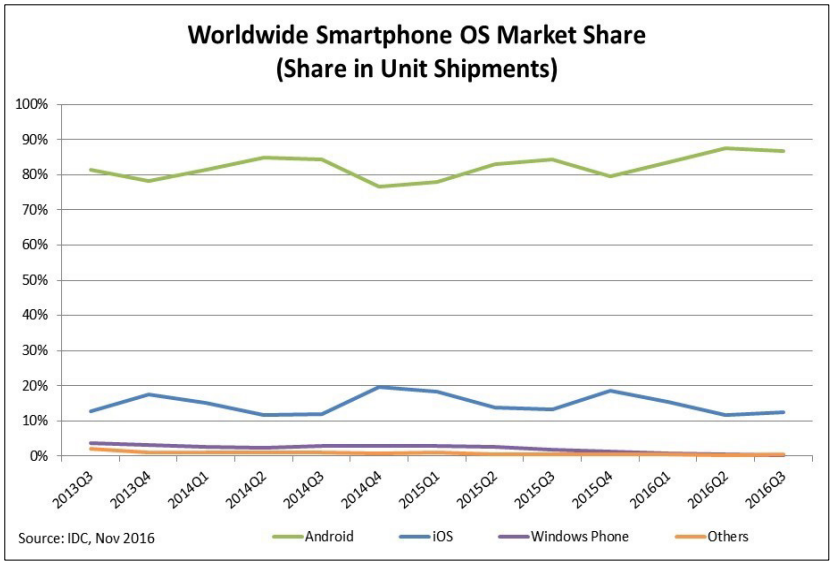
uc3m | Universidad Carlos III de Madrid

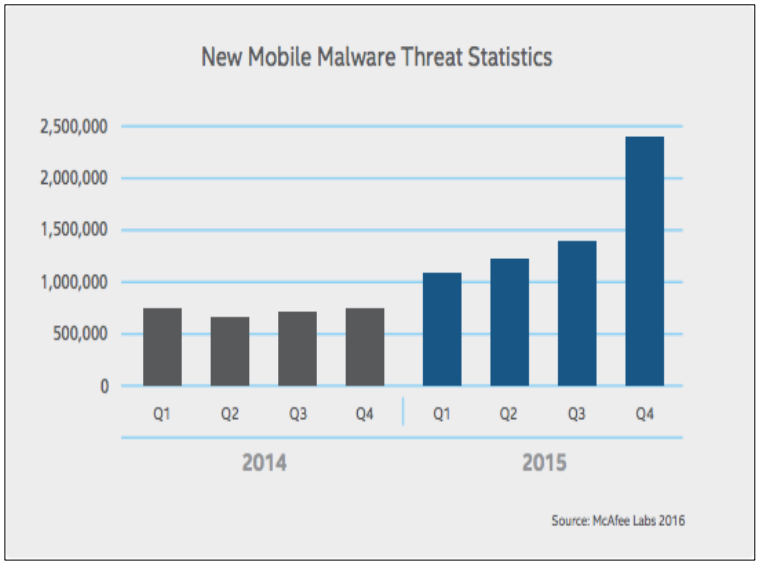
UCL

12th ACM Asia Conference on Computer and Communications Security
**ASIACCS 2017**

# Introduction

- Popularity of Android
- Malicious and Potentially Harmful Apps (PHAs)
- Lots of Analysis tools proposed in the last years



# Outline

- Background & Contributions
- TriFlow Overview
- Results
- Discussion
- Conclusions

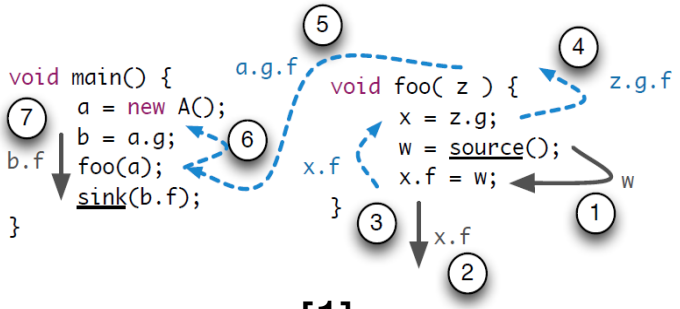
# Outline

- **Background & Contributions**
- TriFlow Overview
- Results
- Discussion
- Conclusions

# Background & Contributions

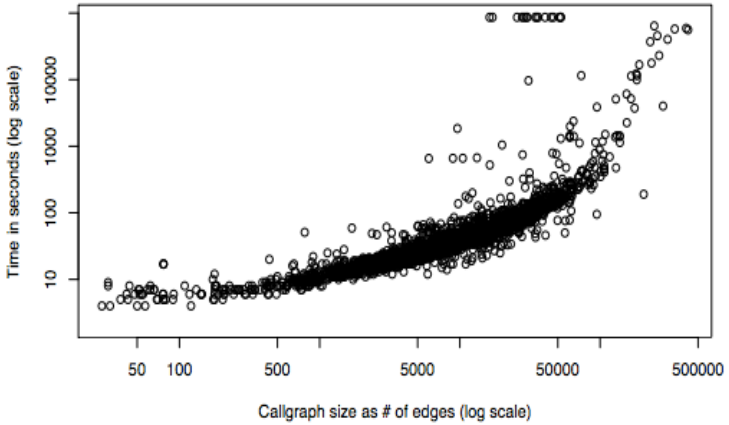
## Info-flows

- Information Flow Analysis
- How to extract flows?
  - Static Analysis (e.g. FlowDroid)
  - Dynamic Analysis (e.g. TaintDroid)
- Limitations of info-flow analysis techniques:
  - Accuracy (false positives, false negatives)
  - Scalability (memory, time)




[1]

[2]





[1] Arzt, Steven, et al. "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps." *Acm Sigplan Notices* 49.6 (2014): 259-269.

[2] Avdiienko, Vitalii, et al. "Mining apps for abnormal usage of sensitive data." *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015.

# Background & Contributions
## Risk metrics

- Info-flows are good to study the behavior of apps
- What if we rely upon flows to measure the risk posed by an app?
  - Identifying apps with potentially dangerous behaviors
- What are the applications?
  - Communicate risk to final users
  - Triage
- Previous risk metrics:
  - Meta-data from Manifest file (permissions and components)
  - Market data (review score, size, developer reputation)
  - Features in the source code (API methods)
  - Others (Apps' behavior, network usage, root exploits)

# Background & Contributions

## Main contributions

- Info-flow based risk scoring mechanism
- Key Features:
  - Notion of speculative info-flows (predicting the presence of flows)
  - Fast: Applications to triage
  - Complement to other risk scoring tools
- Tool:
  - **TriFlow** is publicly available at:

https://github.com/OMirzaei/TriFlow

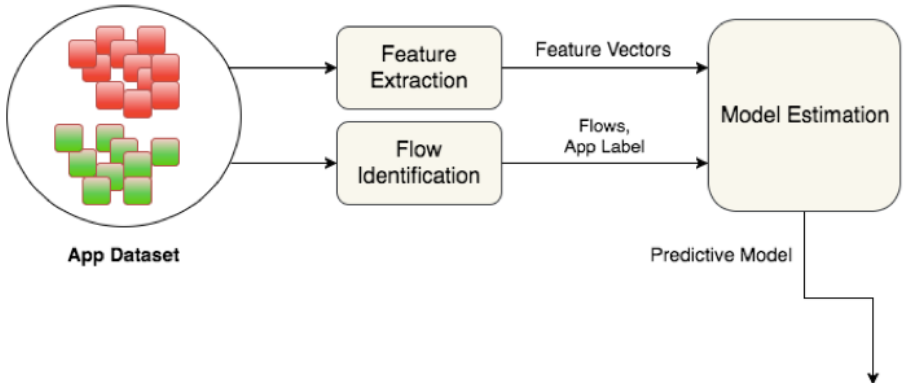
# Outline

- Background & Contributions
- **TriFlow Overview**
- Results
- Discussion
- Conclusions

# TriFlow Overview

## Prediction of flows

- Key idea:
  - Static features whose presence correlates with the presence of flows
- Creating a predictive model:
  - Using a  dataset of apps
  - Needs ground truth, i.e. actual flows for each app (FlowDroid)
  - Estimating **P(flow | features)**



# TriFlow Overview
## Weighting flows

- Goals:
  - Identifying “malicious” info-flows:

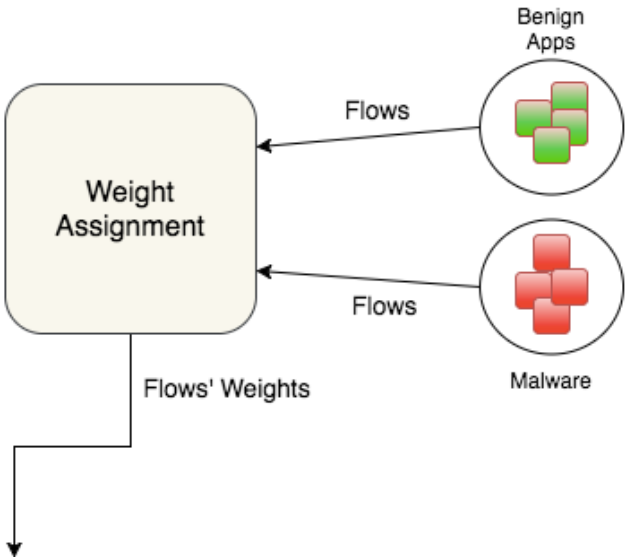
    **Flows appearing mostly in malware but not in benign apps**

  - Filtering out info-flows which are common in both classes
- $I(f) = -P_M(f) \log_2 P_B(f)$

$P_M(f)$: Frequency of flow in malware

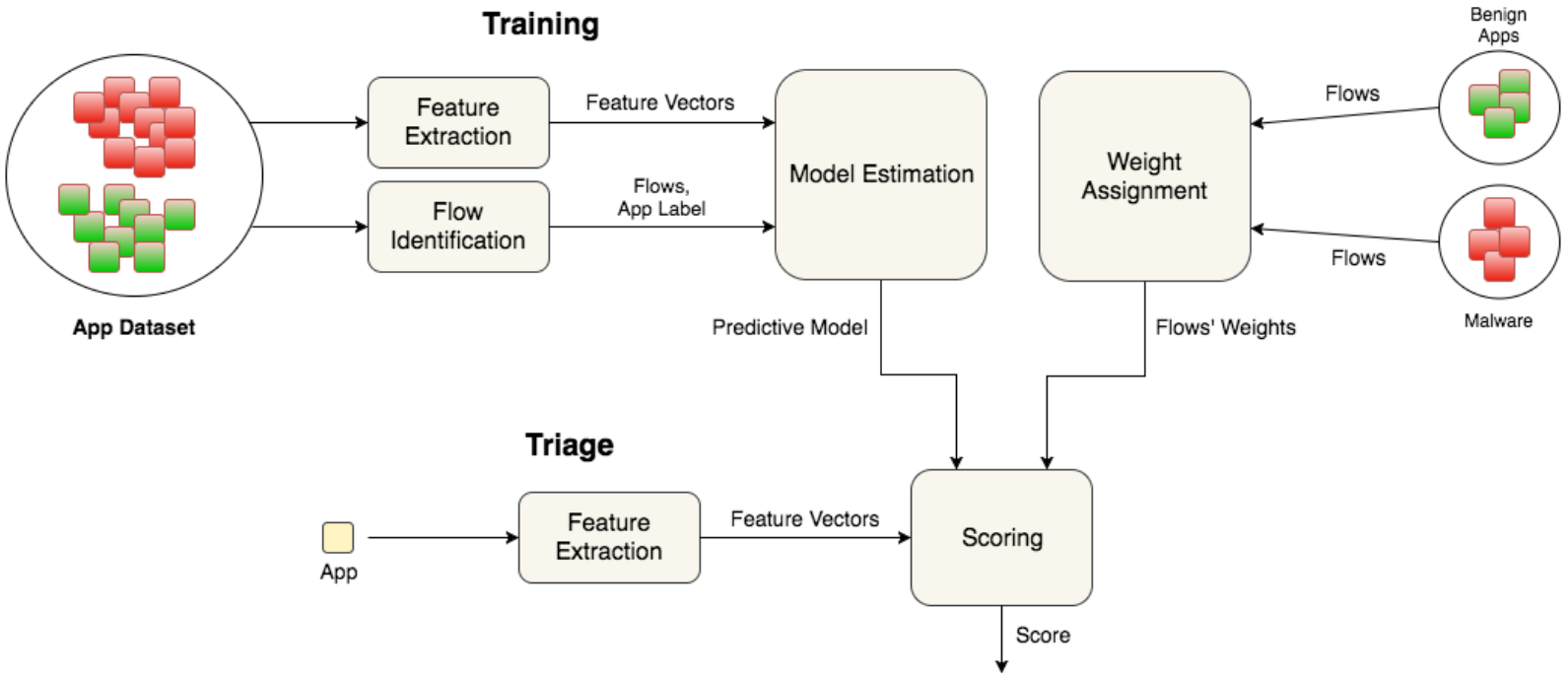
$P_B(f)$: Frequency of flow in Benign apps

$I(f)$: Weight of flow (Maliciousness)



# TriFlow Overview

## Overall System




$$R = \sum_{j=1}^{N} P_j * I_j \qquad j = 1, 2, 3, \ldots, N \ (Total\ number\ of\ flows)$$

# Outline

- Background & Contributions
- TriFlow Overview
- **Results**
- Discussion
- Conclusions

# Results

## Datasets

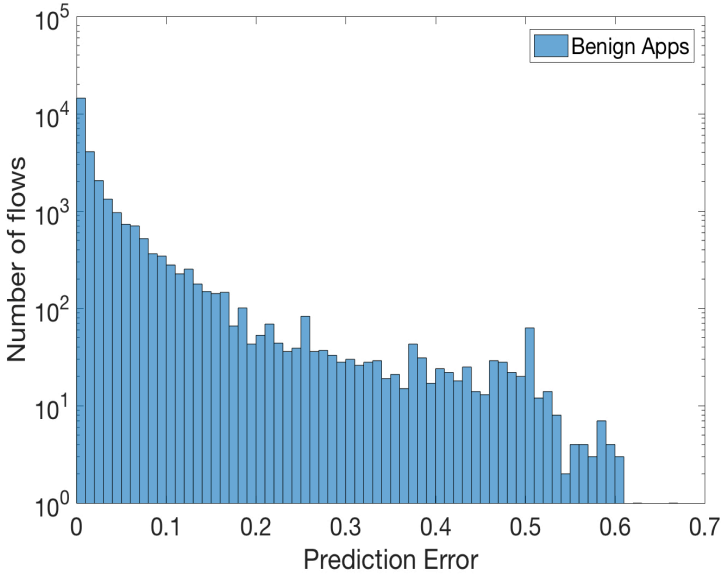
| Type | Dataset | Type | Samples |
|---|---|---|---|
| Malware (MW) | Drebin [4] | Malware | 5,560 |
| Goodware (GW) | Google Play | Goodware | 11,456 |
| | | **Total** | 17,016 |

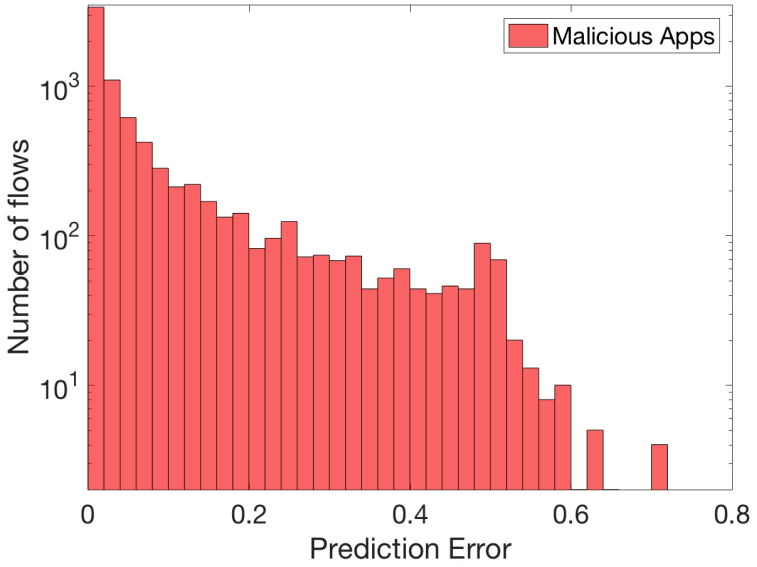
| Mode | Split | Ratio | Samples |
|---|---|---|---|
| Modeling (Training) | 4,000 MW<br>4,000 GW | 1:1 | 8,000 |
| Triage (Testing) | 1,560 MW<br>7,456 GW | 1:5 | 9,016 |

# Results

## Prediction of flows

| Dataset | Mean | Std. Dev. | Median |
|---|---|---|---|
| *Drebin* | 0.0861 | 0.1272 | 0.0278 |
| *GooglePlay* | 0.0361 | 0.0734 | 0.0094 |
| *All* | 0.0376 | 0.0784 | 0.0089 |




1.04% of flows are predicted with error = 0
90% of flows are predicted with error < 0.25

4.31% of flows are predicted with error = 0
83% of flows are predicted with error < 0.1
90% of flows are predicted with error < 0.25

# Results

## Weighting flows

- In 75% of flows: $I(f) = 0$
- In almost 25% of flows: $0 < I(f) \leq 0.5$
- In almost 1% of flows: $0.5 < I(f) \leq 1$
- In very rare flows: $I(f) > 1$

*Observed mainly in Malware*

| Source | Sink | $I(f)$ |
|---|---|---|
| TM.getDeviceId() | String.startsWith() | 0.69 |
| TM.getDeviceId() | OutputStream.write() | 0.26 |
| TM.getDeviceId() | Intent.putExtra() | 0.52 |
| TM.getDeviceId() | String.substring() | 0.28 |
| TM.getDeviceId() | URL.openConnection() | 0.37 |
| TM.getSubscriberId() | String.startsWith() | 0.88 |
| TM.getSubscriberId() | OutputStream.write() | 0.24 |
| TM.getSubscriberId() | HttpURLCon.setRequestMethod() | 0.25 |
| TM.getSubscriberId() | URL.openConnection() | 0.42 |
| TM.getSubscriberId() | Intent.putExtra() | 0.58 |
| TM.getSimCountryIso() | Log.i() | 0.37 |
| TM.getSimCountryIso() | String.substring() | 0.25 |
| TM.getSimOperator() | Log.v() | 0.31 |
| TM.getNetworkOperator() | String.startsWith() | 0.32 |
| TM.getNetworkOperator() | String.substring() | 1.18 |
| TM.getLine1Number() | URL.openConnection() | 0.20 |
| TM.getLine1Number() | Log.v() | 0.52 |
| TM.getLine1Number() | String.startsWith() | 0.53 |
| TM.getSimSerialNumber() | String.startsWith() | 0.98 |
| TM.getSimSerialNumber() | String.substring() | 1.09 |
| gsm.SM.getDefault() | gsm.SM.sendTextMessage() | 0.82 |
| SM.getDefault() | SM.sendTextMessage() | 1.81 |
| NetworkInfo.getExtraInfo() | Log.d() | 0.68 |
| NetworkInfo.getExtraInfo() | String.startsWith() | 0.45 |
| WebView.getSettings() | WebS.setAllowFileAccess() | 0.67 |
| WebView.getSettings() | WebS.setGeolocationEnabled() | 0.46 |
| WebView.getSettings() | WebS.setPluginsEnabled() | 0.50 |
| System.getProperties() | String.substring() | 0.45 |
| PI.getBroadcast() | SM.sendTextMessage() | 1.28 |
| HashMap.get() | SM.sendTextMessage() | 1.33 |

`TM`: TelephonyManager, `SM`: SmsManager, `PI`: PendingIntent, `HttpURLCon`: HttpURLConnection, `WebS`: WebSettings.

# Results

## Weighting flows

**MEAN**

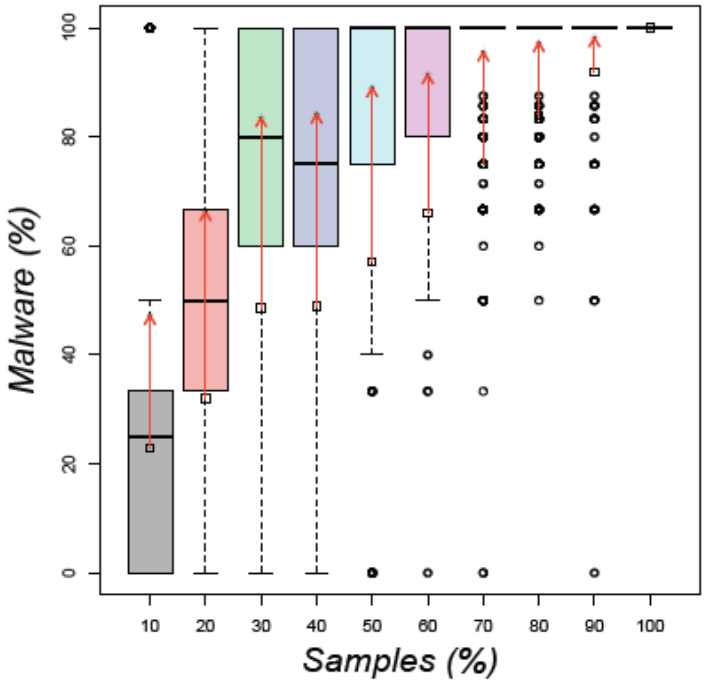
| | LOG | FILE | NETWORK | SMS_MMS | AUDIO | NO_CATEGORY | LOCATION_INFORMATION |
|---|---|---|---|---|---|---|---|
| NETWORK_INFORMATION | 0,0369 | 0,0074 | 0,0111 | 0,1767 | N/A | 0,044 | N/A |
| CALENDAR_INFORMATION | 0,0104 | 0,0096 | 0,0063 | N/A | N/A | 0,0148 | N/A |
| LOCATION_INFORMATION | 0,0342 | N/A | 0,031 | 0,0054 | N/A | 0,0173 | N/A |
| DATABASE_INFORMATION | 0,0277 | 0,0157 | 0,022 | 0,0655 | 0,0032 | 0,0179 | N/A |
| ACCOUNT_INFORMATON | 0,0027 | N/A | N/A | N/A | N/A | 0,032 | N/A |
| UNIQUE_IDENTIFIER | 0,0824 | 0,0079 | 0,3059 | 0,0919 | N/A | 0,0508 | N/A |
| BLUETOOTH_INFORMATION | N/A | N/A | N/A | N/A | N/A | 0,0031 | N/A |
| NO_CATEGORY | 0,0284 | 0,0173 | 0,0382 | 0,0799 | 0,0097 | 0,0222 | 0,0088 |

**MAX**

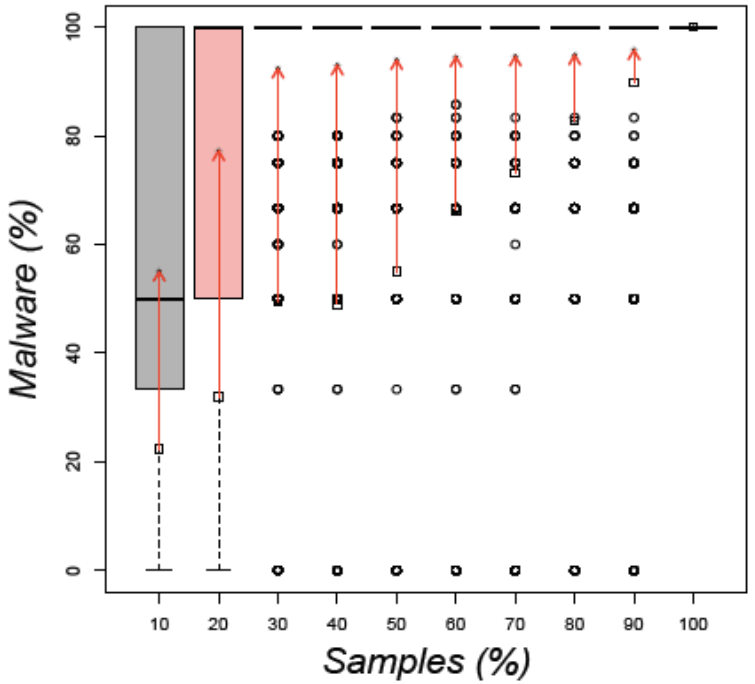
| | LOG | FILE | NETWORK | SMS_MMS | AUDIO | NO_CATEGORY | LOCATION_INFORMATION |
|---|---|---|---|---|---|---|---|
| NETWORK_INFORMATION | 0,684 | 0,0096 | 0,0257 | 1,8161 | N/A | 1,1881 | N/A |
| CALENDAR_INFORMATION | 0,0421 | 0,0128 | 0,0075 | N/A | N/A | 0,1284 | N/A |
| LOCATION_INFORMATION | 0,1403 | N/A | 0,1175 | 0,0128 | N/A | 0,1626 | N/A |
| DATABASE_INFORMATION | 0,2092 | 0,0544 | 0,0471 | 0,2336 | 0,0032 | 0,2766 | N/A |
| ACCOUNT_INFORMATON | 0,0028 | N/A | N/A | N/A | N/A | 0,1056 | N/A |
| UNIQUE_IDENTIFIER | 0,5216 | 0,0187 | 0,4279 | 0,1536 | N/A | 1,0901 | N/A |
| BLUETOOTH_INFORMATION | N/A | N/A | N/A | N/A | N/A | 0,0032 | N/A |
| NO_CATEGORY | 0,6032 | 0,4618 | 0,5292 | 1,3315 | 0,0376 | 1,1776 | 0,016 |

# Results

## Scoring and Prioritizing




(a) RSS (Sarma et al. [34]).




(b) TRIFLOW.

# Results

## Explanation of Scores

**Application Name = 55d7e1c74ed3630f7c8d7a892c049aca.apk** (Trackplus family)

Total Score = 8.002e-05

**[UNIQUE_IDENTIFIER, LOG]** = 2.17e-05 (27.09% of the score)

<android.telephony.TelephonyManager: java.lang.String **getDeviceId()**>,
<android.util.Log: int **i**(java.lang.String,java.lang.String)>, 1.20e-05

<android.telephony.TelephonyManager: java.lang.String **getDeviceId()**>,
<android.util.Log: int **d**(java.lang.String,java.lang.String)>, 8.03e-06

<android.telephony.TelephonyManager: java.lang.String **getDeviceId()**>,
<android.util.Log: int **e**(java.lang.String,java.lang.String)>, 1.58e-06

<android.telephony.TelephonyManager: java.lang.String **getDeviceId()**>,
<android.util.Log: int **w**(java.lang.String,java.lang.String)>, 2.80e-08

**[DATABASE_INFORMATION, LOG]** = 1.43e-08 (0.018% of the score)

<android.database.sqlite.SQLiteDatabase: android.database.Cursor
**query**(java.lang.String,java.lang.String[],java.lang.String,java.lang.String
[],java.lang.String,java.lang.String,java.lang.String,java.lang.String)>,
<android.util.Log: int **d**(java.lang.String,java.lang.String)>, 1.43e-08

…

# Outline

- Background & Contributions
- TriFlow Overview
- Results
- **Discussion**
- Conclusions

# Discussion

- Accuracy
  - TriFlow might miss some flows due to reflection
- Risk Notion
  - Apps that have higher scores are not only malware
- Datasets
  - Number and representativeness of apps in both datasets
- Evasion Attacks
  - Avoid using some sources/sinks (flows)
  - Making flows undetectable (reflection)
  - Replacing flows

# Outline

- Background
- Contributions
- TriFlow Overview
- Results
- Discussion
- **Conclusions**

# Conclusions

- **Predicting** info-flows in Android apps
- **Weighting** flows based on **“rare equals risky”** intuition
- A **new risk metric** based on info-flows
- An efficient **triage** system
- A **complement** to static and dynamic analysis tools

# Your Questions

Thanks!

**Email:** omid.mirzaei@uc3m.es
**Website:** http://www.seg.inf.uc3m.es/~omirzaei/