# DroidCat: Effective Android Malware Detection and Categorization via App-Level Profiling

Haipeng Cai, Na Meng, Barbara Ryder, and Daphne Yao

# Introduction

- App-Level Profiling
- Static Analysis
  - information flows, calling structures
  - malicious code pattern
  - excessive permissions
  - frequently used apis in malware
- Dynamic Analysis
  - system/API call traces
  - differentiates API call counts and strictly matches API signatures

Static:abnormal information flows ;
calling structures; malicious code
pattern; excessive permissions;
frequently used apis in malware.
Dynamic:

Dynamic:behavior-based techniques
model program -- behavioral profiles,
with system/API call traces or resource
usage; Use machine learning to help
analysis; utilize histogram or chains or
dependencies of system calls.
differenciates API call counts and strictly
matches API signatures

图例：FakeInst私自发送SMS信息

# Motivation

```java
// in ad.notify.Settings::getImei(Context context)
// m6 returns 'phone'; cls returns 'android.telephony.TelephonyManager'
TelephonyManager tm = context.getSystemService(m6(b, b−1, x|76));
Class c = Class.forName(mdb.cls(ci));
Method m = c.getMethod(mdb.met(mi),null); //met returns 'getDeviceId'
return m.invoke(tm, null);

// in NotificationApplication::onCreate(); cls returns 'ad.notify.Settings'
Class c = Class.forName(mdb.cls(ci)); //met returns 'getImei'
Method m = c.getMethod(mdb.met(mi), new Class<Context>[1]);
adUrl += m.invoke(null, context);
```

- use reflection to invoke methods including Android APIs to access privileges resources. (class and method names are retrieved from a database)

- exploit SMS service

```java
// in ad.notify.SmsItem::send(String str, String str2)
// cls returns 'android.telephony.SmsManager'
Class c = Class.forName(mdb.cls(ci)); //met returns 'sendTextMessage'
Method m = c.getMethod(mdb.met(mi), new Class<Object>[5]);
SmsManager smsManager = SmgManager.getDefault();
m.invoke(smsManager, str, null, str2, null, null)
```

- simple refection

```java
// in ad.notify.OperaUpdateActivity::sendSms(String str, String str2)
Class c = Class.forName(mdb.cls(ci)); // cls returns 'ad.notify.SmsItem'
Method m = c.getMethod("send", new Class<String>[2]);
Boolean bs = m.invoke(null, str, str2);

// in ad.notify.OperaUpdateActivity::threadOperationRun(int i, Object o)
SmsItem smsItem=getSmsItem(ad.notify.NotifyApplication.smsIndex);
Class c = Class.forName("ad.notify.SmsItem");
Field f1 = c.getField("number"); int number = f1.get(smsItem);
Field f2 = c.getField ("text"); Object text = f2.get(smsItem);
sendSms(number, text);
```

# Goals

- Android Malware Detection and categorization
- Complete existing approches

# Existing work

- **Static approaches:**
  - **Advantage:** sound, scalable for **large amount** of apps;
  - **Disadvantage:**
    - Unable to reveal many malware activities because of the **event-driven features** of Android (lifecycle callback; GUI handling)
    - Malicious permissions or APIs might **not been executed or invoked** frequently at runtime(increase FP) or **run-time permission** mechanism
    - Limit capabilities in detecting malicious behaviors that are exercised through **dynamic code constructs**(eg. calling sensitive APIs via reflection)
    - Vulnerable to widely adopted **detection-evading schemes**

# Existing work

- **Dynamic approaches:**
  - **Advantage**: Can provide a complementary way to detect/categorize malware.
  - **Disadvantage**:
    - Can be evaded when app **obfuscates system calls**;
    - **Sensitive API** usage does not necessarily indicate malicious intentions;
    - **Abnormal resource** usage not mean abnormal behaviors
    - Need to capture varied behavioral profiles to against specific profiles.

# Existing work

- DroidFax — from ICSME (B)
  - A Toolkit for Systematic Characterization of Android Applications
- ICCDetector
  - modeled ICC(inter-component communication) patterns to identify malware that exhibits **different ICC characteristics** from benign apps
- MamaDroid
  - model app behaviors based on the transition probabilities between abstracted API calls in the form of Markov chains
- Hybrid approaches:
  - Messaging traffic, file/network operations, system/API calls. (Rely on static code analysis)

# Existing work

TABLE VI

COMPARISON OF RECENT WORKS ON ANDROID MALWARE CLASSIFICATION IN CAPABILITY AND ROBUSTNESS. DET: DETECTION, CAT: FAMILY CATEGORIZATION, SYSC: SYSTEM CALL, RT_PERM: RUN-TIME PERMISSION, RES: RESOURCE, OBF: OBFUSCATION

| Technique | Year | Approach | Classification Capability | | Robustness against Analysis Challenges | | | |
|---|---|---|---|---|---|---|---|---|
| | | | DET | CAT | Reflection | SYSC_OBF | RT_PERM | RES_OBF |
| DroidMiner [78] | 2014 | Static | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ |
| DroidSIFT [79] | 2014 | Static | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ |
| Drebin [15] | 2014 | Static | ✓ | N/A | ✗ | ✓ | ✗ | ✗ |
| MudFlow [80] | 2015 | Static | ✓ | N/A | ✗ | ✓ | ✓ | ✓ |
| Afonso et al. [27] | 2015 | Dynamic | ✓ | N/A | unknown | ✗ | ✓ | ✓ |
| Marvin [18] | 2015 | Hybrid | ✓ | N/A | ✗ | ✓ | ✗ | ✗ |
| Madam [35] | 2016 | Hybrid | ✓ | N/A | unknown | ✗ | ✗ | ✓ |
| ICCDetector [55] | 2016 | Static | ✓ | N/A | ✗ | ✓ | ✓ | ✗ |
| DroidScribe [34] | 2016 | Dynamic | N/A | ✓ | ✓ | ✗ | ✓ | ✓ |
| StormDroid [26] | 2016 | Hybrid | ✓ | N/A | ✗ | ✓ | ✗ | ✓ |
| MamaDroid [81] | 2017 | Static | ✓ | N/A | ✗ | ✓ | ✓ | ✓ |
| DroidSieve [23] | 2017 | Static | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ |
| **DroidCat** | this work | Dynamic | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

- Compare with representative recent peer works:

Almost all the static approaches compared are vulnerable to reflection as they use features based on APIs

# Contributions

- **Open-source dataset & tool:**
  - Evaluated DroidCat via **three complementary studies** versus two state-of-the-art peer approaches as baselines on **34,343** distinct apps(**from 2009 to 2017**).
  - Released access DroidCat and benchmark suites.
- **New Features:**
  - Developed DroidCat, a novel Android app **classification** and **detection** approach based on **new diverse set of features** that capture app behaviors **at runtime through short app-level profiling**.
  - Conduct in-depth **case studies**. Identified the most effective learning algorithm and dynamic features for DroidCat and demonstrate the low sensitivity to the coverage of dynamic inputs.

# Challenges

通过实验部分数据集的描述，说明随着时代的发展，越来越多的应用程序会采用混淆技术

- **Obfuscating evasion**: reflection, resource/system-call obfuscation, use of run-time permissions.

# Method

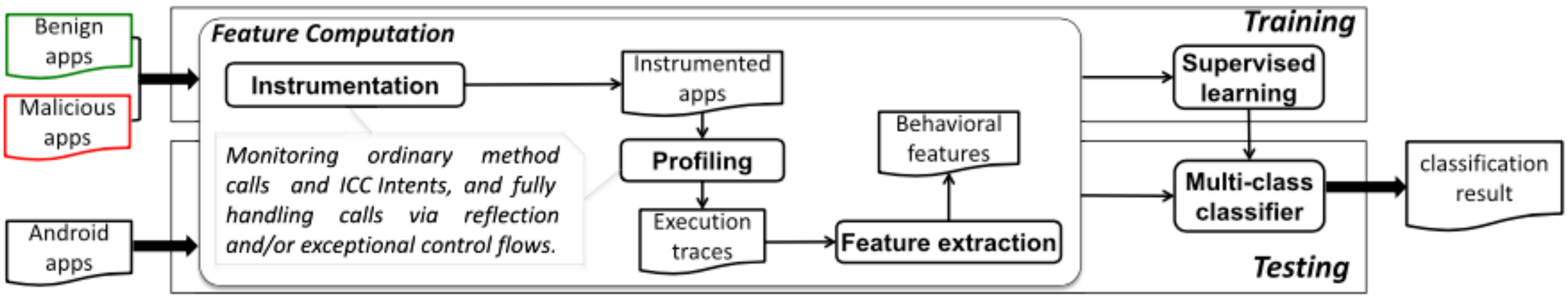



Fig. 3. *DroidCat* overview: it trains a multi-class classifier using benign and malicious apps and then classifies unknown apps.

- Testing
  - 30% newest apps from each class
- Training: 70 metrics, Random Forest algorithm
  - 70% of apps from each class

# Characterization Study

使用DroidFax进行代码覆盖率测量
目的：定义metrics

- 136 benign apps, 135 malicious apps.
- minimum supporting SDK version is **4.4 (API 19) or above**
- instrumented APK file runs successfully with inputs by **Monkey**
- navigating the app with Monkey inputs for **ten minutes covers at least 50%** of user code

# Metrics Definition

**Table1: Metrics For Dynamic Characterization and Feature Selection**

| Dimension | Metrics | Exemplar Metric | Substantially Disparate Metrics | Noticeably Different Metrics |
|---|---|---|---|---|
| **Structure** | 63 | The percentage of method calls whose definitions are in user code. | 15 | 32 |
| **ICC** | 7 | The percentage of external implicit ICCs. | 2 | 5 |
| **Security** | 52 | The percentage of sinks reachable by at least one path from a sensitive source. | 19 | 33 |
| **Total** | **122** | | **36** | **70** |

**Structure Dimension:** contains 63 metrics on the distributions of method calls, their declaring classes, and caller- callee links

**ICC Dimension:** Inter-component communication (Internal, External, Explicit, Inexplicit)

**Security Dimension:** contains 52 metrics to describe distributions of sources, sinks, and the reachability between them through method-level control flows.

# Metrics(Feature) Computation

- Instrument the program for execution trace collection.
  - use Soot to transform each app's APK along with the SDK library (`android.jar`) into Jimple code
- To Run-time monitors for **tracing every method call** (including those targeting SDK APIs and third- party library functions) and **every ICC** *Intent*.
- Use Class Hierarchy Analysis (CHA) [51] to identify all the superclasses
- Fully **tracks** two special kinds of method and ICC calls: (1) those made via ***reflection***, and (2) those due to ***exceptional control flows***

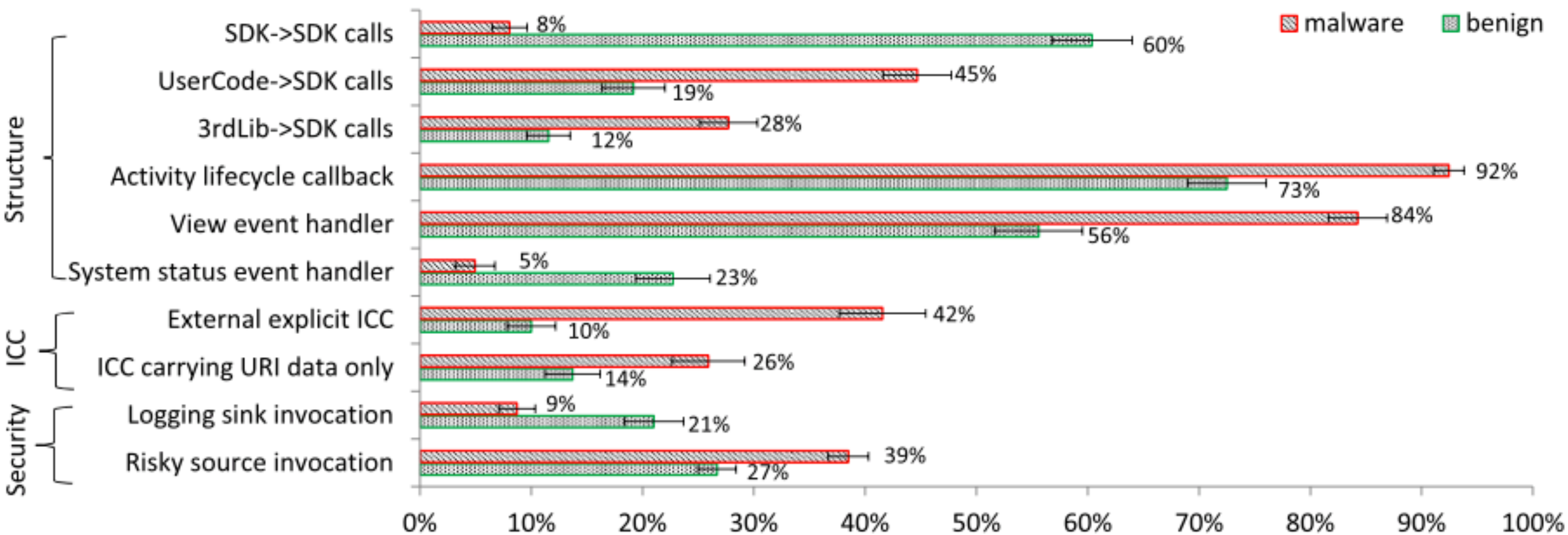
# Metrics(Feature) Selection

| Dimension | Metrics | Exemplar Metric | Substantially Disparate Metrics | Noticeably Different Metrics |
|---|---|---|---|---|
| **Structure** | 63 | The percentage of method calls whose definitions are in user code. | 15 | 32 |
| **ICC** | 7 | The percentage of external implicit ICCs. | 2 | 5 |
| **Security** | 52 | The percentage of sinks reachable by at least one path from a sensitive source. | 19 | 33 |
| **Total** | **122** | | **36** | **70** |

- **Substantially disparate**
  - metric had a mean value difference greater than or equal to **5%**
- **Noticeably different**
  - metric had a difference greater than or equal to **2%**

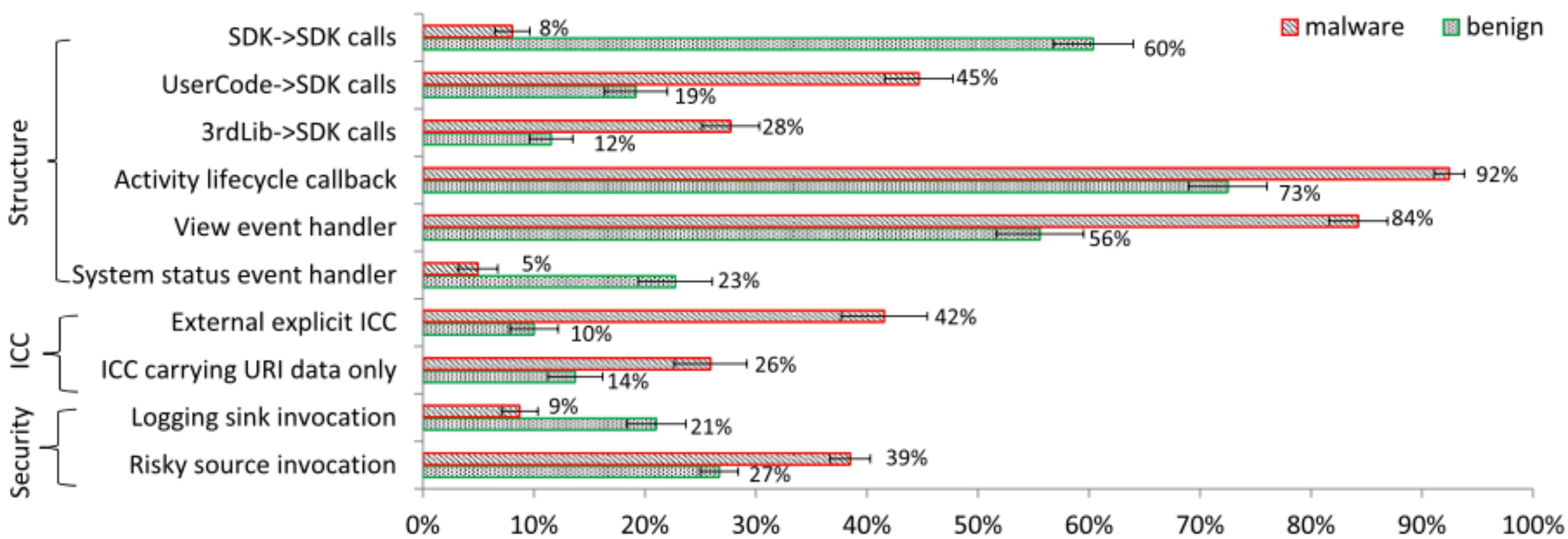
We experimented with various thresholds chosen heuristically, and found these two (5% and 2%) reasonably well represent two major levels of differentiation between our malware and benign samples.

# Metrics(Feature) Selection



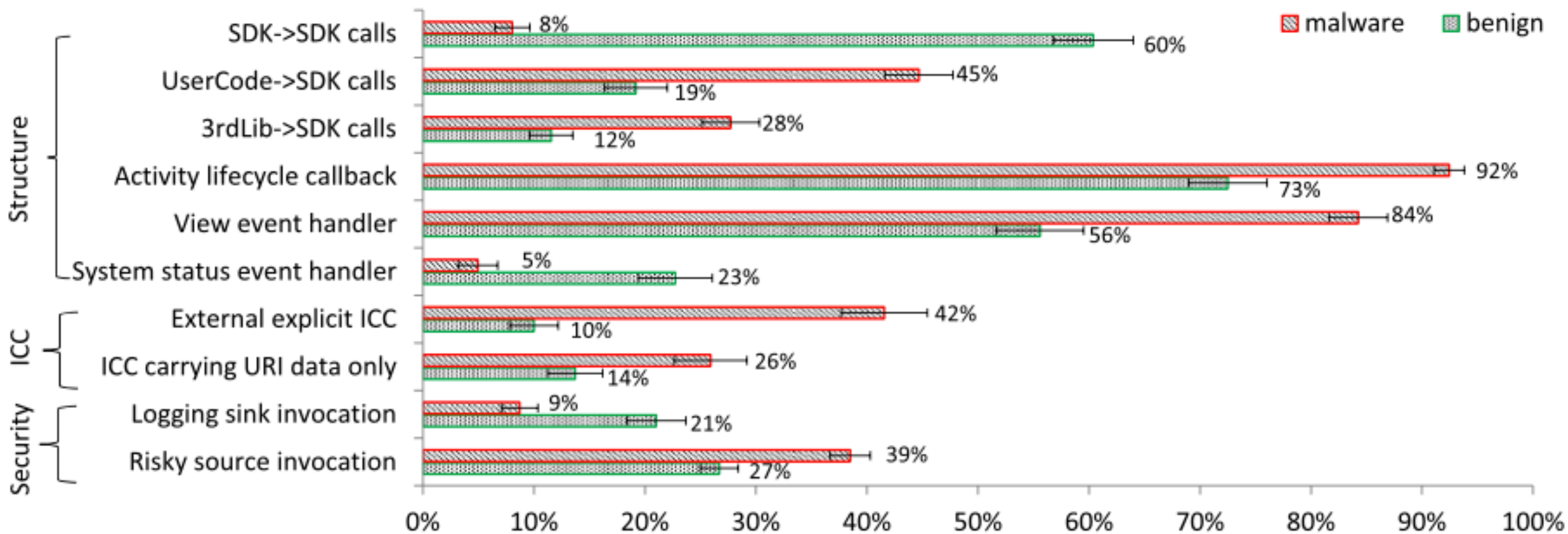

- Malware tended to **invoke SDK APIs** more often **from user code or third-party libraries**, and define **more UI callbacks** indicating that user operations on them may trigger excessive/ unexpected computation.

# Metrics(Feature) Selection




- Malware may use **more explicit ICCs** to potentially attack specific external components, or disseminate potentially malicious URIs more often via ICCs.

# Metrics(Feature) Selection




- Malicious apps exhibit **less logging practice** than benign ones. They execute **more risky sources**, which may lead to sensitive data leakage.

# Evaluation Experiment Setup

- Nexus One **emulator** with API Level 23, 2G RAM, and 1G SD storage for **5 minutes** as triggered by **Monkey** random inputs
- Ubuntu 15.04 workstation with 8G DDR and a 2.6GHz processor.
- **Baselines:**Self-implement **DroidSieve** and **Afonso**

# Dataset

TABLE II

MAIN DATASETS USED IN OUR EVALUATION STUDIES

| Dataset | Period | Benign apps | | Malware | | |
|---|---|---|---|---|---|---|
| | | Source | #Apps | Source | #Apps | #Families |
| D1617 | 2016-2017 | GP,AZ | 5,346 | VS,AZ | 3,450 | 153 |
| D1415 | 2014-2015 | GP,AZ | 6,545 | VS,AZ | 3,190 | 163 |
| D1213 | 2012-2013 | GP,AZ | 5,035 | VS,AZ,DB,MG | 9,084 | 192 |
| D0911 | 2009-2011 | AZ | 439 | VS,AZ,DB,MG | 1,254 | 88 |

- minimum supporting SDK version is **4.4 (API 19) or above**
- instrumented APK file runs successfully with inputs by **Monkey**
- navigating the app with Monkey inputs for **ten minutes covers at least 50%** of user code

# Methodology

Rcall体现了分类模型对正样本的识别能力，值越高，识别能力越强
Precision则是对负样本的区分能力
F1-score是两者的综合，分数越高，说明模型越稳健

- Precision (P)

$$P_i = \frac{\#\text{ of apps belonging to } C_i}{\text{Total }\#\text{ of apps labeled as "}C_i\text{"}}.$$

- Recall (R)

$$R_i = \frac{\#\text{ of apps labeled as "}C_i\text{"}}{\text{Total }\#\text{ of apps belonging to } C_i}.$$

- F1 score (F1)

$$F1_i = \frac{2 * P_i * R_i}{P_i + R_i}.$$

# Study I: Performance Stability

TABLE III

*DroidCat* PERFORMANCE FOR MALWARE DETECTION AND CATEGORIZATION

| Dataset | Detection | | | Categorization | | |
|---|---|---|---|---|---|---|
| | P | R | F1 | P | R | F1 |
| D1617 | 99.31% | 99.27% | 99.28% | 94.79% | 94.74% | 94.54% |
| D1415 | 97.26% | 97.09% | 97.16% | 97.84% | 97.75% | 97.70% |
| D1213 | 96.38% | 96.04% | 96.12% | 99.73% | 99.71% | 99.70% |
| D0911 | 97.19% | 96.96% | 97.00% | 99.48% | 99.43% | 99.44% |
| mean | 97.53% | 97.34% | 97.39% | 97.96% | 97.91% | 97.84% |
| stdev | 1.25% | 1.37% | 1.34% | 2.27% | 2.28% | 2.38% |

- DroidCat achieved mean F1 **high accuracy** of 97.39% and 97.84% for malware detection and categorization, respectively. It was also **stable in classifying** apps from different years within 2009–2017, evidenced by small standard deviations in F1 of 1.34-2.38% across the nine years.

# Study I: Performance Stability

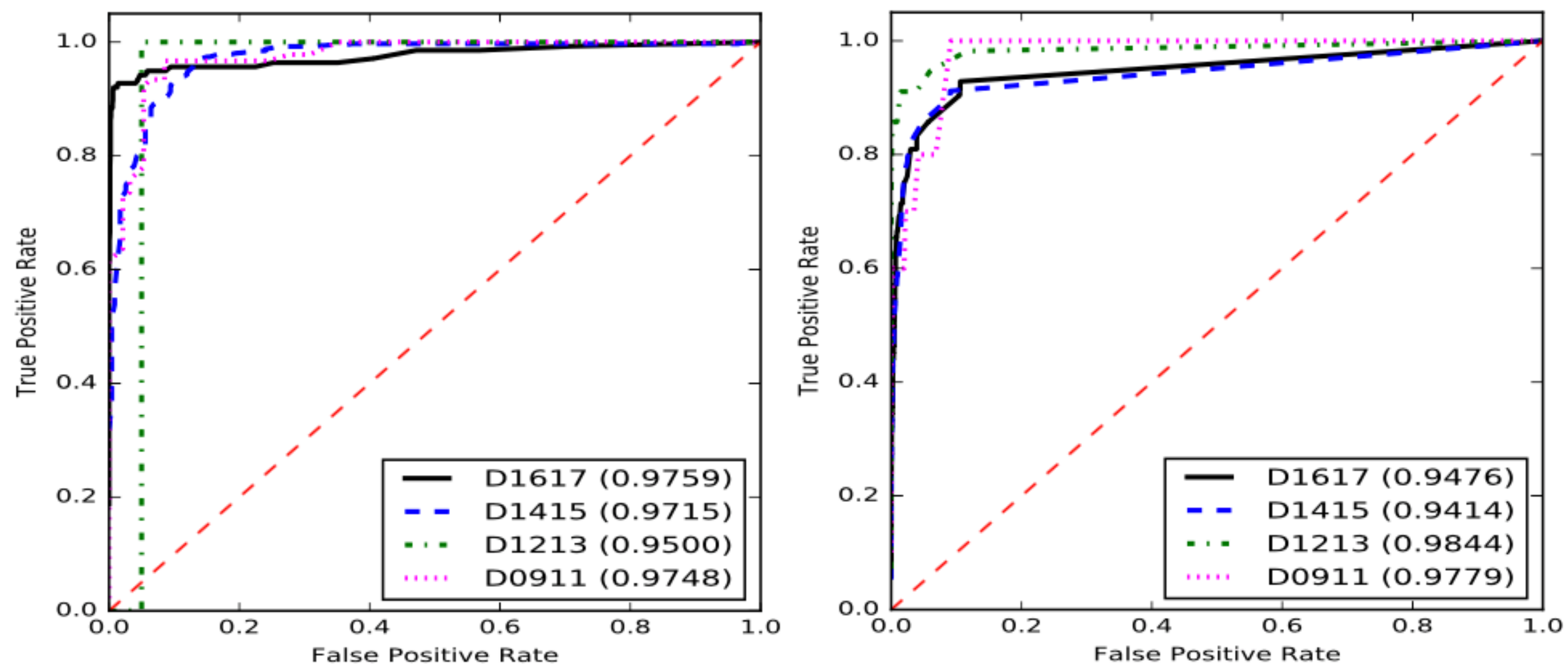



Fig. 4. DroidCat ROC curves with AUCs for malware detection (left) and categorization (right) on four datasets (D0911 through D1617).

- DroidCat achieved AUC of 0.95-0.98 and 0.94- 0.98, for malware detection and categorization, respectively.

# Study II: Comparative Classification Performance

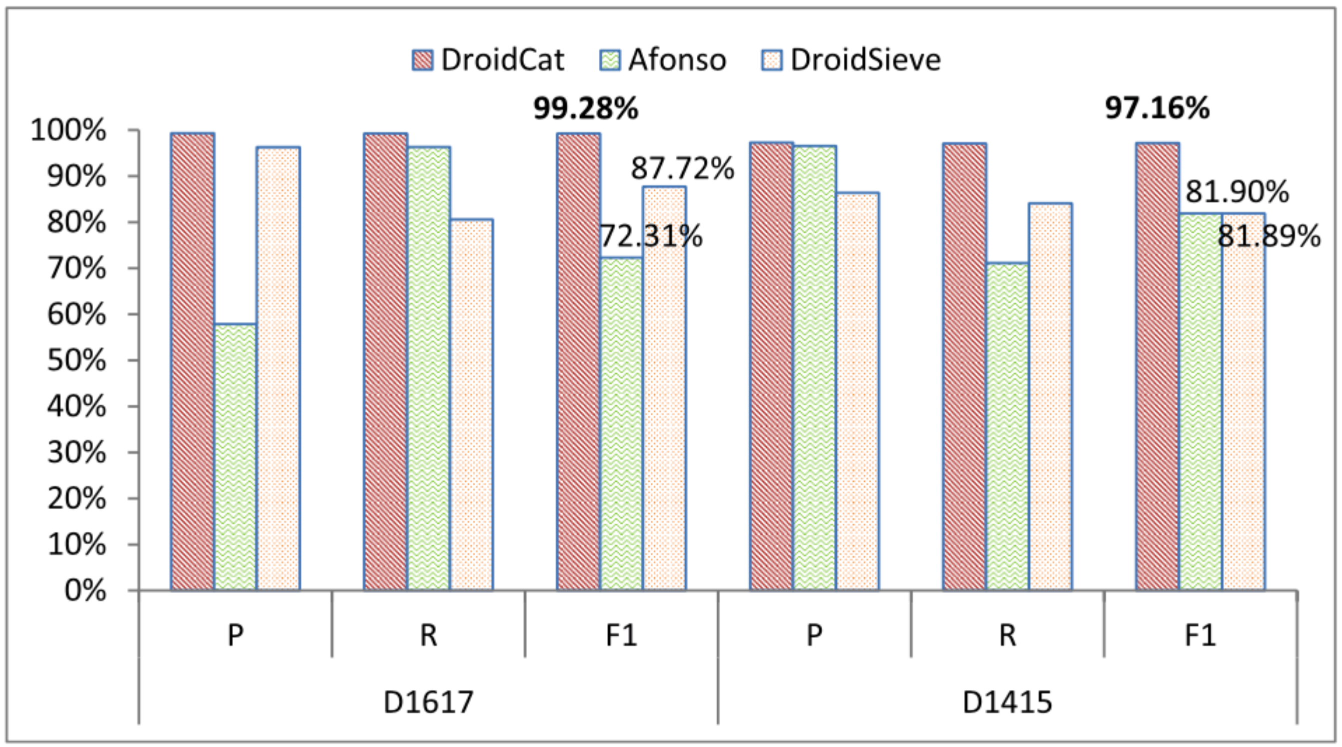



Fig. 5. *DroidCat* versus baselines for malware detection.

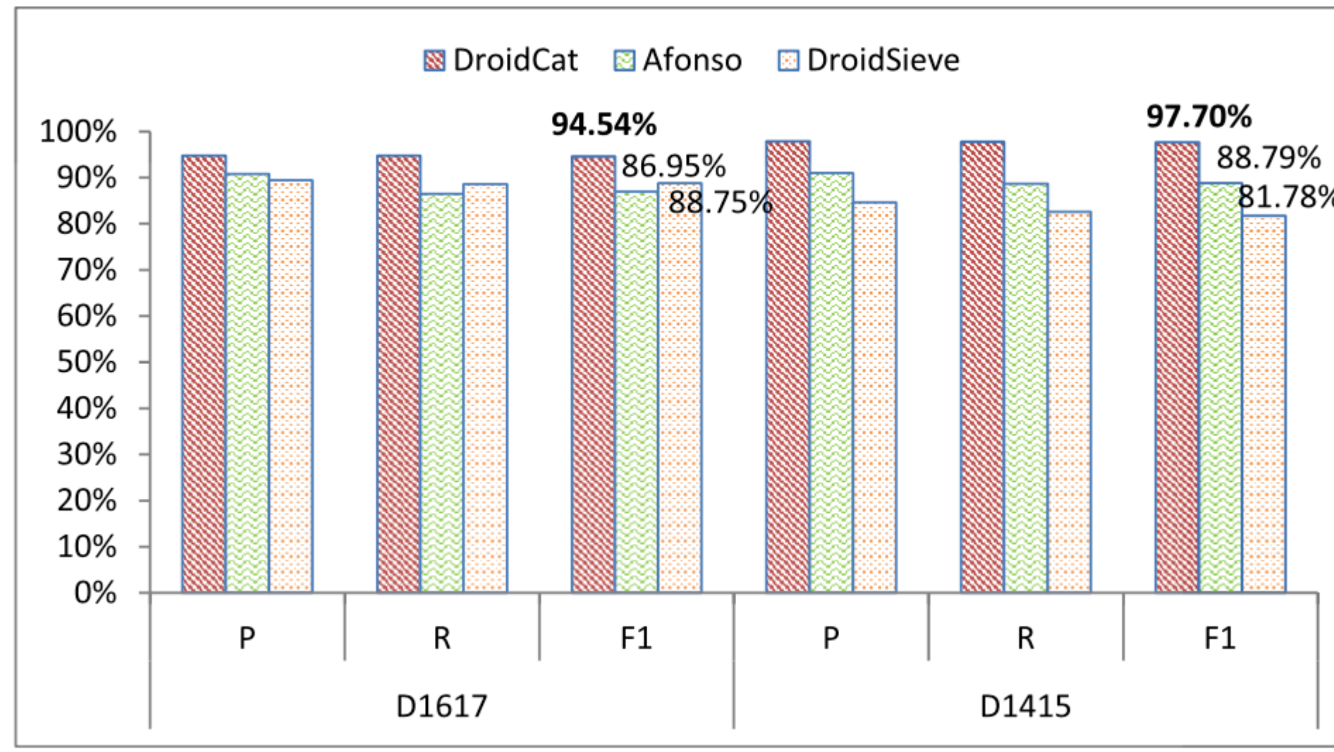



Fig. 6. *DroidCat* versus baselines for malware categorization.

# Study II: Comparative Classification Performance

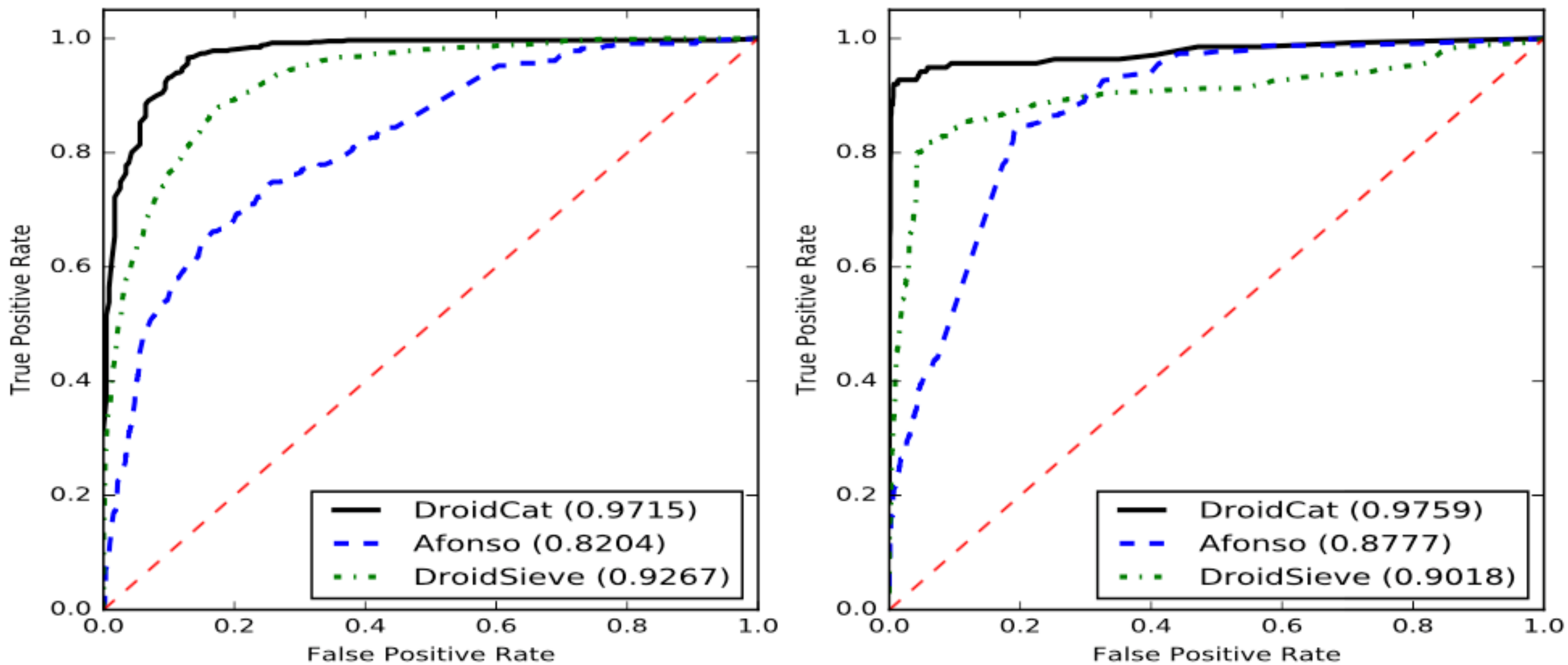



Fig. 7. ROC curves with AUCs of DroidCat versus baselines for malware detection on datasets D1415 (left) and D1617 (right).

# Study II: Comparative Classification Performance

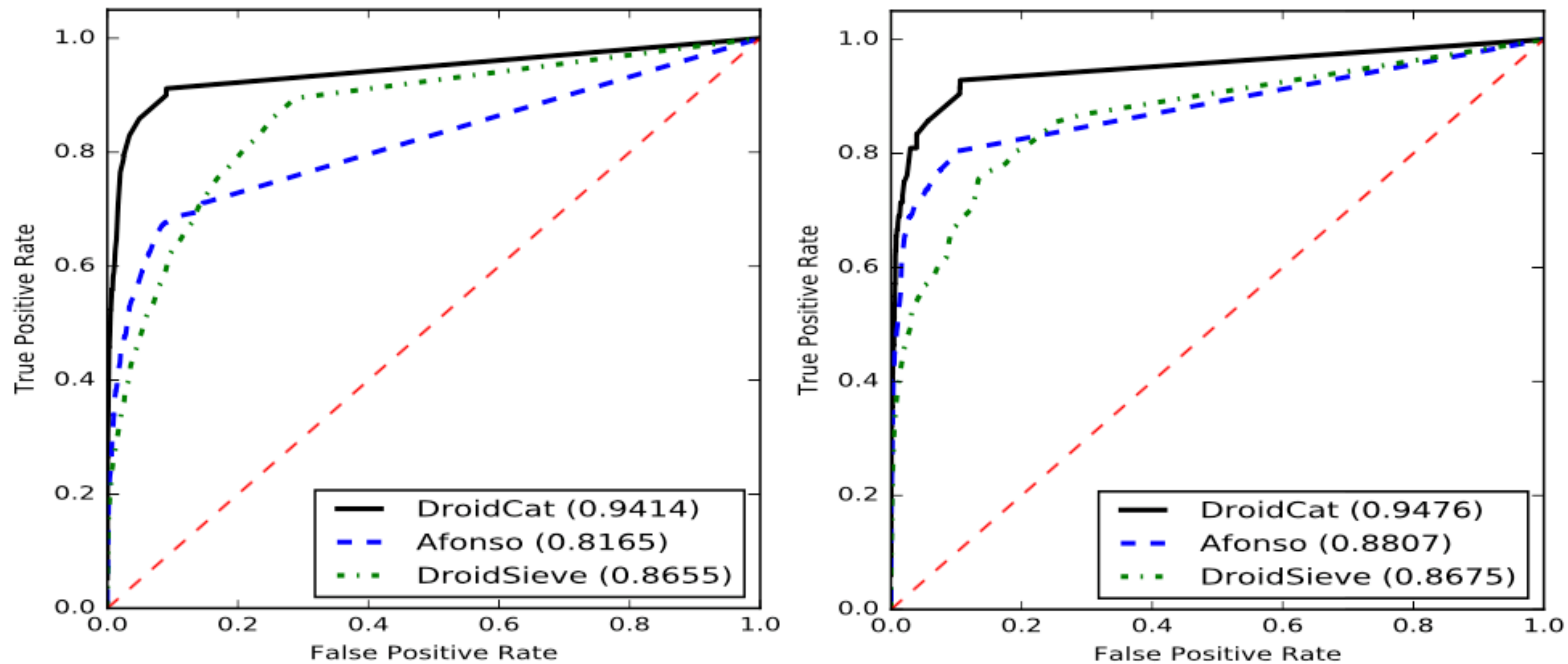



Fig. 8. ROC curves with AUCs of DroidCat versus baselines for malware categorization on datasets D1415 (left) and D1617 (right).

# Study II: Comparative Classification Performance

- DroidCat outperformed the state-of-the- art techniques compared, with up to 27% and 16% higher F1, and 0.15 and 0.13 greater AUC, for malware detection and categorization, respectively. DroidCat also appeared to be noticeably more stable than the two baseline techniques over time when achieving competitive performance.

# Study III : Robustness

| Dataset | Benign apps | | | | Malware (from *Praguard*) | | |
|---|---|---|---|---|---|---|---|
| | Period | Source | #Apps | obf% | Period | #Apps | obf% |
| OBF1617 | 2016-2017 | GP,AZ | 3,196 | 57.38% | 2010-2012 | 1,214 (59 families) | 100% |
| OBF1415 | 2014-2015 | AZ | 4,462 | 25.59% | | | |
| OBF1213 | 2012-2013 | AZ | 4,804 | 12.57% | | | |

# Study III : Robustness

TABLE V

ROBUSTNESS OF *DroidCat* VERSUS BASELINES

| Technique | Perf. | Detection | | | | Cate. |
|---|---|---|---|---|---|---|
| | | OBF1617 | OBF1415 | OBF1213 | Average | |
| *DroidCat* | P | 97.46% | 96.86% | 96.40% | 96.85% | 97.26% |
| | R | 97.34% | 96.71% | 96.19% | 96.69% | 97.07% |
| | **F1** | **97.33%** | **96.66%** | **96.13%** | **96.64%** | **97.06%** |
| *Afonso* | P | 98.43% | 71.07% | 85.37% | 83.91% | 54.55% |
| | R | 52.07% | 86.73% | 93.81% | 79.89% | 56.47% |
| | **F1** | **68.11%** | **78.13%** | **89.39%** | **79.59%** | **51.03%** |
| *DroidSieve* | P | 86.44% | 87.98% | 85.08% | 86.48% | 91.81% |
| | R | 83.34% | 85.94% | 81.77% | 83.67% | 93.49% |
| | **F1** | **80.51%** | **82.67%** | **76.09%** | **79.62%** | **92.27%** |

- DroidCat exhibited **superior robustness** to both state-of-the-art techniques compared, by achieving **96%** to **97%** F1 accuracy on malware that adopted sophisticated obfuscation schemes along with varying sets of benign apps, significantly higher than the two baselines.

# In-Depth Case Studies

- **Setup and Methodology**
  - 287 benign apps, 388 malware (characterization+2016-2017).
  - 15 popular malware families(DroidDream, BaseBridge, DroidKungFu / FakeInst, OpFake)
- **Results**
  - For malware categorization, *DroidCat* performed perfectly (with 100% F1) for the majority (11) of the (15) studied families.
  - Previous tools studied [24] achieved no more than 54% detection rate
  - In malware detection mode, *DroidCat* worked even more effectively

# In-Depth Case Studies

- **Effects of Design Factors**
  - **Feature Set Choice:** 70 features works the best; Structure features significantly better than ICC and security features.
  - **Most Important Dynamic Features:** two subcategories of *Structure* features contributed the most: (1) distribution of **method/class invocation over the three code layers**, and (2) **callback invocation for lifecycle management**.
  - **Learning Algorithm Choice:** RF, with 128 trees
  - **Input Coverage:** performance of DroidCat did not appear to be very sensitive to the user- code coverage of run-time inputs.

# Efficiency

- The primary source of analysis overhead of all the three techniques compared is the **cost for feature extraction**
- *DroidSieve* uses very large feature vectors (over 20,000 features per app — High storage require.
- *DroidSieve* was the most efficient among the three techniques — High speed.

# Limitations and Threats to Validity

- samples from each period may not be representative of the app population of that period
- learning-based malware detectors are subject to class imbalances in training datasets
- features that can be contrived, thus it may be vulnerable to sophisticated attacks such as mimicry and poisoning
- open-source might cause adversarial attacks target at features

# Conclusion

- **DroidCat:** dynamic app classification technique that detects and categorizes Android malware with high accuracy
- **Insight:** Features that capture the structure of app execution and ICC and security sensitive accesses can greatly handle the obfuscation schemes
- **Contribution:** diverse set of dynamic features, open-source code
- **Performance:** DroidCat achieved significantly higher accuracy than the peer approaches studied for both malware detection and family categorization
- **Strength:** superior *robustness* of DroidCat against analysis challenges use of evasion schemes; superior stability of our approach in achieving high classification performance
- **Weakness:** may be vulnerable to sophisticated attacks such as mimicry and poisoning

## “Thanks”

*–zzb*

# Others

- **关于DroidFax，已经做了app分类，且工具可用**
- **关于DroidCat，早在17年就已经发表成文，数据集一模一样(Virginia tech) conputer science technical reports**
- **此文是18年TIFS会议上发表**