StormDroid: A streaminglized Machine Learning-Based System for Detecting Android Malware

Sen Chen,
Minhui Xue,
Zhushou Tang,
Lihua Xu,
Haojin Zhu

Malware Detection in Android

- 1.6 million apps in Google Play Store in July 2015
    - Many more in third-party websites
- Malware Rates - Attacked devices surged 75% from 2013-2014
- Easy to publish apps in android.. 1 in 5 are malware
- Existing malware tools detect only widely known malwares
- Innovative ways in infecting devices
    - Third party developer stolen keys
    - Zero day exploits to get root access

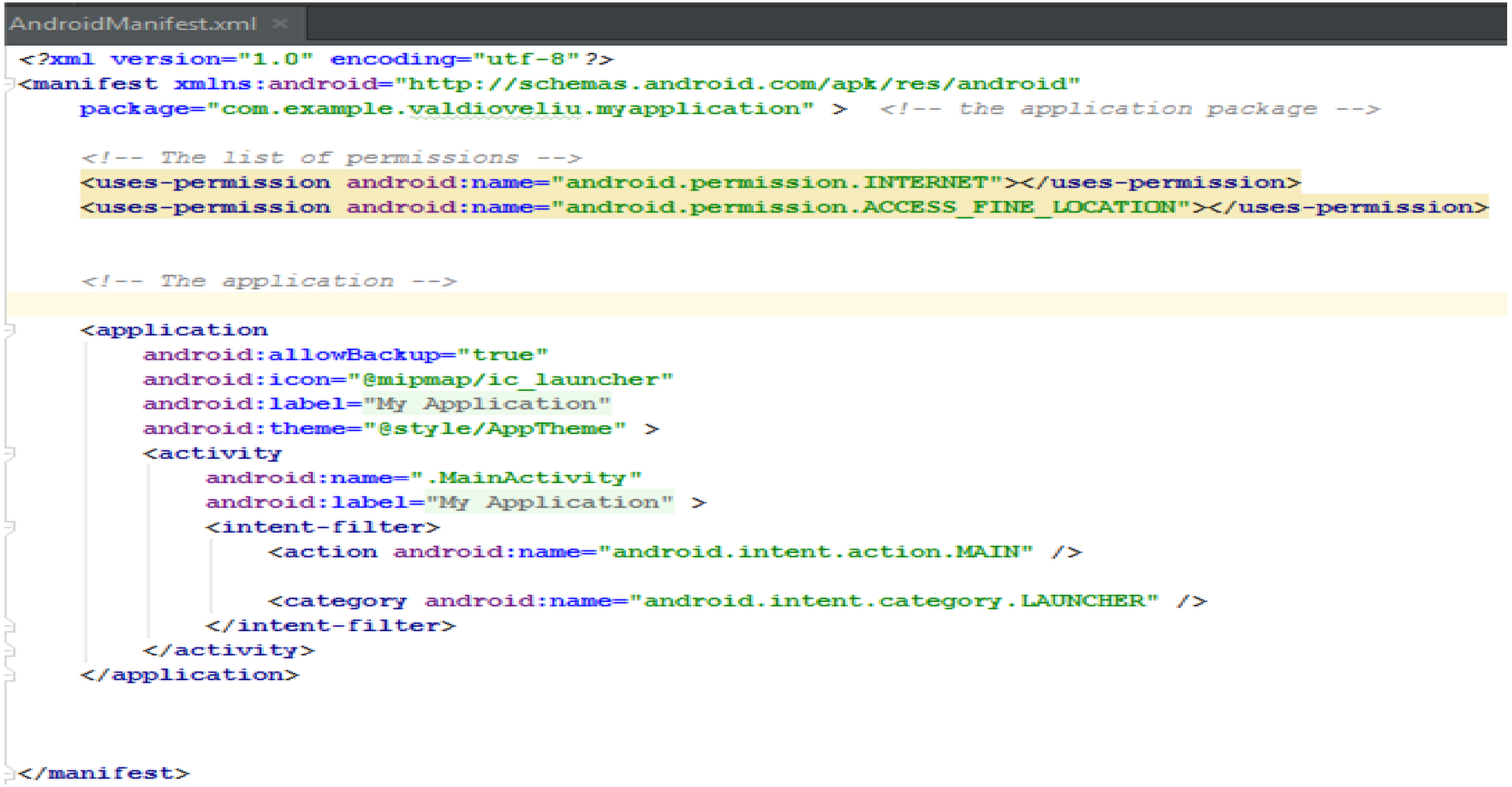
Countermeasures

- Existing countermeasures
  - Signature-based - Once Android markets find a potential malicious app, they will record its signature of the corresponding app for a more in-depth detection later.
  - Behaviour-based - prior work is mostly in Static Analysis
- Behaviour-based - StormDroid
  - Static Analysis - identifies suspicious traces of data to detect known threats
  - Dynamic analysis - Observes actual execution but leads to excessive consumption of OS

Machine Learning for Malware Detection

- Machine Learning helps sift through large sets of applications for malware detection
- Shortcomings of existing techniques in Machine Learning:
    - Features are restricted to Permissions & Sensitive API calls
    - Lack of large-scale data sets for training
    - Validation measures don’t fare well in reality - 10-fold cross validation
    - Unreasonable amount of time taken while processing a large-scale dataset

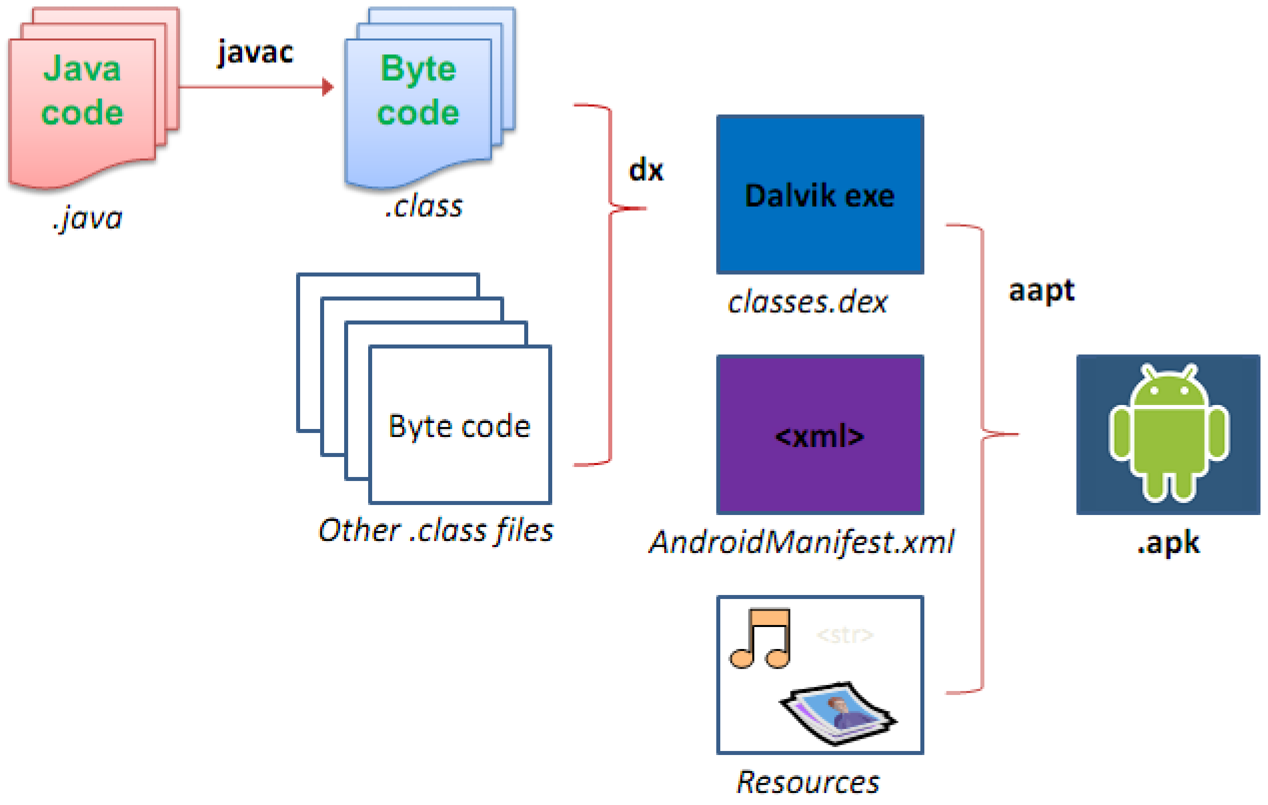
Background - Android Manifest



AndroidManifest.xml ×

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.valdioveliu.myapplication" >  <!-- the application package -->

    <!-- The list of permissions -->
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>


    <!-- The application -->

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="My Application" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>



</manifest>
```

Compiling APK



Security Approaches

- Market Protection
  - **Signing**
  - **Review by playstore**
- Platform Protection
  - **Sandboxing** - VM for each app
  - **Permissions** - either a benign or a malicious app may require the same permissions
    - New versions have dangerous permissions which aren’t granted during installation time

StormDroid Framework

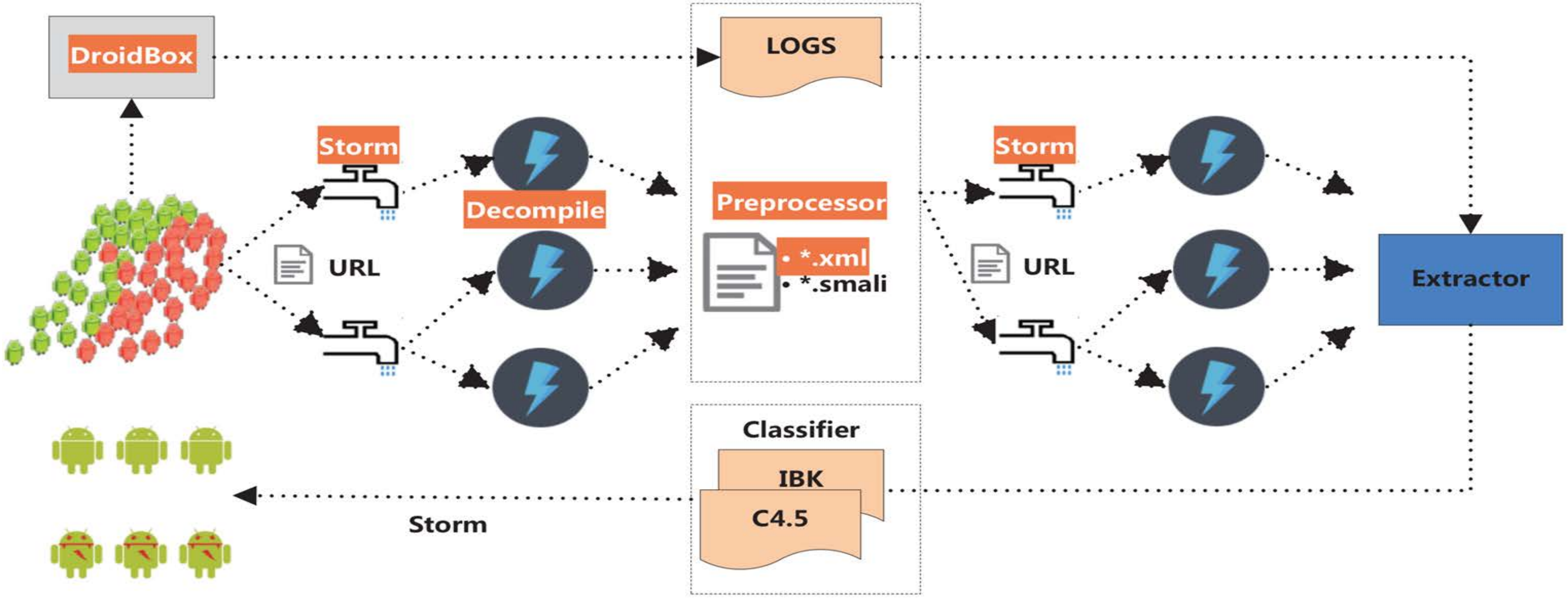


Figure 1: The *StormDroid* Framework for Android Malware Detection

StormDroid

Three phases in execution:

- Preamble - reverse engineering to get resource files
- Feature extraction - extraction of features from combined set of contributed features and creation of binary input vector
- Classification - ML models for classification of an app as benign or malicious

Framework cntd..

Work flow of the detection process is in following topology:

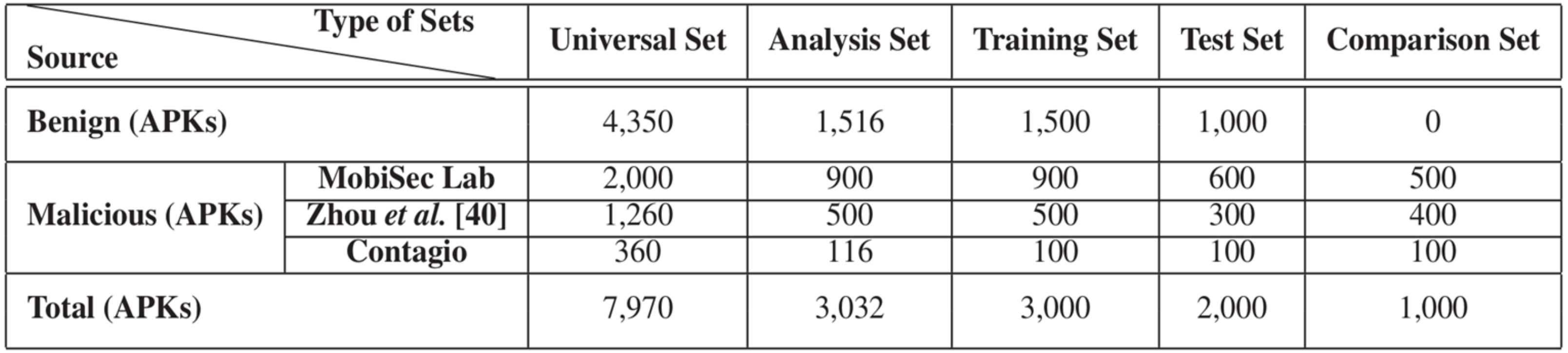
- Submitted app is first disassembled to extract its features
  - static profiling tools: apktool, dex2jar, java decompilation tool

- Differential metrics of the app are calculated
- Run intersection analysis and output a binary input vector
- All the data associated with the app are in a single stream
- Concurrently processes multiple streams
  - enables a market to efficiently detect a large number of submissions.

Classification

- Training performed on 3000 apps
- Total app samples - 7970 apk files
  - 4350 benign apps
  - 3620 malicious apps - includes phishing, trojans, spyware, root exploits

Table 1: Data Sets for Android Malware Detection



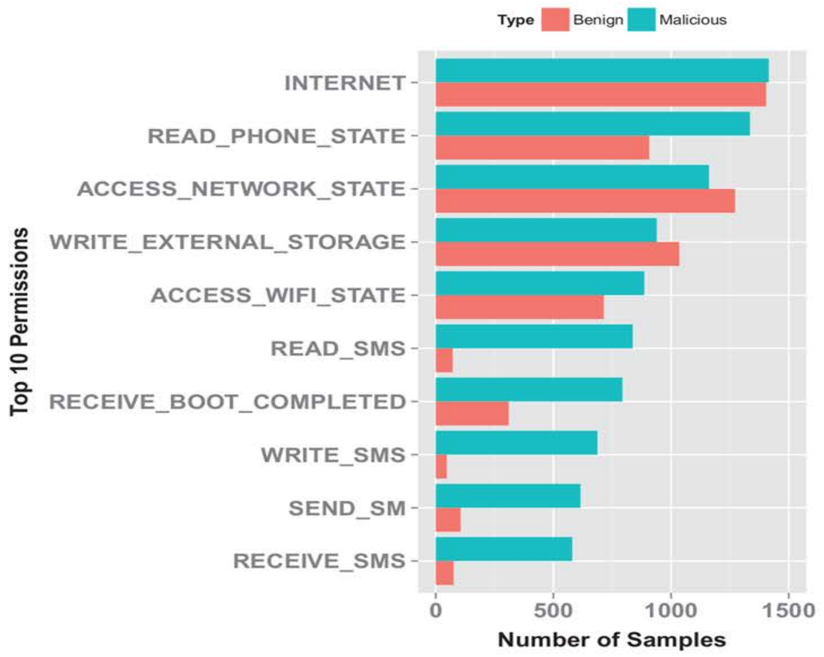
Source \ Type of Sets		Universal Set	Analysis Set	Training Set	Test Set	Comparison Set
Benign (APKs)		4,350	1,516	1,500	1,000	0
Malicious (APKs)	MobiSec Lab	2,000	900	900	600	500
	Zhou *et al.* [40]	1,260	500	500	300	400
	Contagio	360	116	100	100	100
Total (APKs)		7,970	3,032	3,000	2,000	1,000

Feature Extraction

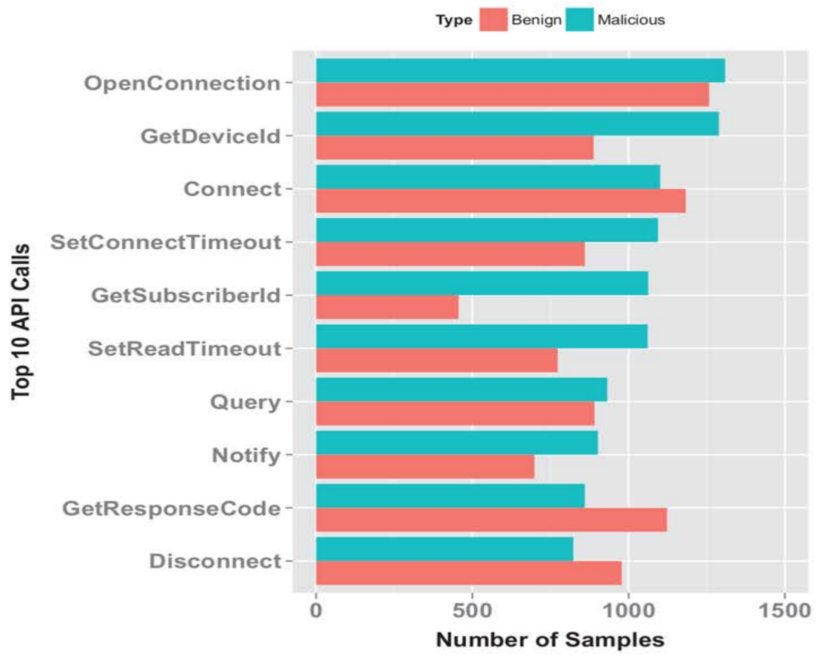
- Features
  - Well received features
    - Permissions
    - Sensitive API Calls - obtain Smali files from the static decompiling
      - Telephony
      - SMS/MMS
      - Network/Data
  - Newly-defined features
    - Sequence
    - Dynamic Behaviour

Feature extraction contd..

Permission settings & Sensitive API calls are indeed relevant to the benign or malware behaviors



(a) Comparison of top 10 requested permissions by 3,032 benign and malicious apps



(b) Comparison of top 10 requested sensitive API calls by 3,032 benign and malicious apps

Feature extraction - Sequences

- **Subtraction-Differential metric:** D1 (resp. D2) as the set of top values of d(s,m,b) (resp. d(s,b,m)) that outnumber the threshold 200

$$\begin{aligned} m_s &= \text{a malicious app } m \text{ w.r.t. \# sensitive API call } s; \\ b_s &= \text{a benign app } b \text{ w.r.t. \# sensitive API call } s; \\ d_{(s,m,b)} &= \text{difference between } m_s \text{ and } b_s; \\ d_{(s,b,m)} &= \text{difference between } b_s \text{ and } m_s. \end{aligned}$$

  ➔ **D=D1∪D2**

- **Logarithm-differential metric:** top 16 values that are greater than 0.4 (set L1) and the bottom 11 values of that are less than 0.05 (set L2)

$$\lg\left\{\left(\frac{m_s}{b_s+1}\right)+1\right\}$$

  ➔ **L=L1∪L2**

Feature Extraction - Sequences

- **Subtraction-Logarithm metric**
  - ➔$S = D \cap L$
  - ➔if the APK contains at least one of the features either in set $D1 \cap L1$ or in set $D2 \cap L2$,
    - ◆ Add weights $+(d(s,m,b)/1,516)$ or $-(d(s,b,m)/1516)$ to sum, respectively;
  - ➔if the (sum value of the set S) > 0.4, the corresponding sequence is heuristically marked as ‘1’ otherwise, it is marked as ‘0’

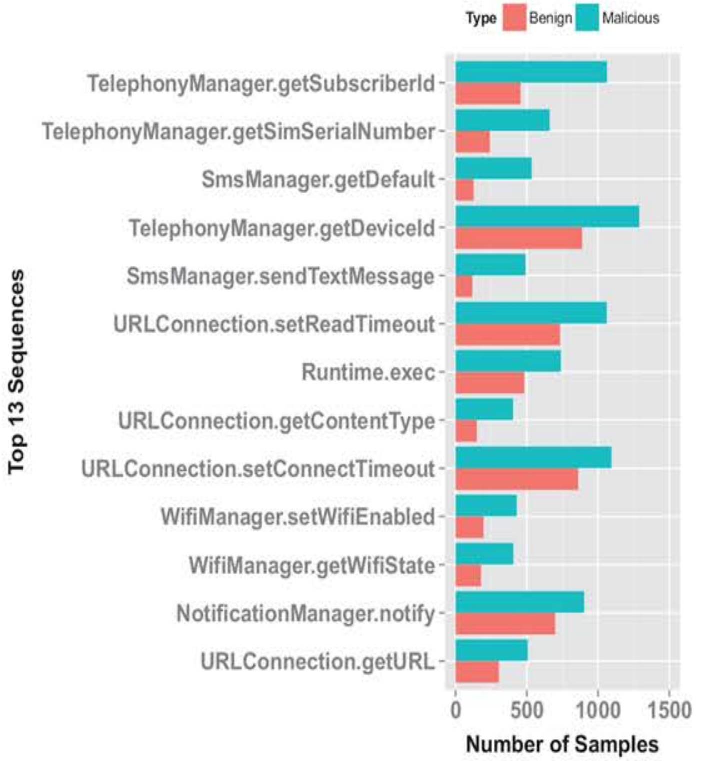


Figure 3: Top 13 differences of sensitive API calls between malicious and benign apps by 3,032 benign and malicious samples

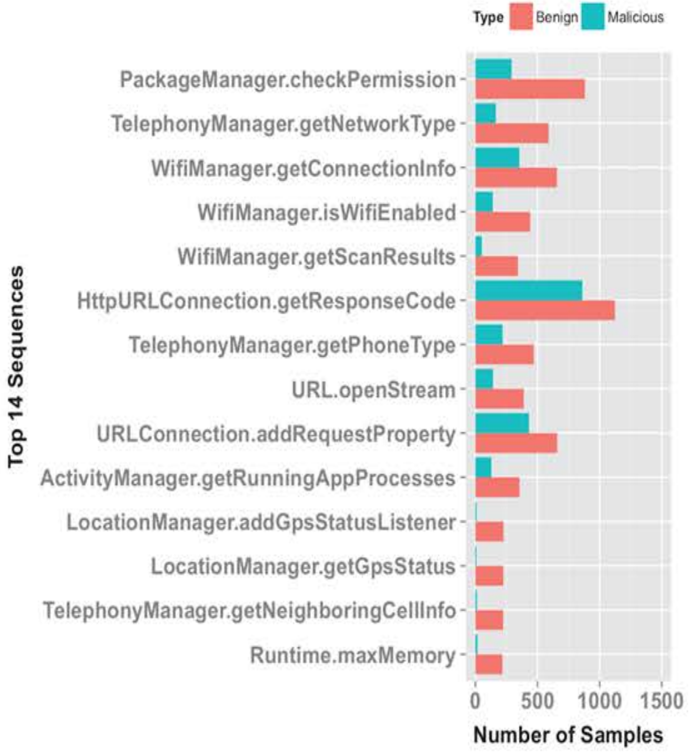


Figure 5: Top 14 differences of sensitive API calls between benign and malicious apps by 3,032 benign and malicious samples

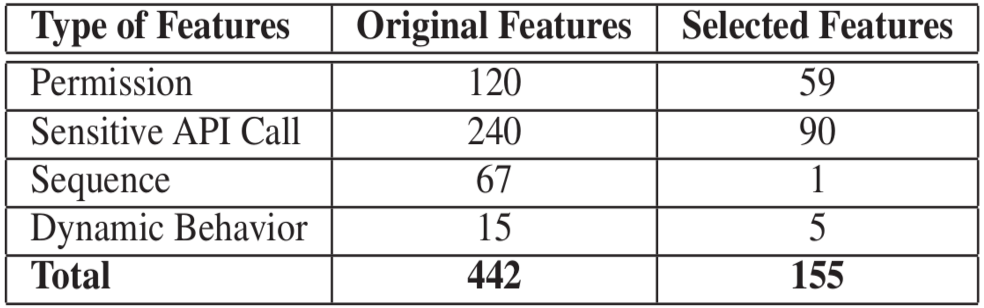
Feature extraction - Dynamic Behaviour

- Apk file is run in DroidBox 6
  - Incoming/outgoing network data
  - File read and write operations
  - Started services and loaded classes through DexClassLoader
  - Information leaks via the network, file and SMS
  - Circumvented permissions
  - Cryptography operations performed using Android API
  - Sent SMS and phone calls
  - two images showing the temporal order of the operations and a treemap to check similarity between analyzed packages.
- Static analysis of the saved log files to extract the top features of dynamic behaviors.

Feature extraction contd..

Several well-known features do not help distinguish between benign and malicious apps, which will increase system overhead. They choose 1,516 benign and malicious APKs to prune well-known features of benign and malicious apps in all categories.

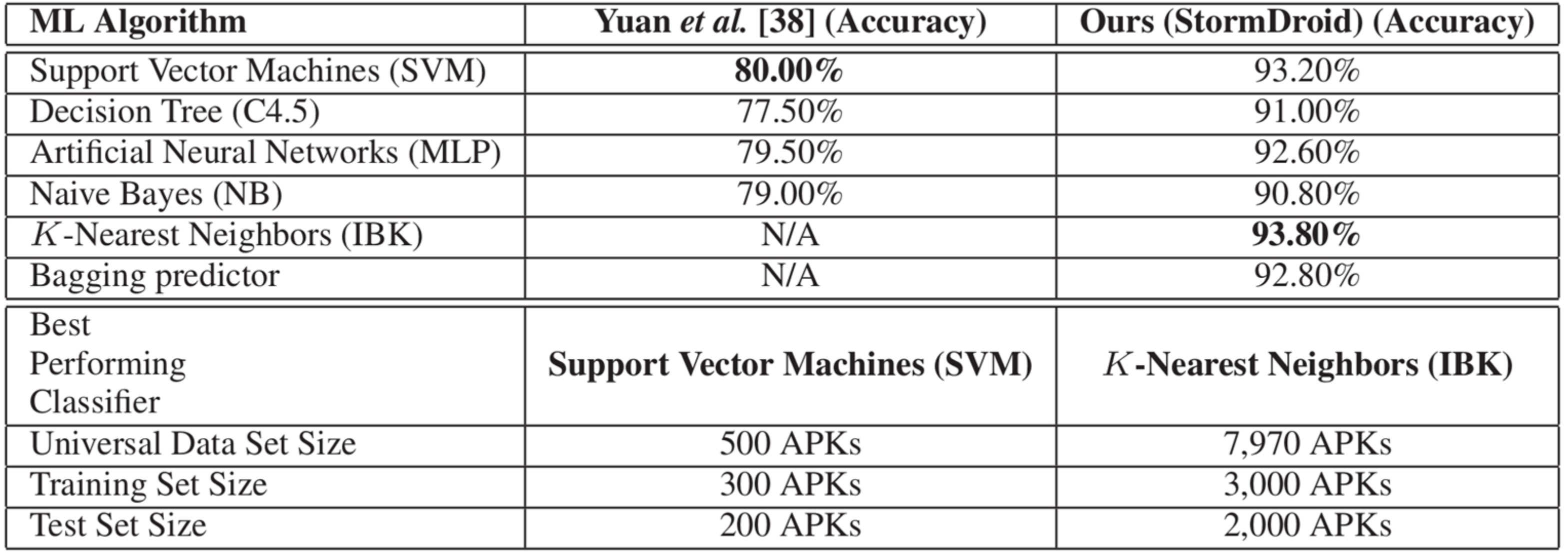
Table 2: Features for Machine Learning



Type of Features	Original Features	Selected Features
Permission	120	59
Sensitive API Call	240	90
Sequence	67	1
Dynamic Behavior	15	5
Total	**442**	**155**

Results

Table 5: Comparative results of our work and the previous work



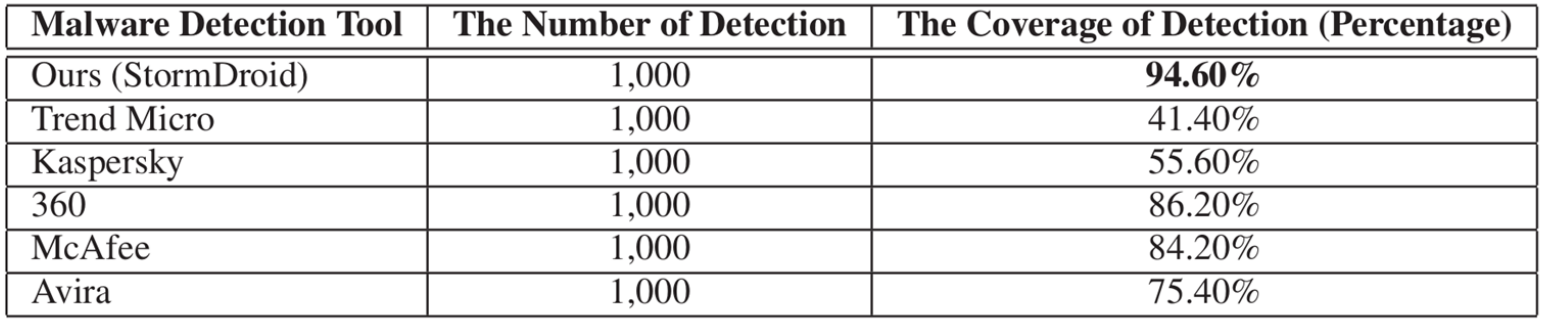
ML Algorithm	**Yuan *et al.* [38] (Accuracy)**	**Ours (StormDroid) (Accuracy)**
Support Vector Machines (SVM)	**80.00%**	93.20%
Decision Tree (C4.5)	77.50%	91.00%
Artificial Neural Networks (MLP)	79.50%	92.60%
Naive Bayes (NB)	79.00%	90.80%
K-Nearest Neighbors (IBK)	N/A	**93.80%**
Bagging predictor	N/A	92.80%
Best Performing Classifier	**Support Vector Machines (SVM)**	**K-Nearest Neighbors (IBK)**
Universal Data Set Size	500 APKs	7,970 APKs
Training Set Size	300 APKs	3,000 APKs
Test Set Size	200 APKs	2,000 APKs

Evaluation

Randomly 1000 malicious apps are chosen for comparison

- ❑ As per the authors, this helps understand coverage and avoid over-fitting

Table 6: Reference experiment: The coverage of other leading malware detection tools



Malware Detection Tool	The Number of Detection	The Coverage of Detection (Percentage)
Ours (StormDroid)	1,000	**94.60%**
Trend Micro	1,000	41.40%
Kaspersky	1,000	55.60%
360	1,000	86.20%
McAfee	1,000	84.20%
Avira	1,000	75.40%

Scalability

- StormDroid outperforms single thread by approximately three times in each group

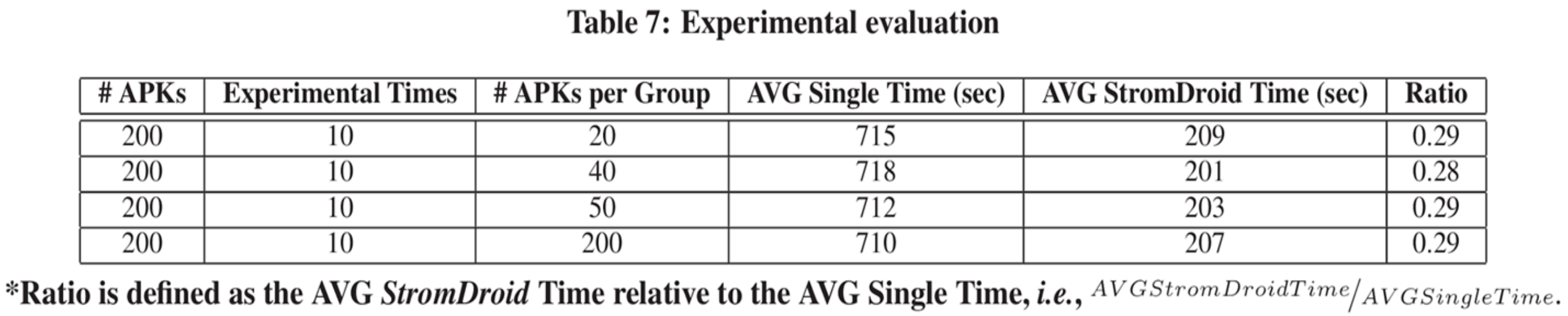


Table 7: Experimental evaluation

# APKs	Experimental Times	# APKs per Group	AVG Single Time (sec)	AVG StromDroid Time (sec)	Ratio
200	10	20	715	209	0.29
200	10	40	718	201	0.28
200	10	50	712	203	0.29
200	10	200	710	207	0.29

***Ratio is defined as the AVG *StromDroid* Time relative to the AVG Single Time, *i.e.*,** $^{AVGStromDroidTime}/_{AVGSingleTime}$.

Thoughts

- Evolving malware requires evolving malware detectors
    - Recent malware samples should be collected constantly to evolve the model
    - Attacks against learning techniques
        - Malwares can incorporate benign features to affect detection scores
        - Frequent retraining on representative datasets can mitigate such attacks

- Decompilation to source code is more difficult than to smali files
    - Repackaging doesn’t affect StormDroid
    - But even standard code obfuscation techniques makes reverse engineering very difficult. It impairs the StormDroid Framework