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
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.valdiouceliu.myapplication" ->
<!-- the application package -->

<!-- The list of permissions -->
<uses-permission android:name="android.permissionINTERNET"></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>
Compiling APK

1. **Java code** compiled to **Byte code** using `javac` to generate `.java` and `.class` files.

2. **Byte code** processed with `dx` to generate **Dalvik exe**.
   - `.class` files are renamed to **classes.dex**.
   - **AndroidManifest.xml** and **Resources** are also included.

3. **XML** files are handled by `aapt` to create **.apk** file.
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
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>
<thead>
<tr>
<th>Source</th>
<th>Type of Sets</th>
<th>Universal Set</th>
<th>Analysis Set</th>
<th>Training Set</th>
<th>Test Set</th>
<th>Comparison Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign (APKs)</td>
<td></td>
<td>4,350</td>
<td>1,516</td>
<td>1,500</td>
<td>1,000</td>
<td>0</td>
</tr>
<tr>
<td>Malicious (APKs)</td>
<td>MobiSec Lab</td>
<td>2,000</td>
<td>900</td>
<td>900</td>
<td>600</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>Zhou et al. [40]</td>
<td>1,260</td>
<td>500</td>
<td>500</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>Contagio</td>
<td>360</td>
<td>116</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Total (APKs)</td>
<td></td>
<td>7,970</td>
<td>3,032</td>
<td>3,000</td>
<td>2,000</td>
<td>1,000</td>
</tr>
</tbody>
</table>
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

  \[ m_s = \text{a malicious app } m \text{ w.r.t. } \# \text{ sensitive API call } s; \]

  \[ b_s = \text{a benign app } b \text{ w.r.t. } \# \text{ sensitive API call } s; \]

  \[ d_{(s,m,b)} = \text{difference between } m_s \text{ and } b_s; \]

  \[ d_{(s,b,m)} = \text{difference between } b_s \text{ and } m_s. \]

  \[ \rightarrow D = D_1 \cup D_2 \]

- **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)

  \[ \log \left\{ \left( \frac{m_s}{b_s + 1} \right) + 1 \right\} \]

  \[ \rightarrow L = L_1 \cup L_2 \]
Feature Extraction - Sequences

- **Subtraction-Logarithm metric**

  \[ S = D \cap L \]

  \rightarrow \text{if the APK contains at least one of the features either in set } D_1 \cap L_1 \text{ or in set } D_2 \cap L_2 ,

  \begin{itemize}
  \item Add weights \((d(s,m,b)/1516)\) or \(-(d(s,b,m)/1516)\) to sum, respectively;
  \end{itemize}

  \rightarrow \text{if the (sum value of the set } S \text{) } > 0.4, \text{ 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>
<thead>
<tr>
<th>Type of Features</th>
<th>Original Features</th>
<th>Selected Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permission</td>
<td>120</td>
<td>59</td>
</tr>
<tr>
<td>Sensitive API Call</td>
<td>240</td>
<td>90</td>
</tr>
<tr>
<td>Sequence</td>
<td>67</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic Behavior</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>442</strong></td>
<td><strong>155</strong></td>
</tr>
</tbody>
</table>
### Results

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

<table>
<thead>
<tr>
<th>ML Algorithm</th>
<th>Yuan et al. [38] (Accuracy)</th>
<th>Ours (StormDroid) (Accuracy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Vector Machines (SVM)</td>
<td>80.00%</td>
<td>93.20%</td>
</tr>
<tr>
<td>Decision Tree (C4.5)</td>
<td>77.50%</td>
<td>91.00%</td>
</tr>
<tr>
<td>Artificial Neural Networks (MLP)</td>
<td>79.50%</td>
<td>92.60%</td>
</tr>
<tr>
<td>Naive Bayes (NB)</td>
<td>79.00%</td>
<td>90.80%</td>
</tr>
<tr>
<td>K-Nearest Neighbors (IBK)</td>
<td>N/A</td>
<td>93.80%</td>
</tr>
<tr>
<td>Bagging predictor</td>
<td>N/A</td>
<td>92.80%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Best Performing Classifier</th>
<th>Support Vector Machines (SVM)</th>
<th>K-Nearest Neighbors (IBK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal Data Set Size</td>
<td>500 APKs</td>
<td>7,970 APKs</td>
</tr>
<tr>
<td>Training Set Size</td>
<td>300 APKs</td>
<td>3,000 APKs</td>
</tr>
<tr>
<td>Test Set Size</td>
<td>200 APKs</td>
<td>2,000 APKs</td>
</tr>
</tbody>
</table>
Evaluation

Randomly 1000 malicious apps are chosen for comparison

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

<table>
<thead>
<tr>
<th>Malware Detection Tool</th>
<th>The Number of Detection</th>
<th>The Coverage of Detection (Percentage)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours (StormDroid)</td>
<td>1,000</td>
<td>94.60%</td>
</tr>
<tr>
<td>Trend Micro</td>
<td>1,000</td>
<td>41.40%</td>
</tr>
<tr>
<td>Kaspersky</td>
<td>1,000</td>
<td>55.60%</td>
</tr>
<tr>
<td>360</td>
<td>1,000</td>
<td>86.20%</td>
</tr>
<tr>
<td>McAfee</td>
<td>1,000</td>
<td>84.20%</td>
</tr>
<tr>
<td>Avira</td>
<td>1,000</td>
<td>75.40%</td>
</tr>
</tbody>
</table>
Scalability

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

Table 7: Experimental evaluation

<table>
<thead>
<tr>
<th># APKs</th>
<th>Experimental Times</th>
<th># APKs per Group</th>
<th>AVG Single Time (sec)</th>
<th>AVG StromDroid Time (sec)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>10</td>
<td>20</td>
<td>715</td>
<td>209</td>
<td>0.29</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>40</td>
<td>718</td>
<td>201</td>
<td>0.28</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>50</td>
<td>712</td>
<td>203</td>
<td>0.29</td>
</tr>
<tr>
<td>200</td>
<td>10</td>
<td>200</td>
<td>710</td>
<td>207</td>
<td>0.29</td>
</tr>
</tbody>
</table>

*Ratio is defined as the AVG StromDroid Time relative to the AVG Single Time, i.e., \( \frac{\text{AVG StromDroid Time}}{\text{AVG Single Time}} \).
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