SpotCheck:
On-Device Anomaly Detection for Android

Mark Vella & Christian Colombo

SINCONF 2020
Overview

Problem:

*Mobile devices are increasingly targeted by malware, posing privacy and financial threats. App store and on-device scanning are however limited mainly due to signature-based detection.*

* A novelty detection layer is needed.

Contributions:

1) Re-purposing of Kernel Principal Component Analysis (KPCA) and Variational Autoencoders (VAE), as used for network anomaly detection (AD), for Android AD

2) A novel process memory dump approach, from which to derive app behavior, as compared to a system-call-trace baseline

3) Openly available datasets capturing benign/malicious app behaviour for both representations
State-of-the-art

App store & on-device protection

Static & dynamic analysis

Stringent permission granting

Discourage unknown sources
State-of-the-art

App store & on-device protection

Static & dynamic analysis

Malware samples

Stringent permission granting

Discourage unknown sources
SpotCheck

Known malware scan

Malware analysis

New malware/ Evasive variant

Malware Classification (Triage)

Official app store

3rd party app store
SpotCheck

Approach:
- On-device execution sampling
- Prioritize suspicious apk submission for malware triage
- Along with the suspicious execution trace

Known malware scan

Execution segment

SpotCheck: Anomaly Detection

Malware analysis

Anomalous execution trace

Malware Classification (Triage)

New malware/Evasive variant

Official app store

3rd party app store

APK

APK

APK
App behavior representation i/ii

- Process memory approach:
  - Less invasive but represents only the residue of execution – time-critical
App behavior representation

- **Call trace**
  - Linux system call histogram
  - Successfully used for malware classification
  - In-line hooking on non-rooted devices is possible

\[ x \overset{\text{def}}{=} \langle \text{accept, access, bind, chdir, ...}, \text{writev} \rangle \]

- **Process memory dump**
  - `android.content.Context.getSystemService()` manager class histogram
  - HPROF – with `android.os.Debug.dumpHprofData()`
  - ArtMethod→data_ patching possible On non-rooted devices is possible

\[ x \overset{\text{def}}{=} \langle \text{AccessibilityManager, AccountManager, ...}, \text{WindowManager} \rangle \]

- **Normalization**

\[ \hat{x} \overset{\text{def}}{=} \frac{a_i}{\|x\|_1}, \ldots, \frac{a_n}{\|x\|_1} \quad (\|\hat{x}\|_1 = 1) \]
KPCA-based AD

- **Premise for AD**
  - For learned: $\gamma, W_2$
  - The lossy inverse transform $X_n = Z_2.W_2^T$ minimizes reconstruction error only in the case datapoints are from the same distribution of $X$
  - Returns a higher reconstruction error otherwise

\[
X_m = \phi(X_n), \text{ where } m > n
\]

\[
\phi(X^T)\phi(X) = k(X) \xrightarrow{eigendecomposition} W\lambda W^{-1}
\]

where $k(x, y) \overset{def}{=} e^{-\gamma||x-y||^2}$, where $\gamma = \frac{1}{2\sigma^2} > 0$
**Premise for AD**

- For learned $\phi, \theta$: $\hat{x}^{(i)}$ is similar to $x^{(i)}$ but only if $x^{(i)}$ is derived from $P(X)$
- Similarity defined in terms of a reconstruction probability

$$P(x^{(i)}) \leftarrow \frac{1}{L} \sum_{l=1}^{L} P(x^{(i),l}; \mu_{\hat{x}^{(i),l}}, \sigma_{\hat{x}^{(i),l}}^2)$$
Dataset

Google Play

2K + 1K apps

https://github.com/mmarrkv/spotcheck_ds

2K + 1K apps

Fraida

MAT

Exerciser Monkey
dumpsys

calllog

HPROF

CSV
## Results i/ii

<table>
<thead>
<tr>
<th>Dataset / AUC ROC</th>
<th>KPCA</th>
<th>VAE*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android AD (calllog)</td>
<td>0.708</td>
<td>0.694</td>
</tr>
<tr>
<td>Android AD (HPROF)</td>
<td>0.69</td>
<td>0.712</td>
</tr>
<tr>
<td>NSL-KDD (DoS)</td>
<td>0.59</td>
<td>0.795</td>
</tr>
<tr>
<td>NSL-KDD (Probe)</td>
<td>0.821</td>
<td>0.944</td>
</tr>
<tr>
<td>NSL-KDD (R2L)</td>
<td>0.712</td>
<td>0.777</td>
</tr>
<tr>
<td>NSL-KDD (U2R)</td>
<td>0.712</td>
<td>0.782</td>
</tr>
</tbody>
</table>

- **Successful re-purposing from network AD** (An & Cho, 2015)
  - **Note:** Probe is particularly noisy on the network level

- **KPCA-HPROF**
  - F1/recall/pres – **0.88/0.97/0.8**
  - **Note 1:** 0.2 imprecision results in benign apps being sent for malware triage, rather than apps being immediately flagged as malicious
  - **Note 2:** 0.03 non-recalled malware could in reality be offset by considering multiple execution samples in a multi-device deployment setting

* Android AD topology: 50-25-2/NLL-Gaussian
Results ii/ii

- Digging deeper into Android AD using HPROF
  - Latent spaces KPCA vs VAE
Conclusion and Next steps

- We have shown that KPCA & VAE can work for Android AD
- The process memory approach is promising, and which in turn is conducive to practical implementation
- Planned experimental improvements
  - App behavior representation: timely memory dumps
    - A meet-in-the-middle with sys call traces
  - AD modeling
    - VAE – Supervised learning: a loss function that pushes the latent distribution away from labeled anomalies
- Closing the loop
  - Generate anomalous execution traces for malware sandbox triage to use
    - Static app re-writing to mark decision points close to entry point, and handler code
    - Direct sandbox execution accordingly
Q&A

Mark.Vella@um.edu.mt