How My SVM nailed your Malware
Implementing Machine Learning into Android Malware Analysis

Nikhil P Kulkarni | @nikchillz
whoami

• Nikhil P Kulkarni
• An active member of the Null (Bangalore Chapter)
• An old school Bug Bounty Hunter
• Currently working as a Security Engineer at FourthWall Security, Bangalore.
Agenda

• Introduction
• The Motive
• The Objectives & Goals
• The Methods used to obtain the motive
• Graph Kernels?
• The process
• The SVM
Flame Framework is our project that we built based on Open Source Python modules for analyzing and detecting Android malware. These modules allow to extract labeled call graphs from Android APKs or DEX files and apply an explicit feature map that captures their structural relationships. Additional modules provide classes for designing classification experiments and applying machine learning for detection of malicious structure.
Why this project?

• Some of the Obvious Reasons
  • Android being the leader in the Mobile Operating System Market and also the most targeted.
  • More than a Billion devices are running Android.
  • Extreme Digitization in the developing nations.
  • Existence of Third party Application Stores that might be hosting malicious apps.
Objectives & Goals

• Check the feasibility of the Machine Learning Algorithms for Android Malware Analysis.
• Build it using the Functional Call Graphs method.
• Computing based on the similarity between the structured objects.
Formal Problem

• Can ML be used as way to perform Android Malware Analysis?
• Trying to find the best fitting hypothesis or quadratic equations that could make the graphs into labels.
Already Available Models

• Machine Learning methods have already been tried out on Malware Analysis before.

• Unsupervised
  • K Means Clustering Algorithms
    • Ended up with Large amount of False Positives.

• Supervised
  • Sequential Minimal Optimization Neural Networks.
  • J48 Decision Tree (ID3)
  • Random Forests
The Model in a Nutshell

Collect Samples

Collecting the Malware Samples as data sets, and read their patterns.

Analyze

Analysis of the apk files are being made based on the API Calls USING THE FCG.

Learn

Find malicious patterns using Structural analysis of functional call graphs with Machine Learning.

Analyzing the malwares using Machine Learning.

The .apk file samples are collected in large numbers and are used as datasets. The datasets of .apk files include both malicious and non-malicious samples. These samples are then tested using the Machine Learning approach in order to make the machine learn about the patterns in the API Calls being made and the permissions being requested for by the app.
Training Datasets Used
A wide range of datasets being used for the Malware Analysis

The Dataset is the API Call Graphs that are generated from the APK Files that can either be malicious/non-malicious.
Parameters Collected

• Initial Parameters Collected:
  • App Name
  • Application Size
  • Calculated SHA256
  • App Type
  • Permissions
Feature Space

• Function and API Names
• Function and API Calls
• Consider this

Node A

Node B

Node C
Learning it with Graphs

• Motivation: Study the relationships between the structured objects.

• Ex: Graph Comparison Problem

G

G′
The Graph Kernels

• Weisfeiler – Lehman Graph Kernel
• Neighborhood Hash Graph Kernel
Call Graphs for the Functions
Weisfeiler – Lehman Graph Kernel

• Main Functionality why Weisfeiler – Lehman Graph was considered:
  • Sorting: To represent each node ‘v’ as a sorted list $L_v$ of its neighbours ($O(m)$)
  • Compression: Compress the list into a hash value $h(L_v)$ ($O(m)$)
  • Relabeling: Relabeling the $v$ with the $h(L_v)$ as its new node label ($O(n)$)
WLGK and its Family

- $K_{WL} (G,G')$
- $K_{WLsubtree} (G,G')$
- $K_{WLshortestpath} (G,G')$
$K_{WL} (G, G')$
KWLsubtree (G,G')
**KWLshortestpath** \((G, G')\)

- Take The shortest path as an instance.
- Compute the shortest path and take the start label.
- Then compression into the sub structure.
Drawbacks of WLGK

• Firstly Diagonal Dominance Problem.
• Secondly, It does not consider the partial similarities between sub structures.
Drawbacks of WLGK

- Feature Space associated with the graph Kernel grows exponentially.
Neighborhood Hash Graph Kernels

- Bit represented Node Labels

Original Graph

Replaced with 4 Bit Labels
Neighborhood Hash Graph Kernels

• Matching Co-efﬁcients
  • Sort Hash Values
  • Count Common Labels
  • Computing of the Coefﬁcients
FCG obtained from NHGK
## Analysis of APK Files

### APK Files List

<table>
<thead>
<tr>
<th>S.No.</th>
<th>App Name</th>
<th>Size (Bytes)</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>app1.1.0.apk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>app2.1.0.apk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>app3.1.0.apk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>app4.1.0.apk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>app5.1.0.apk</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Permissions Analysis

<table>
<thead>
<tr>
<th>S.No.</th>
<th>Permission Type</th>
<th>Sensitivity</th>
<th>Summary</th>
<th>Unknown Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>com.android.permission</td>
<td>dangerous</td>
<td>Unknown permission from android reference</td>
<td>Allows an application to view the status of all networks.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>android.permission</td>
<td>normal</td>
<td>view network status</td>
<td></td>
<td>Allows an application to create network sockets.</td>
</tr>
<tr>
<td>3</td>
<td>android.permission</td>
<td>dangerous</td>
<td>record audio</td>
<td></td>
<td>Allows application to access the audio record path.</td>
</tr>
<tr>
<td>4</td>
<td>android.permission</td>
<td>dangerous</td>
<td>record audio</td>
<td></td>
<td>Allows application to access the audio record path.</td>
</tr>
</tbody>
</table>
Extraction of API Calls
Modules & Dependencies

The Open Source Modules
- AndroGuard
- NetworkX
- MatPlot Lib
- NumPy & SciPy
- Scikit-Learn
- CPickle

• AndroGuard
• NetworkX
• MatPlot Lib
• NumPy & SciPy
• Scikit-Learn
• CPickle
The Support Vector Machine
Doing it the SVM Way

- malicious
  - Non-malicious
SVM 101

- Malicious
- Non-malicious
Doing it the SVM Way

- malicious
  - non-malicious
You could do it this way too

- Malicious
  - Non-malicious
Holy Zucks...!!!, a misclassified node

- malicious
- Non malicious

I got misclassified
Defining the Margin

- Malicious
  - Non-malicious
Maximizing the Margin

- Malicious
  - Non-malicious

These are the Support Vectors datapoints that the margin pushes up against
## Results

As per the Training Set fed into the framework

### Confusion Matrix

<table>
<thead>
<tr>
<th>True Condition</th>
<th>Predicted Condition</th>
<th>PREDICTED CONDITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condition Positive</td>
<td>Predicted Condition Positive</td>
<td>TRUE POSITIVE (TP) Actual Malicious files that were correctly classified as Malicious</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FALSE NEGATIVE (FN) Malicious files that were incorrectly classified as Non-Malicious</td>
</tr>
<tr>
<td></td>
<td>Predicted Condition Negative</td>
<td>750</td>
</tr>
<tr>
<td>Condition Negative</td>
<td>Predicted Condition Positive</td>
<td>FALSE POSITIVE (FP) Non-Malicious files that were incorrectly classified as Malicious</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRUE NEGATIVE (TN) All the remaining files, that were correctly classified as Non-Malicious</td>
</tr>
<tr>
<td></td>
<td></td>
<td>27</td>
</tr>
</tbody>
</table>

### Accuracy

- **78%**

### False Positives

- **3%**

**Accuracy**

MALICIOUS AND NON-MALICIOUS THAT WERE RIGHTELY CLASSIFIED

FALSE POSITIVES

NON-MALICIOUS FILES THAT WERE INCORRECTLY CLASSIFIED AS MALICIOUS
Our Process

Procedure we followed while designing the Analyzer

- Data Analysis
- Talk to Experts
- Revision
- Fixes in Ideas
- Code Review
- Revise
- Project Brief
- Research
- Ask the Community
- Improvements
- Development
- Learning the Industry Standards
- Launch

Colors:
- Meet
- Community Outreach
- Improvements
- Development
- Fixes
- Project Launch
Conclusion

• Machine Learning Algos could be used for Malware Analysis, but as a complimentary feature to the Dynamic Analysis.
• Getting Feature Space right is indeed a Big Deal.
• Needs high Computation Speed and Processing Power.
• These models can be generalized to most adware with a few extra features, but it does would need some more research.
• It still has got its own drawbacks in terms of considering what kind of obfuscation level this would be able to dethrow.
The Team

MANVI SETH
@manviseth

MUDIT SAXENA
@muditsaxena1

PIYUSH BAIVAW
@piyushbaivaw

NIKHIL.P.KULKARNI
@nikchillz
References

- [http://prosec-project.org/docs/2013b-aisec.pdf](http://prosec-project.org/docs/2013b-aisec.pdf)
- [https://orbilu.uni.lu/bitstream/10993/17251/1/history_matters.pdf](https://orbilu.uni.lu/bitstream/10993/17251/1/history_matters.pdf)
Thank You