API CALL SEQUENCE ANALYSIS FOR MALWARE DETECTION

Zeynep ALTIPARMAK, Hatice Medine ÇAYIR

Computer Engineering, Adana Alparslan Turkes Science and Technology University, Adana, TURKEY

ABSTRACT

In the age of constantly developing technology, the number of uses of smart phones is increasing day by day. The newly developed devices aim to provide a more practical and easy life to their users. Android, the most popular smartphone operating system currently in use, is becoming the target of malicious people due to its open source. Detection of malware is important to prevent privacy, security and commercial losses. In this study, it is aimed to detect malware by examining Android API call directories.

Keywords: static analysis, android, malware detection, classification.

1. INTRODUCTION

In the age of constantly developing technology, the number of uses of smart phones is increasing day by day. The newly developed devices aim to provide a more practical and easy life to their users. Developed by Google, Android OS is the most popular smartphone operating system currently in use. According to statistics data, there are currently 2 billion Android users in the world. This means that Android has a 72.84% market share.[1] However, the open-source nature of the android platform makes it a target by malicious people.[2] In addition to causing privacy and security vulnerabilities, malware can also cause commercial damage. Users do not know whether the applications they install on their smartphones and the permissions they encounter are malicious or benign. There have been many different researches on Android mobile security and it is still a subject that is being researched. While developing these researches, static, dynamic and hybrid analysis methods are used. In this research, a static analysis was made by examining API call sequences for the detection of Android malware.

2. MATERIALS AND METHODS

2.1 Dataset

In this study, a data set containing 50 malware APKs and 50 benign APKs from Github repo[3] and instructor's archieve.

2.2 Analysis methods

Approach	Advantages	Disadvantages
Static Analysis	Fast and safe. Low resource consumption. Multipath malware analysis. More secure than dynamic analysis. High accuracy.	Can't analyse obfuscated and encryption malware. Can't detect unknown malware.
Dynamic Analysis	Can analyse obfuscated and encryption malware. Better accuracy than static analysis. Can Detect both known and unknown malware.	Slow and unsafe. High resource consumption. Time-consuming and vulnerable. Limited to code reachability.
Hybrid Analysis	Better than static and dynamic analysis. Highest accuracy.	More time and resource consuming. Highest complexity.

Fig.1. Analysis differences [4]

3. EXTRACT APK FILES AND ANAYLYSIS

In order to analyze the call sequences of the APIs, we had to analyze each APK file. We used the "androguard" tool while doing these operations. Thanks to androguard, we were able to access all methods, API labels, API ids and call sequences in APIs.

Although Androguard provided us with a lot of information, the callgraph files we extracted using this tool were too large and unsuitable for processing.

For this reason, by changing the tool we use, we were able to see the API calls and the relationship between them through a script used with DroidASAT[5]. In addition, the callgraphs we extracted using this script were more convenient to process and the file size was smaller. Therefore, our transactions have become even easier.

We ran this script that our instructor shared with us with each apk file and we obtained an API call list with a hierarchical structure showing the APIs that each API calls.

```
Children of <org.andengine.engine.options.AudioOptions: boolean needsMusic()>:
       <org.andengine.engine.options.MusicOptions: boolean needsMusic()>
Children of <org.andengine.engine.options.TouchOptions: boolean needsMultiTouch()>:
Children of <org.andengine.util.color.Color: int getARGBPackedInt()>:
       <org.andengine.util.color.ColorUtils: int convertRGBAToARGBPackedInt(float,float,float,float)>
Children of <org.andengine.util.color.Color: int getABGRPackedInt()>:
Children of <org.andengine.util.color.Color: void set(float,float,float,float)>:
       <org.andengine.util.color.Color: void packABGR()>
Children of <org.andengine.extension.ui.livewallpaper.BaseLiveWallpaperService: void onDestroy()>:
       <org.andengine.extension.ui.livewallpaper.BaseLiveWallpaperService: void onDestroyResources()>
       <org.andengine.extension.ui.livewallpaper.BaseLiveWallpaperService: void onGameDestroyed()>
       <org.andengine.util.debug.Debug: void e(java.lang.String,java.lang.Throwable)>
       <android.service.wallpaper.WallpaperService: void onDestroy()>
       <org.andengine.engine.Engine: void onDestroy()>
```

We processed these API list files, which we obtained separately for each APK, with a code we wrote through Python, and gave each API a unique ID and wrote this API-ID pair list to a file for later use.

```
36 org.apache.harmony.security.asn1.BerInputStream: int next()
37 java.lang.System: java.lang.String getProperty(java.lang.String,java.lang.String)
38 air.Mobilependol.AppEntry: void onPrepareDialog(int,android.app.Dialog,android.os.Bundle)
39 air.Mobilependol.AppEntry: void finishFromChild(android.app.Activity)
40 libcore.util.ZoneInfoDB: void readIndex()
41 java.net.InetAddress: java.net.InetAddress[] lookupHostByName(java.lang.String)
42 java.lang.Character: boolean isISOControl(char)
```

Using the API-ID pairs we obtained from the API call files we extracted for each APK, we re-examined them with a Python code and transformed each sequence into a tree. By identifying the API calls contained in these trees with their IDs, we listed our API sequences containing API IDs for each APK.

```
458 6290-->6714-->6902-->3179-->3247
459 6290-->6714-->6902-->6992-->7025
460 6290-->6714-->6902-->6992-->7026
461 6290-->6714-->6902-->6994-->7039
462 6290-->6714-->6902-->6992-->7027-->7048
463 6290-->6714-->6902-->6992-->7027-->7049
464 6290-->6714-->6902-->6993-->7029-->7059-->7062
465 6290-->6714-->6902-->6993-->7029-->7059-->7061-->8735
466 6290-->6714-->6902-->6993-->7029-->7059-->7061-->7064
467 6290-->6714-->6902-->6993-->7029-->7059-->7061-->2504
468 6290-->6714-->6902-->6993-->7029-->7059-->7061-->3416
469 6290-->6714-->6902-->6993-->7029-->7059-->7061-->8736
470 6290-->6714-->6902-->6993-->7029-->7059-->7061-->9314
```

We made API sequence lists specific to each APK separately for benign and malware APKs. We obtained common API sequences by comparing each malware API sequence list with other malware API sequence lists. We obtained common benign sequences by applying the same procedure to benign samples.

```
45-->435-->1313-->1851
45-->435-->1315-->1976-->2407
45-->435-->1315-->1976-->2408-->2668
45-->435-->1315-->1976-->2408-->2669
45-->435-->1315-->1976-->2408-->2671
45-->435-->1315-->1976-->2408-->2667-->2702
45-->435-->1315-->1976-->2408-->2670-->2810
45-->435-->1315-->1976-->2408-->2672-->2856
```

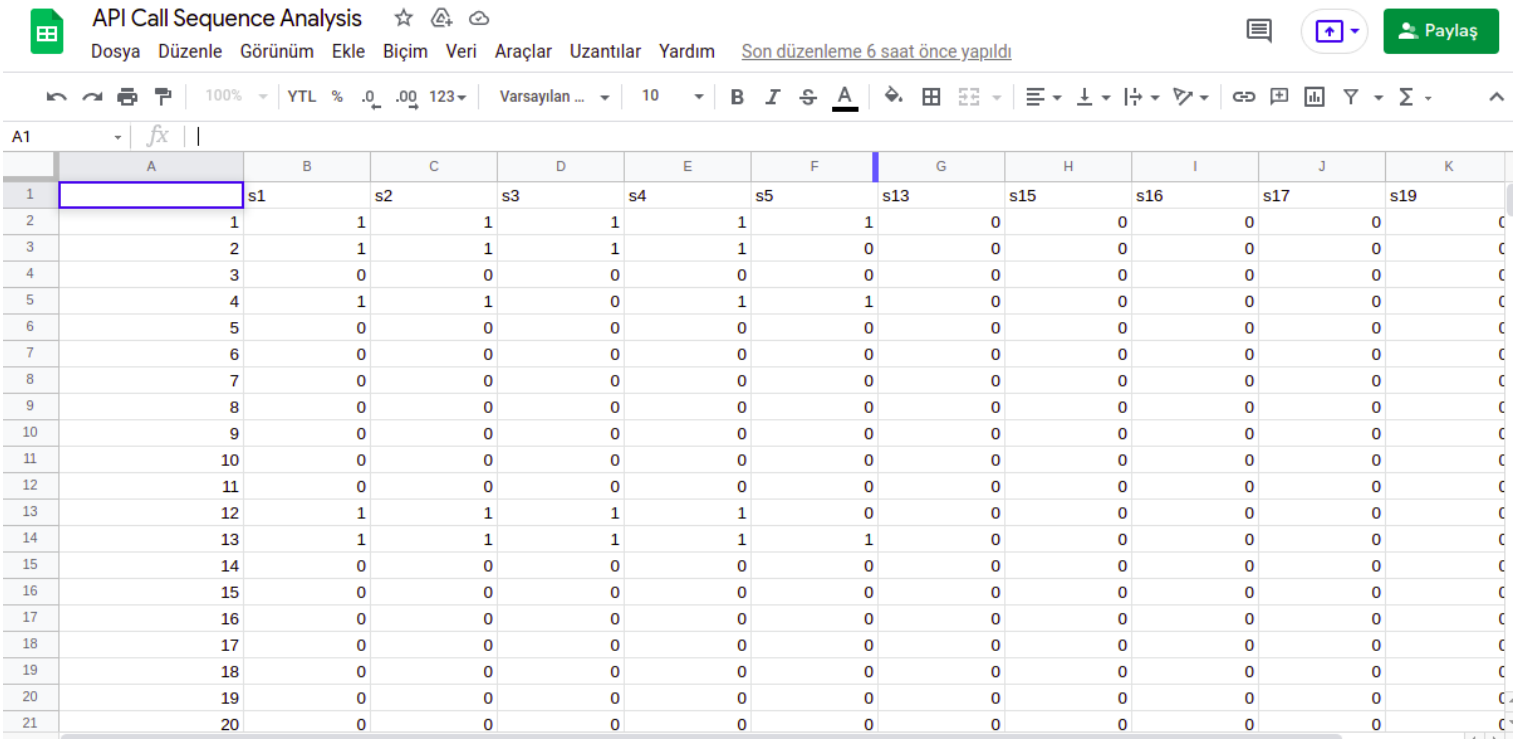
By comparing the common sequence lists we obtained separately for malware and benign samples, we obtained common API sequences of malware-benign samples, API sequences specific to malware samples and API sequences specific to benign samples.

In order to make feature selection using all the API data we obtained, we focused on each API by considering the number of API call sequences found in APKs. We determined our features by choosing our high-weight API calls.

```
('218-->777-->1286-->1959\n', 21)
('277-->904-->4008-->4032-->1842-->2187-->2606-->2764-->3026\n', 1)
('14844-->14924\n', 1)
('13889-->5896-->6289\n', 3)
('157-->682\n', 21)
('232-->819-->1153-->1937-->2418-->2496-->2857\n', 6)
('4541-->571\n', 1)
('191-->736-->1479-->1780-->2392-->2542-->2733-->3058\n', 15)
('88-->508\n', 21)
('13893-->13984-->14935-->14957\n', 1)
('9404-->4945-->5365-->6489\n', 1)
('13872-->378-->1516-->1916-->2328\n', 3)
('345-->1063-->1188-->1860-->2200-->2677-->2868-->2967-->3103-->3177\n', 15)
```

4. RESULTS

The accuracy score of the classification was calculated using the Naive Bayes, Random Tree and Random Forest classification algorithms in the WEKA tool of the CSV file created using 92 instances and 10 features. These scores are 95%, 94% and 95%, respectively.



API Call Sequence Analysis

Dosya Düzenle Görünüm Ekle Biçim Veri Araçlar Uzantılar Yardım Son düzenleme 6 saat önce yapıldı

Paylaş

	A	s1	s2	s3	s4	s5	s13	s15	s16	s17	s19
2	1	1	1	1	1	1	0	0	0	0	0
3	2	1	1	1	1	0	0	0	0	0	0
4	3	0	0	0	0	0	0	0	0	0	0
5	4	1	1	0	1	1	0	0	0	0	0
6	5	0	0	0	0	0	0	0	0	0	0
7	6	0	0	0	0	0	0	0	0	0	0
8	7	0	0	0	0	0	0	0	0	0	0
9	8	0	0	0	0	0	0	0	0	0	0
10	9	0	0	0	0	0	0	0	0	0	0
11	10	0	0	0	0	0	0	0	0	0	0
12	11	0	0	0	0	0	0	0	0	0	0
13	12	1	1	1	1	0	0	0	0	0	0
14	13	1	1	1	1	1	0	0	0	0	0
15	14	0	0	0	0	0	0	0	0	0	0
16	15	0	0	0	0	0	0	0	0	0	0
17	16	0	0	0	0	0	0	0	0	0	0
18	17	0	0	0	0	0	0	0	0	0	0
19	18	0	0	0	0	0	0	0	0	0	0
20	19	0	0	0	0	0	0	0	0	0	0
21	20	0	0	0	0	0	0	0	0	0	0

Fig. 2. CSV file

```
Correctly Classified Instances          87               95.6044 %
Incorrectly Classified Instances         4                4.3956 %
Kappa statistic                          0.9119
Mean absolute error                      0.0483
Root mean squared error                  0.2068
Relative absolute error                  9.6511 %
Root relative squared error             41.3338 %
Total Number of Instances               91

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.979     0.068     0.939      0.979    0.958      0.913    0.962     0.952     M
               0.932     0.021     0.976      0.932    0.953      0.913    0.962     0.974     B
Weighted Avg.  0.956     0.046     0.957      0.956    0.956      0.913    0.962     0.962

=== Confusion Matrix ===

  a  b   <-- classified as
 46  1 |  a = M
  3 41 |  b = B
```

Fig. 3. Naive Bayes classification

```
Correctly Classified Instances          86               94.5055 %
Incorrectly Classified Instances         5                5.4945 %
Kappa statistic                          0.8898
Mean absolute error                      0.0847
Root mean squared error                  0.2299
Relative absolute error                 16.9424 %
Root relative squared error             45.9464 %
Total Number of Instances               91

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.979     0.091     0.920       0.979   0.948      0.892    0.948     0.938     M
               0.909     0.021     0.976       0.909   0.941      0.892    0.948     0.946     B
Weighted Avg.  0.945     0.057     0.947       0.945   0.945      0.892    0.948     0.942

=== Confusion Matrix ===

  a  b   <-- classified as
 46  1 |  a = M
  4 40 |  b = B
```

Fig. 4. Random Tree classification

```
Correctly Classified Instances          87               95.6044 %
Incorrectly Classified Instances         4                4.3956 %
Kappa statistic                          0.9119
Mean absolute error                      0.0869
Root mean squared error                  0.2168
Relative absolute error                 17.3878 %
Root relative squared error             43.3298 %
Total Number of Instances               91

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.979     0.068     0.939       0.979   0.958      0.913    0.957     0.947     M
               0.932     0.021     0.976       0.932   0.953      0.913    0.957     0.969     B
Weighted Avg.  0.956     0.046     0.957       0.956   0.956      0.913    0.957     0.957

=== Confusion Matrix ===

  a  b   <-- classified as
 46  1 |  a = M
  3 41 |  b = B
```

Fig. 5. Random Forest classification

In future studies, extensive research can be developed on the classification of malware and being APKs using different types of analysis, using more sample sets and more features.

REFERENCES

[1] https://www.statista.com/topics/876/android/#dossierKeyfigures

[2] KABAKUŞ, Abdullah Talha, İbrahim Alper DOĞRU, and Aydın ÇETİN. "Android kötücül yazılım tespit ve koruma sistemleri." Erciyes Üniversitesi Fen Bilimleri Enstitüsü Fen Bilimleri Dergisi 31.1 (2015): 9-16.

[3] https://github.com/ashishb/android-malware

[4] Analysis and Classification of Android Malware using Machine Learning Algorithms, Sharma, Challa

[5] Alam, Shahid, Soltan Abed Alharbi, and Serdar Yildirim. "Mining nested flow of dominant APIs for detecting android malware." Computer Networks 167 (2020): 107026.