Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Martin Kanev
11th of May 2021

Machine Learning for Android Malware Detection Using Permissions

Abstract

The massive number and growth of Android devices leads to an increase in the potential threat from malicious applications. The traditional security methods are rendered ineffective towards the new and more sophisticated malwares. It still remains a challenge to find an effective malware detection mechanism. Therefore, in this paper it is proposed an approach that focuses on one of the main lines of defenses for Android systems- namely the permissions' system. This project looks at a large data set of 2000 applications of a small size (<300KB) and trains three different classification algorithms: Support Vector Machine, Random forest and Naïve Bayes. The first two show a promising predictive power of 77% and 75%, respectively, and not so good performance from the third (57%). All three methods are compared and a set of important permission features is presented.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and

include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Contents

Abstract 2

Introduction 6

Background 7

Android application 8

Security approaches 8

Market protection 8

Platform protection 9

Data set 9

Overview 9

Methodology 9

Preamble 9

Feature Extraction 10

Embedding in a vector 10

Classification 10

Evaluation. 11

Limitations 14

Related work 14

Future work 15

Conclusion 15

References 17

Appendices 20

Table of contents for the zip file 20

Gantt chart 20

Introduction

Nowadays we have reached a point in time, where almost everyone who lives in a developed country has a smart phone. The most common operating system for mobile devices is Android. Since, we have come to a point where we rely on our mobile devices so much, many people with malicious intents have focused their resources in compromising our devices in some way. Today, we trust our smart phones for storing our private data: emails, SMS messages, appointments, pictures, files. We also use our phones for bank authentication and many other work-related activities. Android users can download apps from Google play store, but they can also download them from other third-party markets. Despite the marketplace, malicious applications exist, and we are vulnerable to them. Our exposure to a malign software has increased immensely throughout the years [26].

It has become increasingly important for our devices to be secure [11]. In recent years the common countermeasure is using one of the following two techniques for malware detection: signature-based or behavior-based. This work will focus on the signature-based approach, also known as the static analysis one. Static analysis consists of observing unpacked apps statically for identifying and detecting suspicious traces of data. While this method it is useful for detecting threats, it relies on existing known treats and therefore its main drawback is its difficulty of detecting malicious apps that contain a zero-day attacks [10]. The other approach, behavior-based, mainly refers to the dynamic analysis, where the application is being observed during its execution. However, this approach also has a drawback - dynamic analysis can be very computationally expensive and might not be feasible for many Android devices [4]. On top of this, users want to preserve their battery of their devices and many people avoid applications that are power intensive.

Trying to cope with these issues, many machine learning solutions have been proposed. Using a data set of applications to detect malicious apps based on common feature [6]. However, there are a couple of problems that hinder this type of approach. First, it is not easy to identify a good set of features, for now the most common is to use some type of a combination of Android permissions and API calls [19][24]. The second, problem is that some of the providers for data sets for malware and benign apps have stopped maintaining their data collections and have stopped providing their sets. In addition, only a few providers of large data sets are available. An example of a work that has encountered this problem is Yuan *et al*. [23] where they use only a small number of apps (500) for its training and testing.

Android as a platform has a few security mechanisms. One of the most famous ones is using the Android permission system. For an application to do a certain task on your device, such as fixing your calendar, accessing your gallery, or sending an SMS message, it requires the user's permission to do so. However, people have the

tendency to accept these permissions without even much reading them [12]. This tendency has become a fact, because many of the permission requests look exactly the same. Therefore, malicious apps are hardy constrained by the Android permission system.

Considering the abovementioned challenges and the problem with the permissions, I have decided to focus my work onto using different machine learning methods for classifying lightweight apps (<300 KB) as malicious or benign based on permissions and then comparing the different machine learning algorithms. My proposed approach follows 4 basic stages. *Preamble*, where the apps are being prepared for feature extraction. *Feature extraction*, where the permissions are being extracted. *Classification*, where the different machine learning algorithms are used. *Evaluation*, comparison of the performance metrics of the machine learning algorithms. The work has been inspired from different works that have been using either permissions or permissions and some other feature set for the classification. They are later discussed in the Related work section.

A short summary of the contributions that have been achieved by the project are:

- To the best of my knowledge the other approaches that have focused on permissions only, have used only one machine learning algorithm for their classification. Whereas this project has shown how a few algorithms perform on the same data set.
- A further investigation into the area of malware detection using machine learning.
- A lightweight approach, using the full set of permissions extracted.

The approach that was taken relies on static analysis and it is prone to obfuscation technique and dynamically loaded malicious code. These limitations are further discussed in the Limitations section.

The rest of the report will be divided into the following sections. Background, Data set, Overview, Methodology, Limitations, Related work, Future work and Conclusion.

Background

This section will provide an overview of the Android application and the APK resource files. It will also outline two of the most notable Android OS security approaches.

Android application

The application is built from four different types of components: Activities, Services, Broadcast Receivers and Content Providers. Activities are responsible for the GUI (graphical user interface), they are responsible for the interactivity with the user. Services and Broadcast Receivers are working in the background providing functionality. Content Providers manage access to a central repository of data.

Android apps are written in java and then together with data and resource files are compiled into a single archive file (APK file). Android devices use the APK to install the wanted application.

An APK file has different components:

- XML manifest file with information about the application, and the components declaration, such as: Activities, Services, Broadcast Receivers and Content Providers and the permissions
- Classes.dex – the .java files are compiled to .dex file which is Davlik executable file that runs in its own instance of a Davlik Virtual machine
- /res directory containing images, icons...
- /lib directory for compiled code
- /META-INF directory having the list of resources, app certificate…
- Resources.arsc – a compiled resource file.

Security approaches

Market protection

No market can guarantee that the apps that are in it are benign. However, there are two main approaches for market protection that try and keep malicious apps away. Unfortunately, both of them have proven that they are not sufficient and that malicious apps are still being downloaded and installed.

- **Signing**. Most markets require that developers sign their apps. All developers use a certificate to sign their executables. This makes sure that the app is not corrupted using a cryptographic key. Android checks if the certificate is valid and will not install the app if it is not signed.
- **Review**. Most apps are reviewed and analyzed upon submission and if they are found to be malicious, they record the signature and for a traceback and do a potential further detection.

Platform protection

The Android platform has two main methods to try and protect the user from a malicious app.

- **Sandboxing**. The idea is to limit the environment in which the app operates. This can improve the security of the app. However, it can cause problems when two apps need to communicate between each other.
- **Permissions.** Prior to an app being installed, the system will show the user a list of permissions and will require confirmation in order to proceed. However, because permissions require some technical knowledge, look similar and the user usually cannot use the app if he/she does not agree, people just blindly agree, without much of a consideration.

Data set

The data set that was used for this project was taken from AndroZoo [1]. They have a very big collection of apps. They have both malicious and benign apps. The apks have been fetched using a librabry (1). The data set used for the project is 2000 (1000 malicious and 1000 benign) apks. The data set is composed from lightweight apks with less than 300 KB, from December 2015 to 2021. The apks are from Google play store and appchina, which the name suggests is a Chinese market for Android apps. AndroZoo uses a VirusTotal (2) number for the number of anti-virus programs that have detected the application as malicious. For the data set were used only apps that have been flagged by two or more anti-virus programs.

Overview

The idea of the project is that malicious applications will use a certain set of permissions, whereas a benign application will use a different set. Therefore, after using different classification algorithms we will be able to provide a model that will be able to classify unseen data(apks) to malicious or benign. Then we will be able to compare the performance of the different algorithms. In addition, it will be able to improve the model and to do a feature visualization of the features with the highest weight in the decisions.

Methodology

Preamble. For the extraction of the components of the apks, it was used a library for reverse engineering of Android APKs- androguard (3).

(1) https://github.com/ArtemKushnerov/az
(2) https://www.virustotal.com/gui/services-overview
(3) https://github.com/androguard/androguard

Feature Extraction. For the feature extraction it was used androguard (3), which aided for the extraction of the permissions from the AndroidManifest.xml file. This file declares, which permissions the app must have in order to operate. The file is contained in the root directory and the Android OS must first process it before moving to the installation part of the program.

Embedding in a vector. For example, an app can use a permission READ_LOGS which can be considered a dangerous one, because it reads various system log files. However, it might be part of a benign application that is just required for the normal functionality. Therefore, it is important to look what other permissions are being used in combination with it. The approach that was taken for this project was to create a vector for each application with ones and zeros. Extracting the permissions from each apk (manifest file) has made the permissions (features) set.

The vector for each application is generated as follows: 1 if the permission is in the manifest file in the certain apk and 0 for all the other permission that are not. In this way a 2D matrix was created with 425 different permissions together with a classifier - 1 if the application is malicious and 0 if the application is benign.

Classification. The above-mentioned matrix is split for training and testing with a ratio of 80/20 respectively. For the classification it was taken the approach to use scikit-learn library for the different models, namely - Support Vector Machine, Random Forest and Naïve Bayes.

- Support Vector Machine (SVM) - the reason it was chosen as an algorithm for the project was, because it is a well-known algorithm for classification problems with relatively low computational cost. The idea behind it is to find a hyperplane in an N-dimensional space that can distinctly classify the data points. The best hyperplane is the one with the maximum margin (maximum distance between data points of both classes).

- Random forest – another very good algorithm for classification. It operates by constructing multiple decision trees at training time on various sub-samples of the data set and uses averaging in order to enhance the predicative powers and to reduce potential over-fitting.

- Naïve Bayes – its usage was inspired from a paper [22]. Another reason to choose this algorithm was, because it is good for classifying categorical data and also it assumes that all features are independent, which is not the case for our project. Therefore, the expectation was that it will do much worse than the others. The algorithm uses the Bayes Theorem of conditional probability in order to rank probability of one event happening given that a set of features have happened. It is very good for tackling high dimensionality problems.

Evaluation.

For the evaluation of the different classifier models the following performance metrics were used (with the help of a confession matrix, where TP = true positive, TN= true negative, FP = false positive, FN= false negative):

- Recall – it is calculated as follows (TP/TP+FN). It is the probability that an actual positive will test positive.
- Precision – it is calculated as follows (TP/TP + FP). It is the probability that an actual positive is positive from the retrieved positive instances.
- Accuracy – it is calculated as follows ((TP + TN) / (TP + TN + FP + FN)). It is the fraction of predictions our model got right.
- F1-score – it is calculated as follows (2*((precision*recall) / (precision + recall))). It is the harmonic mean of the precision and recall.

First the Support Vector Machine model was trained. It was decided to use a linear kernel. The results can be seen in Fig 1. Then the model was improved by changing the kernel to a Radial Basis Function (RBF) and tuning the C and gamma parameters and a slightly better performance metrics were received, which can be seen in Fig 2. The performance metrics for the Random Forest and the Naïve Bayes model can be seen in Fig 3 and Fig 4. The Support Vector Machine and the Random forest have very similar performance metrics with the Random forest having slightly better results for almost all metrics. In addition, having better results when compared to the SVM with the linear kernel. This is, because Random forest searches for the best feature from a random subset of features, this in general helps to produce a better model. The Random forest model was slightly improved, by tuning the n_estimator, which says how many trees there will be. Usually, the more trees that are generated the better model. However, this adds to the computational cost. Experiments were conducted using different values and the lowest value was kept that gives fairly good results. As it can be seen in Fig 4 the Naïve Bayes, did not do very well. This was expected, due to the reasons mentioned in the Classification subsection.

```
[[114 107]
 [ 23 157]]
              precision    recall  f1-score   support

           0       0.83      0.52      0.64       221
           1       0.59      0.87      0.71       180

    accuracy                           0.68       401
   macro avg       0.71      0.69      0.67       401
weighted avg       0.73      0.68      0.67       401
```

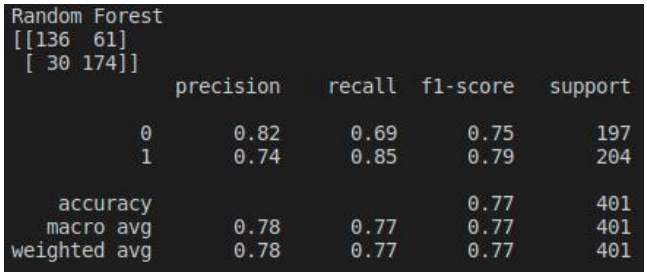
Figure 1. RBF kernel SVM

```
SVM
[[128  69]
 [ 31 173]]
              precision    recall  f1-score   support

           0       0.81      0.65      0.72       197
           1       0.71      0.85      0.78       204

    accuracy                           0.75       401
   macro avg       0.76      0.75      0.75       401
weighted avg       0.76      0.75      0.75       401
```

Figure 2. RBF kernel SVM

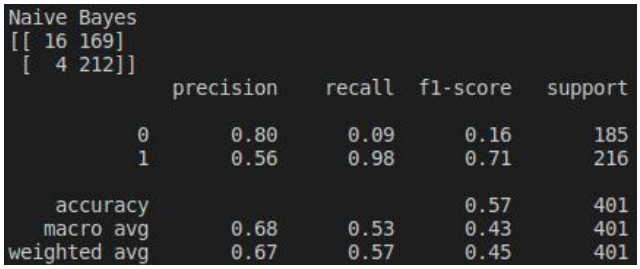


```
Random Forest
[[136  61]
 [ 30 174]]
              precision    recall  f1-score   support

           0       0.82      0.69      0.75       197
           1       0.74      0.85      0.79       204

    accuracy                           0.77       401
   macro avg       0.78      0.77      0.77       401
weighted avg       0.78      0.77      0.77       401
```

Figure 3. Random forest



```
Naive Bayes
[[ 16 169]
 [  4 212]]
              precision    recall  f1-score   support

           0       0.80      0.09      0.16       185
           1       0.56      0.98      0.71       216

    accuracy                           0.57       401
   macro avg       0.68      0.53      0.43       401
weighted avg       0.67      0.57      0.45       401
```

Figure 4. Naïve Bayes

Feature visualisation and importance. For the feature visualisation and importance, the results of the SVM model were used. It has given a weight for the different features and Fig 5 was produced, by showing the top 20 most relevant features (the positive ones), which aided in the decision to classify an apk as malicious and the top 20 (negative ones) that aided in the decision to classify an apk as benign. From this figure it can be seen of the top permissions are different from the ones other papers have reached. We can see that some of the top permissions from Yerima *et al*.[21] Fig 6 are not in our figure Fig 5. For example, SEND_SMS is used way more in malicious apps than in benign. However, because the size of the applications of our data set is small, we are left with a different set of top features. That means that similar models trained on different data sets might produce a different real time result.

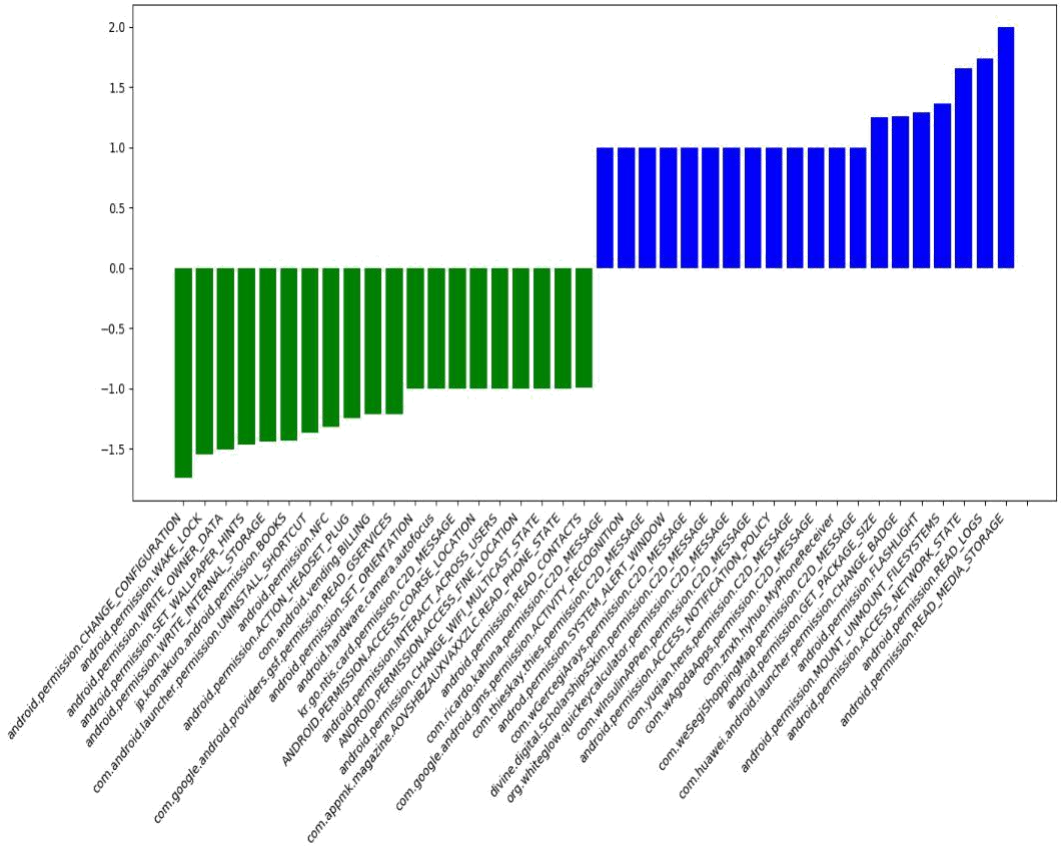


Figure 5. (top 20 positive for both benign and malicious)

Permission	Benign	Malicious
INTERNET	68.50 (1)	93.38 [1]
ACCESS_NETWORK_STATE	30.97 (2)	42.98 [8]
READ_PHONE_STATE	24.99 (3)	80.99 [2]
WRITE_EXTERNAL_STORAGE	24.14 (4)	59.50 [4]
ACCESS_COARSE_LOCATION	18.17 (5)	43.80 [7]
ACCESS_FINE_LOCATION	17.22 (6)	35.53 [12]
WAKE_LOCK	13.07 (7)	23.14 [18]
VIBRATE	12.84 (8)	23.14 [19]
ACCESS_WIFI_STATE	8.09 (9)	28.92 [16]
RECEIVE_BOOT_COMPLETED	7.99 (10)	23.14 [20]
READ_CONTACTS	7.50 (11)	47.11 [6]
GET_TASKS	5.32 (12)	5.78 [30]
CALL_PHONE	5.10 (13)	31.40 [14]
SEND_SMS	4.83 (14)	64.46 [3]
SET_WALLPAPER	4.75 (15)	30.57 [15]
CAMERA	4.35 (16)	5.78 [30]
GET_ACCOUNTS	4.31 (17)	4.95 [31]
RECEIVE_SMS	4.29 (18)	40.49 [10]
WRITE_SETTINGS	3.90 (19)	7.44 [27]
PROCESS_OUTGOING_CALLS	3.64 (20)	4.13[36]

Figure 6. Yerima *et al.*[21] top permissions

Limitations

Some of the limitations for the project are that:

- The performance metrics received from the project are lower compared to other projects that have decided to combine permissions with API calls or using other combinations. For example, Arp *et al*. [2] proposed a Drebin tool that uses eight different sets of features (hardware components, requested permissions, app components, filtered intents, restricted API calls, user permissions, suspicious API calls and network addresses) they have received some very good performance metrics with an accuracy of 94%. Another project of Scalas *et al*. [18] proposed R-pack droid, which can be found in the Google play store. They relied on collecting permissions combined with API packages. Chen *et al*. [5] have create StromDroid, which is similar to the previous two models and it uses permissions and API calls. In general, these different papers have all hinted that using solely android permissions might not be enough for creating a model that has an accuracy in the 90^{th} percentile.
- Another limitation that is common for projects that rely on machine learning is that there is a possibility of mimicry and poisoning attacks [13][14][15][20] and that they can mislead the model. There are obfuscation strategies, such as code-reordering, repackaging, or inserting made up code. The attacker can incorporate benign features. For example, inserting bonus permissions that are not dangerous and this is highly likely to lower the detection rate of the algorithm [16][25].
- In order not to write manually the detection patterns, the project relied on machine learning to generate the decisions for the classification. However, the machine learning algorithm is highly dependent on the data set that is being provided. Since, crawling yourself a big market for malicious and benign samples requires additional technical knowledge, the project is highly reliant on the few providers of such sets of apks. These providers have certain conditions in order to gain access to their collections, therefore it can pose a limitation for other people, who don't cover these conditions [2].
- Decomplication of an apk can have some limitations. In practice, it is possible for an apk file to not be able to be decompiled and this can hinder that extraction of feature. However, such limitation was not present in this project.

Related work

There are many related papers written on malware detection using machine learning. However, some of the following have made a greater focus on the permissions.

Felt *et al.* [8] try to establish whether applications are overprivileged. That means that an application requested a permission that it never used. Out of there data set of 940 applications, they have found out that one third is overprivileged. The have also observed that most of the developers try to make applications that are least privileged, but unfortunately, they fail because of poor API documentation.

Another work by Felt *et al.* [9] has examined both paid and free applications and have found that most of them 93% from the free apps and 82% form the paid use at least one dangerous permission according them, but still the majority of the people gave access to the dangerous permission. This further proves that people don't pay attention to the permissions.

Enck *et al.* [7] their work has analyzed different permissions and tried to find specific combinations of permissions that are considered dangerous and therefore, if an application has this combination has a high chance of being malicious. For example, WRITE_SMS and SEND_SMS.

Barrera *et al.* [3] their work has done some analysis using self-organizing maps over different permission-based security models. Some of their observations were that the INTERNET permission is the most common one to be requested, because it is in the root for requesting advertisements. Also, permissions that are for location are usually requested in pairs (fine and coursed locations).

Sarma *et al*. [17] their work has focused on using machine learning model - SVM with a combination of assessing the risk and the benefits of the permissions required. It was accomplished based on the category (e.g. Books, Games) in which the application falls in and if this permission is common among such categories.

Future work

As a potential future work, the features set can be combined with another set of features. This can be API calls, a set of activities, a set of receivers. The project can also be tested on a different data set, where the apk size is in the range of 1MB to 50MB. In addition, a comparison to other classification algorithms can be included, such as Logarithmic regression, k-Nearest Neighbors, Decision trees. Finally, the data for the models can be used for resampling using k-fold cross validation.

Conclusion

In this work, it was proposed an approach that solely focuses on the full set of collected permissions. It was observed the performance of three different models over a randomly selected large data set. Random forest had the best performance metrics with an accuracy of 77%, followed by Support Vector Machine, with 75% and in the end, the Naïve Bayes approach with an accuracy of 57%. The different

models have demonstrated that using permissions is a good starting point, but not the best approach. I hope that this study draws more attention to the issue of malicious apps in the Android markets and that it also inspires more and better detection methods in this area. Optimistically, within time the markets will have less and less malicious applications in them and this will allow users not to worry whether and application is malicious or benign.

References

[1] Allix, K., Bissyandé, T.F., Klein, J. and Le Traon, Y., 2016, May. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)* (pp. 468-471). IEEE.

[2] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T., 2014, February. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss* (Vol. 14, pp. 23-26).

[3] Barrera, D., Kayacik, H.G., Van Oorschot, P.C. and Somayaji, A., 2010, October. A methodology for empirical analysis of permission-based security models and its application to android. In *Proceedings of the 17th ACM conference on Computer and communications security* (pp. 73-84).

[4] Cavallaro, L., Saxena, P. and Sekar, R., 2008, July. On the limits of information flow techniques for malware analysis and containment. In *International conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (pp. 143-163). Springer, Berlin, Heidelberg.

[5] Chen, S., Xue, M., Tang, Z., Xu, L. and Zhu, H., 2016, May. Stormdroid: A streaminglized machine learning-based system for detecting android malware. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (pp. 377-388).

[6] Demontis, A., Melis, M., Biggio, B., Maiorca, D., Arp, D., Rieck, K., Corona, I., Giacinto, G. and Roli, F., 2017. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, *16*(4), pp.711-724.

[7] Enck, W., Ongtang, M. and McDaniel, P., 2009, November. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security* (pp. 235-245).

[8] Felt, A.P., Chin, E., Hanna, S., Song, D. and Wagner, D., 2011, October. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 627-638).

[9] Felt, A.P., Greenwood, K. and Wagner, D., 2011, June. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development* (pp. 7-7). USENIX Association.

[10] Grace, M., Zhou, Y., Zhang, Q., Zou, S. and Jiang, X., 2012, June. Riskranker: scalable and accurate zero-day android malware detection. In *Proceedings of the*

10th international conference on Mobile systems, applications, and services (pp. 281-294).

[11] Hu, J.W., Zhang, Y. and Cui, Y.P., 2020, July. Research on Android ransomware protection technology. In *Journal of Physics: Conference Series* (Vol. 1584, No. 1, p. 012004). IOP Publishing.

[12] Motiee, S., Hawkey, K. and Beznosov, K., 2010, July. Do Windows users follow the principle of least privilege? Investigating user account control practices. In *Proceedings of the Sixth Symposium on Usable Privacy and Security* (pp. 1-13).

[13] Newsome, J., Karp, B. and Song, D., 2006, September. Paragraph: Thwarting signature learning by training maliciously. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 81-105). Springer, Berlin, Heidelberg.

[14] Perdisci, R., Dagon, D., Lee, W., Fogla, P. and Sharif, M., 2006, May. Misleading worm signature generators using deliberate noise injection. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (pp. 15-pp). IEEE.

[15] Perdisci, R., Dagon, D., Lee, W., Fogla, P. and Sharif, M., 2006, May. Misleading worm signature generators using deliberate noise injection. In *2006 IEEE Symposium on Security and Privacy (S&P'06)* (pp. 15-pp). IEEE.

[16] Rastogi, V., Chen, Y. and Jiang, X., 2013, May. Droidchameleon: evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security* (pp. 329-334).

[17] Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C. and Molloy, I., 2012, June. Android permissions: a perspective combining risks and benefits. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies* (pp. 13-22).

[18] Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C.A., Martinelli, F. and Giacinto, G., 2018. R-PackDroid: practical on-device detection of Android ransomware. *CoRR, abs/1805.09563*.

[19] Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C.A., Martinelli, F. and Giacinto, G., 2019. On the effectiveness of system API-related information for Android ransomware detection. *Computers & Security*, *86*, pp.168-182.

[20] Venkataraman, S., Blum, A. and Song, D., 2008. Limits of learning-based signature generation with adversaries.

[21] Yerima, S.Y., Sezer, S. and Muttik, I., 2014, September. Android malware detection using parallel machine learning classifiers. In *2014 Eighth international conference on next generation mobile apps, services and technologies* (pp. 37-42). IEEE.

[22] Yerima, S.Y., Sezer, S., McWilliams, G. and Muttik, I., 2013, March. A new android malware detection approach using bayesian classification. In *2013 IEEE 27th international conference on advanced information networking and applications (AINA)* (pp. 121-128). IEEE.

[23] Yuan, Z., Lu, Y., Wang, Z. and Xue, Y., 2014, August. Droid-sec: deep learning in android malware detection. In *Proceedings of the 2014 ACM conference on SIGCOMM* (pp. 371-372).

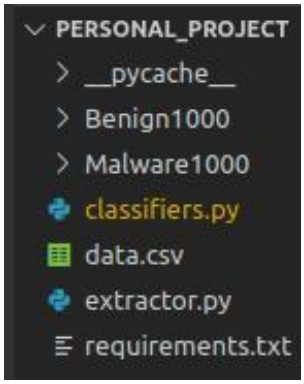
[24] Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M. and Yang, M., 2020, October. Enhancing State-of-the-art Classifiers with API Semantics to Detect Evolved Android Malware. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (pp. 757-770). [24]

[25] Zheng, M., Lee, P.P. and Lui, J.C., 2012, July. ADAM: an automatic and extensible platform to stress test android anti-virus systems. In *International conference on detection of intrusions and malware, and vulnerability assessment* (pp. 82-101). Springer, Berlin, Heidelberg.

[26] Zhou, Y. and Jiang, X., 2012, May. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy* (pp. 95-109). IEEE.

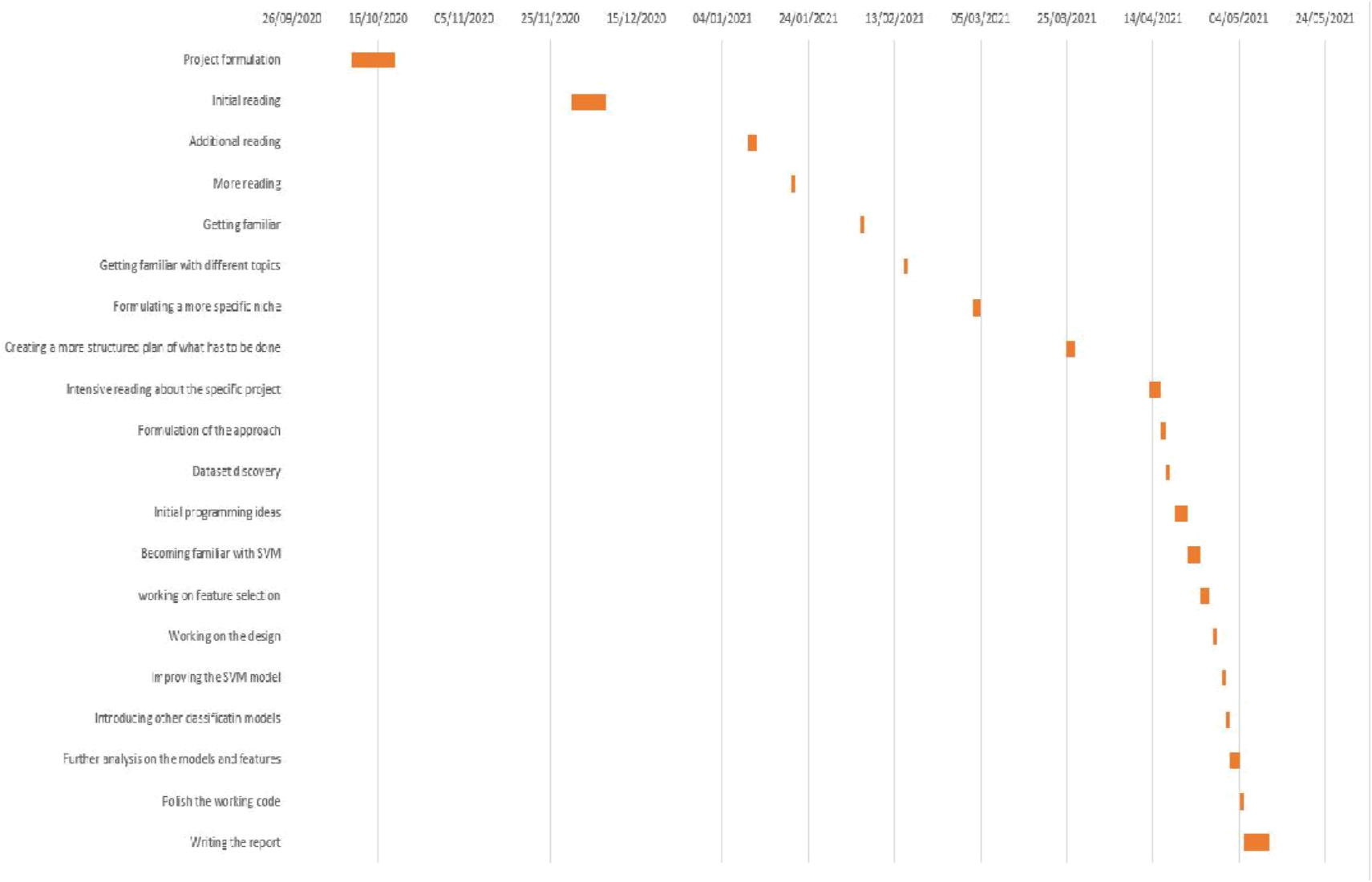
Appendices

Table of contents for the zip file



Gantt chart

The time management of the project was excellent in the last month. However, this differed from what was imagined in the progress report. The Gantt chart is on the following page.


26/09/2020
16/10/2020
05/11/2020
25/11/2020
15/12/2020
04/01/2021
24/01/2021
13/02/2021
05/03/2021
25/03/2021
14/04/2021
04/05/2021
24/05/2021
Project formulation
Initial reading
Additional reading
More reading
Getting familiar
Getting familiar with different topics
Formulating a more specific niche
Creating a more structured plan of what has to be done
Intensive reading about the specific project
Formulation of the approach
Dataset discovery
Initial programming ideas
Becoming familiar with SVM
working on feature selection
Working on the design
Improving the SVM model
Introducing other classificatin models
Further analysis on the models and features
Polish the working code
Writing the report