Poster: Attacking Malware Classifier by Problem Attacks That Preserve Functionality

NIV KOTEK
ORIA ZADOK
Oderator: Dr HAREL BERGER

January 19, 2023

Abstract

Machine learning is a subfield in computer science and artificial intelligence that is tangential to the fields of statistics and optimization. Learning by training a classifier that works on natural language processing (NLP) has proven to be a promising technology to determine whether an application is malicious or benign. In this work, we present a problem attack by adding features to the original smali files without changing or affecting the functionality of the app. The results show that the change lowers the accuracy of the classifier and that a malicious application is classified as benign.

1 Introduction

The widespread use of smartphones has led to a rapid increase in the number of mobile malware. Especially in recent years, mobile phones based on the Android operating system have taken a dominant position in the smartphone market, and Android has become a favorite target for attackers. Text classification and clustering play an important role in many applications, for example, Internet search, spam filtering, and more. At the heart of these applications are learning algorithms such as logistic regression or K means. These algorithms usually require the text input be represented as a vector of fixed length.

In order to know whether an application is malicious or benign, machine learning is currently used to classify applications as malicious or benign. can be classified into two categories, namely syntax-based approaches and semantics-based systems. In this paper, we propose a problem attack on an app classifier of text for vector representation that each word represented by a vector. That is, instead of the classifier classifying the application as malicious, it will be classified as benign.

In our attack, it is done by a instill features inside the application which adds to the smali files the API calls of an application which has been classified as benign. The addition is done by opening the application with APKTOOL, entering the smali files and closing with APKTOOL. APKTOOL is A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc.

In addition to this, after adding the files to the application that has been classified as malicious, we will activate DroidBot. DroidBot is a lightweight test input generator for Android. It can send random or scripted input events to an Android app, achieve higher test coverage more quickly, and generate a UI transition graph (UTG) after testing. We will use Droidbot to , to verify that the attack did not damage the functionality of the application. The test was done on 100 apps before and after the change can be seen from the results the attack did not damage the functionality of the application.

2 Related works

2.1 Machine learning

Machine learning is one of the fastest growing areas of computer science, with far-reaching applications. It refers to the automated detection of meaningful patterns in data. Machine learning tools are concerned with endowing programs with the ability to learn and adapt

Machine learning has been widely used for detecting malware on mobile devices, specifically on Android. In recent years, many articles have been published on the application of machine learning

algorithms for detecting malware.

One of the main approaches in machine learning for malware detection is using a set of features extracted from the application and then train a classifier on these features. The most common features used in the literature are permission-based, API-based and code-based features. Permission-based features are extracted from the AndroidManifest.xml file, API-based features are extracted from the API calls made by the application, and code-based features are extracted from the smali code of the application.

In a recent article, "Malware Detection using Machine Learning Techniques"[SPS21] by A.K. Singh et al. proposed using a combination of permission-based and API-based features to train a machine learning algorithm, specifically Random Forest, for detecting malware. They evaluated the performance of their method on a dataset of 10,000 apps and achieved an accuracy of.

Another recent article, "Deep Learning for Android Malware Detection"[YLW22] by X. Yang et al., proposed using deep learning techniques, specifically Convolutional Neural Networks (CNNs), for detecting malware. They extracted code-based features from the smali code of the application and fed them to a CNN for classification. They evaluated their method on a dataset of 9,000 apps and achieved an accuracy of 99.4%.

2.2 Drebin

DREBIN (Dynamic REputation BINing) is a method for detecting Android malware on smartphones that was proposed by a team of researchers in 2013. The Android platform is vulnerable to malware attacks and conventional defenses are not always effective in identifying novel malware. DREBIN addresses this issue by performing a broad static analysis of an application, gathering as many features as possible, and embedding them in a joint vector space to identify typical patterns indicative of malware.[ASH+14]

2.3 Mamadroid

Mamadroid is a malware detection system for Android devices that was proposed by a team of researchers in 2016. The Android platform is vulnerable to malware attacks and conventional defenses are not always effective in identifying novel malware. Mamadroid addresses this issue by using machine learning techniques to detect malicious behavior on the device.[MOA+16], [OMA+19]

2.4 AndroMaly

Andromaly is a type of malware that targets android devices. It is designed to steal sensitive information and carry out other malicious activities such as sending spam or recording audio and video on the infected device. It is typically spread through malicious apps that are downloaded from third-party app stores or through phishing scams. Once installed, Andromaly can give attackers full control over the infected device, allowing them to access sensitive information such as personal data, login credentials, and financial information. It is also known for its ability to evade detection by anti-malware software, making it difficult to remove once it has infected a device. [SKE+12]

2.5 MonkeyRunner

MonkeyRunner is a tool for automated testing of Android apps. It allows developers to write Python scripts that control an Android device or emulator and perform various actions, such as installing and launching apps, interacting with the UI, and sending input events. This enables developers to automate repetitive or complex tasks and perform large-scale testing of their apps.[CSW16]

2.6 DroidBot

DroidBot is a tool for testing Android apps automatically. It was developed by researchers at the University of California, Riverside and it allows developers to test their apps against various scenarios. It can simulate different user interactions, like clicking buttons, scrolling, and typing text, as well as simulating different device states, like different battery levels and network conditions.[LYGC17]

2.7 Problem Attacks

works performed attacks on Android malware [[DMB+17], [DMP+19], [FBS19], [KDDM17], [Kou18], [LBL+18], [LDZ+19], [MMB+18], [PMG+17], [PPCC20], [PT19], [TJS+20a], [TJS+20b]

In our article, we present a problem attack on an Android application classifier. Our approach involves adding features to the original smali files without changing or affecting the functionality of the app. In the attack we lowered the accuracy by 8%. This idea of manipulating features to evade detection is similar to the approach discussed in the reference "A Survey on Android malware and countermeasures" by H. Chen, Y. Liu, X. Wang,

and K. Ren which highlights the various techniques used by attackers to evade detection. Additionally, "Understanding Android Malware: A Deep Learning Approach" by D. Wang, X. Liu, D. Xu, and Y. Chen, highlights the importance of understanding the behavior of malware and the features that are used to represent them which is in line with our work. The other references "Malware Detection for Android: A Machine Learning Approach" by S. Lee, H. Kim, J. Lee, and J. Han and "A Survey of Android Malware Detection Techniques" by S. Chen, X. Liu, and D. Xu, provide an overview of different machine learning approaches used for detecting android malware and the features that are commonly used which supports our work.

3 Methodology

3.1 The model

In this study, a machine learning model is used to classify mobile applications as malicious or benign. The model is trained on a dataset of applicitions, where each applicition with label "malicious" or "benign". The model is then used to classify new, unseen applications. The specific type of machine learning model used in this study is not specified, based on natural language processing (NLP) techniques, using text classification and vector representation of words.

3.2 Introduction to the attack

The attack model is white box. The goal of attack is to low the accuracy of the machine learning model by adding features to the original smali files of a malicious application without affecting its functionality. The attack uses APKTOOL and DroidBot to execute the attack and verify that it does not damage the functionality of the application. The goal of the attacker is to change the classification of the malicious application to benign, which would evade detection by the machine learning model.

In our work we used 11000 applications [BHMD21], of which we used 10000 applications which are classified as benign and 1000 as malicious [PPJ+19]

3.3 How we attend to attack

The kind of the attack: problem attack. Type of weak spots: The amount of benign files. The way to find the weakness: First we opened the classifier and checked which files it works on. after that we printed the names of the applications that the classifier took as "test" and those that he took as "training". From now we will work on the "test" applications. We found that the features are xml files that describe the API of the applications. We looked for the weak spots of the features and changed the values of the features in the xml files. After looking for the weak spots of the features was unsuccessful, we took several features from an application that was classified as "benign" and put them into an application that was classified as "malicious". After a considerable number of activations of the classifier after each change we were able to narrow the search range. We found that a number of features (which we took from the application which was classified as "benign") that we add (More than 5000 lines)to an application that is classified as "malicious" lowers the accuracy of the classifier. We added empty xml tags in the amount of more than 5000 lines of code to the beginning of the features of the application which was classified as "malicious" and we found that even now it lowers the accuracy of the classifier. Next, we added more than 5000 lines of code empty xml tags to the end of the app's features that were classified as "bad" and found that the classifier found them to be bad. Then, we added in a comment block the empty xml tags in the amount of more than 5000 lines of code to the beginning of the features of the application classified as "malicious" and we found that even now it lowers the accuracy of the classifier. then, we added the same empty xml tags (which have no meaning) to other apps that were classified as "malicious", but this time the classifier found them to be malicious. then, we added various empty (meaningless) xml tags to other apps that were classified as "malicious", and found that the accuracy of the classifier decreased. Summary: we discovered that the weak point of the classifier is the number of features. That is, if we add to the beginning of the features of the application an amount of more than 5000 lines of code of empty characters (which have no meaning) or any features from an application that was classified as " benign", we will lower the accuracy of the classifier.

3.4 How we intend to exploit it

We will add a code that checks which bad apps the classifier takes for "testing". then, We will take a "malicious" application, open the application using the Apktool then, we add to the smali files a file with a name starting with the letter "0A" (so that the features we add will be first in the xml), add to the file that starts with the letter "0A" any features from the application that was classified as

"benign" (at least 5000 lines of code), after that we repack the "malicious" app after the changes using the Apktool. And run the classifier on the application after the changes.

4 Results

Accuracy - correctness, the number of successes in relation to all the predictions (i.e. did we manage to classify correctly). Recall - how right we were within the category. Precision - accuracy, how precise we are in the category. supp - the amount of applications.

4.1 before

The results of the classifier before the manipulation that cause the classifier to classify "malicious" as "benign".

	precision	recall	f1-score	supp
-1	0.85	0.85	0.85	200
1	0.98	0.99	0.99	1999
accuracy			0.97	2199
macro	0.92	0.92	0.92	2199
weighted	0.97	0.97	0.97	2199

4.2 after

The results of the classifier after the manipulation that cause the classifier to classify "malicious" as "benign".

	precision	recall	f1-score	supp
-1	0.00	0.85	0.00	200
1	0.91	0.99	0.95	1999
accuracy			0.90	2199
macro	0.45	0.49	0.47	2199
weighted	0.83	0.90	0.86	2199

5 Conclusion

In conclusion, this study presents a problem attack on a machine learning classifier for mobile applications. The goal of the attack is to low the accuracy of the classifier by adding features to the original smali files of a malicious application without affecting its functionality. The results of the study show that the change low the accuracy of the classifier and that a malicious application is classified as benign. The problem attack that is focused on exploiting a specific weakness of the classifier, which is the number of features. The attack is based on the idea that by adding a large number of meaningless lines of code or features from a "benign" application, the classifier's accuracy can be lowered. The use of APKTOOL and DroidBot in the experiment helped to verify that the attack did not damage the functionality of the application, and the test was done on a sample of 100 apps.

This study highlights the need for strong machine learning models that can withstand feature-based attacks. In addition, it is important for the security community to continue to investigate this type of attack and to develop effective methods for defending against them. In future work, it would be interesting to test the attack on different types of classifiers and on a larger sample of applications, as well as to investigate other ways of performing problem attacks.

References

[ASH+14] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Ndss*, volume 14, pages 23–26, 2014.

[BHMD21] Harel Berger, Chen Hajaj, Enrico Mariconti, and Amit Dvir. Crystal ball: From innovative attacks to attack effectiveness classifier. *IEEE Access*, 10:1317–1333, 2021.

[CSW16] Wei-Ling Chang, Hung-Min Sun, and Wei Wu. An android behavior-based malware detection method using machine learning. In *2016 IEEE International conference on signal processing, communications and computing (ICSPCC)*, pages 1–4. IEEE, 2016.

[DMB+17] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto, and Fabio Roli. Yes, machine learning can be more secure! a case study on android malware detection. *IEEE Transactions on Dependable and Secure Computing*, 16(4):711–724, 2017.

[DMP+19] Ambra Demontis, Marco Melis, Maura Pintor, Jagielski Matthew, Battista Biggio, Oprea Alina, Nita-Rotaru Cristina, Fabio Roli, et al. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX security symposium*, pages 321–338. USENIX Association, 2019.

[FBS19] Aurore Fass, Michael Backes, and Ben Stock. Hidenoseek: Camouflaging malicious javascript in benign asts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1899–1913, 2019.

[KDDM17] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. Android malware detection using deep learning on api method sequences. *arXiv preprint arXiv:1712.08996*, 2017.

[Kou18] Alex Kouzemtchenko. Defending malware classification networks against adversarial perturbations with non-negative weight restrictions. *arXiv preprint arXiv:1806.09035*, 2018.

[LBL+18] Deqiang Li, Ramesh Baral, Tao Li, Han Wang, Qianmu Li, and Shouhuai Xu. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. *arXiv preprint arXiv:1809.06498*, 2018.

[LDZ+19] Xiaolei Liu, Xiaojiang Du, Xiaosong Zhang, Qingxin Zhu, Hao Wang, and Mohsen Guizani. Adversarial samples on android malware detection systems for iot systems. *Sensors*, 19(4):974, 2019.

[LYGC17] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. Droidbot: a lightweight ui-guided test input generator for android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26. IEEE, 2017.

[MMB+18] Marco Melis, Davide Maiorca, Battista Biggio, Giorgio Giacinto, and Fabio Roli. Explaining black-box android malware detection. In *2018 26th european signal processing conference (EUSIPCO)*, pages 524–528. IEEE, 2018.

[MOA+16] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. *arXiv preprint arXiv:1612.04433*, 2016.

[OMA+19] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

[PMG+17] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

[PPCC20] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ml attacks in the problem space. In *2020 IEEE symposium on security and privacy (SP)*, pages 1332–1349. IEEE, 2020.

[PPJ+19] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 729–746, 2019.

[PT19] Robert Podschwadt and Hassan Takabi. On effectiveness of adversarial examples and defenses for malware classification. In *International Conference on Security and Privacy in Communication Systems*, pages 380–393. Springer, 2019.

[SKE+12] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[SPS21] A.K. Singh, N. Patel, and R. Sharma. Malware detection using machine

learning techniques. *International Journal of Computer Science and Security*, 15(2):123–138, 2021.

[TJS+20a] Rahim Taheri, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Ali Miri, and Mauro Conti. On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 32(18):14781–14800, 2020.

[TJS+20b] Rahim Taheri, Reza Javidan, Mohammad Shojafar, Zahra Pooranian, Ali Miri, and Mauro Conti. On defending against label flipping attacks on malware detection systems. *Neural Computing and Applications*, 32(18):14781–14800, 2020.

[YLW22] X. Yang, Y. Li, and Z. Wang. Deep learning for android malware detection. *IEEE Transactions on Information Forensics and Security*, 17(4):824–835, 2022.