Evading Static and Dynamic Android Malware Detection Mechanisms

Teenu S. John(✉) and Tony Thomas(✉)

Indian Institute of Information Technology and Management-Kerala,
Thiruvananthapuram, India
{teenu.john,tony.thomas}@iiitmk.ac.in

Abstract. With the widespread usage of Android mobile devices, malware developers are increasingly targeting Android applications for carrying out their malicious activities. Despite employing powerful malware detection mechanisms, an adversary can evade the threat detection model by launching intelligent malware with fine-grained feature perturbations. Since machine learning is widely adopted in malware detection owing to its automatic threat prediction and detection capabilities, attackers are nowadays targeting the vulnerability of machine learning models for malicious activities. In this work, we demonstrate how an adversary can evade various machine learning based static and dynamic Android malware detection mechanisms. To the best of our knowledge, this is the first work that discusses adversarial evasion in both static and dynamic machine learning based malware detection mechanisms.

Keywords: Android · Adversarial malware · Evasion techniques

1 Introduction

Nowadays Android malware are increasing rapidly. According to the McAfee mobile threat report issued in 2020, there is an increase in the malicious apps targeting Android operating system [3]. These malware can hide themselves after installation, mimick the legitimate applications icon and also use advanced evasion technique that downloads the malicious code after sometime. The conventional signature based detection mechanisms [1] cannot detect such obfuscated evasive malware. Hence many malware detection mechanisms are adopting machine learning to detect unknown malware. The advantage of machine learning is that it can automatically learn and predict the malware behaviour from raw data [34]. In Android, there are many works that show how machine learning can be used effectively for static, dynamic and hybrid malware detection mechanisms [42].

This work is done as a part of Centre for Research and Innovation in Cyber Threat Resilience project (CRICTR 2020-21), which is funded by Kerala State Planning Board.

In static Android malware detection, the Android application is examined without executing it [51]. The features used for static analysis are permissions, intents, static API calls, opcodes etc. The advantage of employing static analysis technique is that it has high code coverage. The drawback of static detection mechanism is that, it cannot detect obfuscated malware. Dynamic analysis on the other hand runs the application in an emulator and then captures the malicious behaviour of the application by examining the run time features such as system calls, dynamic API calls etc. [13,25,26,45]. The disadvantage of dynamic detection mechanism is that it has less code coverage. Moreover, malware developers can evade the dynamic analysis by examining the specific API's invoked by the application when it is made to run in a virtual machine. For example, if an application is executed in a virtual machine, then the *TelephonyManager.getDevice id()* API returns zero [47]. Petsas et al. [38] proposed an attack against virtual machines by examining the dynamic sensor information and VM-related intricacies of the Android emulator to evade detection. Wenrui et al. [24] proposed a mechanism to evade Android emulator runtime analysis using an evasive component that identifies whether the events are coming from a real user or from an automated tool. To solve the problems related to static and dynamic malware detection, several mechanisms have been proposed in the past that use hybrid analysis. The hybrid analysis uses a combination of both static and dynamic features for detection. The drawback of hybrid detection is that the resource consumption is more [12] when compared to static and dynamic malware detection mechanisms.

To detect malware, the static, dynamic or hybrid features of the application are fed into unsupervised or supervised machine learning classifiers. In supervised machine learning malware detection, the machine learning model is trained with thousands of benign and malware samples. However, adversaries can evade the most powerful machine learning models used for the malware detection by crafting malware that exploits the vulnerability of machine learning models. These malware are called adversarial malware. Adversarial malware pose a serious threat nowadays and is an emerging area of research [36].

There are some works that show how static detection mechanism using permissions and API calls can be easily evaded using adversarial attack. In [5], the authors evade the Drebin [11] detection method using some feature perturbation techniques. However, a research on how adversary evades the opcode based Android malware detection is yet to be explored and is an interesting area of study. This is because the opcodes contain valuable information to detect malware that employ code repackaging. The recent CoronaVirus application is one such ransomware [14] that uses code repackaging. The effectiveness of opcodes lies in the fact that despite obfuscation, the same family of Android malware share the same code parts and hence can be identified by examining the opcode patterns of the application [7,45].

Likewise in dynamic malware detection, system calls are very effective features for detecting obfuscated malware [16,33]. This is because, system call captures the interaction of the application with the operating system and hence reveal the actual behaviour of the application even if the application adopts dynamic loading, encryption and other techniques for evasion. Vinod et al. [49] showed label flipping attack against Android system call based malware detection where they poison the training data with adversarial samples. However, their attack injects individual system calls rather than sequences of system calls that is not effective.

In this paper, we show how static Android malware detection mechanism using opcodes and dynamic Android malware detection mechanism using system calls can be evaded by the adversary. We employ frequency based evasion in both static and dynamic malware detection mechanisms whereby the attacker injects the most frequent benign code sequences into the malware to subvert the detection. Our attack is realistic which shows that injecting a few sequences of benign code can evade the robust machine learning based malware detection mechanisms. Moreover, our attack is resilient against feature selection approaches [23] since we inject opcode sequences that replicate benign application behaviour.

The contributions in this paper are the following:-

1. We explore how an adversary injects benign opcode sequences to evade the static Android malware detection mechanism. For this we evade the mechanism employed in [16], which is a malware detection mechanism using opcode n- grams. This is the first work that shows adversarial attacks in the form of benign opcode injection.
2. We explore how the adversary injects benign system call sequences for evading dynamic Android malware detection mechanism. To show this, we evade the mechanism employed in [44] which is a malware detection mechanism using graph signal processing. We show that Android system call based malware detection mechanism using the powerful graph signal processing technique can be evaded by injecting a few benign system call sequences.

2 Related Works

The malware detection mechanism employed by the popular malware detection companies like Kaspersky [2] and Norton [4] use machine learning to detect polymorphic and obfuscated malware. However their detection capabilities can be deteriorated by an adversary that employs feature perturbations to evade detection. According to the recent threat report issued by Kaspersky [9], the adversarial attack against machine learning based malware detection can cause misidentified Trojans to infect millions of devices. There are many works in the past that discuss about adversarial attacks and defenses in Android malware detection. This section discusses about the attacks and defenses against Android malware detection classifiers that are implemented in the past.

There are two types of adversarial attacks. They are data poisoning and evasion attacks [32]. Data poisoning attacks are launched by contaminating the training instances of the classifier. Evasion attack on the other hand finely perturbs the applications features to evade detection. The evasion attack can be either problem space attacks or feature space attacks [39].

Chen al [19] proposed a data poisoning attack in Android malware detection where the adversary contaminates the training data with malicious samples. However, their attack required gradient information about the classifier. Abaid et al. [5] proposed an evasion attack in which evades the Drebin detection mechanism. They constructed attackers with different capabilities and showed that adversarial evasion is a feasible threat. Their attack reduced the detection accuracy of the classifier from 100% to 0%. However, their attack removed some features from the Android application, which may cause the application to lose its malware functionality.

The evasion attack can be either problem space attacks or feature space attacks [39].

In problem space attacks, the attacker transforms the malicious application to a new variant sample that is valid and realistic. Pierazzi et al. [39] proposed a problem space attack in android malware where the adversary employs opaque predicates by carefully constructing obfuscated conditions or program code that returns $False$ but evades static detection. Their attack can be detected with the help of feature selection techniques mentioned in [31]. Yang et al. [53] proposed a problem space attack in which they craft adversarial malware samples. Their technique alters the semantics of the application, and the generated malware may loose its functionality. Rosenberg et al. [40] proposed an attack against the API call based malware detection mechanism. They added artifacts into the application which can be detected using dynamic analysis.

The feature space attack on the other hand makes fine grained feature perturbations on various static and dynamic features of the application for evasion. Gross et al. [27] proposed a feature space attack using Jacobian matrix perturbation to evade Drebin Android malware detection. However the generated malware can be detected using undeclared classes and unused permissions. Li et al. [35] crafted attack against Drebin android malware detection mechanism. Their attack employed multiple generative methods to craft malware that do not ruin the malware functionality. They also proposed an ensemble technique to defend against adversarial attack.

Demontis et al. [23] proposed a secure learning technique to detect adversarial attacks in Android malware detection. However, their technique cannot detect malware that replicated benign applications behaviour. Chen et al. [18] proposed an ensemble based defense against adversarial attacks in Android malware detection. They used a feature selection approach to detect adversarial malware. The disadvantage of all these defensive mechanisms is that they can only detect adversarial attacks that perturb syntactic features like permissions [19,23,28,53,53]. Moreover, perturbing syntactic features like Android permissions can be easily achieved unlike semantic features. Since many malware detection mechanisms are extensively using the information of the dex files for malware detection [20] the attack that manipulates the features of the dex file is critical.

In this work, we perturb the features in the classes.dex file by injecting Dalvik opcodes that occur in the benign Android applications. Chen et al. [21] proposed a similar attack that manipulates the API control flow graph to evade detection. However, their attack inserts *Nop* API calls that can be detected using white list filtering [21]. Moreover their attack requires sophisticated adversarial feature perturbation techniques to achieve high evasion rate. In this paper, we investigate evasion attack in the form of feature space attack in which the attacker subverts the malware detection mechanism by injecting the features of the benign application. Our attack is resilient against feature selection approaches as mentioned in [23]. In Android, malware developers can easily download evasive malware by launching an update attack [5] when compared to data poisoning attacks. This motivated us to explore evasion attacks in static and dynamic Android malware detection mechanisms. We believe that our work will help security researchers to develop suitable defensive mechanisms against adversarial attacks in Android malware detection classifiers.

3 Machine Learning for Android Malware Detection

There are many works that show the effectiveness of machine learning for malware detection. This section discusses about the various machine learning models implemented in the past for malware detection. In [54] the authors proposed a classifier fusion approach that combines several machine learning classifiers for detecting malware. They combined various classifiers like J48, Random Tree-100, and Voted Perceptron, REPTree and Random Tree-9. Among the various machine learning models used for malware detection, Support Vector Machines(SVM) are found to be extremely useful for detecting unknown malware. Justin et al. [41] proposed a malware detection mechanism using SVM to detect Android malware with control flow graphs(CFG). Shifu et al. [30] proposed Hindroid, a mechanism using standard multi-kernel learning with SVM to detect Android malware. Canfora et al. [17] proposed a malware detection mechanism using sequences of system calls with SVM for building a fingerprint of the Android malware applications. Wen et al. [52] proposed a malware detection approach based on big data analytics and SVM by extracting various static and dynamic features of Android malware. Besides SVM, decision trees and random forest also gave excellent results in detecting malware. Peiravain et al.[37] proposed a malware detection method with permissions and API calls to detect Android malware using decision trees. Alam et al. [6] proposed a malware detection mechanism using Random Forest. The features used were battery consumption, CPU usage, memory related features, permissions etc. Moutaz et al. [8] proposed a malware detection mechanism using API calls and permissions. Their detection mechanism gave 94.3% F-measure with random forest classifier. Among all other machine learning models, deep neural network gained popularity owing to its ability to detect malware without manual feature engineering [48].

Venkatraman et al.[46] proposed a malware detection with deep neural network and their detection mechanism gave 96% accuracy. However, all these malware detection mechanisms using machine learning can be evaded by an adversary that crafts intelligent malware using adversarial machine learning.

4 Method of Attack

In this work, we evade the state of the art malware detection mechanisms [44] and [16] that use system calls and opcode n- grams as features. To inject the features of benign application to the malware application, we use TF-IDF feature selection. We chose TF-IDF since both of these malware detection mechanisms [16,44] use the frequency counts of the opcode n- grams and system calls for constructing the features for malware detection.

5 Evading Opcode Based Android Malware Detection Mechanism

To evade the opcode based malware detection mechanism, an adversary may employ code injection attack to inject benign dalvik code parts to the malware application or may insert junk codes to evade the detection mechanism. Since a good feature selection approach can easily detect junk code insertion, we launch attacks in the form of benign opcode injection to test whether the classifier is able to detect malware. This section discusses about how an adversary can evade opcode n- gram based Android malware detection mechanism mentioned in [16]. In [16], the opcode n- grams obtained from benign and malware applications are given to SVM(Support Vector Machines) and Random Forest for malware detection. An accuracy of 95.67% accuracy was obtained using opcode 5- grams with SVM classifier while with Random Forest, an accuracy of 96.88% was obtained using opcode 2- grams.

5.1 Preprocessing

We replicated the experimental setup mentioned in [16] to explore how adversary evades the detection mechanism. For this, we took 5560 malware applications from the Drebin dataset [11] and collected 5560 benign applications. Table 1 shows the malware families that were taken for the experiments as mentioned in [16]. We took all the benign application categories as mentioned in the original work. The benign applications were downloaded from Google Playstore and were uploaded to VirusTotal to check for malicious behaviour. Using apktool [50], we first extracted the .dex files from the apk. Then by using smali tool [29], we extracted the smali files from the .dex files. These smali files contain the opcodes of an apk and can be used to construct opcode n- grams.

Table 1. Android malware familes

SI no	Android malware family	Obfuscation	Malware type
1	DroidKungFu	Repackaging, string encryption, native payload	Trojan
2	Fakeinstaller	Renaming	Trojan
3	Plankton	Dynamic loading	Trojan,botnet
4	Opfake	Renaming	Trojan
5	GinMaster	Renaming	Trojan
6	Basebridge	Renaming	Trojan
7	Kmin	–	Trojan
8	Geinimi	Renaming	Trojan
9	Adrd	Renaming	Trojan
10	DroidDream	Renaming	Botnet

5.2 Training the Classifier

We trained the classifier as mentioned in [16]. We took opcode n- grams with $n =$ 2 and 5 since they gave the maximum accuracy when compared to $n =$ 1,3,4. We took top 2000 number of opcode n- grams that distinguish benign from malware application by using the technique mentioned in the original paper. We trained the two classifiers Support Vector Machine(SVM) and Random Forest(RF) as mentioned in the original work. We obtained an accuracy 96.3% with 1000 number of opcode 2-grams and an accuracy of 95.3% on 2000 number of opcode 5-grams. The accuracy values were approximately equal to that of the original work. Table 2 shows this.

5.3 Testing the Classifier

To evaluate the performance of the classifier, we used metrics such as True Positive Rate(TPR), False positive(FPR), True negative Rate (TNR), False Negative Rate (TNR), Accuracy, Recall, F-measure. True Positives(TP) refers to the number of malware applications that are correctly classified as malware by the classifier. True Negatives(TN) refers to the number of goodware applications that are correctly classified as goodware. False Positives(FP) refers to the number of benign applications that are incorrectly classified as malware. False

Table 2. Performance of [16] before the attack

Number of opcodes	n-gram	Classifier	TPR	FPR	TNR	FNR	Accuracy	Precision	Recall	F-measure
1000	2- gram	Random forest	0.963	0.027	0.973	0.037	0.963	0.972	0.963	0.968
2000	5- gram	SVM	0.962	0.055	0.945	0.038	0.953	0.945	0.962	0.953

Table 3. Performance of [16] After the Attack

l	TPR	FPR	TNR	FNR	Accuracy	Precision	Recall	F-measure
50	0.751	0.027	0.973	0.249	0.862	0.965	0.751	0.844
100	0.489	0.039	0.961	0.511	0.725	0.926	0.489	0.640
150	0.305	0.033	0.967	0.695	0.636	0.902	0.305	0.455
200	0.191	0.027	0.973	0.809	0.582	0.876	0.191	0.313

Negatives(FN) represents the number of malware applications incorrectly classified as goodware applications. The accuracy, precision and F-measure are computed as follows:

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F - measure = \frac{2 \times precision \times recall}{precision + recall} \tag{4}$$

5.4 Dalvik Opcode Injection

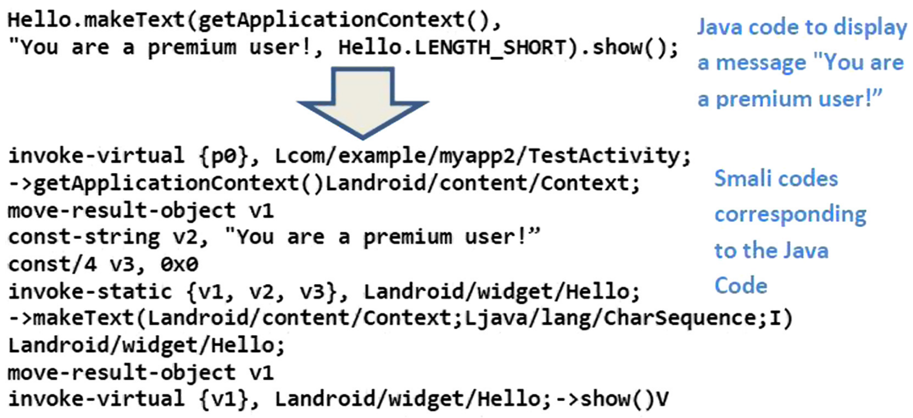
The aim of the Dalvik opcode injection attack is to evade the classifier by injecting benign opcode n- grams. We aim to use opcode injection attack rather than opcode elimination attack since the latter may destroy the malware functionality. The attack is achieved by using a JADX tool to find the java files corresponding to the smali files and injecting opcode sequences corresponding to the malicious class files.

We assume that the attacker has complete knowledge about the classifier and the features. To evade the detection mechanism the attacker injects benign opcode n- grams. We computed the most frequent benign opcode n- grams obtained using the TF-IDF method as mentioned before and injected them to evade the detection mechanism employed in [16]. Table 4 shows the top five opcode 5- grams obtained using TF-IDF method. In addition to the benign opcode n- grams obtained using TF-IDF, we also injected opcode n- grams for displaying text messages inside the malicious application to mimick the legitimate application behaviour. Figure 1 shows this. In this figure, the java code and its corresponding dalvik code to display a text message is shown. Here we aim to explore how injecting junk or random text messages can evade the opcode n-gram based detection mechanism.

We conducted the experiments on Random Forest classifier with opcode 2-grams, since it gave maximum accuracy in the original work. For testing the performance of the classifier, we took 500 benign application and 500 opcode injected malware applications. We took 50 samples from each of the Android malware families listed in Table 1. The performance of the classifier when we inject l benign opcode n- grams is shown in Table 3. When l increases, the FNR also increases which shows that the injected malware can evade the detection mechanism employed in [16]. Figure 2 shows how the detection accuracy of the classifier is reduced when we increase the value of l.

Table 4. Top five opcode 5- grams obtained from TF-IDF Method.

SI no	Opcode 5- grams	Category
1	array-length,const/1,const/,const-string,goto	Malware
2	const/1,const/,const/high1,const-string,const-wide/	Malware
3	aput-object,array-length,const/1,const/,const-string	Malware
4	aget-object,aput-char,aput-object,array-length,check-cast	Malware
5	iget-object,iput,iput-boolean,iput-object,move-exception	Malware
1	iget-object,iput-object,move-result-object,return-object,return-void	Benign
2	check-cast,const/,goto ,if-eqz,if-nez	Benign
3	mul-int/lit,new-instance,return,return-object,return-void	Benign
4	move-result-object,new-instance,nop,return-object,return-void	Benign
5	move-result,move-result-object,new-instance,nop,return-object	Benign


```
Hello.makeText(getApplicationContext(),
"You are a premium user!, Hello.LENGTH_SHORT).show();
```

```
invoke-virtual {p0}, Lcom/example/myapp2/TestActivity;
->getApplicationContext()Landroid/content/Context;
move-result-object v1
const-string v2, "You are a premium user!"
const/4 v3, 0x0
invoke-static {v1, v2, v3}, Landroid/widget/Hello;
->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)
Landroid/widget/Hello;
move-result-object v1
invoke-virtual {v1}, Landroid/widget/Hello;->show()V
```

Fig. 1. Injecting benign opcodes for displaying text messages

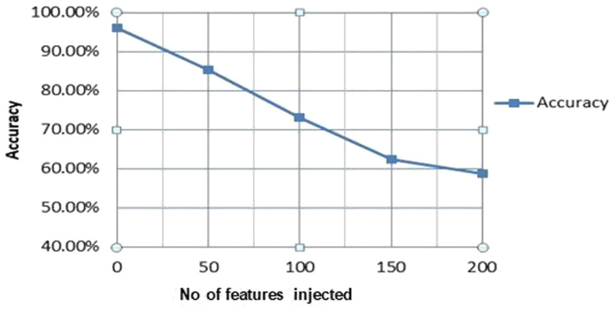


Fig. 2. Accuracy Values of [16] After Injecting l number of opcode $n-$ grams

6 Evading System Call Graph Based Android Malware Detection Mechanism

System call based Android malware detection mechanisms are found to be extremely powerful in detecting malware that evade static detection mechanisms [33]. Most of the system call based Android malware detection mechanisms are using frequency of occurrence of the system calls for detecting malware [17,25]. This is because certain system calls like *read()*, *write()* etc. are frequently invoked by the malware than goodware. This technique can be evaded by using a system call injection attack in which the malware injects some rare or benign system calls at runtime [15]. In this section, we show how an adversary can evade the system call based Android malware detection mechanism in [44]. The malware detection mechanism in [44] employs graph signal processing mechanism to detect Android malware. In this mechanism, the frequency of occurrence of the system calls are taken as the signals and then a graph shift operation is applied to the signals to obtain the processed graph signals. These graph signals are then fed into the machine learning classifiers to check whether the application is malicious or not.

6.1 Preprocessing and Signal Extraction

We took 2500 malware and goodware applications as mentioned in the original work [44] to replicate the experimental setup. The malware samples were taken from Drebin [11], AMD [10], and Contagio minidump [22]. We took 1,2,5,6,9,10 malware families mentioned in Table 1 and also the malware families in Table 5 as mentioned in [44] for conducting the experiments. The benign applications were downloaded from Google Playstore and checked with VirusTotal to check for malicious behaviour. We also eliminated semantically similar Android malware and took all the malware families mentioned in [44] and replicated the experimental set up. The Android applications were made to run in an emulator by injecting thousand pseudorandom events like key press event, touch event etc. to achieve high code coverage. We collected system calls using strace utility [43] and eliminated irrelevant system calls as mentioned in [44] and only selected relevant opcodes for malware detection to replicate the features for classification. After selecting the relevant opcodes, we constructed system call digraph and extracted the graph signals.

6.2 Training the Classifier

We trained the classifier as mentioned in the original paper. We took Random Forest Classifier, since it gave maximum accuracy. We took 80% samples for training and remaining 20% for training as mentioned in the original work [44].

6.3 Testing the Classifier

The accuracy, precision, recall and F-measure was computed as in Section 3.1. Table 6 shows the performance matrix of the classifier.

6.4 System Call Injection

The system call graph signal based detection mechanism takes the frequency of occurrence of the system call for constructing the graph signal. This mechanism can be evaded by injecting benign system call codes that mimick legitimate application behaviour. We model a perfect knowledge attack where the attacker has complete knowledge about the features and the classification model. The attacker can gather malware and benign system calls from public repositories and examine the most frequent system calls that are occurring in malware and benign applications. We inject a sequence of system calls rather than individual system calls since the application may not work properly if we do so. To inject a system call sequence, we first computed the most frequently occurring benign system calls from goodware applications using the TF-IDF method. We found that certain system calls like *unlink()*, *mkdir()*, *chmod()* are frequently invoked by the benign application. Our attack is similar to the attack as mentioned in [15]. We carefully selected the benign applications that are having the most frequent benign system call counts and then injected those system call sequences to evade the detection. We took 10 malware samples from each of the malware family and made a test set of 270 system call injected malware samples and 270 benign samples. Table 7 shows how the detection accuracy is reduced.

Table 5. Android malware familes

SI No	Android malware family	Obfuscation	Malware type
1	Andrup	String encryption, renaming	Adware
2	GoldDream	Repackaging, string encryption, native payload	Backdoor
3	Boxer	Renaming, string encryption	Trojan-SMS
4	FakeTimer	–	Trojan
5	Lotoor	Renaming, dynamic loading	HackerTool
6	Rumms	String encryption, renaming, dynamic loading	Trojan-SMS
7	NandroBox	Repackaging	Trojan
8	MMarketPay	Renaming	Trojan
9	Penetho	–	Exploit
10	Mercor	–	Trojan
11	FakeDoc	Renaming	Trojan
12	FakePlayer	Renaming	Trojan
13	Vidro	–	Trojan SMS
14	Tesbo	Repackaging, string encryption, renaming	Trojan
15	AndroRat	–	Backdoor
16	Mseg	Repackaging, renaming	Trojan
17	SpyBubble	–	Trojan
18	MobileTX	–	Trojan
19	Zitmo	Renaming	Trojan
20	Lnk	–	Trojan
21	FakeDoc	Renaming	Trojan

Table 6. Performance of [44] before the attack

TPR	FPR	TNR	FNR	Accuracy	Precision	Recall	F-measure
0.971	0.041	0.959	0.029	0.965	0.959	0.971	0.965

Table 7. Performance of [44] after the attack

TPR	FPR	TNR	FNR	Accuracy	Precision	Recall	F-measure
0.400	0.00	1.00	0.600	0.700	1.00	0.400	0.571

7 Conclusion and Future Work

In this paper, we showed that how an adversary can evade the static and dynamic Android malware detection mechanism employed by some of the state of the art machine learning models. We showed that by injecting only a few number of features, adversaries can induce misclassification. In future, we plan to model a limited knowledge attack and a blackbox attack to evade the system call and opcode based malware detection mechanisms. This is to explore how the adversary evades the detection model with less or no knowledge about the classifier. We also plan to develop suitable mechanisms to detect adversarial malware.

References

1. A guide to malware detection techniques and beyond. https://www.cynet.com/blog/a-guide-to-malware-detection-techniques-av-ngav-and-beyond/. Accessed 08 Sept 2020
2. Machine learning methods for malware detection. https://media.kaspersky.com/en/enterprise-security/Kaspersky-Lab-Whitepaper-Machine-Learning.pdf. Accessed 08 Oct 2020
3. Mcafee mobile threat report 2020. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf. Accessed 08 Sept 2020
4. Securing against malware using artificial intelligence. https://www.nortonlifelock.com/blogs/feature-stories/securing-against-malware-using-artificial-intelligence. Accessed 08 Oct 2020
5. Abaid, Z., Kaafar, M.A., Jha, S.: Quantifying the impact of adversarial evasion attacks on machine learning based android malware classifiers. In: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA), pp. 1–10. IEEE (2017)
6. Alam, M.S., Vuong, S.T.: Random forest classification for detecting android malware. In: 2013 IEEE international conference on green computing and communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, pp. 663–669. IEEE (2013)
7. Alazab, M., et al.: A hybrid wrapper-filter approach for malware detection. J. Netw. **9**(11), 2878–2891 (2014)
8. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., Awajan, A.: Intelligent mobile malware detection using permission requests and API calls. Future Gen. Comput. Syst. **107**, 509–521 (2020)
9. Alexander Chistyakov, A.A.: Ai under attack. https://media.kaspersky.com/en/business-security/enterprise/machine-learning-cybersecurity-whitepaper.pdf. Accessed 08 Oct 2020
10. Amd: http://amd.arguslab.org/ (2015)
11. Arp, D., Spreitzenbarth, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and explainable detection of android malware in your pocket (2014)
12. Arshad, S., Shah, M.A., Wahid, A., Mehmood, A., Song, H., Yu, H.: Samadroid: a novel 3-level hybrid malware detection model for android operating system. IEEE Access **6**, 4321–4339 (2018)
13. Azab, A., Alazab, M., Aiash, M.: Machine learning based botnet identification traffic. In: 2016 IEEE Trustcom/BigDataSE/ISPA, pp. 1788–1794. IEEE (2016)

14. Barth, B.: Coronavirus app locks android screens with repackaged malware. https://www.scmagazine.com/home/security-news/cybercrime/about-coronavirus-app-locks-android-screens-with-repackaged-malware/, https://www.scmagazine.com/home/security-news/ cybercrime/about-coronavirus-app-locks-android-screens-with-repackaged-malware/. Accessed 08 Sept 2020
15. Bhandari, S., Panihar, R., Naval, S., Laxmi, V., Zemmari, A., Gaur, M.S.: Sword: semantic aware android malware detector. J. Inf. Secur. Appl. **42**, 46–56 (2018)
16. Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., Visaggio, C.A.: Effectiveness of opcode ngrams for detection of multi family android malware. In: 2015 10th International Conference on Availability, Reliability and Security, pp. 333–340. IEEE (2015)
17. Canfora, G., Medvet, E., Mercaldo, F., Visaggio, C.A.: Detecting android malware using sequences of system calls. In: Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, pp. 13–20 (2015)
18. Chen, L., Hou, S., Ye, Y.: Securedroid: enhancing security of machine learning-based detection against adversarial android malware attacks. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 362–372 (2017)
19. Chen, S., et al.: Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. Comput. Secur. **73**, 326–344 (2018)
20. Chen, T., Mao, Q., Yang, Y., Lv, M., Zhu, J.: Tinydroid: a lightweight and efficient model for android malware detection and classification. Mob. Inf. Syst. **2018** (2018)
21. Chen, X., et al.: Android HIV: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inf. Forensics Secur. **15**, 987–1001 (2019)
22. Contagio: http://contagiodump.blogspot.com/ (2015)
23. Demontis, A., et al.: Yes, machine learning can be more secure! a case study on android malware detection. IEEE Trans. Depend. Secur. Comput. (2017)
24. Diao, W., Liu, X., Li, Z., Zhang, K.: Evading android runtime analysis through detecting programmed interactions. In: Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks, pp. 159–164 (2016)
25. Dimjašević, M., Atzeni, S., Ugrina, I., Rakamaric, Z.: Evaluation of android malware detection based on system calls. In: Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, pp. 1–8 (2016)
26. Du, Y., Wang, J., Li, Q.: An android malware detection approach using community structures of weighted function call graphs. IEEE Access **5**, 17478–17486 (2017)
27. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435 (2016)
28. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) ESORICS 2017. LNCS, vol. 10493, pp. 62–79. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66399-9_4
29. Gruver, B.: Smali/baksmali tool (2015)
30. Hou, S., Ye, Y., Song, Y., Abdulhayoglu, M.: Hindroid: an intelligent android malware detection system based on structured heterogeneous information network. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1507–1515 (2017)
31. Íncer Romeo, Í., Theodorides, M., Afroz, S., Wagner, D.: Adversarially robust malware detection using monotonic classification. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics, pp. 54–63 (2018)

32. John, T.S., Thomas, T.: Adversarial attacks and defenses in malware detection classifiers. In: Handbook of Research on Cloud Computing and Big Data Applications in IoT, pp. 127–150. IGI global (2019)
33. John, T.S., Thomas, T., Emmanuel, S.: Graph convolutional networks for android malware detection with system call graphs. In: 2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP), pp. 162–170. IEEE
34. Lee, J., Kim, J., Kim, I., Han, K.: Cyber threat detection based on artificial neural networks using event profiles. IEEE Access **7**, 165607–165626 (2019)
35. Li, D., Li, Q.: Adversarial deep ensemble: evasion attacks and defenses for malware detection. IEEE Trans. Inf. Forensics Secur. **15**, 3886–3900 (2020)
36. Li, D., Li, Q., Ye, Y., Xu, S.: Sok: Arms race in adversarial malware detection. arXiv preprint arXiv:2005.11671 (2020)
37. Peiravian, N., Zhu, X.: Machine learning for android malware detection using permission and API calls. In: 2013 IEEE 25th international conference on tools with artificial intelligence, pp. 300–305. IEEE (2013)
38. Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S.: Rage against the virtual machine: hindering dynamic analysis of android malware. In: Proceedings of the Seventh European Workshop on System Security, pp. 1–6 (2014)
39. Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L.: Intriguing properties of adversarial ml attacks in the problem space. In: 2020 IEEE Symposium on Security and Privacy (SP), pp. 1332–1349. IEEE (2020)
40. Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y.: Generic black-box end-to-end attack against state of the art API call based malware classifiers. In: Bailey, M., Holz, T., Stamatogiannakis, M., Ioannidis, S. (eds.) RAID 2018. LNCS, vol. 11050, pp. 490–510. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00470-5_23
41. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: 2012 European Intelligence and Security Informatics Conference, pp. 141–147. IEEE (2012)
42. Souri, A., Hosseini, R.: A state-of-the-art survey of malware detection approaches using data mining techniques. Hum.-centric Comput. Inf. Sci. **8**(1), 3 (2018)
43. strace: https://strace.io/ (2015)
44. Surendran, R., Thomas, T., Emmanuel, S.: Gsdroid: graph signal based compact feature representation for android malware detection. Expert Syst. Appl. 113581 (2020)
45. Venkatraman, S., Alazab, M.: Use of data visualisation for zero-day malware detection. Secur. Commun. Netw. **2018** (2018)
46. Venkatraman, S., Alazab, M., Vinayakumar, R.: A hybrid deep learning image-based analysis for effective malware detection. J. Inf. Secur. Appli. **47**, 377–389 (2019)
47. Vidas, T., Christin, N.: Evading android runtime analysis via sandbox detection. In: Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, pp. 447–458 (2014)
48. Vinayakumar, R., Alazab, M., Srinivasan, S., Pham, Q.V., Padannayil, S.K., Simran, K.: A visualized botnet detection system based deep learning for the internet of things networks of smart cities. IEEE Trans. Ind. Appl. (2020)
49. Vinod, P., Zemmari, A., Conti, M.: A machine learning based approach to detect malicious android apps using discriminant system calls. Future Gen. Comput. Syst. **94**, 333–350 (2019)

50. Winsniewski, R.: Android-apktool: A tool for reverse engineering android APK files. **10**, 2020 (2012)
51. Wu, D.J., Mao, C.H., Wei, T.E., Lee, H.M., Wu, K.P.: Droidmat: android malware detection through manifest and API calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security, pp. 62–69. IEEE (2012)
52. Wu, W.C., Hung, S.H.: Droiddolphin: a dynamic android malware detection framework using big data and machine learning. In: Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, pp. 247–252 (2014)
53. Yang, W., Kong, D., Xie, T., Gunter, C.A.: Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 288–302 (2017)
54. Yerima, S.Y., Sezer, S.: Droidfusion: a novel multilevel classifier fusion approach for android malware detection. IEEE Trans. Cybern. **49**(2), 453–466 (2018)