How Can We Craft Large-Scale Android Malware? An Automated Poisoning Attack

Sen Chen*, Minhui Xue†, Lingling Fan*, Lei Ma‡, Yang Liu*, Lihua Xu§
*Nanyang Technological University, Singapore †Optus Macquarie University Cyber Security Hub, Australia
‡Harbin Institute of Technology, China §New York University Shanghai, China
ecnuchensen@gmail.com

***Abstract*—Android malware, is one of the most serious threats to mobile security. Today, machine learning-based approach is one of the most promising approaches in detecting Android malware. However, our previous experiments show that sophisticated attackers can craft large-scale Android malware to pollute training data and pose an automated poisoning attack on machine learning-based malware detection systems (e.g., DREBIN, DROIDAPIMINER, STORMDROID, and MAMADROID), and eventually mislead the detection tools. We further examine how machine learning classifiers can be mislead under four different attack models and significantly reduce detection accuracy. Apart from Android malware, to better protect mobile devices, we also discuss a general threat model of Android devices to investigate the capabilities of different attackers.**

***Index Terms*—Android malware detection, Adversarial machine learning, Poisoning attack**

I. INTRODUCTION

Currently, Android malware detection is one of the most important security research topics. Many state-of-the-art techniques are used for this task. For example, signature-based, behavior-based, data flow-based, and learning-based approach. Among them, machine learning-based approach is one of the most promising techniques for Android malware detection [1], [2], [6], [8]. However, sophisticated attackers are able to generate crafted samples into training dataset by reverse-engineering to conduct an automated large-scale *poisoning attack*.[1]

Based on our investigation, most previous work using machine learning mainly focused on detection accuracy and assumed that feature extraction process is under an ideal environment [9]. In this paper, we describe a threat model within a specific class of attacks (i.e., *poisoning attack*), with three types of attackers (i.e., weak attacker, strong attacker, and sophisticated attacker). In *poisoning attack*, attackers are assumed to take in charge of part of the samples or add seeds to mislead the classifiers. Through reverse-engineering, the attackers are able to gain an in-depth understanding of the Android apps, and further adopt a customized crafting algorithm to generate large-scale crafted samples. These crafted samples are misclassified by the original machine learning classifiers, therefore, they can actively tamper with the machine learning classifiers by injecting large-scale well-crafted samples into training data to reduce the detection accuracy significantly.

[1]More details can be found in [5].

In this paper, we show a significantly high misclassification rate on malicious applications when using 564 extracted features. we investigate the ability of poisoning attack on the four most recent machine-learning detection systems in academia: DREBIN [2], DROIDAPIMINER [1], MAMADROID [9], and STORMDROID [6]. The results show that almost all the existing machine learning-based Android malware detection systems are suffering from the poisoning attack [5], [7].

Besides the poisoning attack in the adversary model, mobile devices also suffer from various potential threats. In order to better protect mobile devices, we also propose a general *threat model* to characterize attackers in different scenarios. In this paper, (1) we introduce an automated poisoning attack under three different classes. Moreover, we show a customized crafting algorithm, to generated well-crafted samples automatically using syntax features. (2) we show that the poisoning attack is able to mislead DREBIN, DROIDAPIMINER, STORMDROID, and MAMADROID, reducing the classification accuracy. (3) we also propose a general threat model of mobile devices to investigate the capabilities of different attackers to help mobile protection.

II. POISONING ATTACK

The goal of adversarial sample crafting in malware detection is to mislead the detection system, causing the classification result to change according to the attackers' expectations. In this paper, we focus on *poisoning attack* that results in malware being misclassified as benign (false negative).

We start the definition by denoting a sample set by $\{(x_i, y_i) \in (\mathcal{X}, \mathcal{Y})\}_{i=1}^{n}$, where n is the total number of samples, and $x_i = \{x_{i1}, x_{i2}, \cdots x_{im}\}$ is the feature vector of the i^{th} sample. Each component $x_{ij} \in x_i$ indicates the existence of the j^{th} component, if it exists, then $x_{ij} = 1$, otherwise, $x_{ij} = 0$. $y_i \in \{0, 1\}$ indicates the attribute of the i^{th} sample (i.e., 0 for benign, 1 for malicious). $\mathcal{X} \subseteq \{0, 1\}^m$ is a m-dimensional feature space. In this paper, we consider binary classifiers with two output classes (benign ot malicious).

The attackers attempt to evade binary classifiers by adding a non-zero displacement feature vector θ_i to $x_i|_{y_i=1}$, with the ultimate aim of $y_i = 0$. For example, attackers may add good features into malware to fool binary classifiers. Since attackers aim to evade the classifiers to classify malware as benign ones, we define our problem as crafting an adversarial sample x^*, misclassified by the function f (where $f : x \to y = f(x)$), as a benign sample x, and formalized as follows [11]:

$$x^* = x + \theta_x \quad \text{s.t.} \quad f(x+\theta_x) \neq f(x) \tag{1}$$

where $\theta_x = x$ is the minimal perturbation yielding misclassification. We assume the attacker has full access to the used classifier, and can inject as many variants' features as possible. Therefore, followed by Equation (1), we further denote $x_{ij}^{\max}$ and $x_{ij}^{\min}$ as the maximum and the minimum values that the j^{th} feature of the i^{th} sample can take. A poisoning attack can be formalized as follows:

$$A_f(x_{ij}^{\min} - x_{ij}) \leq \theta_{ij} \leq A_f(x_{ij}^{\max} - x_{ij}), \ \forall j \in [1, m] \tag{2}$$

where $x_{ij}^{\max} = 1$, $x_{ij}^{\min} = 0$, and $A_f \in [0, 1]$ indicates the aggressiveness of attacks. $A_f = 0$ indicates no attacks, while $A_f = 1$ indicates the most aggressive attacks. To check the consequences of causal attacks and elaborate the challenges, we develop an adversarial model with three types of attackers with the corresponding A_f values as follows:

- *Weak attacker ($A_f = 0.33$).* Our weak attacker is totally unaware of the statistical properties of the training features or labels. This attacker simply fakes additional labels with random binary features to poison the training dataset.
- *Strong attacker ($A_f = 0.67$).* Our strong attacker is aware of the features we use for training and can has access to our ground-truth dataset (which comes from public sources). This attacker can manipulate partial features in the training data. However, this attacker is resource constrained and cannot manipulate any mobile application statistics which would require more time. The strong attacker crafts features by randomly selecting public available Android malware and then faking additional labels, so that the partial training labels can become nearly identical.
- *Sophisticated attacker ($A_f = 1$).* Our strongest attacker, named sophisticated attacker, has full knowledge of our training feature set. Additionally, this attacker has sufficient time and economic resources to create arbitrary mobile application statistics. Therefore, the sophisticated attacker can fully manipulate almost all training features, which creates scenarios where relatively benign mobile applications and real-world malicious mobile applications appear to have nearly identical attributes at the training phase.

To achieve this, we leverage the adversarial crafting algorithm [10] based on the Jacobian matrix

$$J_f = \frac{\partial f(\mathcal{X})}{\partial \mathcal{X}} = \left[\frac{\partial f_i(x)}{\partial x_j}\right]_{i \in \{0,1\}, j \in [1,m]}$$

where $f_i(x)$ indicates the output of sample x (either malicious or benign), $i = 0$ indicates x is benign and $i = 1$ indicates x is malicious.

To craft adversarial samples, we construct a feature database with 73 benign features and 102 malicious features. We then take two steps: (1) we compute the gradient of f with respect to x to estimate the direction where a perturbation in x would change the output of f; (2) we choose a perturbation θ of x with maximal positive gradient into our target class $\mathcal{Y}_{y_i=0|y_i=1}$, denoted as y^*, and we then customize the adversarial crafting algorithm [10] according to our adversarial model with three types of attackers (A_f), to indicate the probability (Equation (2)) of adding a specific feature. After computing the gradient, we iteratively choose a target feature whose gradient is the largest for our target class and then update its value in x to obtain our new input vector. We then re-update the gradient and repeat this process until either we reach the bounded allowed changes (loop bound) or we successfully achieve a misclassification.

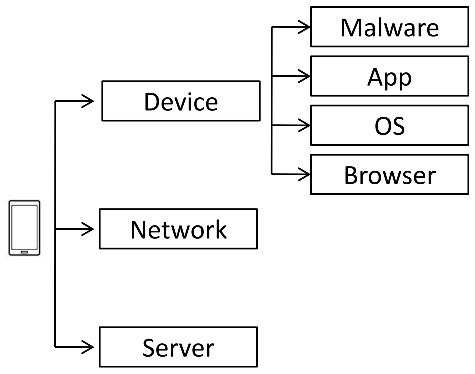

Fig. 1. General Threat Model of Mobile Devices

III. General Threat Model and Adversarial Attacks

Our adversarial model mentioned above (based on Android malware) is only one of the potential attack surfaces of mobile devices. We consider as many attack surfaces as possible and propose a general threat model of mobile devices. Based on the understanding of the threat model, we can investigate whether the existing protection mechanisms (e.g., sandbox, signature, permission-based mechanism, access control, Binder mechanism, memory management mechanism) are safe or not, and evaluate the attack efforts.

As shown in Fig. 1, the potential threats (or attack surfaces) are mainly from three aspects where attackers may exploit: *Device, Network and Server*. There are a variety of entry points when attackers are targeting the mobile device, such as mobile malware (e.g., fake apps [12]), operate system, web browser and application itself [3], [4] (i.e., security issues in applications of mobile device). The detailed descriptions of the threat model are as follows.

A. Device-Based Attackers

The threats of malware. As for this adversarial attacker, we assume a crafted malicious application is pre-installed on the same device with benign applications. Still, it is restricted by the sandbox isolation. In such a case, this malicious application is only able to access the public available data, take screenshots of users' input, receive implicit Intents, etc. Therefore, all sensitive data that leaks to these sinks can be obtained by the malware. Attackers can either monitor the application status (running or not) via malware or even take in charge of the mobile device via remote control. Malware mainly do harm to the data confidentiality.

The threats of application. Mobile applications occupy an extremely large proportion. Despite the great convenience provided to customers, potential security issues are hiding behind the prosperity. Sensitive data leakage and insecure communication are the main security issue categories. For example, an attacker is able to obtain data stored in internal storage (e.g., SharedPreference, SQLite) or external storage (e.g., SD Card), and also from debugging output. It can be imagined that these security issues in mobile applications could be the time bomb counting down for large-scale destruction.

The threats of OS. Rooted mobile devices allow users to alter or replace system applications and settings, run specialized applications that require administrator-level permissions. It can result in the exposure of sensitive data. Besides, SMS point of attackers can use mobile device text messages in place of e-mail messages in order to prompt users to visit illegitimate websites and enter sensitive information such as usernames, passwords and verification code. A zero-day attack happens once that flaw, or software/hardware vulnerability, is exploited and attackers release malware before a developer has an opportunity to create a patch to fix the vulnerability. It is an advanced attack of mobile devices.

The threats of browser. Research has shown that mobile users are three times more likely than desktop users to submit personal information to phishing websites. The sender can either be a malicious app that links users to a spoofed screen or a benign app whose link is intercepted by another party and is sent to a spoofed target. Browser-based attackers can execute remote code through exploit of browser vulnerabilities to take in charge of the mobile device.

B. Network-Based Attackers

Network-based attackers are in the middle of applications and their corresponding servers. Under this circumstance, attackers are able to obtain all communication data between applications and the corresponding servers. If the communication data is plain without encryption, attackers can immediately read it and subsequently subvert applications. If the communication data is encrypted with a weak or vulnerable cryptographic algorithm, it is still possible in practice for attackers to crack the communication. Hence, network-based attackers can harm data confidentiality, integrity and availability.

For network-based attackers, we can set up a proxy server with FIDDLER as a relay of applications (Man-in-the-middle), and 1) intercept all plain data in communication; 2) disguise as a genuine server to copy information; 3) tamper exchanged data. Besides the above attacks, other common network-based attacks contain packet sniffing, session hacking, DNS poisoning, SSL strip, etc.

C. Server-Based Attackers

Server handles substantial core data of applications, if misconfigured, the data may be leaked to other parties by accident. Apart from problems of its own, they also suffer from the following attacks.

Cross-site scripting (XSS) is a kind of injecting attack that injects malicious JavaScript code into a secure website. If the page is vulnerable to this attack, it'll return user input to the browser without proper sanitization. This attack is often used to run code automatically when a user visits a page, taking control of a user's PC browser. After taking control of the browser, the attacker can leverage that control into a variety of attacks, such as content injection or malware propagation.

Cross-site request forgery (CSRF) is an attack that involves creating HTTP (Web) requests based on knowledge, and tricking a user or browser into submitting these requests. If a Web app is vulnerable, the attack can execute transactions or submissions that appear to come from the user. CSRF is normally used after an attacker has already taken control of a user's session, either through XSS or other methods.

IV. Evaluation on Attacks against the Detection

Our collected dataset contains 16,000 samples as the training data and 4,000 samples as the test data. The benign samples are downloaded from Google Play Store, and the other malicious samples are collected from four parties (e.g., Genome Project,[2] DREBIN Project,[3] Contagio Mobile Website,[4] and Pwnzen Infotech Inc.[5]). We select four state-of-the-art machine learning-based systems for Android malware detection (i.e., DREBIN [2], DROIDAPIMINER [1], MAMADROID [9], and STORMDROID [6]), to investigate the ability of poisoning attack under Support Vector Machine (SVM) since SVM is the only jointly-used algorithm by the four systems.

We demonstrate that by poisoning their training set, it is possible to mislead their corresponding classifiers. We then further discuss the results as follows.

Misclassification of Machine Learning Classifiers. We mimic the sophisticated attack of poisoning attacks to observe how ineffective these four malware detection systems perform. Specifically, we first assume we control a subset of samples, and then automatically generate a large-scale crafted samples by the following tow principles: (1) we can only add or remove features. We must preserve the utility of the modified application, which we achieve by only adding features from benign set, and only those that do not interfere with the functionality of the application. (2) We can add a restricted number of features. More specifically, we customize the adversarial crafting algorithm [10] according to our adversarial model, to indicate the probability of adding or removing a specific feature.

[2]http://www.malgenomeproject.org/
[3]https://www.sec.cs.tu-bs.de/∼danarp/drebin/
[4]http://contagiodump.blogspot.com/
[5]http://www.pwnzen.com/

TABLE I
MISCLASSIFICATION RATE COMPARISON OF ADVERSARIAL DETECTION

Detection Tool	DroidAPIMiner	Drebin	MaMaDroid	StormDroid
Misclassification rate (FN)	80.05%	75.20%	68.95%	65.35%

Based on the above crafting algorithm, we can apply it for machine learning mechanisms that are relied on syntax features, such as DROIDAPIMINER, DREBIN, and STORMDROID. MAMADROID relies on application behaviors using Markov chain modeling, while we use call sequences to craft the features. Specifically, we extract a set of call sequences that are frequently used by benign samples in our dataset, and then add them to the malicious samples to poison the training data, and further mislead the classifiers.

Table I shows that we obtain 80.05%, 75.20%, 68.95%, and 65.35% misclassification rates on DROIDAPIMINER, DREBIN, MAMADROID, and STORMDROID, respectively. We further discuss the results as follows.

- The poisoning attack is able to mislead the machine learning-based detection systems.
- STORMDROID achieves lower misclassification rates than other systems. However, it can still be attacked through poisoning attack, which indicates that it still suffers from crafted samples, although it uses the newly-defined and dynamic behavior features.
- MAMADROID uses transitional call sequences, rather than single API calls, to train its classifier, therefore, we craft its feature space through pre-generated call sequences from benign samples, to achieve a significant misclassification (i.e., 68.95%).
- DROIDAPIMINER, DREBIN, STORMDROID, and MAMADROID can be thwarted if we embed native code (as a strong attacker) and dynamic code loading with reflection (as a sophisticated attack), because malicious code is loaded or determined at runtime. The attackers can pollute training data using a large-scale crafted samples through these techniques.

In summary, we conjecture that almost all the state-of-the-art machine-learning-based malware detection systems are suffering from the poisoning attack we exhibited [5], [7].

V. CONCLUSION

We showed how the machine learning-based classifiers can fail against poisoning attack by our automated customized crafted algorithm, although machine learning-based approaches are able to help solve many security problems, such as Android malware detection. We argued that it is essential to inform researchers considering how attackers will adapt to the conventional detection, as well as to inform developers working on the next-generation malware detection systems. We also proposed a general threat model of mobile devices and showed mainly attack surfaces corresponding different potential adversarial attackers. It would be helpful to understand the existing security mechanisms of mobile devices and the attackers' efforts.

ACKNOWLEDGMENTS

We appreciate the reviewers' constructive feedback. This work is partially supported by NSFC Grant 61502170, NTU Research Grant NGF-2017-03-033 and NRF Grant CRDCG2017-S04.

REFERENCES

[1] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Security and Privacy in Communication Networks*. Springer, 2013, pp. 86–103.

[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[3] S. Chen, G. Meng, T. Su, L. Fan, Y. Xue, Y. Liu, L. Xu, M. Xue, B. Li, and S. Hao, "Ausera: Large-scale automated security risk assessment of global mobile banking apps," *arXiv preprint arXiv:1805.05236*, 2018.

[4] S. Chen, T. Su, L. Fan, G. Meng, M. Xue, Y. Liu, and L. Xu, "Are mobile banking apps secure? what can be improved?" in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2018, pp. 797–802.

[5] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, vol. 73, pp. 326–344, 2018.

[6] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "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*. ACM, 2016, pp. 377–388.

[7] S. Chen, M. Xue, and L. Xu, "Towards adversarial detection of mobile malware: poster," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 415–416.

[8] L. Fan, M. Xue, S. Chen, L. Xu, and H. Zhu, "Poster: Accuracy vs. time cost: Detecting Android malware through pareto ensemble pruning," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1748–1750.

[9] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MAMADROID: Detecting Android malware by building Markov chains of behavioral models," in *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*, 2017.

[10] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE, 2016, pp. 372–387.

[11] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the 2014 International Conference on Learning Representations*. Computational and Biological Learning Society, 2014.

[12] C. Tang, S. Chen, L. Fan, L. Xu, Y. Liu, Z. Tang, and L. Dou, "A large-scale empirical study on industrial fake apps," in *Proceedings of the 41th ACM/IEEE International Conference on Software Engineering, ICSE 2019*, 2019.