α*Cyber*: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model

Shifu Hou, Yujie Fan
Yiming Zhang, Yanfang Ye*
Dept. of CDS, Case Western Reserve University, OH, USA

Jingwei Lei, Wenqiang Wan
Jiabin Wang, Qi Xiong, Fudong Shao
Tencent Security Lab, Tencent, Guangdong, China

ABSTRACT

The explosive growth and increasing sophistication of Android malware call for new defensive techniques that are capable of protecting mobile users against novel threats. To combat the evolving Android malware attacks, systems of HinDroid and AiDroid have demonstrated the success of heterogeneous graph (HG) based classifiers in Android malware detection; however, their success may also incentivize attackers to defeat HG based models to bypass the detection. By far, there has no work on adversarial attack and/or defense on HG data. In this paper, we explore the robustness of HG based model in Android malware detection at the first attempt. In particular, based on a generic HG based classifier, (1) we first present a novel yet practical adversarial attack model (named *HG-Attack*) on HG data by considering Android malware attackers' current capabilities and knowledge; (2) to effectively combat the adversarial attacks on HG, we then propose a resilient yet elegant defense paradigm (named *Rad-HGC*) to enhance robustness of HG based classifier in Android malware detection. Promising experimental results based on the large-scale and real sample collections from Tencent Security Lab demonstrate the effectiveness of our developed system α*Cyber*, which integrates our proposed defense model *Rad-HGC* that is resilient against practical adversarial malware attacks on the HG data performed by *HG-Attack*.

CCS CONCEPTS

• **Artificial Intelligence** → **General**; • **Database applications** → *Data mining*; • **Security and Protection** → Invasive Software.

KEYWORDS

Android malware detection; heterogeneous graph (HG); node classification; adversarial attack and defense on HG.

ACM Reference Format:
Shifu Hou, Yujie Fan, Yiming Zhang, Yanfang Ye, Jingwei Lei, Wenqiang Wan, and Jiabin Wang, Qi Xiong, Fudong Shao. 2019. α*Cyber*: Enhancing Robustness of Android Malware Detection System against Adversarial Attacks on Heterogeneous Graph based Model. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM'19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357875

*Corresponding author: yanfang.ye@case.edu

CIKM '19, November 3–7, 2019, Beijing, China

ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357875

1 INTRODUCTION

Due to the mobility and ever expanding capabilities, smart phones have become increasingly ubiquitous in people's everyday life performing tasks such as social networking and online banking. Android, as an open source and customizable operating system (OS) for smart phones, is currently dominating the market by 74.85% [19]. However, due to its large market share and open source ecosystem of development, Android attracts not only the developers for producing legitimate Android applications (apps), but also attackers to disseminate malware (***mal***icious soft***ware***) that deliberately fulfills the harmful intent to smart phone users. Driven by considerable economic profits, there has been explosive growth of Android malware - i.e., according to Tencent Security Lab [23], there have been 4, 687, 008 newly generated Android malware that infected more than 61 million smart phones in the first half of 2018. The large volume of increasingly sophisticated Android malware has posed serious threats to smart phone users, such as stealing user's credentials, pushing unwanted apps or advertisements (ads) [14]. Therefore, the detection of Android malware is of major concern to both anti-malware industry and researchers.

Attackers and defenders always engage in a never-ending arms race. At each round, both of them try to analyze methodologies and vulnerabilities of each other, and develop their own optimal strategies to overcome the opponents [1], which has led to considerable countermeasures of variability and sophistication between them. For example, Android malware attackers employ techniques such as repackaging and obfuscation to bypass the signature-based detection and defeat attempts to analyze their inner mechanisms [26]. To combat the evolving Android malware attacks, systems applying data mining and machine learning techniques have been developed for Android malware detection [8, 14, 18, 22, 24, 25], where different kinds of classification models are constructed based on different feature representations to detect malicious apps. Different from most of the existing works that merely leveraged content-based information (i.e., statically or dynamically extracted features from Android apps) for malware detection, HinDroid [14] and AiDroid [25] were proposed which considered higher-level semantic relations among apps and other types of entities (e.g., Application Programming Interfaces (APIs) called by apps, smart phones where apps installed, signatures signed by app developers, etc.) and introduced structured heterogeneous graphs (HGs) to model such complex relations for Android malware detection. These systems

resting on HG based models have been successfully deployed in anti-malware industry [8, 14, 25]. However, the success of deployments may also incentivize attackers to defeat HG based models to bypass the detection. To put this into perspective, as shown in Figure 1, driven by considerable profits, malware attackers are organized within the complicated and decentralized ecosystem, which enables them to have powerful capabilities: by exploiting vulnerabilities and/or utilizing social engineering tactics (e.g., induced installation), attackers are capable of downloading apps to compromised devices from the Command and Control (C&C) servers and executing them on-demand. Such mechanism makes evasion attacks together with poisoning attacks on HG based models realistic: under the cover of injected apps that perturb the relational (non-i.i.d.) nature of the data in HG (i.e., poisoning attacks), the target apps (i.e., new malware) can be better "protected" to bypass the detection (i.e., evasion attacks).

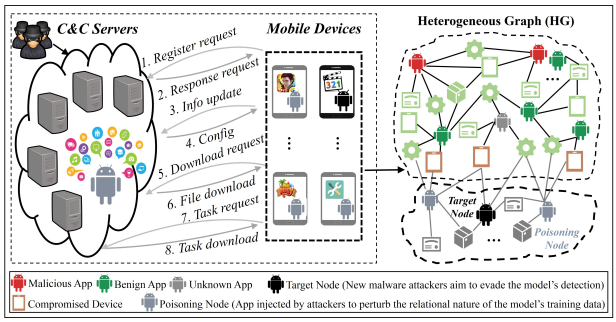


Figure 1: Adversarial attacks on heterogeneous graph (HG).

With the popularity of machine learning based models deployed for various applications, the issues of understanding their security in adversarial settings have been widely studied ranging from traditional learning models (e.g., Support Vector Machine (SVM) or logistic regression [17]) towards deep neural networks [4, 10, 16, 21]. However, majority of these works are based on the assumption that data samples are independent. Although there have been a few studies of adversarial settings on non-i.i.d. samples (e.g., graph data) [2, 6, 28, 29], these works only considered homogeneous graph data. Due to the heterogeneous property (i.e., graph consisting of multi-typed entities and relations), it is difficult to directly apply these existing adversarial settings on heterogeneous graph. **By far, there has no work on adversarial attack and/or defense on heterogeneous graph data.** Our work in this paper aims to bridge this gap with the application in Android malware detection.

In this paper, to explore the robustness of HG based classifiers in Android malware detection, built upon the preliminary work [14, 25], we first construct a generic HG based classification model: we extract the API call sequences from runtime executions of Android apps to capture their behaviors; and then we further analyze higher-level semantic relations such as whether two apps have similar behaviors, whether they co-exist in the same smart phone that can be identified by its unique International Mobile Equipment Identity (IMEI) number, and whether they are signed by the same developer or produced by the same company (i.e., affiliation); later, we present a structured HG to model such complex relations and exploit meta-path based embedding approach to learn the representations of nodes (i.e., apps) fed to the downstream classifier. Based on the constructed HG based classifier, we first present a novel yet practical adversarial attack model (named *HG-Attack*) on HG data by considering Android malware attackers' current capabilities and knowledge. Then, to effectively combat the adversarial attacks on HG, we further propose a resilient yet elegant defense model (named *Rad-HGC*) to enhance robustness of HG based classifier in Android malware detection. Promising experimental results based on the large-scale and real sample collections from Tencent Security Lab demonstrate the effectiveness of our developed system *αCyber* (as shown in Figure 2), which integrates our proposed defense model *Rad-HGC* that is resilient against practical adversarial malware attacks on HG performed by *HG-Attack*. The major contributions of our work in this paper can be summarized as follows:

- ***Novel yet practical adversarial attacks on HG data:*** In the adversarial point of view, to conduct a practical attack, attackers need to answer the following question: *how to optimally inject poisoning nodes (i.e., apps) to influence the relational nature of the data in HG to make the classifier maximally misclassify the target node (i.e., new malware) as benign*? Note that the efficacy of adversarial attacks is also constrained in the capabilities and knowledge attackers possess. In this paper, *HG-Attack* model is proposed to answer the above question, by taking consideration of Android malware attackers' current capabilities (e.g., the compromised devices they have and the limit of poisoning nodes they can leverage) and knowledge (i.e., the information of the training data and learning algorithms).
- ***Resilient yet elegant defense model against adversarial attacks on HG:*** Like the game of packing and unpacking between malware attackers and defenders [26], to be resilient against the adversarial attacks on HG, defenders need to resolve the following puzzle: *how to uncover the poisoning nodes (i.e., injected apps) in the HG in order to detect the target node (i.e., new malware)?* To solve this problem, *Rad-HGC* is proposed to enhance the robustness of HG based model against the adversarial attacks while not compromising its detection accuracy.
- ***A practical and robust system against adversarial Android malware attacks on HG based model:*** We obtain two large-scale and real sample collections from Tencent Security Lab: (1) the first data set is the historically accumulative data including 1,389,408 apps uploaded by 70,184 users (i.e., IMEIs) and its generated HG (i.e., denoted as *HG-1* consisting of 1,389,408 nodes with five different entity types and 20,576,125 edges with six types of relations); and (2) the second data set is generated based on *HG-1* which further incorporates 13,129 new apps uploaded by 2,817 mobile users (i.e., IMEIs). Based on these data collections, we develop a system named *αCyber*, which integrates our proposed defense model *Rad-HGC* that is resilient against practical adversarial malware attacks performed by *HG-Attack*.

2 HETEROGENEOUS GRAPH BASED MODEL FOR ANDROID MALWARE DETECTION

In this section, we define the Android malware detection problem based on a generic HG based classification model.

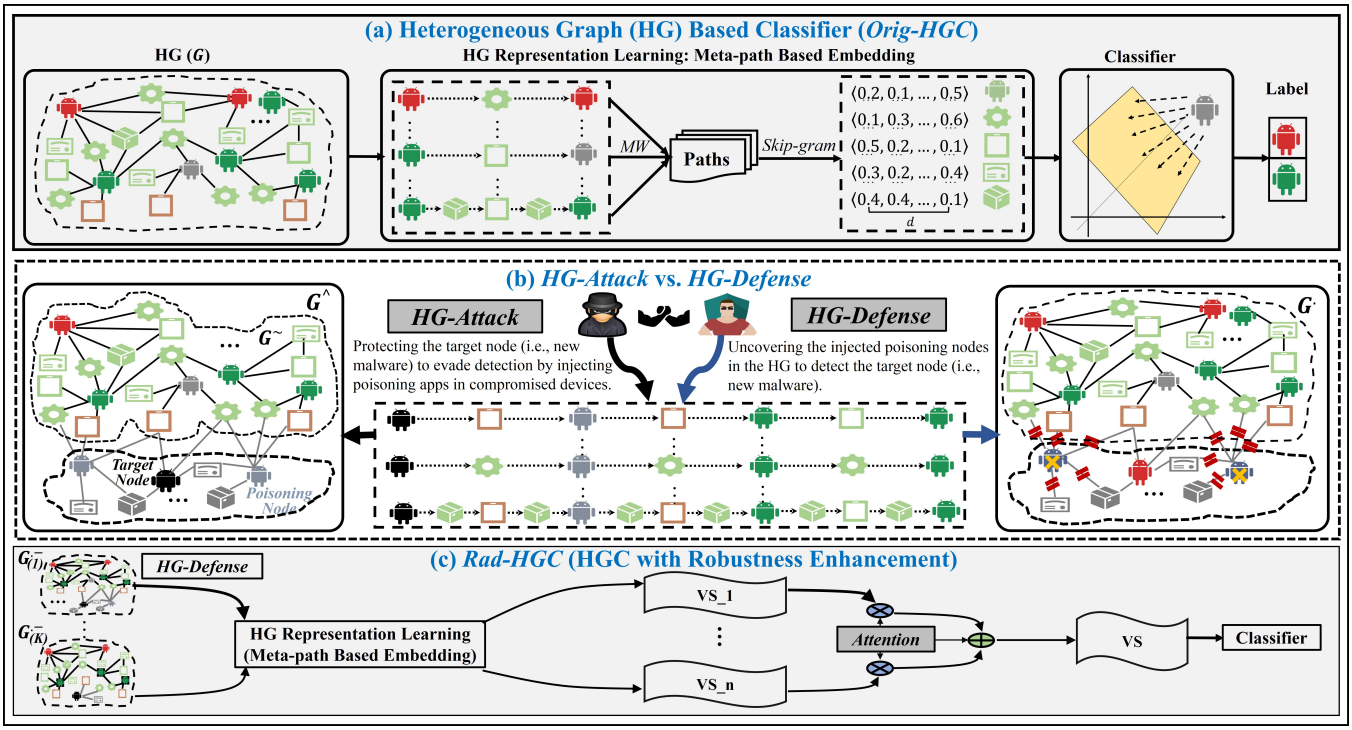


Figure 2: An architecture overview of α*Cyber*. In α*Cyber*, a generic HG based classifier (denoted as *Orig-HGC*) is first introduced; and then novel yet practical adversarial model on HG (denoted as *HG-Attack*) is presented; later, integrating our proposed defense mechanism (i.e., *HG-Defense*), a detection model with robustness enhancement (denoted as *Rad-HGC*) is devised against the adversarial attacks on HG based model in Android malware detection.

2.1 Feature Extraction

To detect Android malware, we consider both content- and relation-based information to comprehensively describe the Android apps.

Content-based Feature Extraction. API calls are used by Android apps in order to access Android OS functionality and system resources. Therefore, we extract the sequences of API calls in the application framework from runtime executions of Android apps to capture their behaviors. For example, the sequence of API calls (*startActivity*, *checkConnect*, *sendSMS*, *finishActivity*) denotes the intention of sending SMS messages without user's concern by a malicious "TigerEyeing" trojan.

Relation-based Feature Extraction. To detect the increasingly sophisticated Android malware, we further extract the following kinds of relations: (1) ***R1***: The *app-invoke-API* means if an app invokes an API call during runtime execution. (2) ***R2***: The *app-exist-IMEI* indicates if an app exists (i.e., is installed) in a smart phone (i.e., IMEI). (3) ***R3***: The *app-certify-signature* means if an app is certified by a signature (i.e., every app run on the Android platform must be signed by the developer). (4) ***R4***: Package name (a.k.a. Google Play ID) is a unique name to identify a specific app. Companies conventionally use their reversed domain names to begin their package names (e.g., "com.tencent.mobileqq"). We extract domain name from package name to denote the relation between an app (e.g., "mobileqq") and its affiliation (e.g., "tencent.com"); and then we generate the *app-associate-affiliation* to describe if an app is associated with an affiliation. (5) ***R5***: To represent that a smart phone has a set of apps signed by a particular developer, we extract the *IMEI-have-signature* to indicate if a device has a specific signature. (6) ***R6***: To denote that a smart phone installs a set of apps associated with a specific affiliation, we generate the *IMEI-possess-affiliation* to describe if a smart phone possesses a particular affiliation.

2.2 Heterogeneous Graph Construction

To depict apps, APIs, IMEIs, signatures, affiliations and the rich relations among them (i.e., *R1-R6*), we introduce HG to model them, which is able to be composed of multi-typed entities and relations.

Definition 1. *A **heterogeneous graph (HG)** [20] is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with an entity type mapping $\phi: \mathcal{V} \rightarrow \mathcal{A}$ and a relation type mapping $\psi: \mathcal{E} \rightarrow \mathcal{R}$, where $\mathcal{V}$ denotes the entity set and $\mathcal{E}$ is the relation set, $\mathcal{A}$ denotes the entity type set and $\mathcal{R}$ is the relation type set, and the number of entity types $|\mathcal{A}| > 1$ or the number of relation types $|\mathcal{R}| > 1$. The **network schema** [20] for the $\mathcal{G}$, denoted as $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, is a graph with nodes as entity types from $\mathcal{A}$ and edges as relation types from $\mathcal{R}$.*

HG not only provides the network structure of data associations, but also a high-level abstraction of categorical association. Based on the definitions above, the network schema shown in Figure 3 enables the apps to be represented in a comprehensive way that utilizes their semantic and structural information.

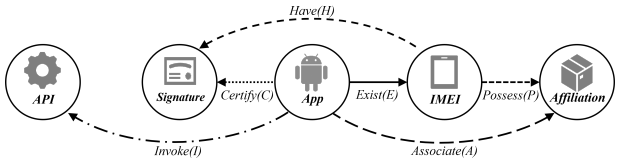


Figure 3: Network schema in our application.

To formulate the relatedness among entities in HG, the concept of meta-path has been proposed [20]: a ***meta-path*** $\mathcal{P}$ is a path defined on the network schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \ldots \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \ldots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is length of $\mathcal{P}$. Based on the network schema shown in Figure 3, incorporated anti-malware experts' domain knowledge, we design six meaningful meta-paths to characterize the relatedness over apps at different views (i.e., *PID1-PID6* shown in Figure 4). For example, *PID1* depicts that two apps are related if they both invoke the same API (e.g., two malicious mobile video players both invoke the API of "*requestAudioFocus*").

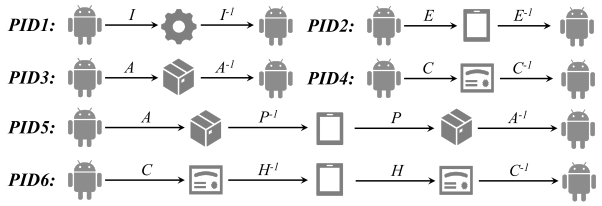


Figure 4: Meta-paths built for Android malware detection.

2.3 Classifier Based on Heterogeneous Graph

After constructing the HG, the problem of Android malware detection can be considered as node classification in HG. To efficiently solve this problem, we first present the concept of HG representation learning [7, 9]: given a HG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the representation learning task is to learn a function $g : \mathcal{V} \rightarrow \mathbb{R}^d$ that maps each node $v \in \mathcal{V}$ to a vector in a d-dimensional space $\mathbb{R}^d$, $d \ll |\mathcal{V}|$ that are capable to preserve the structural and semantic relations among them. To learn the presentations of nodes in HG, various embedding methods [7–9] have been proposed. In this work, without loss of generality, we exploit metapath2vec [7] which employed meta-path based random walks and heterogeneous skip-grams to learn the latent representations for HG such that the semantic and structural correlations between different types of nodes could be persevered. After employing metapath2vec to learn representations of nodes (e.g., apps) in HG, we denote our dataset D to be of the form $D = \{\mathbf{x}_i, y_i\}_{i=1}^n$ of n apps, where $\mathbf{x}_i \in \mathbb{R}^d$ is the learned representation for app i, and y_i is the class label of app i ($y_i \in \{+1, -1, 0\}$, +1: malicious, −1: benign, and 0: unknown). To this end, the problem of Android malware detection can be stated in the form of: $h : \mathcal{X} \rightarrow \mathcal{Y}$ which assigns a label $y \in \mathcal{Y}$ (i.e., −1 or +1) to an input app $\mathbf{x} \in \mathcal{X}$ through the learning function h (i.e., without loss of generality, in this paper, we use SVM as the downstream classifier after learning representations of apps in HG). We denote this original HG based classifier as *Orig-HGC* throughout the paper.

3 ADVERSARIAL ATTACKS ON HG

In Android malware detection, a HG based detection system is to detect malicious apps based on the classifier trained on HG data and prevent them from interfering users' smart phones. In contrast, attackers would like to perform adversarial attacks on HG (i.e., deliberate perturbations of HG data that can lead to misclassification) to violate the security context. Since the efficacy of adversarial attacks is constrained in the capabilities and knowledge attackers possess, we first present the power of Android malware attackers based on the current development of malware industry.

3.1 Preliminary and Problem Definition

Goal of attackers. Given a newly developed malicious app $\mathbf{x} \in \mathcal{X}^+$ that has bypassed content-based classifiers (i.e., classifiers only take content-based features as input and assume apps are independent), the goal of Android malware attackers is to devise a model that can enable it to be misclassified as benign by the HG based classifier (i.e., $\mathbf{x} \in \mathcal{X}^-$), with optimal perturbations on HG.

Capabilities of attackers. According to different capabilities attackers have, they can perform: (1) *evasion attack* which exploits misclassification without affecting training distribution, or (2) *poisoning attack* which influences the training data that is used to construct the classifier, or (3) both of them. As previously mentioned, in Android malware industry, the new kind of C&C malware enables attackers to leverage C&C servers to create powerful networks of compromised devices capable of downloading and executing apps on-demand. For example, a malicious app in "SDKSmartPush" family with package name of "com.i***soft.***.gamecenter" and signature hash value of "20B***12F" is able to query the C&C servers to download and execute other malicious apps to push unwanted ads in users' smart phones; it is also capable of downloading and executing benign apps such as mobile games and tool apps on-demand. Such powerful capabilities enable attackers to *concurrently perform evasion and poisoning attacks on HG*: they can cleverly devise tactics to inject poisoning apps in the compromised devices to protect the new malware (i.e., target node in HG) to evade the detection.

Knowledge of attackers. Attacker can have different levels of knowledge about (i) the training data (i.e., the HG $\mathcal{G}$), and (ii) the learning algorithms (i.e., HG representation learning algorithm g and classification function h in our case). As apps in users' smart phones (including compromised and non-compromised devices) are dynamically generated, attackers can only have partial knowledge about the training data (i.e., a subset of HG data). Practically, in our work, we assume attackers can have complete knowledge about the learning algorithms of *Orig-HGC* and use a subset of the data to devise a surrogate model to perform attacks on the HG.

Problem Definition. Based on the capabilities and knowledge of attackers, the problem of an adversarial attack on the HG based classifier (i.e., *Orig-HGC*) can be defined as: given a subset of HG data (i.e., a sub-HG) $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$, an adversarial attack is to optimally inject poisoning nodes $\mathcal{V}_P$ along with the target node v_t (i.e. new malicious app $\mathbf{x}$) into $\mathcal{G}$, leading to an adversarial HG $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ where $\hat{\mathcal{V}} = \tilde{\mathcal{V}} \cup \mathcal{V}^*$ ($\{\mathcal{V}_P, v_t\} \subseteq \mathcal{V}^*$) and $\hat{\mathcal{E}} = \tilde{\mathcal{E}} \cup \mathcal{E}^*$ ($\mathcal{E}^*$ is the set of new generated edges after injecting $\mathcal{V}^*$ in $\tilde{\mathcal{G}}$); taking $\hat{\mathcal{G}}$ as part of training data, the attack will lead the learning model (i.e., employing HG representation learning algorithm g and classification function h) to maximumly misclassify v_t as benign.

3.2 *HG-Attack* : Adversarial Attack Model

Based on the above definition of adversarial attack on *Orig-HGC*, to make the classifier h maximumly misclassify v_t as benign, attackers can perturb the embeddings - representations of nodes (i.e., apps) - learned from g which are fed to train the classifier h. In *Orig-HGC*, the embeddings are learned from metapath2vec (i.e., g) which first generates a set of meta-path guided random walks that serve as a training corpus for the skip-gram model; in this way, the learned embeddings are largely dependent on the generated corpus. By injecting poisoning nodes along with target node in the given $\tilde{\mathcal{G}}$, the attacker necessarily changes a set of possible random walks and thus influences the training corpus and subsequent embeddings. *How to optimally inject poisoning nodes (i.e., apps) to influence the embeddings to maximize the misclassification of v_t?* Following observation enlightens us for the solution to answer this question.

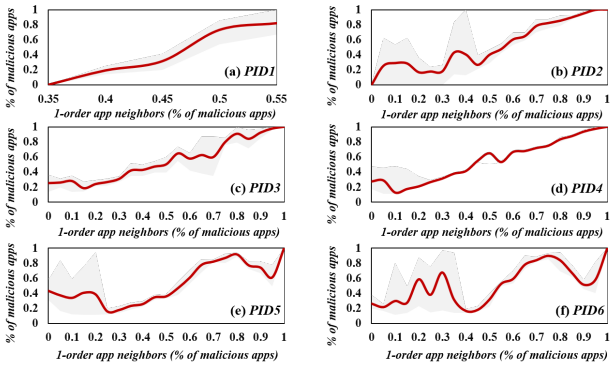


Figure 5: 1-order *app-app* neighborhood relations under different meta-path schemes (i.e., *PID1-PID6*).

Based on a large-scale and real data collection from Tencent Security Lab, including 1,389,408 apps uploaded by 70,184 users (i.e., 217,107 are malicious, 547,366 are benign, and 624,935 are unknown) and its constructed HG (i.e., ***HG-1*** as described in Section 5.1). As shown in Figure 5, guided by the six designed meta-paths (i.e., *PID1-PID6*), we observe that the more malicious apps the node (i.e., app) neighbors the higher probability the node is classified as malicious, and vice versa. Based on this observation, to make target node v_t bypass the detection, attackers can devise tactics to cleverly inject poisoning nodes to perturb the relational nature of the data (i.e., optimally link target node to benign apps via injected nodes). To this end, given $\tilde{\mathcal{G}}$, the problem of adversarial attack turns to maximize the likelihood of target node v_t connecting with benign apps by optimally injecting poisoning nodes $\mathcal{V}_P$ in $\tilde{\mathcal{G}}$ while minimize the probability of v_t being predicted as malicious.

To solve this problem, we first assume that if $v_p \in \mathcal{V}_P$ is successfully injected along with v_t in $\tilde{\mathcal{G}}$, then it should be capable of connecting v_t with benign apps in $\tilde{\mathcal{G}}$ (i.e., it should have high connectivity after being injected). To determine how to inject v_p, based on given $\tilde{\mathcal{G}}$, we would like to first measure its connectivity by using its estimated frequency f_{v_p} (i.e., number of occurrences) via its neighbors in the random walks. We formulate f_{v_p} as:

$$f_{v_p} = \sum_{\tilde{v} \in N(v_p)} \frac{f_{\tilde{v}}}{d_{\tilde{v}} + 1}, \tag{1}$$

where $N(v_p)$ denotes 1-order neighbors of node v_p; $f_{\tilde{v}}$ denotes the number of occurrences of $\tilde{v}$ in the random walks generated from $\tilde{\mathcal{G}}$ using metapath2vec; $d_{\tilde{v}}$ is the degree of node $\tilde{v}$ in $\tilde{\mathcal{G}}$. Then, to further estimate the likelihood of v_p being predicted as malicious (denoted as c_{v_p}), we devise following approach for computation:

$$c_{v_p} = \frac{1}{f_{v_p}} \cdot \sum_{\tilde{v} \in N(v_p)} \frac{f_{\tilde{v}} c_{\tilde{v}}}{d_{\tilde{v}} + 1}, \tag{2}$$

where $c_{\tilde{v}}$ is the probability of $\tilde{v}$ being malicious measured by label propagation (e.g., LLGC algorithm [30]) on $\tilde{\mathcal{G}}$.

With the above estimated f_{v_p} and c_{v_p}, once v_p along with v_t being injected in $\tilde{\mathcal{G}}$ to generate the adversarial $\hat{\mathcal{G}}$, the information of each of its neighborhood nodes (i.e., $\hat{v} \in N(v_p)$) in $\hat{\mathcal{G}}$ will be further updated as follows:

$$f_{\hat{v}} = f_{\hat{v}} + \frac{f_{v_p}}{d_{v_p}}, \tag{3}$$

$$c_{\hat{v}} = \frac{1}{f_{\hat{v}}} \cdot \left(\frac{f_{v_p} c_{v_p}}{d_{v_p}} + f_{\hat{v}} c_{\hat{v}}\right). \tag{4}$$

Accordingly, we would like to maximize f_{v_t} while minimize c_{v_t}; to this end, the objective function can be defined as:

$$\mathcal{L}(f, c) = \left(\sum_{\hat{v} \in N(v_t)} \frac{f_{\hat{v}}}{d_{\hat{v}}}\right)^{-1} \cdot \sum_{\hat{v} \in N(v_t)} \frac{f_{\hat{v}} c_{\hat{v}}}{d_{\hat{v}}}. \tag{5}$$

To perform practical adversarial Android malware attacks, attackers are also constrained in their capabilities: (1) they can only inject $v_p \in \mathcal{V}_P$ in the devices (i.e., IMEIs) that are compromised; and (2) they have limited number of compromised devices. Therefore, we impose two budget constraints δ_1 and δ_2 on the attacks: (i) $|\mathcal{E}_{(v_p, v_{IMEI})}| \leq \delta_1$ which limits the number of poisoning nodes injected to compromised devices; (ii) $|\mathcal{M}_{compromised}| \leq \delta_2$ which limits the number of compromised devices. To this end, given $\tilde{\mathcal{G}}$, an adversarial attack is to solve the following optimization problem:

$$\begin{aligned} &\underset{N(v_p)}{\text{argmin}}\ \mathcal{L}(f, c) \\ &\text{s.t.} \quad |\mathcal{E}_{(v_p, v_{IMEI})}| \leq \delta_1, |\mathcal{M}_{compromised}| \leq \delta_2. \end{aligned} \tag{6}$$

To solve this optimization problem, we propose *HG-Attack* model based on the following strategies to perform realistic adversarial attacks on $\tilde{\mathcal{G}}$: (i) greedily selecting those APIs with low $c_{\tilde{v}}$ ($c_{\hat{v}}$ in later iterations) to generate poisoning apps $\mathcal{V}_P$; (ii) greedily choosing compromised devices (s.t. $|\mathcal{M}_{compromised}| \leq \delta_2$) with minimum $c_{\tilde{v}}$ ($c_{\hat{v}}$ in later iterations) to inject $\mathcal{V}_P$ until minimum c_{v_t} or δ_1 is reached. The detailed implementation of *HG-Attack* is given in Algorithm 1 (we empirically set $\xi_1 = 0.15, \xi_2 = 0.5, \xi_{exp} = 0.1, \delta_1 = 500$). In *HG-Attack*, given a target node (i.e., a new malicious app), to simulate the compromised devices its attacker owns, we retrieve the devices (i.e., IMEIs) that include this app's signature. As signature can indicate the ownership of an app (i.e., app's signature can only be used with the corresponding private key that is owned by a specific developer), if a device installs app(s) with the same signature as the target node, we assume the attacker is capable of accessing this device. In our experiments, we further remove the signatures whose private keys have been exposed in the Internet, since these signatures could be publicly accessible for all developers.

Algorithm 1: *HG-Attack* : Adversarial Attack on HG.

Input: $\tilde{\mathcal{G}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{E}})$; v_t: target node; δ_1: the number of poisoning nodes; $\mathcal{M}_{compromised}$: compromised devices; ξ_1, ξ_2, ξ_{exp}: user-pecified thresholds.
Output: Adversarial $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$.

Get the edges between v_t and the nodes in $\tilde{\mathcal{G}}$ as $\tilde{\mathcal{E}}_{v_t}$;
Initialize $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$ where $\hat{\mathcal{V}} = \tilde{\mathcal{V}} + v_t$, $\hat{\mathcal{E}} = \tilde{\mathcal{E}} + \tilde{\mathcal{E}}_{v_t}$;;
Calculate $c_{\hat{v}}$ for each node $\hat{v} \in \hat{\mathcal{V}}$ via LLGC;
$n = 1$;
while $n < \delta_1$ *or* $c_{v_t} > \xi_{exp}$ **do**
  Create poisoning node v_p and set its associated edges as $\hat{\mathcal{E}}_{v_p} = \emptyset$;
  for $v_i \in \tilde{\mathcal{V}}$ *and* $\phi(v_i) = A_{api}$ **do**
    if $(v_i, v_t) \in \tilde{\mathcal{E}}_{v_t}$ *and* $c_{v_i} < \xi_1$ **then**
      Update $\hat{\mathcal{E}}_{v_p}$ by $\hat{\mathcal{E}}_{v_p} = \hat{\mathcal{E}}_{v_p} + (v_i, v_p)$;
    end
    if $(v_i, v_t) \notin \tilde{\mathcal{E}}_{v_t}$ *and* $c_{v_i} < \xi_2$ **then**
      Update $\hat{\mathcal{E}}_{v_p}$ by $\hat{\mathcal{E}}_{v_p} = \hat{\mathcal{E}}_{v_p} + (v_i, v_p)$;
    end
  end
  Select $v_{IMEI} \in (\tilde{\mathcal{V}} \cap \mathcal{M}_{compromised})$ with minimal $c_{v_{IMEI}}$;
  Update $\hat{\mathcal{E}}_{v_p}$ by $\hat{\mathcal{E}}_{v_p} = \hat{\mathcal{E}}_{v_p} + (v_{IMEI}, v_p)$;
  Randomly select v_{IMEI} sujects to $(v_t, v_{IMEI}) \in \tilde{\mathcal{E}}_{v_t}$;
  Update $\hat{\mathcal{E}}_{v_p}$ by $\hat{\mathcal{E}}_{v_p} = \hat{\mathcal{E}}_{v_p} + (v_{IMEI}, v_p)$;
  Update $\hat{\mathcal{V}}$ by $\hat{\mathcal{V}} = \hat{\mathcal{V}} + v_p$;
  Update $\hat{\mathcal{E}}$ by $\hat{\mathcal{E}} = \hat{\mathcal{E}} + \hat{\mathcal{E}}_{v_p}$;
  Recalculate $c_{\hat{v}}$ for each node $\hat{v} \in \hat{\mathcal{V}}$;
  $n++$;
end
Return $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \hat{\mathcal{E}})$;

4 DEFENSE FROM ADVERSARIAL ATTACKS

To enhance robustness of a generic HG based classifier (i.e., *Orig-HGC*) against the adversarial attacks, in following sections, we first present our defense mechanism (named *HG-Defense*) and then propose our defense model (named *Rad-HGC*).

4.1 *HG-Defense*: Defense Mechanism

The key idea to enhance robustness of the HG based classifier against adversarial attacks is to uncover those poisoning nodes injected in the HG. Recall that, in *HG-Attack*, to protect target node v_t bypassing the detection of *Orig-HGC*, attackers inject poisoning nodes in the given $\tilde{\mathcal{G}}$ to maximize the likelihood of v_t neighboring with benign apps while to minimize the probability of it being predicted as malicious. That is, a poisoning node in the adversarial $\hat{\mathcal{G}}$ will have following properties: (i) in order to maximize the likelihood of v_t neighboring with benign apps, it might have high connectivity in the $\hat{\mathcal{G}}$ which can be implied by its frequency in the generated random walks; and (ii) as it connects both malicious app (i.e., target node) and benign apps, after performing label prorogation on $\hat{\mathcal{G}}$, the likelihood of it being considered as either malicious or benign could be low. Therefore, we formulate the probability of a node $\hat{v}_i$ in $\hat{\mathcal{G}}$ being an injected poisoning node as:

$$I_{\hat{v}_i} \propto \frac{f_{\hat{v}_i}}{|(c_{\hat{v}_i} - \beta)|}, \tag{7}$$

where β is user-specific parameter (in this paper we empirically set $\beta = 0.45$). To normalize $I_{\hat{v}_i}$, we further define it as:

$$I_{\hat{v}_i} = \lambda f_{\hat{v}_i}(1 - (c_{\hat{v}_i} - \beta)^2), \tag{8}$$

where λ is a rescaling parameter to keep $I_{\hat{v}_i}$ in the range of $[0, 1]$.

Based on the above definition, we can see that the maximum of $I_{\hat{v}_i}$ is attained when node $\hat{v}_i$ in $\hat{\mathcal{G}}$ is estimated with greatest chance as an injected poisoning node, and vice versa. Then we propose to uncover the possible poisoning nodes to reconstruct a $\dot{\mathcal{G}}^- = (\dot{\mathcal{V}}^-, \dot{\mathcal{E}}^-)$: given a pseudo random function $\mathcal{R}(.) \in (0, 1)$,

$$\begin{aligned} \dot{\mathcal{V}}^- &= \dot{\mathcal{V}} - \hat{v}_i, \quad \text{s.t.} \quad \mathcal{R}(.) \leq I_{\hat{v}_i}, \\ \dot{\mathcal{E}}^- &= \dot{\mathcal{E}} - (\hat{v}_i, \hat{v}_j), \quad \text{s.t.} \quad \hat{v}_j \in \dot{\mathcal{V}}, (\hat{v}_i, \hat{v}_j) \in \dot{\mathcal{E}}. \end{aligned} \tag{9}$$

The proposed method to uncover the possible injected poisoning nodes in $\hat{\mathcal{G}}$ is named *HG-Defense*, which aims to enhance robustness of *Orig-HGC* against adversarial attacks.

4.2 *Rad-HGC*: Defense Model

To enhance the robustness while not compromising detection accuracy, in this section, we further propose an attention-based framework to aggregate a set of reconstructed $\dot{\mathcal{G}}^-$s to build the classifier.

Although a classifier can be directly trained based on a reconstructed $\dot{\mathcal{G}}^-$ using the proposed *HG-Defense*, we expect it should cover as many features as possible in the original feature space to assure the integrity of original $\mathcal{G}$ (i.e., the HG before attack). To address this challenge, we first define the integrity of original feature space a classifier built on as:

$$T_{\dot{\mathcal{G}}_S^-} = \frac{|\cup_{k=1}^{K} \dot{\mathcal{V}}_k^-|}{|\dot{\mathcal{V}}|}, \quad \text{s.t.} \quad T_{\dot{\mathcal{G}}_S^-} \geq \eta, \tag{10}$$

where K is the number of reconstructed $\dot{\mathcal{G}}^-$s using *HG-Defense*, η can be empirically set as $1 - (\sum_{\dot{v}_i \in \dot{\mathcal{V}}} I_{\dot{v}_i}/|\dot{\mathcal{V}}|)$.

To this end, given K reconstructed $\dot{\mathcal{G}}^-$s, for each of them, we employ metapath2vec to learn its node (i.e., app) representations. To fuse the embeddings learned from each $\dot{\mathcal{G}}^-$, because of its effectiveness in various machine learning tasks [27], we propose an attention framework to learn attention weights of different $\dot{\mathcal{G}}^-$s to obtain final embedding for each app. Specifically, for a node v_{app} (i.e., a node with type of app), we define the attention weight of its embedding learned based on the k_{th} reconstructed $\dot{\mathcal{G}}^-$ using a softmax unit as following:

$$\alpha_{v_{app}}^{k_{th}} = \frac{\exp({\mathbf{z}_{k_{th}}}^T \cdot \mathbf{e}_{v_{app}}^C)}{\sum_{k=1}^{K} \exp({\mathbf{z}_k}^T \cdot \mathbf{e}_{v_{app}}^C)}, \tag{11}$$

where $\mathbf{z}_{k_{th}} \in \mathbb{R}^{|K|*d}$ is the attention vector learned based on the k_{th} reconstructed $\dot{\mathcal{G}}^-$ (i.e., d is the dimension of the embeddings) and $\mathbf{e}_{v_{app}}^C$ is the concatenation of node v_{app}'s embeddings w.r.t. all $\dot{\mathcal{G}}^-$s. A higher $\alpha_{v_{app}}^{k_{th}}$ means that the k_{th} reconstructed $\dot{\mathcal{G}}^-$ is more

informative for node v_{app}. After obtaining the attention weight of each $\dot{\mathcal{G}}^-$, the final embedding of node v_{app} is given by the following:

$$\mathbf{e}_{v_{app}} = \sum_{k=1}^{K} \alpha^k_{v_{app}} \cdot \mathbf{e}^k_{v_{app}}, \tag{12}$$

where $\mathbf{e}^k_{v_{app}}$ is node v_{app}'s embedding learned based on the k_{th} $\dot{\mathcal{G}}^-$. The final obtained embeddings of all nodes (i.e., apps) will then be fed to the downstream classifier to train the model.

We name the above proposed defense model as *Rad-HGC*, whose detailed implementation is given in Algorithm 2.

Algorithm 2: *Rad-HGC* against Adversarial Attacks.

Input: $\dot{\mathcal{G}} = (\dot{\mathcal{V}}, \dot{\mathcal{E}})$.
Output: Class label for unlabeled node $\dot{v} \in \dot{\mathcal{V}}$.

```
t = 0;
while 1 do
    t++;
    G_t^- = HG-Defense(G);
    Learn node embeddings for G_t^- via metapath2vec;
    Calculate T_{G_S^-} using Eq. (10);
    if T_{G_S^-} ≥ η then
        break;
    end
end
K = t;
for k = 1 to K do
    Calculate the attention weight of graph G_k^- for nodes
     (i.e., apps) using Eq. (11);
end
Get the final node (i.e., app) embeddings using Eq. (12) to
 train SVM;
Use the trained model to predict unlabeled node v̇ ∈ V̇;
```

HG-Defense($\dot{\mathcal{G}}$)

```
Initialize G^- = (V̇^-, Ė^-) where V̇^- = V̇, Ė^- = Ė;
Calculate f_v̇, c_v̇ for each node in V̇;
for v̇_i ∈ V̇ do
    Calculate I_{v̇_i} using Eq. (8);
    Get a pseudo random number from R(.);
    if R(.) ≤ I_{v̇_i} then
        Collect the edges associated with v̇_i as Ė^-_{v̇_i};
        Update V̇^- by V̇^- = V̇^- − v̇_i;
        Update Ė^- by Ė^- = Ė^- − Ė^-_{v̇_i};
    end
end
Return G^- = (V̇^-, Ė^-);
```

5 EXPERIMENTAL RESULTS AND ANALYSIS

In this section, we conduct four sets of experimental studies using large-scale and real sample collections from Tencent Security Lab to fully evaluate the performance of $\alpha Cyber$: (1) we first evaluate the performance of our proposed adversarial attack model *HG-Attack*; (2) and then we evaluate the effectiveness of our proposed defense model *Rad-HGC* against adversarial attacks on HG; (3) later, we perform parameter sensitivity, scalability and stability evaluations of *Rad-HGC*; (4) finally, we compare the performance of *Rad-HGC* with other popular Android malware detection systems.

5.1 Experimental Setup

Through Tencent Mobile Manager (i.e., one of the most popular mobile security products in China), its users can scan and upload the Andorid apps for detection (i.e., these data are fully de-identified). We obtain the large-scale and real data collections from Tencent Security Lab: (1) The first data set is the historically accumulative data collected from Aug. 30, 2018 till Nov. 30, 2018, which contains 1,389,408 apps uploaded by 70,184 users (i.e., IMEIs) (i.e., 217,107 of them are detected as malicious, 547,366 are benign, and 624,935 are unknown). After feature extraction (note that, for content-based features, 75,419 benign apps and 72,868 malware are uploaded with runtime API sequences) and based on the designed network schema, the constructed HG has 1,865,034 nodes (i.e., 1,389,408 nodes with type of app, 331 nodes with type of API, 70,184 nodes with type of IMEI, 228,976 nodes with type of signature, and 176,135 with type of affiliation) and 20,576,125 edges including *R1-R6* relations. We denote this constructed HG as ***HG-1***. (2) The second data set is the newly generated data on Dec. 1, 2018, including 13,129 new apps uploaded by 2,817 users. Based on *HG-1*, we incorporate this newly collected data to generate an updated HG (denoted as ***HG-2***) which contains 1,882,978 nodes (i.e., 1,402,537 nodes with type of app, 331 nodes with type of API, 70,571 nodes with type of IMEI, 231,462 nodes with type of signature, and 178,077 with type of affiliation) and 21,721,829 edges. For those newly uploaded 13,129 apps, to obtain the ground truth, we ask anti-malware experts of Tencent Security Lab for further analysis - i.e., 9,456 of them are labeled as benign and 3,673 are malicious. To quantitatively assess the Android malware detection performance of different methods, we use the measures shown in Table 1 for evaluations.

The experimental studies are conducted under the environment of ubuntu 16.04 operating system, plus two Intel Xeon E5-2620 v4 CPU, 4-way SLI GeForce GTX 1080 Ti Graphics Cards and 80 GB of RAM. Other parameters include the dimension of node embedding $d = 128$, neighborhood size $w = 5$, iteration time $epoch = 5$ for skip-gram model. We use sklearn.svm with RBF kernel in our experiments as the downstream classifier and the penalty is empirically set to be 50 while other parameters are set by default.

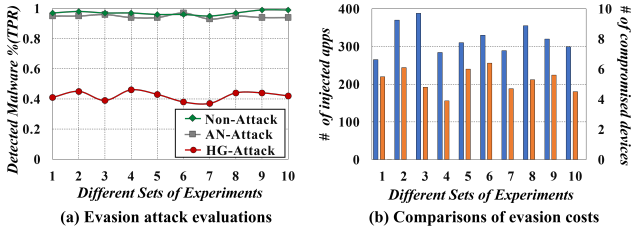
Table 1: Performance indices of Android malware detection

Indices	Description
TP	# of apps correctly classified as malicious
TN	# of apps correctly classified as benign
FP	# of apps mistakenly classified as malicious
FN	# of apps mistakenly classified as benign
$Precision$	$TP/(TP + FP)$
$Recall$	$TP/(TP + FN)$
ACC	$(TP + TN)/(TP + TN + FP + FN)$
$F1$	$2 * Precision * Recall/(Precision + Recall)$

5.2 Evaluation of Attack Model *HG-Attack*

Since this is the first work considering adversarial attacks on HGs, there are no known baselines. In this set of experiments, similar to the way how the works [2, 6, 28, 29] evaluate their attacks, we compare our proposed *HG-Attack* with the ***baseline*** of *anonymous attack (AN-Attack)*: given the same experimental settings as *HG-Attack* (e.g., same budgets of δ_1 and δ_2), attackers are assumed to be capable of randomly injecting the randomly generated poisoning nodes to the compromised devices they possess.

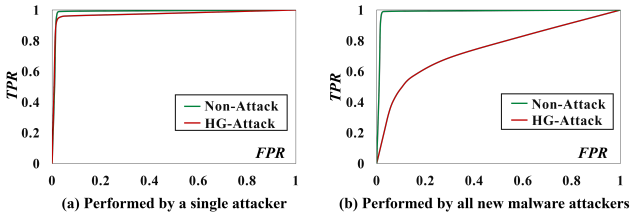
Based on the data collections described in Section 5.1, we first evaluate the ***effectiveness of evasion attacks*** for the proposed *HG-Attack*. We conduct 10 sets of experiments: each set includes 100 new malicious apps randomly selected from the data collected on Dec. 1, 2018; the goal of the attacks is to inject poisoning nodes in the *HG-1* to make the *Orig-HGC* maximumly misclassify these new malicious apps. Figure 6.(a) shows different performances of *HG-Attack* and *AN-Attack* in each set of experiments, from which we can see that: (1) *AN-Attack* almost fails in performing the evasion attacks for each set of experiments; while (2) *HG-Attack* shows powerful capabilities to evade the detection of *Orig-HGC* (i.e., the average TPR of *Orig-HGC* is brought down from 97.1% to 41.9%). The success of *HG-Attack* lies in its novel yet practical adversarial attack mechanism. We then evaluate the evasion cost (i.e., the numbers of compromised devices and injected nodes) of *HG-Attack* in each set of experiments. Figure 6.(b) shows that *HG-Attack* is able to compromise *Orig-HGC* with an efficient cost (i.e., to evade a new malware, the median number of compromised devices *HG-Attack* leverages is 5, while the median number of nodes injected is 298).



(a) Evasion attack evaluations (b) Comparisons of evasion costs

Figure 6: Evaluation of evasion attacks.

We then evaluate the ***effectiveness of poisoning attacks*** for the proposed *HG-Attack*. We first randomly select a new malicious app collected on Dec. 1, 2018; as signature can indicate the ownership of an app, we then retrieve the devices that include this app's signature (i.e., "4CD***194") to simulate the compromised devices this malware attacker owns (i.e., $\delta_2 = 5$); later, we use *HG-Attack* to inject all new malicious apps that are with this signature along with 38 poisoning apps generated by *HG-Attack* based on *HG-1*. Figure 7.(a) shows that the F1 of *Orig-HGC* drops from 0.9599 to 0.9412 (i.e., those three new malware all successfully evade the detection). Based on the same setting, we then further test the worst case of *Orig-HGC* being attacked by *HG-Attack* (i.e., using all the new malware collected on Dec. 1, 2018 to perform the attacks). The results are shown in Figure 7.(b), from which we can see that the performance of *Orig-HGC* significantly degrades under such setting (i.e., F1 drops from 0.9599 to 0.5168). The results demonstrate the effectiveness of poisoning attacks performed by *HG-Attack*.



(a) Performed by a single attacker (b) Performed by all new malware attackers

Figure 7: Evaluation of poisoning attacks.

5.3 Evaluation of Defense Model *Rad-HGC*

Since no other defenses against adversarial attacks on HG based models exist, we evaluate our proposed defense *Rad-HGC* against the adversarial attacks (i.e., *HG-Attack* and *AN-Attack* on *HG-1* using all new malware collected on Dec. 1, 2018) and without attacks by comparisons with two ***baselines*** (i.e., *Orig-HGC* and *AU-HGC*): (1) *Orig-HGC* is the original HG based classifier described in Section 2.3; (2) different from *Rad-HGC*, *AU-HGC* performs arbitrary guess of injected nodes in the HG while other settings are the same as *Rad-HGC*. Table 2 and Figure 8 show the experimental results.

Table 2: Evaluation of defense model *Rad-HGC*

HG-Attack	F1	ACC	TP	FP	TN	FN
Orig-HGC	0.5168	0.7855	1506	649	8807	2167
AU-HGC	0.8077	0.8987	2794	451	9005	879
Rad-HGC	0.9350	0.9631	3488	300	9156	185
AN-Attack	F1	ACC	TP	FP	TN	FN
Orig-HGC	0.9308	0.9600	3537	390	9066	136
AU-HGC	0.9317	0.9604	3545	391	9065	128
Rad-HGC	0.9372	0.9641	3516	314	9142	157
Non-Attack	F1	ACC	TP	FP	TN	FN
Orig-HGC	0.9599	0.9768	3636	267	9189	37
AU-HGC	0.9535	0.9732	3608	287	9169	65
Rad-HGC	0.9565	0.9749	3613	269	9187	60

From the results, we can see that our proposed defense *Rad-HGC* outperforms other defense models (*Orig-HGC* and *AU-HGC*) against the adversarial attacks (*HG-Attack* and *AN-Attack*). To further illustrate, (1) Based on the most powerful *HG-Attack*, *Rad-HGC* stays resilient (i.e., almost retains the F1 before attack), while *Orig-HGC* (i.e., F1 drops from 0.9599 to 0.5168) and *AU-HGC* (i.e., F1 drops from 0.9535 to 0.8077) are both vulnerable to such attack; the success of *Rad-HGC* lies in its well-designed defense mechanism to uncover the possible injected poisoning nodes. (2) For *AN-Attack*, as it almost fails in attacking HG based model, there are not great differences for different defense models. In the case of without attacks (*Non-Attack*), compared with *Orig-HGC* (i.e., F1 of 0.9599), due to its resilient yet elegant defense framework by aggregating a set of reconstructed HGs with attention strategy, *Rad-HGC* retains the detection accuracy (i.e., F1 of 0.9565).

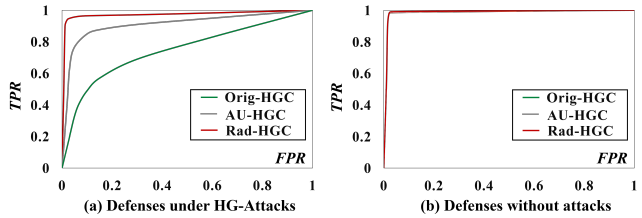


Figure 8: Comparisons of different defenses.

5.4 Evaluation of Parameter Sensitivity, Scalability and Stability

In this set of experiments, we first examine how different choices of parameters (number of walks per node r and walk length l) will affect the performance of *Rad-HGC* in Android malware detection. As shown in Figure 9.(a) and (b), we can see that the performance tends to be stable when we varied the number of walks per node from $r = 5$ to $r = 25$ or walk length $l = 30$ to $l = 60$ in the model. Thus, *Rad-HGC* is not strictly sensitive to these parameters and is able to reach high performance under a cost-effective parameter choice. We also examine the running time with different sizes of training data generated in terms of the number of devices (i.e., IMEIs). From Figure 9.(c), we can see that the running time is quadratic to the number of training samples. When dealing with more data, approximation or parallel algorithms can be developed. We then run the experiments using new collected apps from Dec. 2-8, 2018 to assess the detection performance: as shown in Figure 9.(d) and (e), *Rad-HGC* is stable over a long time span in detecting new Android malware, which achieves an impressive average 0.9812 true positive rate (*TPR*) at 0.0289 false positive rate (*FPR*). We can conclude that *Rad-HGC* is feasible in practical use for Android malware detection.

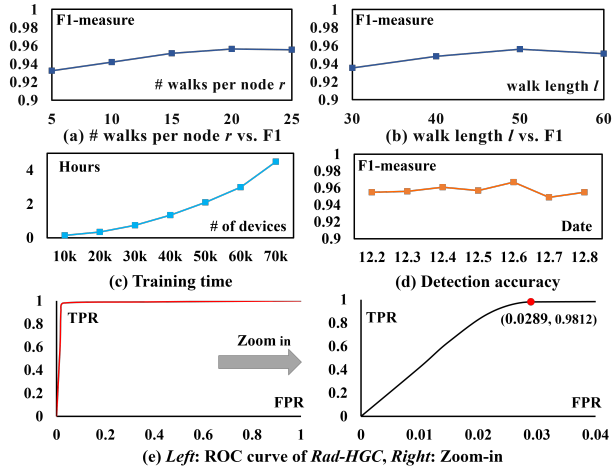


Figure 9: Parameter sensitivity, scalability and stability.

5.5 Comparisons with Other Detection Systems

In this section, based on the labeled 3,673 Android malware described in Section 5.1, we evaluate the performance of our developed detection model *Rad-HGC* in comparisons with some popular commercial mobile security products (i.e., Lookout: 10.25-03571d4, Norton: 4.4.0.4302) and HG based Android malware detection systems (i.e., HinDroid [14] and AiDroid [25]). Table 3 shows the detection results of different Android malware families. From Table 3, we can see that *Rad-HGC* performs better than others in the overall detection of recent collected Android malware; in particular, it outperform others in the detection of C&C malware families such as "BlackBaby" and "SDKSmartPush" (i.e., C&C malware is a new kind of malware that attackers can leverage C&C servers to create powerful networks of compromised devices capable of downloading and executing apps on-demand).

Table 3: Comparisons with other detection systems

Family	#	Norton	Lookout	HinDroid	AiDroid	***Rad-HGC***
FakeBank	167	166	164	165	166	166
CryptoMiner	155	154	151	153	154	154
AppCracked	357	354	341	354	353	355
MalPlayer	256	249	242	249	250	251
GameTrojan	212	208	205	209	210	209
BlackBaby	129	108	105	109	79	128
SDKSmartPush	321	298	275	299	259	319
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Others	985	924	908	943	913	961
Total	3,673	3,497	3,442	3,516	3,426	3,613
DetectionRate	–	95.21%	93.71%	95.72%	93.27%	**98.37%**

6 SYSTEM DEPLOYMENT AND OPERATION

Our defensed model *Rad-HGC* in α*Cyber* has already been incorporated into Tencent's Mobile Security product that provides anti-malware service for over 5,000,000 users. *Rad-HGC* has been used to predict the daily sample collection from the Tencent Security Lab which includes around 15,000 newly uploaded apps per day. Note that Android malware techniques are constantly evolving and new malicious apps are produced on a daily basis. To detect the increasingly sophisticated Android malware, our developed system has been upgraded to train the model by incorporating newly detected malicious and benign apps everyday. Our system *Rad-HGC* has been deployed and tested based on the real daily sample collections for over 150 days.

For the development of the system, it has been spent over $325K, $145K of which is on the hardware equipment. Due to the high performance in the detection of evolving Android malware, the developed system *Rad-HGC* has greatly saved human labors and cost: over 35 anti-malware analysts and developers at Tencent Security Lab are utilizing the system on the daily basis. Practically, an anti-malware analyst can manually analyze 20-40 Android apps per day. Using the developed system *Rad-HGC*, the analysis of daily collected unknown apps (i.e., about 15,000 apps/day) based on millions of labeled data can be performed within hours with multiple servers. This has benefited over 5,000,000 smart phone users of Tencent's Mobile Security product.

7 RELATED WORK

In recent years, there have been ample studies on developing Android malware detection systems using data mining and machine learning techniques [3, 5, 8, 11–15, 18, 22, 24, 25]. However, most of the existing systems merely utilize content-based features for the detection. To combat the increasingly sophisticated Android malware, in the preliminary work, besides content-based information, HinDroid [14] and AiDroid [25] were proposed which considered higher-level semantic relations among apps and other types of entities (e.g., APIs, IMEIs, signatures, etc.) and introduced structured HGs to model such complex relations for malware detection, which have been successfully deployed in anti-malware industry. As attackers and defenders always engage in a never-ending arms race, the success of deployments may also incentivize attackers to defeat the HG based models to bypass the detection.

The robustness of learning-based models has been widely studied including traditional shallow learning models [17] and emerging deep learning models [4, 10, 16, 21]. However, majority of these works are based on the assumption that data samples are independent. Adversarial attacks and/or defenses for graph (i.e., non-i.i.d. samples) learning tasks have been sparse with a few exceptions of [2, 6, 28, 29]. Unfortunately, these works only considered adversarial settings (i.e., particularly attacks) on homogeneous graph data. Due to the heterogeneous property, it is difficult to directly apply these existing adversarial settings on heterogeneous graph. By far, there has no work on adversarial attack/defense on heterogeneous graph data. Our work in this paper is the first attempt to bridge this gap with the application in Android malware detection.

8 CONCLUSION

To combat the increasingly sophisticated Android malware, systems of HinDroid and AiDroid were developed, which have demonstrated the success of HG based classifiers in Android malware detection; however, their success may also incentivize attackers to defeat HG based models to bypass the detection. To date, there has no work on adversarial attack and/or defense on HG data. This work is the first study on the robustness of HG based classifier in Android malware detection. To explore the security of a generic HG based classifier, we first propose a novel yet practical adversarial attack model (i.e., *HG-Attack*) on HG data by considering Android malware attackers' current capabilities and knowledge; then, to effectively combat the adversarial attacks, we further propose a resilient yet elegant defense model (i.e., *Rad-HGC*) to enhance robustness of HG based classifier in Android malware detection. Promising experimental results based on the large-scale and real sample collections from Tencent Security Lab demonstrate the effectiveness of our developed system *αCyber*, which integrates our proposed defense model *Rad-HGC* that is resilient against practical adversarial malware attacks on HG performed by *HG-Attack*.

ACKNOWLEDGMENTS

S. Hou, Y. Fan, Y. Zhang, Y. Ye's work is partially supported by the NSF under grants CNS-1618629, CNS-1814825, CNS-1845138, OAC-1839909, and III-1908215, the NIJ 2018-75-CX-0032, the WV HEPC Grant (HEPC.dsr.18.5), and the WVU RSA grant (R-844).

REFERENCES

[1] Battista Biggio, Giorgio Fumera, and Fabio Roli. 2014. Security evaluation of pattern classifiers under attack. *TKDE* 26, 4 (2014), 984–996.
[2] Aleksandar Bojcheski and Stephan Günnemann. 2019. Adversarial attacks on node embeddings. In *ICLR*.
[3] Haipeng Cai, Na Meng, Barbara Ryder, and Daphne Yao. 2019. Droidcat: Effective android malware detection and categorization via app-level profiling. *IEEE TIFS* 14, 6 (2019), 1455–1470.
[4] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE S&P*.
[5] Lingwei Chen, Shifu Hou, Yanfang Ye, and Shouhuai Xu. 2018. Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks. In *ASONAM*. IEEE, 782–789.
[6] Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. 2018. Adversarial Attack on Graph Structured Data. In *PMLR*.
[7] Yuxiao Dong, Nitesh V Chawla, and Ananthram Swami. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*. ACM, 135–144.
[8] Yujie Fan, Shifu Hou, Yiming Zhang, Yanfang Ye, and Melih Abdulhayoglu. 2018. Gotcha-sly malware! Scorpion: a metagraph2vec based malware detection system. In *KDD*.
[9] Tao-Yang Fu, Wang-Chien Lee, and Zhen Lei. 2017. HIN2Vec: Explore Meta-paths in Heterogeneous Information Networks for Representation Learning. In *CIKM*.
[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *NIPS*. 2672–2680.
[11] William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. 2016. DL4MD: A deep learning framework for intelligent malware detection. In *DMIN*. 61–67.
[12] Shifu Hou, Aaron Saas, Lifei Chen, and Yanfang Ye. 2016. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In *WIW*. IEEE, 104–111.
[13] Shifu Hou, Aaron Saas, Yanfang Ye, and Lifei Chen. 2016. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In *WAIM*. Springer, 54–66.
[14] Shifu Hou, Yanfang Ye, Yangqiu Song, and Melih Abdulhayoglu. 2017. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *KDD*. ACM, 1507–1515.
[15] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2019. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE TIFS* 14, 3 (2019), 773–788.
[16] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. Towards deep learning models resistant to adversarial attacks. In *ICLR*.
[17] Shike Mei and Xiaojin Zhu. 2015. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. In *AAAI*. 2871–2877.
[18] Andrea Saracino, Daniele Sgandurra, Gianluca Dini, and Fabio Martinelli. 2018. Madam: Effective and efficient behavior-based android malware detection and prevention. *TDSC* (2018).
[19] Statcounter. 2019. Mobile Operating System Market Share Worldwide. In *http://gs.statcounter.com/os-market-share /mobile/worldwide*.
[20] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *PVLDB* (2011).
[21] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. 2014. Intriguing properties of neural networks. In *ICLR*.
[22] K. Tam, S. Khan, A. Fattori, and L. Cavallaro. 2015. CopperDroid: Automatic Reconstruction of Android Malware Behaviors.. In *NDSS*.
[23] TencentSecurity. 2018. Mobile phone security report for the first half of 2018. In *https://m.qq.com/security_lab/news _detail_471.html*.
[24] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. 2014. DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications. In *ESORICS*.
[25] Yanfang Ye, Shifu Hou, Lingwei Chen, Jingwei Lei, Wenqiang Wan, Jiabin Wang, Qi Xiong, and Fudong Shao. 2019. Out-of-sample Node Representation Learning for Heterogeneous Graph in Real-time Android Malware Detection. In *IJCAI*. 4150–4156.
[26] Yanfang Ye, Tao Li, Donald Adjeroh, and S Sitharama Iyengar. 2017. A Survey on Malware Detection Using Data Mining Techniques. *ACM Computing Surveys (CSUR)* 50, 3 (2017).
[27] Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. 2016. Image captioning with semantic attention. In *CVPR*.
[28] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial attacks on neural networks for graph data. In *KDD*.
[29] Daniel Zügner and Stephan Günnemann. 2019. Adversarial Attacks on Graph Neural Networks via Meta Learning. In *ICLR*.
[30] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. 2004. Learning with local and global consistency. In *NIPS*.