AiDroid: When Heterogeneous Information Network Marries Deep Neural Network for Real-time Android Malware Detection

Yanfang Ye[*1], **Shifu Hou**[1], **Lingwei Chen**[1], **Jingwei Lei**[2], **Wenqiang Wan**[2],
Jiabin Wang[2], **Qi Xiong**[2], **Fudong Shao**[2]

[1]Department of Computer Science and Electrical Engineering
West Virginia University, Morgantown, WV, USA
yanfang.ye@mail.wvu.edu, {shhou, lgchen}@mix.wvu.edu
[2]Tencent Security Lab, Tencent, Guangdong, China
{lingfonglei, johnnywan, luciferwang, keonxiong, joeyshao}@tencent.com

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. In this paper, we first extract the runtime Application Programming Interface (API) call sequences from Android apps, and then analyze higher-level semantic relations within the ecosystem to comprehensively characterize the apps. To model different types of entities (i.e., *app*, *API*, *IMEI*, *signature*, *affiliation*) and the rich semantic relations among them, we then construct a structural heterogeneous information network (HIN) and present meta-path based approach to depict the relatedness over apps. To efficiently classify nodes (e.g., apps) in the constructed HIN, we propose the *HinLearning* method to first obtain in-sample node embeddings and then learn representations of out-of-sample nodes without rerunning/adjusting HIN embeddings at the first attempt. Afterwards, we design a deep neural network (DNN) classifier taking the learned HIN representations as inputs for Android malware detection. A comprehensive experimental study on the large-scale real sample collections from Tencent Security Lab is performed to compare various baselines. Promising experimental results demonstrate that our developed system *AiDroid* which integrates our proposed method outperforms others in real-time Android malware detection. *AiDroid* has already been incorporated into Tencent Mobile Security product that serves millions of users worldwide.

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, online banking, and entertainment. Android, as an open source and customizable operating system (OS) for smart phones, is currently dominating the smart phone market by 77.32% (Statcounter 2018). 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 the smart phone users (e.g., stealing user credentials, pushing unwanted apps or advertisements). Because of lacking trustworthiness review methods, developers can easily upload their Android apps including repackaged apps and malware to the official marketplace (i.e., Google Play). The

presence of other third-party Android markets (e.g., Opera Mobile Store, Wandoujia) makes this problem worse. Driven by the considerable economic profits, there has been explosive growth of Android malware which posed serious threats to the smart phone users - i.e., it's reported that there have been $4,687,008$ newly generated Android malware that infected more than 61 million smart phones in the first half of 2018 (TencentSecurity 2018). To evade the detection of mobile security products (e.g., Norton, Lookout and Tencent Mobile Security), Android malware has turned to be increasingly sophisticated. For example, as shown in Figure 1, the "TigerEyeing" trojan is a new kind of Command and Control (C&C) malware that pretends to be legitimate apps (e.g., mobile games, system tools) and only executes to perform the profitable tasks on-demand. The explosive growth and increasing sophistication of Android malware call for new defensive techniques that are capable of protecting smart phone users against novel threats.
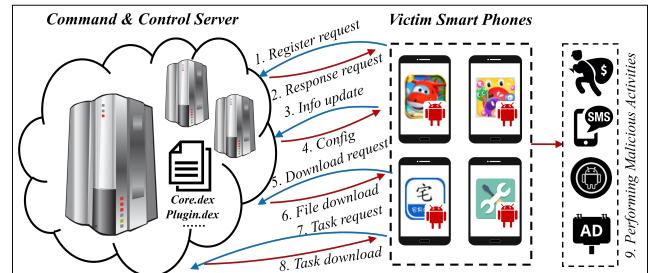

Figure 1: Increasingly sophisticated Android malware.

To combat the evolving Android malware attacks, by collaboration with smart phone vendors, in this paper, we first extract the Application Programming Interface (API) call sequences from runtime executions of Android apps in users' smart phones to capture their behaviors. To comprehensively characterize Android apps, we further analyze higher-level semantic relationships within the ecosystem, 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, whether they are signed by the same developer or produced by the same company (i.e., affiliation), etc. To model such complex relationships, we present a structured heterogeneous information network (HIN) (Sun and Han 2012) and use meta-path based approach (Sun et al. 2011) to build up relatednesses over apps. To efficiently classify nodes (e.g., apps) in the constructed HIN, HIN embed-
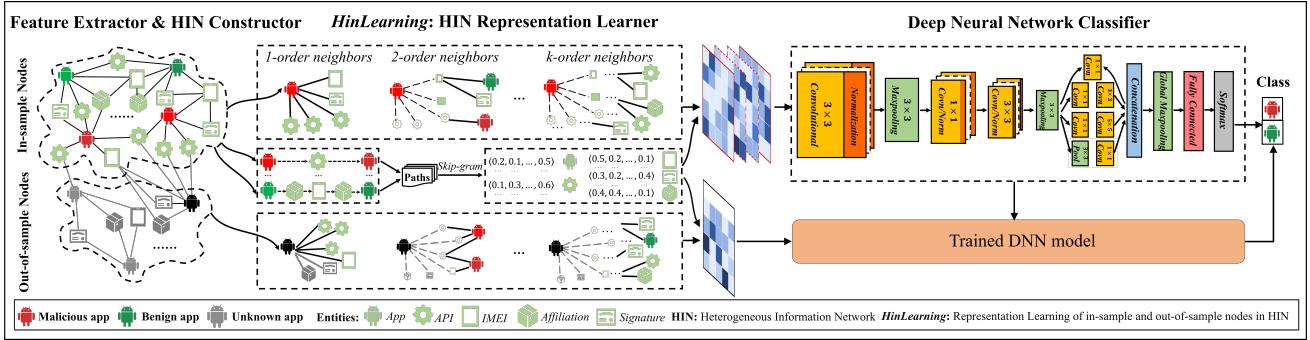
Figure 2: Overview of the developed system *AiDroid* for real-time Android malware detection.

ding methods (Fu, Lee, and Lei 2017; Dong, Chawla, and Swami 2017; Fan et al. 2018) have been proposed. However, most of these existing methods are primarily designed for static networks, where all nodes are known before learning. As our application requires real-time prediction of new coming nodes (i.e., unknown apps) to detect Android malware, it is infeasible to rerun HIN embeddings whenever new nodes arrive, especially considering the fact that rerunning HIN embeddings also results in the need of retraining the downstream classifier. How to efficiently learn the representations of out-of-sample nodes in HIN, i.e. nodes that arrive after the HIN embedding process, remains largely unanswered. To solve this problem, we propose the *HinLearning* method to first obtain in-sample node embeddings and then learn the representations of out-of-sample nodes in HIN, which is capable to preserve the heterogeneous property of HIN and also allows fast and scalable learning. Afterwards, we design a deep neural network (DNN) classifier leveraging the advantages of convolutional neural networks (CNNs) and Inception for Android malware detection. We develop a system *AiDroid* for real-time Android malware detection, which has the following major traits:

- Besides runtime behaviors extracted from Android apps, we further analyze the complex relationships within the ecosystem (i.e., *app-API*, *app-IMEI*, *app-signature*, *app-affiliation*, *IMEI-signature*, *IMEI-affiliation* relations) to characterize Android apps. We then present HIN to represent the Android apps and exploit meta-path based approach to depict the relatednesses over apps. This provides a comprehensive solution that is capable to be more resilient against Android malware's evasion tactics.

- We are the first to propose the method (denoted as *HinLearning*) to efficiently learn the representations for out-of-sample nodes in HIN using in-sample node embeddings while without rerunning/adjusting them, which makes the downstream classifier feasible for classifying new arriving nodes (e.g., apps) without retraining. Though it's proposed for real-time Android malware detection, the *HinLearning* method is a general framework which is able to learn desirable node representations in HIN (i.e., including in-sample and out-of-sample nodes) and thus can be further applied to various speed-sensitive dynamic network mining tasks (e.g., node classification, clustering).

- We provide a comprehensive experimental study based on the large-scale real sample collections from Tencent Se-

curity Lab, which demonstrates the effectiveness and efficiency of our developed system *AiDroid*. It has already been incorporated into Tencent Mobile Security product that server millions of users worldwide.

2. System Overview

The overview of our developed system *AiDroid* for real-time Android malware detection is shown in Figure 2, mainly consisting of the following components:

- **Feature Extractor.** Through the installed Tencent Mobile Security product, smart phone users can upload the extracted API call sequences of Android apps as well as the meta-data to *AiDroid*. It will then analyze various relationships among different types of entities (i.e., *app*, *API*, *IMEI*, *signature*, *affiliation*) to depict the Android apps. (See Section 3.1. for details.)

- **HIN Constructor.** In this module, based on the features extracted from the previous component, a structural HIN is first presented to model the relationships among different types of entities; and then different meta-paths are built from the HIN to capture the relatedness over apps from different views (i.e., with different semantic meanings). (See Section 3.2. for details.)

- **HIN Representation Learner.** In *HinLearning*, based on the meta-path schemes built from the previous module, a heterogeneous in-sample node embedding method HINE is first proposed to learn the low-dimensional representations for in-sample nodes in HIN; then, Hin2Img method is devised to learn representations of out-of-sample nodes using learned in-sample node embeddings, which is also capable to enrich representations of in-sample nodes. (See Section 3.3. for details.)

- **DNN Classifier.** After representation learning using *HinLearning*, the learned representations of in-sample nodes with type of app will be fed to the designed DNN (see Section 3.4. for details) to train the classification model, based on which new arriving nodes (i.e., out-of-sample nodes with type of app) can be predicted as either benign or malicious.

3. Proposed Method

In this section, we introduce the detailed approaches of how we represent the Android apps, and how to solve Android malware detection (i.e., classification) problem based on the representations.

3.1. Feature Extraction

Dynamic Behavior 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, a sequence of API calls (*StartActivity*, *checkConnect*, *getPhoneInfo*, *receiveMsg*, *sendMsg*, *finishActivity*) extracted from the previous mentioned "TigerEyeing" trojan represents its typical behaviors of connecting to the C&C server in order to fetch the configuration information; while another sequence of its extracted API calls (*startActivity*, *checkConnect*, *sendSMS*, *finishActivity*) denotes its intention of sending SMS messages without user's concern. Note that, to simplify the further computation, each API call in the extracted sequences will be mapped to an integer ID.

Relation-based Feature Extraction. Besides the API call sequences extracted from an Android app that can be used to represent its behaviors, to detect the increasingly sophisticated Android malware, we further consider the following kinds of relationships.

- **R1**: To describe the relation between an app and an API call it invokes during runtime execution, we build the ***app-invoke-API*** matrix $\mathbf{I}$ where each element $i_{i,j} \in \{0,1\}$ denotes if app i invokes API call j.

- **R2**: To describe the relation that an app exists (i.e., is installed) in a smart phone (i.e., IMEI), we generate the ***app-exist-IMEI*** matrix $\mathbf{E}$ where each element $e_{i,j} \in \{0,1\}$ means if app i exists in phone j.

- **R3**: Every app run on the Android platform must be signed by the developer. To depict such relationship, we build the ***app-certify-signature*** matrix $\mathbf{C}$ whose element $c_{i,j} \in \{0,1\}$ denotes if app i is certified by signature j.

- **R4**: Package name (a.k.a. Google Play ID) is the unique identifier for an Android app. Companies conventionally use their reversed domain name to begin their package names (e.g., "com.tencent.mobileqq"). We extract the domain name from the 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*** matrix $\mathbf{A}$ where each element $a_{i,j} \in \{0,1\}$ indicates whether app i is associated with affiliation j.

- **R5**: To represent the relation that a smart phone has a set of apps signed by a particular developer, we create the ***IMEI-have-signature*** matrix $\mathbf{H}$ where each element $h_{i,j} \in \{0,1\}$ denotes if smart phone i has signature j.

- **R6**: To denote the relation that a smart phone installs a set of apps associated with a specific affiliation, we generate the ***IMEI-possess-affiliation*** matrix $\mathbf{P}$ where each element $p_{i,j} \in \{0,1\}$ denotes whether smart phone i possesses affiliation j.

3.2. HIN Construction

In order to depict apps, APIs, IMEIs, signatures and affiliations as well as the rich relationships among them (i.e., *R1-R6*), it is important to model them in a proper way so that different kinds of relations can be better and easier handled. We introduce how to use HIN, which is capable to be composed of different types of entities and relations, to represent the apps by using the features extracted above. We first present the concepts related to HIN as follows.

Definition 1 *A **heterogeneous information network (HIN)** (Sun and Han 2012) 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** (Sun and Han 2012) for a HIN $\mathcal{G}$, 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 $\mathcal{R}$.*

HIN not only provides the network structure of the data associations, but also provides a high-level abstraction of the categorical association. For our case, we have five entity types (i.e., app, API, IMEI, package, signature) and six types of relations among them (i.e., *R1-R6*). Based on the definitions above, the network schema for HIN in our application is shown in Figure 3, which enables the apps to be represented in a comprehensive way that utilizes their semantic and structural information.
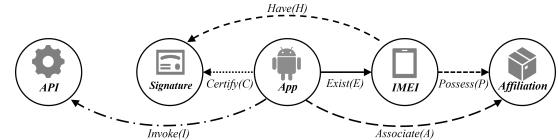

Figure 3: Network schema for HIN in our application.

To formulate the relatedness among entities in HIN, the concept of meta-path has been proposed (Sun et al. 2011): a ***meta-path*** $\mathcal{P}$ is a path defined on the graph of network schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and is denoted in the form of $A_1 \xrightarrow{R_1} A_2 \xrightarrow{R_2} \dots \xrightarrow{R_L} A_{L+1}$, which defines a composite relation $R = R_1 \cdot R_2 \cdot \dots \cdot R_L$ between types A_1 and A_{L+1}, where $\cdot$ denotes relation composition operator, and L is length of $\mathcal{P}$. In our application, based on the HIN schema shown in Figure 3, incorporated the domain knowledge from anti-malware experts, we design six meaningful meta-paths to characterize the relatedness over apps at different views (i.e., *PID1-PID6* shown in Figure 4).
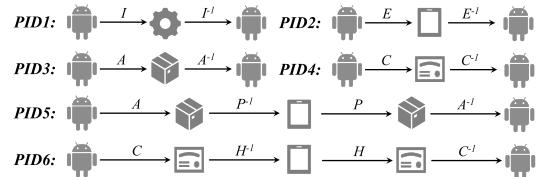

Figure 4: Meta-paths built for Android malware detection (the symbols are the abbreviations shown in Figure 3).

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*"); while *PID5* describes that two apps are connected if their associated affiliations are possessed by (i.e., co-occur in) the same phone. To measure the relatedness over HIN entities (e.g., apps), traditional representation learning for HIN (Sun et al. 2011) mainly focuses on factorizing the

matrix (e.g., adjacency matrix) of a HIN to generate latent-dimension features for the nodes in this HIN. However, the computational cost of decomposing a large-scale matrix is usually very expensive, and also suffers from its statistical performance drawback (Grover and Leskovec 2016). Since Android malware detection is a speed sensitive application and requires cost-effective solutions, scalable representation learning method for HIN, especially for out-of-sample nodes, is in need.

3.3. *HinLearning*: Representation Learning of In-Sample and Out-of-Sample Nodes in HIN

To address the above challenge, we first formalize the problem of HIN representation learning as follow.

Definition 2 *HIN Representation Learning (Fu, Lee, and Lei 2017; Dong, Chawla, and Swami 2017). Given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the representation learning task is to learn a function $f : \mathcal{V} \rightarrow \mathbb{R}^d$ 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 solve the problem of HIN representation learning, due to the heterogeneous property of HIN (i.e., network consisting of multi-typed entities and relations), it is difficult to directly apply the conventional homogeneous network embedding techniques (e.g., DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015), node2vec (Grover and Leskovec 2016)) to learn the latent representations for HIN. To address this issue, HIN embedding methods (Fu, Lee, and Lei 2017; Dong, Chawla, and Swami 2017; Fan et al. 2018) have been proposed, which are capable to preserve the semantic and structural correlations between different types of nodes . However, most of these existing methods are primarily designed for static networks, where all nodes are known before learning. In our application (i.e., real-time Android malware detection), it is infeasible to re-run HIN embeddings whenever new nodes arrive, especially considering the fact that rerunning HIN embeddings also results in the need of retraining the downstream classifier. How to efficiently learn the representations of out-of-sample nodes in HIN, i.e. nodes that arrive after the HIN embedding process, remains largely unanswered. To solve this problem, we first propose heterogeneous in-sample node embedding (HINE) model to learn in-sample node embeddings that is able to preserve the heterogeneous property of HIN; then, we devise Hin2Img to learn out-of-sample node representations and also enrich in-sample node representations using previous learned in-sample node embeddings without rerunning/adjusting HIN embeddings.

HINE: Heterogeneous In-sample Node Embedding. We first propose a random walk strategy guided by different meta-paths to map the word-context concept in a text corpus into a HIN; then we exploit skip-gram to learn effective in-sample node representations for a HIN.

Given a source node v_j in a homogeneous network, the traditional random walk is a stochastic process with random variables $v_j^1, v_j^2, ..., v_j^k$ such that v_j^{k+1} is a node chosen at random from the neighbors of node v_k. The transition probability $p(v_j^{i+1}|v_j^i)$ at step i is the normalized probability

distributed over the neighbors of v_j^i by ignoring their node types. However, this mechanism is unable to capture the semantic and structural correlations among different types of nodes in a HIN. Here, we show how we use different built meta-paths to guide the random walker in a HIN to generate the paths of multiple types of nodes. Given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_{\mathcal{G}} = (\mathcal{A}, \mathcal{R})$, and a set of different meta-paths $\mathcal{S} = \{\mathcal{P}_j\}_{j=1}^n$, each of which is in the form of $A_1 \rightarrow ...A_t \rightarrow A_{t+1}... \rightarrow A_l$, we put a random walker to traverse the HIN. The random walker will first randomly choose a meta-path $\mathcal{P}_k$ from $\mathcal{S}$ and the transition probabilities at step i are defined as:

$$p(v^{i+1}|v_{A_t}^i, \mathcal{S}) =$$
$$\begin{cases} \frac{\lambda}{|\mathcal{S}|} \frac{1}{|N_{A_{t+1}}(v_{A_t}^i)|} & \\ \quad \text{if } (v^{i+1}, v_{A_t}^i) \in \mathcal{E}, \phi(v_{A_t}^i) = A_{app}, \phi(v^{i+1}) = A_{t+1} \\ \frac{1}{|N_{A_{t+1}}(v_{A_t}^i)|} & \\ \quad \text{if } (v^{i+1}, v_{A_t}^i) \in \mathcal{E}, \phi(v_{A_t}^i) \neq A_{app}, \\ \quad \phi(v^{i+1}) = A_{t+1}, (A_t, A_{t+1}) \in \mathcal{P}_k \\ 0 & \text{otherwise,} \end{cases}$$
$$(1)$$

where ϕ is the node type mapping function, $N_{A_{t+1}}(v_{A_t}^i)$ denotes the A_{t+1} type of neighborhood of node $v_{A_t}^i$, A_{app} is entity type of *app*, and λ is the number of meta-paths starting with $A_{app} \rightarrow A_{t+1}$ in the given meta-path set $\mathcal{S}$. The walk paths generated by the above strategy are able to preserve both the semantic and structural relations between different types of nodes in the HIN.

After mapping the word-context concept in a text corpus into a HIN via the above proposed meta-path guided random walk strategy (i.e., a sentence in the corpus corresponds to a sampled path and a word corresponds to a node), skip-gram (Mikolov et al. 2013a; Perozzi, Al-Rfou, and Skiena 2014) is then applied on the paths to minimize the loss of observing a node's neighbourhood (within a window w) conditioned on its current representation. The objective function of skip-gram is:

$$\arg\min_{X} \sum_{-w \leq k \leq w, j \neq k} -\log p(v_{j+k}|X(v_j)), \quad (2)$$

where $X(v_j)$ is the current representation vector of v_j, and $p(v_{j+k}|X(v_j))$ is defined using the softmax function:

$$p(v_{j+k}|X(v_j)) = \frac{\exp(X(v_{j+k}) \cdot X(v_j))}{\sum_{q=1}^{|\mathcal{V}|} \exp(X(v_q) \cdot X(v_j))}. \quad (3)$$

Due to its efficiency, we first apply hierarchical softmax technique (Mikolov et al. 2013b) to solve Eq. 3, and employ the stochastic gradient descent to train the skip-gram.

Hin2Img: Out-of-sample node representation learning. Can we use in-sample node embeddings learned by the above proposed HINE to efficiently learn representations of out-of-sample nodes in HIN and also enrich in-sample node representations? To answer this question, we first present the concept of k-order neighbors in HIN as following.

Definition 3 *k-order Neighbors in HIN*. Given a HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, let 1-order neighbors of a node $v_i \in \mathcal{V}$ be $S^{(1)}(v_i)$ so that $S^{(1)}(v_i) = \{v_j | (v_i, v_j) \in \mathcal{E}\}$; then, k-order neighbors $S^{(k)}(v_i)$ of a node v_i (k > 1) can be denoted as $S^{(k)}(v_i) = \{S^{(1)}(v_z) \setminus S^{(k-2)}(v_i), v_z \in S^{(k-1)}(v_i)\}$.

By the above definition, each node (i.e., either in-sample or out-of-sample) in HIN is represented by the following representation matrix:

$$\mathbf{X}(v_i) = [X(v_i), X(S^{(1)}(v_i)), ..., X(S^{(k)}(v_i))], \quad (4)$$

where $\mathbf{X} \in \mathbb{R}^{t \times d}$, each of which denotes d-dimensional node embedding. In our application, as illustrated in Figure 5, for each k, we exploit breadth-first search (BFS) method to find the t_k neighbors in the order of type app, signature, package, IMEI and API, and $t = 1 + \sum_1^k t_k$. Empirically we found $k = 2$ and $t = d$ perform the best in our application, and apply them to our problem throughout the paper. Note that we zero-pad the representation matrix in the corresponding rows when the node embeddings cannot be found (i.e., out-of-sample nodes) or $t < d$.
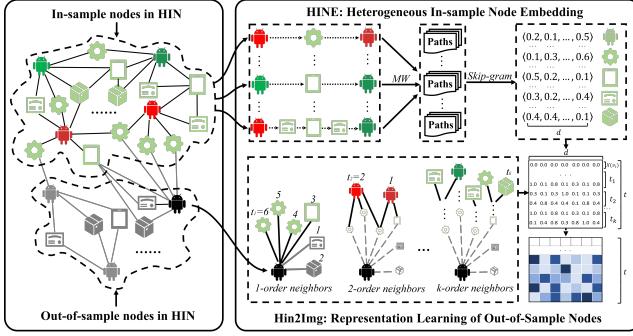
Figure 5: *HinLearning*: node representation learning in HIN.

Based on the in-sample node embeddings which can be learned offline using the proposed HINE, when a new node (e.g., a testing app) arrives, it only take $O(t \times d)$ time to obtain its representation using the proposed Hin2Img; furthermore, the representation learning for the new arriving node doesn't require rerunning HIN embeddings, which makes the downstream classifier workable for classifying the new arriving node without retraining.

3.4. Deep Neural Network

CNNs (LeCun, Bengio, and Hinton 2015) have achieved great success in learning salient features for classification tasks; while the crafty architecture of Inception (Szegedy et al. 2015) has shown high performance and low computational cost under strict constraints on memory and computational budget. Therefore, taking the generated representations of Android apps from previous section as inputs, we devise our deep learning framework leveraging the advantages of CNNs and Inception for real-time detection of Android malware, which is illustrated in Figure 6. In our designed DNN, for training, we take the generated $t \times d$ representation matrix $\mathbf{X}$ of each in-sample node (with type of

app) in HIN as input fed to the multilayer architecture to learn the higher level concept. In this multilayer architecture, the designed DNN first stacks pairs of convolutional layers and normalization layers with different filter sizes and strides followed by maxpoolings to capture universal features, such as curves and edges (Huang, Zhao, and Liu 2018). It then comes with an Inception module (Szegedy et al. 2015) to generate more task-specific features, such as discriminative properties for malicious and benign apps. In the Inception module, the general features will be passed through the mixture of convolutional layers in parallel to take advantage of multi-level feature extraction, in which 1×1 convolutions and 3×3 maxpooling are configured for dimensionality reduction prior to 1×1, 3×3, and 5×5 convolutions that are used for feature learning, followed by a concatenation layer to concatenate the resulting feature representations. After the global maxpooling, based on a pair of fully connected layer and a softmax layer, the designed DNN will train the model for classifying any new arriving node (i.e., out-of-sample node with type of app represented as a $t \times d$ matrix $\mathbf{X}$) as either benign or malicious.
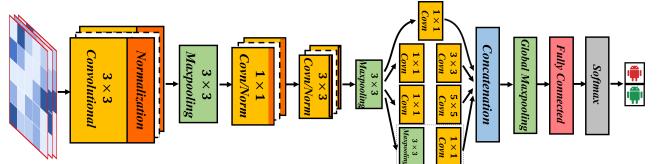

Figure 6: Our designed DNN for malware detection.

Algorithm 1 illustrates the implementation of our developed system *AiDroid* which integrates the above proposed method in detail.

Algorithm 1: *AiDroid*: HIN marries DNN.

Input: HIN $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with schema $\mathcal{T}_\mathcal{G} = (\mathcal{A}, \mathcal{R})$, traning data set D_I (in-sample apps), and testing data set D_O (out-of-sample apps).
Output: $\mathbf{y}$: The labels for the testing data set.

Learn in-sample node embeddings $X(v_i) \in \mathbb{R}^d$ $(i = 1, ..., |\mathcal{V}|)$ using HINE;
for $i = 1 \to |D_I|$ **do**
  Apply Hin2Img: for each k, find k-order neighbors $X(S^{(k)}(v_i))$ and then generate the $t \times d$ matrix $\mathbf{X}(v_i) = [X(v_i), X(S^{(1)}(v_i)), ..., X(S^{(k)}(v_i))]$;
end
Train DNN using $\mathbf{X}$s $\in \mathbb{R}^{t \times d}$;
for $j = 1 \to |D_O|$ **do**
  Use Hin2Img to generate $\mathbf{X}(v_j)$;
  Obtain the label y_{v_j} using trained DNN;
end
return $\mathbf{y}$;

---

4. Experimental Results and Analysis

In this section, we fully evaluate the performance of our developed system *AiDroid* for Android malware detection.

4.1. Data Collection

We obtain the large-scale real sample collection from Tencent Security Lab, which contains 190,696 training app (i.e., 83,784 benign and 106,912 malicious). After feature extraction and based on the designed network schema, the constructed HIN has 286,421 nodes (i.e., 190,696 nodes with type of app, 331 nodes with type of API, 70,187 nodes with type of IMEI, 8,499 nodes with type of signature, and 16,708 with type of affiliation) and 4,170,047 edges including relations of *R1-R6*. The new coming 17,746 unknown apps are used as testing data (to obtain the ground truth, they are further analyzed by the anti-malware experts, 13,313 of which are labeled as benign and 4,433 are malicious).

4.2. Baseline Methods

We validate the performance of our proposed method in *AiDroid* for Android malware detection by comparisons with different groups of baseline methods.

First, based on the constructed HIN described above, we evaluate our proposed HIN representation learn method *HinLearning* by comparisons with following baselines.

In-sample node embedding. We compare our proposed HINE with other network embedding methods including:

- **DeepWalk** and **LINE**: For DeepWalk (Perozzi, Al-Rfou, and Skiena 2014) and LINE (Tang et al. 2015), we ignore the heterogeneous property of HIN and directly feed the HIN for embedding.
- **metapath2vec**: We use each meta-path scheme separately to guide random walks in metapath2vec (Dong, Chawla, and Swami 2017).

For HINE, we divide the designed meta-paths into three sets (i.e., $\mathcal{S}_1 = \{PID1\}$, $\mathcal{S}_2 = \{PID3, PID4\}$, $\mathcal{S}_3 = \{PID2, PID5, PID6\}$) and use the proposed strategy to guide random walks. The parameter settings used for HINE are in line with DeepWalk, LINE and metapath2vec, which are empirically set as: vector dimension $d = 64$ (LINE: 64 for each order (1st- and 2nd-order)), walks per node $r = 20$, walk length $l = 50$ and window size $w = 5$.

Out-of-sample node representation learning. We compare our proposed Hin2Img for out-of-sample node representation learning with following baselines:

- **LocalAvg**: The out-of-sample nodes are represented by averaging embeddings of neighboring in-sample nodes.
- **LabelProp**: Label propagation proposed for multivariate regression problem can be used to learn the representations of out-of-sample nodes. As it has been demonstrated (Ma, Cui, and Zhu 2018) that the vanilla version of label propagation (Zhu and Ghahramani 2002) outperforms others, we hence use it as a baseline.
- **Rerunning**: For comparisons, we also run a baseline by rerunning all node embeddings when new nodes arrive.

Second, we evaluate different types of features for Android malware detection. Our proposed method is general for HINs. Thus, a natural baseline is to see whether the knowledge we add in should be represented as HIN instead of other features. Here we compare two types of features: behavioral sequences and HIN-based features.

- **Behavioral Sequences (*f-1*)**: We devise three baselines based on the extracted API call sequences of Android apps: (1) we build support vector machine (SVM) classifier based on binary (i.e., if an API call is invoked by an app) feature vectors (i.e., Bin+SVM); (2) we exploit Long Short-term Memory (LSTM) (Sutskever, Vinyals, and Le 2014) for sequence modeling, based on which SVM classifier is then built (i.e., LSTM+SVM); and (3) we also train our proposed DNN based on the extracted API call sequences for evaluation (i.e., Seq+DNN).
- **HIN-based Features (*f-2*)**: Based on the constructed HIN, the proposed *HinLearning* is applied for representation learning (i.e., both in-sample and out-of-sample nodes), based on which the designed DNN is used to train the classification model for prediction (i.e., *AiDroid*).

4.3. Comparisons and Analysis

Based on the HIN constructed from the training data, using the developed DNN as downstream classifier and the new coming nodes (i.e., apps) for testing, from Table 1, we can observe that different combinations of in-sample node embedding and out-of-sample representation learning show different performances in Android malware detection: (1) For in-sample node embedding methods, our proposed HINE outperforms all baselines in terms of *ACC* and *F1*. That is to say, HINE learns significantly better node (i.e., app) representation in HIN than current state-of-the-art methods. The success of HINE lies in the proper consideration and accommodation of the heterogeneous property of HIN (i.e., the multiple types of nodes and relations), and the advantage of random walk guided by different meta-paths for sampling the node paths. (2) For out-of-sample representation learning, our proposed Hin2Img consistently and significantly outperforms all baselines (i.e., the detection performance achieve superb 0.9908 *ACC* and 0.9817 *F1*), which even surpasses the rerunning HIN embeddings. Obviously, $t \times d$ representation matrices learned by Hin2Img utilizing 1- and 2-order neighbors are more expressive than other embeddings in depicting the apps for the problem of real-time Android malware detection.

Table 1: Comparisons of different methods.

Metric	In-sample Embedding	Out-of-sample Learning			Rerunning
		LocalAvg	LabelProp	**Hin2Img**	
ACC	DeepWalk	0.9057	0.9214	0.9506	0.9516
	LINE	0.9111	0.9307	0.9690	0.9602
	metapath2vec	0.9289	0.9448	0.9799	0.9722
	HINE	0.9389	0.9533	**0.9908**	0.9843
F1	DeepWalk	0.8267	0.8547	0.9055	0.9076
	LINE	0.8364	0.8705	0.9398	0.9234
	metapath2vec	0.8669	0.8954	0.9606	0.9459
	HINE	0.8849	0.9094	**0.9817**	0.9691

We also show the comparisons in Table 2 for different features in Android malware detection. From the results, we can see that: (1) Based on the extracted API call sequences (i.e., *f-1*), LSTM provides significant improvement in sequence modeling while our proposed DNN outperforms

others in detection Android malware. (2) Compared with content-based features only, HIN-based features (i.e., *f-2*) indeed perform better. The reason behind this is that HIN-based features are more expressive to characterize a complex and comprehensive relatednesses over apps through the designed meta-paths which consist of not only relationships between apps and their invoked API calls, but also higher-level semantics within the ecosystem.

Table 2: Comparisons of different types of features.

Feature	Method	TP	TN	FP	FN
f-1	Bin+SVM	3,926	11,828	1,485	507
	LSTM+SVM	4,115	12,339	974	318
	Seq+DNN	4,168	12,504	809	265
f-2	*AiDroid*	4,395	13,188	125	38

Feature	Method	Recall	Precision	ACC	F1
f-1	Bin+SVM	0.8856	0.7255	0.8877	0.7976
	LSTM+SVM	0.9282	0.8086	0.9271	0.8642
	Seq+DNN	0.9402	0.8374	0.9394	0.8858
f-2	*AiDroid*	0.9914	0.9723	0.9908	0.9817

4.4. Parameter Sensitivity, Scalability and Stability

In this set of experiments, we systematically evaluate the parameter sensitivity, scalability and stability of our developed system *AiDroid*. We first examine how latent dimensions (d) and neighborhood size (w) affect the performance in Android malware detection. As shown in Figure 7.(a) and (b), we can see that *AiDroid* is not strictly sensitive to these parameters and is able to reach high performance under a cost-effective parameter choice. We then run the experiments using new arriving apps from Aug. 1-10, 2018 to assess the average detection time and accuracy. Figure 7.(c) and (d) demonstrate *AiDroid* is scalable and stable over a long time span in detecting newly generated Android malware (i.e., average prediction time: 4.3 ms/app and 0.9891 ACC on average). Figure 7.(e) shows the ROC curve of *AiDroid* based on the data described in Section 4.1 which achieves an impressive 0.9914 true positive rate (*TPR*) at 0.0094 false positive rate (*FPR*). We can conclude that *AiDroid* is indeed feasible in practical use for real-time Android malware detection.
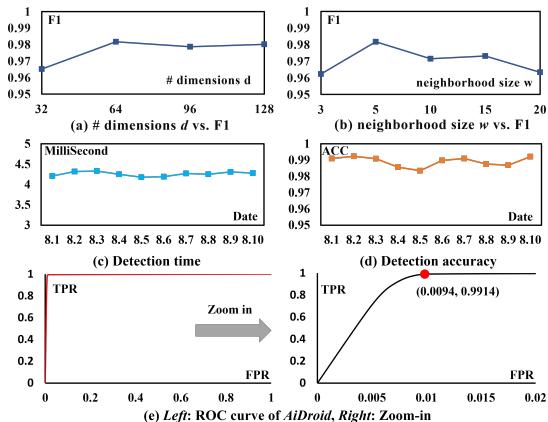

Figure 7: Parameter sensitivity, scalability and stability.

5. Related Work

In recent years, there have been ample research studies on developing intelligent Android malware detection systems using machine learning and data mining techniques (Wu et al. 2012; Wu and Hung 2014; Hou et al. 2016; Chen, Hou, and Ye 2017; Ye et al. 2017; Saracino et al. 2018). For example, DroidDolphin (Wu and Hung 2014) built classifiers based on dynamic analysis, while Droid-Mat (Wu et al. 2012) and DroidMiner (Yang et al. 2014) constructed their models based on static analysis. However, most of the existing systems merely utilize content-based features for the detection. To further address the challenges of Android malware detection, in our preliminary work, Hin-Droid (Hou et al. 2017) was proposed which considered higher-level semantic relations among apps and APIs and introduced HIN for the first time in Android malware detection; but HinDroid was primarily designed for static HIN without considering new arriving nodes.

To solve the problem of network representation learning, after DeepWalk (Perozzi, Al-Rfou, and Skiena 2014), LINE (Tang et al. 2015) and node2vec (Grover and Leskovec 2016) that were proposed for homogeneous network embedding, HIN2vec (Fu, Lee, and Lei 2017), metapath2vec (Dong, Chawla, and Swami 2017), metagraph2vec (Fan et al. 2018), and PME (Chen et al. 2018) have been proposed for HIN representation learning. However, few of them can deal with out-of-sample nodes, i.e., nodes that arrive after the HIN embedding process. Though algorithms (Chang et al. 2015; Zhao et al. 2018) have been proposed to infer embeddings for out-of-sample nodes in HIN, they necessitate adjusting in-sample node embeddings and also the downstream classifier retraining. Efficient representation learning for out-of-sample nodes in HIN without rerunning/adjusting HIN embeddings is in need for our application in real-time Android malware detection.

6. Conclusion

To combat the evolving Android malware attacks, in this paper, we first extract the API call sequences from runtime executions of Android apps and further analyze higher-level semantic relationships within the ecosystem. To depict such complex relations, we introduce HIN for modeling and use meta-path based approach to build up related-nesses over apps. To efficiently classify nodes (i.e., apps) in HIN, we propose the *HinLearning* method to first gain in-sample node embeddings and then learn representations of out-of-sample nodes without rerunning/adjusting HIN embeddings for the first time. Afterwards, we design a DNN classifier leveraging the advantages of CNNs and Inception for Android malware detection. A comprehensive experimental study on the large-scale real data collections from Tencent Security Lab is performed to compare various baselines. Promising experimental results demonstrate that our developed system *AiDroid* outperforms others in real-time Android malware detection, which has been incorporated into Tencent Mobile Security product that serves millions of users worldwide.

References

[Chang et al. 2015] Chang, S.; Han, W.; Tang, J.; Qi, G.-J.; Aggarwal, C. C.; and Huang, T. S. 2015. Heterogeneous network embedding via deep architectures. In *KDD*, 119–128. ACM.

[Chen et al. 2018] Chen, H.; Yin, H.; Wang, W.; Wang, H.; Nguyen, Q. V. H.; and Li, X. 2018. Pme: Projected metric embedding on heterogeneous networks for link prediction. In *KDD*.

[Chen, Hou, and Ye 2017] Chen, L.; Hou, S.; and Ye, Y. 2017. Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks. In *ACSAC*, 362–372. ACM.

[Dong, Chawla, and Swami 2017] Dong, Y.; Chawla, N. V.; and Swami, A. 2017. metapath2vec: Scalable representation learning for heterogeneous networks. In *KDD*.

[Fan et al. 2018] Fan, Y.; Hou, S.; Zhang, Y.; Ye, Y.; and Abdulhayoglu, M. 2018. Gotcha-sly malware!: Scorpion a metagraph2vec based malware detection system. In *KDD*.

[Fu, Lee, and Lei 2017] Fu, T.-Y.; Lee, W.-C.; and Lei, Z. 2017. Hin2vec: Explore meta-paths in heterogeneous information networks for representation learning. In *CIKM*.

[Grover and Leskovec 2016] Grover, A., and Leskovec, J. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM.

[Hou et al. 2016] Hou, S.; Saas, A.; Ye, Y.; and Chen, L. 2016. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In *WAIM*, 54–66.

[Hou et al. 2017] Hou, S.; Ye, Y.; Song, Y.; and Abdulhayoglu, M. 2017. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. In *KDD*, 1507–1515. ACM.

[Huang, Zhao, and Liu 2018] Huang, S.-J.; Zhao, J.-W.; and Liu, Z.-Y. 2018. Cost-effective training of deep cnns with active model adaptation. In *KDD*.

[LeCun, Bengio, and Hinton 2015] LeCun, Y.; Bengio, Y.; and Hinton, G. 2015. Deep learning. *Nature*.

[Ma, Cui, and Zhu 2018] Ma, J.; Cui, P.; and Zhu, W. 2018. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*.

[Mikolov et al. 2013a] Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013a. Efficient estimation of word representations in vector space. In *arXiv preprint arXiv:1301.3781*.

[Mikolov et al. 2013b] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; and Dean, J. 2013b. Distributed representations of words and phrases and their compositionality. In *NIPS*.

[Perozzi, Al-Rfou, and Skiena 2014] Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. Deepwalk: Online learning of social representations. In *KDD*.

[Saracino et al. 2018] Saracino, A.; Sgandurra, D.; Dini, G.; and Martinelli, F. 2018. Madam: Effective and efficient behavior-based android malware detection and prevention. *TDSC*.

[Statcounter 2018] Statcounter. 2018. Mobile operating system market share worldwide. In *http://gs.statcounter.com/os-market-share /mobile/world-wide*.

[Sun and Han 2012] Sun, Y., and Han, J. 2012. Mining heterogeneous information networks: principles and methodologies. *Synthesis Lectures on Data Mining and Knowledge Discovery*.

[Sun et al. 2011] Sun, Y.; Han, J.; Yan, X.; Yu, P. S.; and Wu, T. 2011. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. *PVLDB*.

[Sutskever, Vinyals, and Le 2014] Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

[Szegedy et al. 2015] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *CVPR*.

[Tang et al. 2015] Tang, J.; Qu, M.; Wang, M.; Zhang, M.; Yan, J.; and Mei, Q. 2015. Line: Large-scale information network embedding. In *WWW*.

[TencentSecurity 2018] TencentSecurity. 2018. Mobile phone security report for the first half of 2018. In *https://m.qq.com/security_lab/news _detail_471.html*.

[Wu and Hung 2014] Wu, W.-C., and Hung, S.-H. 2014. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In *RACS*.

[Wu et al. 2012] Wu, D. J.; Mao, C. H.; Wei, T. E.; Lee, H. M.; and Wu, K. P. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *Asia JCIS*, 62–69.

[Yang et al. 2014] Yang, C.; Xu, Z.; Gu, G.; Yegneswaran, V.; and Porras, P. 2014. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *ESORICS*.

[Ye et al. 2017] Ye, Y.; Li, T.; Adjeroh, D.; and Iyengar, S. S. 2017. A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)* 50(3).

[Zhao et al. 2018] Zhao, D.; Li, J.; Tan, Y.; Yang, K.; Ge, B.; and Dou, Y. 2018. Optimization adjustment of human resources based on dynamic heterogeneous network. *Physica A*.

[Zhu and Ghahramani 2002] Zhu, X., and Ghahramani, Z. 2002. Learning from labeled and unlabeled data with label propagation. *CMU Technical report*.