ANDRUSPEX: Leveraging Graph Representation Learning to Predict Harmful App Installations on Mobile Devices

Yun Shen
NortonLifeLock Research Group
yun.shen@nortonlifelock.com

Gianluca Stringhini
Boston University
gian@bu.edu

Abstract—Android's security model severely limits the capabilities of anti-malware software. Unlike commodity anti-malware solutions on desktop systems, their Android counterparts run as sandboxed applications without root privileges and are limited by Android's permission system. As such, PHAs on Android are usually willingly installed by victims, as they come disguised as useful applications with hidden malicious functionality, and are encountered on mobile app stores as suggestions based on the apps that a user previously installed. Users with similar interests and app installation history are likely to be exposed and to decide to install the same PHA. This observation gives us the opportunity to develop predictive approaches that can warn the user about which PHAs they will encounter and potentially be tempted to install in the near future. These approaches could then be used to complement commodity anti-malware solutions, which are focused on post-fact detection, closing the window of opportunity that existing solutions suffer from. In this paper we develop ANDRUSPEX, a system based on graph representation learning, allowing us to learn latent relationships between user devices and PHAs and leverage them for prediction. We test ANDRUSPEX on a real world dataset of PHA installations collected by a security company, and show that our approach achieves very high prediction results (up to 0.994 TPR at 0.0001 FPR), while at the same time outperforming alternative baseline methods. We also demonstrate that ANDRUSPEX is robust and its runtime performance is acceptable for a real world deployment.

1. Introduction

While becoming the most popular mobile OS in the world, Android has also attracted significant interest by malicious parties, who started developing apps with malicious purposes targeting this platform [2], [40], [70], [86]. Millions of these malicious Android apps are observed every year [2], carrying out various types of harmful activity including stealing private information from the victim devices [83], sending premium SMS messages, performing click fraud, and encrypting the victim's data in exchange for a ransom [3], [47]. Similar to what happens for desktop computers, not all malicious Android applications come with clearly harmful content, but some present unwanted components that are often an annoyance to the user (e.g., adware [22], [31], [35], [75]) or without user consent (e.g., click fraud [16]). In this paper, we adopt the terminology used by Google and refer to these potentially unwanted Android apps as *potentially harmful apps*

(PHAs) [25]. Unlike traditional desktop malware, which is often installed by automatically exploiting vulnerabilities in the victim's Web browser and carrying out *drive-by* download attacks [57], Android PHAs tend to be manually installed by victims, who willingly choose to install apps that promise useful functionalities but come together with harmful code or advertising SDKs that hide malicious functionalities [70], [79], [86]. In fact, the preliminary analysis that we performed ahead of this study on a real world dataset of apps installed by millions of real users showed that 93.1% of PHAs are manually installed via either the Google play store or other well known side-loading apps like `com.sec.android.easyMover` and `com.huawei.appmarket`.

To mitigate the threat of PHAs on Android, the security community has followed two directions. The first one is to analyze Android apps to identify malicious behavior [1], [4], [26], [55], [13], [45], [80], [82]. In the real world, this analysis usually takes place when apps are submitted to an Android marketplace. For example, Google uses a vetting system known as *Android Bouncer* [52]. While quite effective in detecting malicious Android apps, these systems suffer from the limitation that the Android ecosystem is quite fragmented, and users install apps from a variety of sources, including alternative marketplaces that do not employ strict enough security measures [2], [77]. To complement market-level detection, a plethora of commercial anti-malware software products are available to users. Unfortunately, the Android security model severely limits their capabilities [21]. Unlike traditional desktop anti-malware software, mobile solutions run as sandboxed apps without root privileges [21] and cannot proactively block malicious downloads or even automatically remove a PHA once they find it; instead, they periodically scan the mobile device for suspicious apps and simply warn the user about newly discovered threats, prompting them to manually remove the detected PHAs. This is far from ideal because detection happens after the PHAs have been installed, leaving a *window of vulnerability* during which attackers can cause harm to the victims and their devices undetected.

In this paper, we propose to mitigate the aforementioned issues by complementing existing on-device anti-malware solutions with the capability of *predicting* future PHA installations on the device. Our intuition is that since the installation of a PHA on Android is usually a consequence of a user's own rational choices, the installation of a certain PHA on a user's device can be predicted by observing the apps that the user has installed

in the past, together with which PHAs "similar" users have installed. For example, users with similar interests (e.g., gaming) might be active on the same third party marketplaces and receive suggestions to install similar PHAs (e.g., disguised as free versions of popular games). If we were able to predict which PHA a user will attempt to install in the near future, we could display a warning to them ahead of time and potentially convince them not to install that app, closing the window of vulnerability between installation and detection introduced by traditional Android anti-malware software.

To achieve an effective prediction of which PHAs will be installed on mobile devices in the future, we develop a system called ANDRUSPEX. Our approach is based on *graph representation learning* [27]. This technique allows us to automatically discover important features in raw data, without the need for feature engineering. ANDRUSPEX first builds a graph of PHA installation events on a global scale, in which edges represent which devices installed which PHAs. It then applies representation learning to learn the low dimensional vertex representation of the PHA installation graph at time t. Finally, ANDRUSPEX predicts new links that will be formed between devices and PHAs that at time $t + \delta$ based on their respective properties and the currently observed links at time t. In production, ANDRUSPEX would then warn the user about suspicious applications that they will encounter and likely be interested in installing in the near future (e.g., on a third party Android marketplace), complementing current on device Android anti-malware solutions that can only act after the fact.

Contributions. Our contributions are summarized as follows;

- We present ANDRUSPEX, a system based on graph representation learning which allows us to predict which PHAs a user might be tempted to install on an Android device ahead of time. To the best of our knowledge, we are the first ones to propose a predictive system in the area of Android malware mitigation.
- We test ANDRUSPEX on a dataset of tens of millions of PHA installations collected from real devices that installed the Android anti-malware program of a major security company. We show that ANDRUSPEX can achieve very good performance prediction (up to 0.994 TPR at 0.0001 FPR), significantly outperforming simpler baseline methods.
- We also show that ANDRUSPEX could be effectively used in the wild since it has both acceptable performance and it does not significantly reduce its detection performance when the system needs to be retrained.

2. Motivation

The main motivation behind our approach is that, unlike desktop malware, PHAs on Android are usually willingly installed by less attentive users [23]. To confirm this motivation, we performed a preliminary analysis of the PHA installation events reported by 2.4M devices that installed a commercial mobile security product during
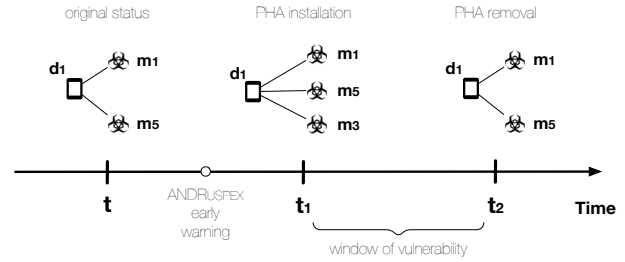

Figure 1: PHA installation over time for a device d_1. The goal of our approach is to correctly predict that d_1 will install m_3 in the future and close the window of vulnerability.

March 2019 (see Section 4 for more details about the dataset). When a PHA is detected, the security product records its installer package names and how it was installed on the device (i.e., pre-installed or user-installed). Matching this information with the official package names of Android market apps we find that 93.1% of the PHAs identified on those devices were manually installed by users through Google Play or alternative marketplaces. This is confirmed by a recent work by Kotzias *et al.* [36], which showed that the Google Play market is responsible for 87% of all PHA installations and for 67% of unwanted app installations. Since the Google Play market is the official market of Android apps, end users may trust apps distributed from Google Play and willingly install them. In short, our preliminary analysis, combined with the fact that anti-malware systems on Android are severely limited by the platform's security model, and cannot proactively block known PHAs nor automatically remove them after they are detected, shows that a predictive approach to PHA mitigation, allowing the user to be warned ahead of time, would be beneficial.

The main research challenge is how to do this effectively. Looking at device activity in isolation does not provide enough information to predict the upcoming PHA installations for a given mobile device. For example, in Figure 1 there is no easy way to predict that m_3 will be installed based on the past installation activity on d_1. One possible solution is to collect abundant additional information about d_1 (e.g., application categories, device types, marketplace information, etc.) and leverage feature engineering to create features for PHA installation events. These features can then be used to train a machine learning model to predict future installations. However, such process requires domain knowledge and is essentially a manual endeavor that is time-consuming and problem-dependent.

Another plausible solution is looking at the installation behavior of multiple devices, treating a device's historical installation activities as its behavioral profile, and grouping devices with similar profiles together [84]. This way, we can predict which PHA a given device's owner will install based upon the PHAs that have been installed by those with similar profiles. For instance, as we can see in Figure 2a (❶), d_1, d_3, and d_4 share similar behavioral profiles. Given how Android marketplaces work, it is likely that m_1 and m_3 are apps that the owner of d_3 is interested in, and they might even been suggested by
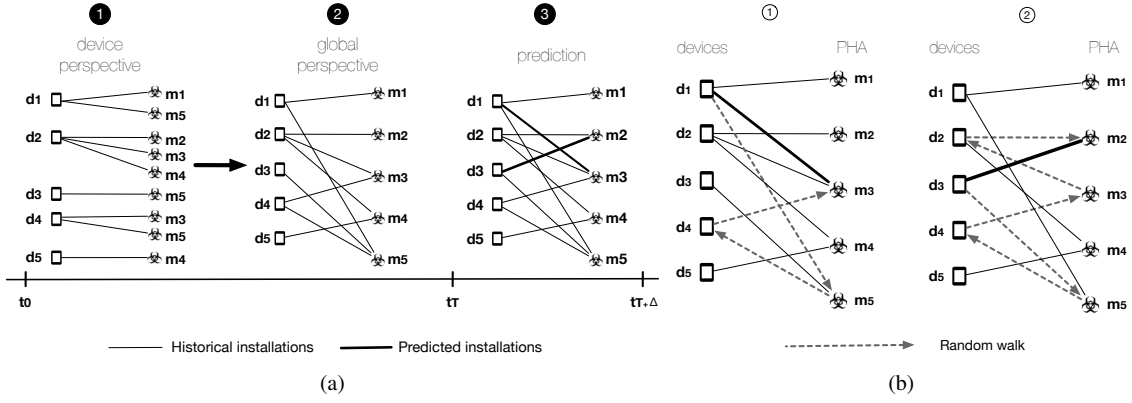
Figure 2: Global view of PHA installation events (Figure 2a) and potential prediction clues captured by random walks (Figure 2b) from this global graph.
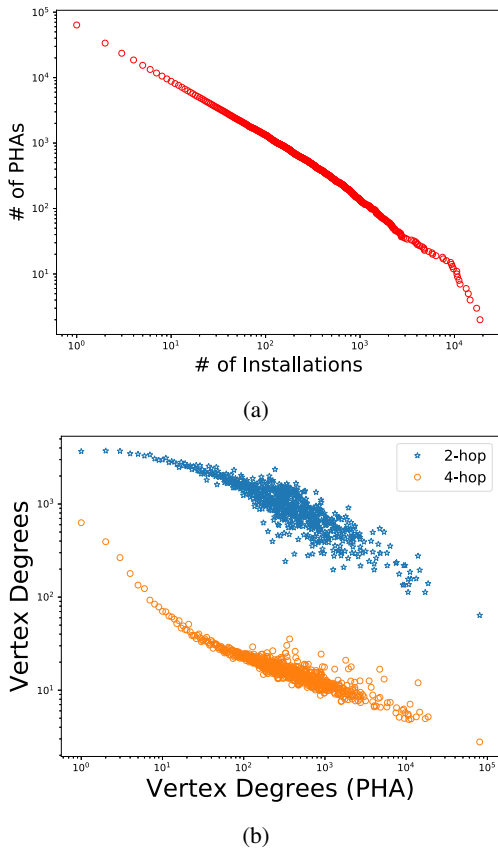

(a)

(b)

Figure 3: PHA installations follow a power law distribution (Figure 3a) and have implicit correlations (Figure 3b) between PHA degrees and the average degrees of all vertices reachable by 2 hops (i.e., by a path $(\mathbf{m}_i, \mathbf{d}_z, \mathbf{m}_j)$, blue) and 4 hops (i.e., by a path $(\mathbf{m}_i, \mathbf{d}_z, \mathbf{m}_j, \mathbf{d}_l, \mathbf{m}_h)$, orange) using one day PHA installation data (see Section 4).

the marketplace itself. As such, it is more likely that $\mathbf{d}_3$ will install those PHAs in the future than a randomly chosen PHA. It is however not trivial to group devices with similar behavioral profiles. For example, we can use the Jaccard similarity coefficient to calculate the similarity between two devices, but this leads to a $O(n^2)$ time complexity method [84] to generate pair wise similarity

for potentially millions of devices, which renders such approach practically infeasible.

To mitigate the aforementioned limitations, we propose to approach the problem of predicting PHA installations in a principled way, by looking at the collective PHA installation events from a global perspective instead of focusing on a device level (see ❷ in Figure 2a). As it can be seen, this graph provides aggregated historical information of how the PHAs have been installed by mobile devices globally, and yields additional structural information by representing the relations between devices and PHAs as a bipartite graph. This graph enables us to capture the collective behavior from all devices, and elicit the similarities between the download activity of different devices.

Leveraging this global installation graph for prediction, however, presents two challenges. The first challenge is the distribution of PHA installations among the devices. We use data on the PHAs installed by real devices over a period of one day (see Section 4) for illustration purposes. As we can see in Figure 3a, the distribution of PHA installations (i.e., vertex degrees from a graph perspective) follows a power law distribution. This indicates that there are popular PHAs with a number of installations that greatly exceed the average and that the majority of the PHA population only appears in a small number of devices. Preferential attachment [5], [50] has been proposed as an effective mechanism to explain conjectured power law degree distributions. We show in Section 5.2 that such model is promising but insufficient to predict PHA installation, which instead requires a model that considers both popular and less popular PHAs at the same time.

The second challenge is modeling the implicit relationships among PHAs with limited information. For example, in the real world, a user may choose to install $\mathbf{m}_1$ due to a deceitful full-screen ad displayed in $\mathbf{m}_5$, and $\mathbf{m}_3$ could be willingly installed by a user on device $\mathbf{d}_4$ because recommended by the same app store from which $\mathbf{m}_5$ was initially installed. Inferring these relationships from observing app installation is challenging. The global installation graph as depicted in Figure 2a ❸ can help identify these hidden relationships. In Figure 3b, we show possible correlations between PHA degrees and the average degrees of all vertices reachable by 2 hops (i.e., by a path $(\mathbf{m}_i, \mathbf{d}_z, \mathbf{m}_j)$, blue) and 4 hops (i.e., by a path

$(\mathbf{m}_i, \mathbf{d}_z, \mathbf{m}_j, \mathbf{d}_l, \mathbf{m}_h)$, yellow). We can observe two interesting aspects. The first aspect is that there is a negative correlation in both cases. This indicates that PHAs with larger installations (i.e., popular PHAs) are co-existing with smaller ones (i.e., less popular PHAs). The second aspect is that the correlation coefficient decreases with the increasing number of hops. For example, the correlation coefficient between PHA degrees and the average degrees of 2-hop vertices is -0.28 while the correlation coefficient of those 4 hops away is -0.11.

Core idea. To address both challenges, in this paper, we put random walk [78], [51] and graph representation learning [6] in use. Graph representation learning allows us to take both popular and unpopular PHAs into account, while random walk enables us to harvest the aforementioned explicit and implicit relationships among PHAs for each device. Take Figure 2b (①) for instance, if we want to predict the next PHA that $\mathbf{d}_1$ will install, we can take a random walk starting from $\mathbf{d}_1$ to capture the aforementioned implicit relationship. Suppose we obtain a random walk $(\mathbf{d}_1, \mathbf{m}_5, \mathbf{d}_4, \mathbf{m}_3)$ as illustrated in Figure 2b (①), offers us an indication that $\mathbf{d}_1$ may potentially install the malicious app $\mathbf{m}_3$ in the near future. Similarly, as we can see in Figure 2b (②), a random walk $(\mathbf{d}_3, \mathbf{m}_5, \mathbf{d}_4, \mathbf{m}_3, \mathbf{d}_2, \mathbf{m}_2)$ may capture the higher proximity relationship between $\mathbf{d}_3$ and $\mathbf{m}_2$, and in turn, facilitate the prediction. In Section 3 we show that leveraging graph representation learning to extract useful information from the random walks enables us to understand the explicit and implicit dynamics between devices and PHAs without a time-consuming feature engineering step, and allows us to make accurate PHA installation predictions up to one week ahead of the real installations.

Problem formulation. We formalize our PHA installation prediction problem as follows. Let $\mathbf{D}$ and $\mathbf{M}$ denote all unique devices and PHAs respectively, and $\mathbf{E}$ denote all observed PHA installation events. We define our global PHA installation graph as a bipartite graph $\mathbf{G}_{[t_0, t_T]} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \mathbf{D} \cup \mathbf{M}$ denotes vertices in the graph, each edge $\mathbf{e} = (\mathbf{d}_i, \mathbf{m_j}, t_k) \in \mathbf{E}$ represents a device $\mathbf{d}_i \in \mathbf{D}$ installs a PHA $\mathbf{m_j} \in \mathbf{M}$ at a particular timestamp $t_k \leq t_T$. Our goal is to learn lower d-dimensional representations of $\mathbf{V}$ in $\mathcal{G}_{[t_0, t_T]}$, denoted as $\Phi_\mathbf{V} \in \mathbb{R}^{|\mathbf{V}| \times d}$, by taking global PHA installation information into consideration. With the learned vertex embeddings $\Phi_\mathbf{V}$, ANDRUSPEX trains a prediction function $f : \mathbf{G}_{(t_0, t_T]} \rightarrow \mathbf{E}_{(t_T, t_{T+\Delta}]}$ that outputs a list of edges (i.e., PHA installations) that are not present in $\mathbf{G}_{[t_0, t_T]}$, but predicted to appear in the graph $\mathbf{G}_{(t_T, t_{T+\Delta}]}$ (see ❸ in Figure 2a for illustration).

Note. In this paper, we do not aim to understand how PHAs are distributed nor do we target to detect unknown PHAs through graph analysis. We also do not investigate, from a behavioral perspective, how PHAs operate and propagate once installed in the devices. Rather, our main goal is to predict the exact PHAs that are likely to be installed by the end users **in the near future** given their historical installation activities and the collective PHA installation events on a global scale.
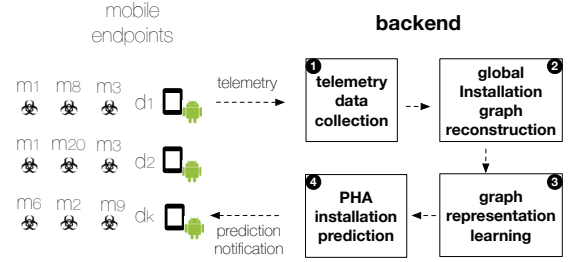

Figure 4: System overview of ANDRUSPEX

3. Methodology

3.1. System Overview

The system overview of ANDRUSPEX is depicted in Figure 4. ANDRUSPEX's frontend component is installed on mobile devices, alongside an existing mobile malware detection product. Every time the malware detection product detects that a PHA has been installed, the mobile device sends data to our backend infrastructure (❶). ANDRUSPEX's backend component uses this data to build a global PHA installation graph (❷). It then leverages graph representation learning (Section 3.3) to capture the collective behavior from all devices and understand the implicit relationships among the PHAs in a latent space (❸). ANDRUSPEX periodically trains a prediction engine based upon these historical installations, and predicts the impending PHA installations on the mobile devices that installed the frontend component in the future (❹). When ANDRUSPEX predicts that a mobile device's user will attempt to install a certain PHA in the near future (for example because this PHA will appear as a suggestion on a third party market) it sends an alert to the mobile device, so that the device's owner can be warned about the threat and reach an informed decision to not install a PHA which might weaken their device security.

3.2. ANDRUSPEX Workflow

Following the system overview in Figure 4, the workflow of ANDRUSPEX at the backend is depicted in Figure 5. Its operation consists of three steps: ❶ reconstruction of the global PHA installation graph, ❷ graph representation learning, and ❸ PHA installation prediction.
Step 1: Reconstruction of the Global PHA installation graph (❶ in Figure 5). The goal of this step is building an installation graph that encapsulates a comprehensive view of how PHAs are installed by mobile devices on a global scale. To build the graph, ANDRUSPEX takes as input the streams of detected PHA installation events generated by the mobile devices. These events are treated as tuples, in the format of $(\mathbf{d}_i, \mathbf{m}_j, t_k)$. Let $\mathbf{A}$ denote the symmetric adjacency matrix of $\mathbf{G}$. $\mathbf{A}_{\mathbf{d}_i, \mathbf{m}_j} = 1$ if $\mathbf{d}_i$ and $\mathbf{m}_j$ are linked in the same installation event, otherwise $\mathbf{A}_{\mathbf{d}_i, \mathbf{m}_j} = 0$. The output of this step is a global PHA installation graph $\mathbf{G}$ represented by the adjacency matrix $\mathbf{A}^{|\mathbf{D}| \times |\mathbf{M}|}$. This matrix will be used by the graph representation learning module in the next step.
Step 2: Graph representation learning (❷ in Figure 5). The core of ANDRUSPEX is learning the low dimensional
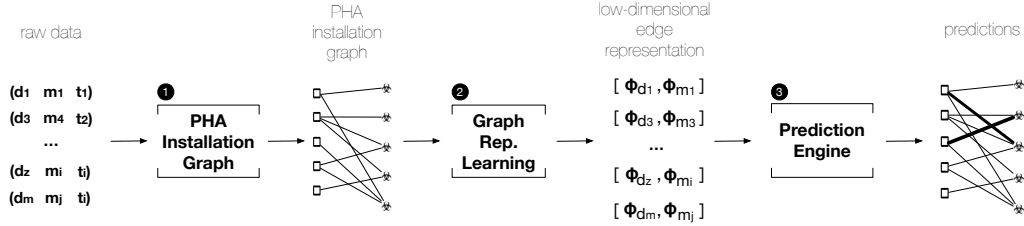
Figure 5: ANDRUSPEX's architecture. ANDRUSPEX collects PHA installation events from the mobile devices that have installed a mobile security product. These events are used to build a global PHA installation graph. ANDRUSPEX then learns the low dimensional vertex representations and uses them to build and validate ANDRUSPEX's predictive model.

vertex representations (i.e., vertex embeddings) of the PHA installation graph. Essentially, ANDRUSPEX takes a number of truncated random walks from each vertex $d_i \in D$ in the graph G. These random walks effectively capture the high-order indirect connections between mobile devices and PHAs. In this way, ANDRUSPEX explicitly builds a high-order proximity transition matrix between D and M. It then factorizes this matrix together with a decay factor to account for the strength of the implicit connections (see technical details in Section 3.3). The output of this step is the low dimensional vertex representations (i.e., Φ_V) that will be used to model PHA installation events in the latent space.

Step 3: PHA installation prediction (❸ in Figure 5). Taking the vertex embeddings as the input, ANDRUSPEX models the observed PHA installation events in the latent space and trains a prediction model to predict future installations. First, ANDRUSPEX models the observed PHA installations by concatenating two low dimensional vertex embeddings in the latent space (see Section 6 for the influence of alternative edge representations). That is, $\phi_e = concat(\phi_{d_i}, \phi_{m_j})$, where $e \in E$. Effectively, each PHA installation event is represented by a $2d$-dimensional vector. ANDRUSPEX then formulates the prediction task as a binary prediction problem for edges (i.e., PHA installations) in the graph where two classes are considered: *positive* or presence of edges and *negative* or absence of edges. ANDRUSPEX samples an equal number of non-existing edges [54] from the observed PHA installation edges (i.e., positive class) to form the negative class and train the prediction model. The output of the PHA installation prediction is a list of edges (i.e., PHA installations) that are not present in $[t_0, t_T]$, but predicted to appear in the $(t_T, t_{T+\Delta}]$. These predictions can then be leveraged to warn the user ahead of time about PHAs that they are likely to encounter and might be enticed to install.

End-to-end solution. We envision ANDRUSPEX to be deployed as an end-to-end solution to warn the end user about PHAs that they will likely encounter in the future. At the backend, ANDRUSPEX continuously collects and monitors PHAs installed on all mobile devices that install the security application. If an end user installs a known PHA, the installation event would trigger the on-device security engine to send the detection back to ANDRUSPEX. The system would then calculate the probability of this end user to install a related PHA based on the telemetry from other devices. If the probability is higher than a threshold, ANDRUSPEX notifies the end user with
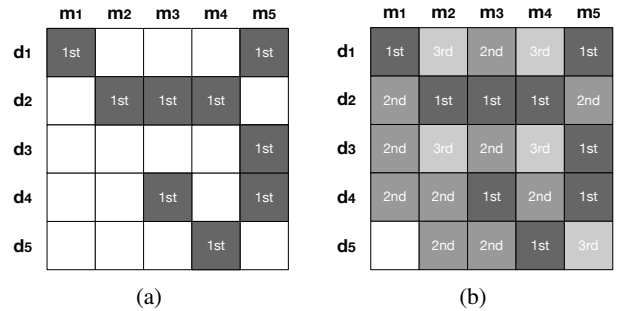

Figure 6: Illustration of the l-order proximity matrix via random walk. Figure 6a shows a 1-order matrix and Figure 6b shows a 3-order proximity matrix for the PHA installation graph shown in Figure 2a. We use grayscale to illustrate the diminishing correlation among vertices (see Figure 3b). A blank cell implies that l-order proximity is 0. As it can be seen, the 3-order proximity matrix (Figure 6b) offers more information comparing to 1-order proximity matrix (Figure 6a).

a summary of why the system reached such a decision. For example, ANDRUSPEX could highlight the key aspects of a PHA distribution campaign as shown in the case studies in Section 6. This summary would be displayed to the end user when they use the security app as in-app messages. The rationale of using in-app messages is to explain to the end user the potential risk associated with the known on-device PHAs. This way, the end user would benefit from an informative explanation rather than a short notification via the status bar. We are planning to collaborate with usable security experts in the future to design an effective in-app warning approach(e.g., modularity based approach [46].

3.3. Graph Representation Learning

The key challenge of ANDRUSPEX is building a model that learns both explicit relations between devices and PHAs, and implicit relationships among PHAs from the global PHA installation graph, and predict the future installations. To address this challenge, we use a graph representation learning model [81]. Representation learning [6], [81] is able to extract useful information to build prediction models *without feature engineering*. This characteristic of representation learning is particularly desirable since it enables us to understand the explicit and

implicit dynamics between devices and PHAs without a time-consuming feature engineering step.

Random walk has been proven to provide insightful transitive associations between two vertices in a graph, and has been successfully applied in numerous settings, e.g., community detection [56], [63], personalized PageRank [53], and Sybil account detection [18], [10], [68]. The core of ANDRUSPEX is building a high-order proximity matrix by conducting truncated random walks with a decay factor on the global PHA installation graph. This proximity matrix captures both direct and indirect relationships between devices and PHAs, and, at the same time, discriminates the strength between different orders of proximity (see Section 2) due to the decay factor. The low dimensional representations are later learned by factorizing this high-order proximity matrix using random walk approximation techniques. We provide the following definition to describe the technical details of ANDRUSPEX in formal terms.

l-order proximity. Let $\mathbf{RW}_{\mathbf{d_0}} = (\mathbf{d}^0, \mathbf{m}_i^1, \mathbf{d}_j^1, \mathbf{m}_z^2, ..., \mathbf{d}_h^{l-1}, \mathbf{m}_k^l, ...)$ denote a random walk starting from a device vertex $\mathbf{d}_0$ on a PHA installation graph $\mathbf{G} = (\mathbf{D} \cup \mathbf{M}, \mathbf{E})$, where superscript denotes the rank of vertex occurrence in the random walk. l-order proximity between $\mathbf{d}_0 \in \mathbf{D}$ and $\mathbf{m}_k^l \in \mathbf{M}$ is defined by a decay function C (see Eq 1) of the rank of occurrence (i.e., $C(l)$) and the transition probability between them in the random walk. If there is no walkable path from $\mathbf{d}$ to $\mathbf{m}_k^l$, the l-order proximity is 0.

Example. Given a random walk $RW_{\mathbf{d}_1^0} = (\mathbf{d}_1^0, \mathbf{m}_5^1, \mathbf{d}_4^1, \mathbf{m}_3^2)$ as illustrated in ① Figure 2b, $\mathbf{m}_3$ is at 2-order proximity of $\mathbf{d}_1$. Similarly, given a random walk $RW_{\mathbf{d}_3^0} = (\mathbf{d}_3^0, \mathbf{m}_5^1, \mathbf{d}_4^1, \mathbf{m}_3^2, \mathbf{d}_2^3, \mathbf{m}_2^3)$ as illustrated in ② Figure 2b, $\mathbf{m}_2$ is at 3-order proximity of $\mathbf{d}_3$.

Figure 6 shows two l-order proximity matrices induced by random walks. Figure 6a shows an example of 1-order proximity matrix and Figure 6b shows a 3-order proximity matrix induced by random walks on the PHA installation graph (see Figure 2a). We can see that, with the increasing value of l, l-order proximity captures more high-order relationship information between devices and PHAs.

Given the l-order proximity matrix, formally, the objective function for learning low dimensional representations (i.e., Φ) is defined in Eq 1

$$\mathcal{L} = \sum_{\substack{1 \le l \le K \\ d_i, (m_j, m_{j'})}} C(l) \mathbb{E}_{\substack{m_j \sim P_{d_i}^l \\ m_{j'} \sim P_N}} \left[\mathcal{F}(\phi_{d_i}^T \phi_{m_{j'}}, \phi_{d_i}^T \phi_{m_j}) \right]$$
$$+ \lambda_\Phi \|\Phi\|_2^2 \tag{1}$$

where $C(l) = 1/l$ denotes the decay function, $P_{d_i}^l(\cdot)$ denotes the l-order probability distribution of a PHA m_j sampled from a random walk $\mathbf{RW}_{\mathbf{d_i}}$ (see Eq 2), P_N denotes a uniform distribution of all items from which a PHA i' was drawn, and $\mathcal{F}(\phi_{d_i}^T \phi_{m_{j'}}, \phi_{d_i}^T \phi_{m_j})$ is a ranking objective function discriminating the observed PHAs installations (i.e., $\phi_{d_i}^T \phi_{m_j}$) from unobserved PHA installations (i.e., $\phi_{d_i}^T \phi_{m_{j'}}$) in the low dimensional embedding space (see Eq 3). Eq 2 is formalized as:

Dataset	Date(s)	# Dev.	# PHA	V	E
DS_1	Mar 1 2019	644,823	63,650	708,473	844,531
DS_2	Mar 1-14 2019	1,272,505	99,464	1,371,969	1,675,952
DS_3	Mar 1-31 2019	1,864,021	131,903	1,995,924	2,509,744

TABLE 1: Summary of experimental datasets.

$$P_{v_x}^l(v_y) = \begin{cases} \frac{\mathbf{A}_{v_x,v_y} deg(v_y)}{\sum_{v_{y'}} \mathbf{A}_{v_x,v_{y'}} deg(v_{y'})} & l=1, v_x \in \mathbf{D} \\ \frac{\mathbf{A}_{v_y,v_x} deg(v_y)}{\sum_{v_{y'}} \mathbf{A}_{v_{y'},v_x} deg(v_{y'})} & l=1, v_x \in \mathbf{M} \\ p_{v_x}^1(v_\alpha) p_{v_\alpha}^{l-1}(v_\beta) p_{v_\beta}^1(v_y) & otherwise \end{cases} \tag{2}$$

approximates the probability of sampling a l-th neighbor v_y for v_x given a random walk $(v_x, v_\alpha, ..., v_\beta, v_y)$, where v_α denotes the vertex after visiting v_x and v_β denotes the vertex before v_y. In this way, it simplifies the cumulative process of counting all intermediate vertices from all possible random walks from v_x to v_y in a recursive manner. Note that if $v_x \in \mathbf{D}$, then $v_\alpha \in \mathbf{M}$. Otherwise, if $v_x \in \mathbf{M}$, then $v_\alpha \in \mathbf{D}$. For example, given a random walk $RW_{\mathbf{d}_1^0} = (\mathbf{d}_1^0, \mathbf{m}_5^1, \mathbf{d}_4^1, \mathbf{m}_3^2)$ as illustrated in ① Figure 2b, v_x is $\mathbf{d}_1^0$, v_α is $\mathbf{m}_5$, v_β is $\mathbf{d}_4$ and v_y is $\mathbf{m}_3^2$).

We define the ranking objective function $\mathcal{F}(\cdot)$ as follows:

$$\mathcal{F}(\phi_{d_i}^T \phi_{m_{j'}}, \phi_{d_i}^T \phi_{m_j}) = \mathbb{1}_{(\delta > \frac{\epsilon}{k})} log(\delta) \tag{3}$$

where $\delta = \phi_{d_i}^T \phi_{m_{j'}} - \phi_{d_i}^T \phi_{m_j}$ and $\mathbb{1}_\zeta : \zeta \to \{0,1\}$ denotes an indicator function for condition ζ, where $\zeta = \delta > \frac{\epsilon}{k}$. we employ a random walk approximation technique [15] to approximate the matrix factorization results. Eq 1 is accordingly minimized using asynchronous stochastic gradient descent [61] in the paper.

4. Dataset

To evaluate ANDRUSPEX's performance, we focus on Android PHA detection data collected by a major security company's mobile security product. This company offers end users to explicitly opt in to its data sharing program during the setup process when the app is run for the first time to help improving its detection capabilities. The end users are shown a dialog about the purpose of the telemetry collection in the license agreement, and how the global privacy policy of the security company safeguards the data. The license agreement specifies that the telemetry is "processed for the purposes of delivering the product by alerting the User to potentially malicious applications as well as improving the app security feature" and is "kept in an encrypted pseudonymized form." To preserve the anonymity of users and their devices, client identifiers are anonymized and it is not possible to link the collected data back to the users and the mobile devices that originated it. The mobile security app only collects detection metadata (e.g., anonymized device identifier, timestamp, file SHA2, etc.), and it cannot inspect network traffic data, hence the company does not collect any actual communication/user data, or other types of PII. The telemetry used in this paper does not contains any personally identifiable information (PII). The product runs periodically scanning the apps that the user installed and records meta information associated with PHAs that are detected based on pre-defined signatures or whose behavior violates the pre-defined policies.

From this dataset, we extract the following information: anonymized device identifier, timestamp, file SHA2, PHA package name, and other detection information (e.g., installation types, installer packages).

We collect 31 days of data over March 2019 to thoroughly investigate the prediction effectiveness, stability, and resilience of a real world deployment of ANDRUSPEX. On average, we collect 2.4 million raw detection events from 1.8 million end users (the majority of the users run Android 6.0 Marshmallow or above) located in over 200 countries and territories per day. From these 31 days of raw data, we build three datasets summarized in Table 1. DS_1 is a single day data using raw data collected on March 1 2019. DS_2 and DS_3 are one week data (between March 1 and March 7 2019) and one month data (between March 1 and March 31 2019). We use all three datasets to compare ANDRUSPEX's performance to those of the baseline methods (see Section 5.2). We later focus on DS_2 and DS_3 and use them to evaluate ANDRUSPEX's prediction and runtime performance given different real world scenarios.

Reproducibility. Note that ANDRUSPEX can operate on fully anonymized data (e.g., device identifiers and PHA packages names can be replaced by other unique identification schemes) to protect the end users' privacy. It is straightforward for the researchers to reproduce ANDRUSPEX's results by generating bipartite networks matching the characteristics stated in Table 1 and Figure 3a.

Data Limitations. It is important to note that the PHA detection data is collected passively. That is, a PHA detection event is recorded when the product detects a potentially harmful application that matches a pre-defined signature including its behavior, communications, and policy violations. Any events pre-emptively blocked by other security products (e.g., application store link blacklists, cloud-based app reputation system) cannot be observed. Additionally, any PHAs that did not match the predefined signatures are also not observed. The prediction model used in this paper can, therefore, predict the PHA observed by the mobile security product from this security company. We discuss more details on the limitations underlying the data used by ANDRUSPEX in Section 7.

5. Evaluation

In this section, we describe the experiments used to evaluate ANDRUSPEX. We designed several experiments that allow us to answer the following research questions:

RQ1: What is ANDRUSPEX's performance in predicting the upcoming PHA installation events and how does its performance compare to baseline and state-of-the-art methods?

RQ2: Can we interpret and explain ANDRUSPEX's prediction performance?

RQ3: How does ANDRUSPEX cope with data latency issues that arise in real world deployment? In other words, does ANDRUSPEX still work if mobile devices report data about the infections that they encountered with a significant delay?

RQ4: What is ANDRUSPEX's prediction performance when we need to retrain the model periodically?

RQ5: Is ANDRUSPEX's runtime performance acceptable for a real world deployment?

5.1. Experimental Setup

Implementation. We implement ANDRUSPEX's graph representation learning component in C++ using the OpenMP [17] library, the PHA graph reconstruction component using Python 3.7.2 and the NetworkX library, and employ the Random Forest [12] implementation from Sklearn as the prediction engine. In terms of baseline implementations (see Section 5.2), preferential attachment is implemented in Python 3.7.2 and the rest are implemented in C++. All experiments are carried out on a server with dual 2.67GHz Xeon CPU X5650 and 256GB of memory.

Training/test data split. For DS_1, all methods are trained on the first 18 hours data (00:00 - 18:00) and tested on the following 6 hours data (18:00 - 24:00). For DS_2, all methods are trained on the first 6 days of data (March 1 - March 6 2019) and tested on the 7th day of data (i.e., March 7 2019). For DS_3, all methods are trained on the first 3 weeks of data (March 1 - March 24 2019) and tested on the 4th week of data (i.e., March 25 - 31 2019). In this setup, we deliberately eliminate the temporal bias that may be introduced using traditional training/test splits [54]. However, different training and test ratios are therefore introduced due to the fact that PHA installations are not evenly distributed. Details of the training and test data are summarized in Table 2.

Prediction formulation. We formulate our prediction task as a binary prediction problem for edges in the network where two classes are considered: positive (i.e., observed edges) and negative (i.e., absence of edges). For both training and test data, we sample an equal number of non-existing edges [54] from the observed PHA installation edges to form the negative class. In this way, we are able to generalize ANDRUSPEX's performance by evaluating its performance on a set of data not used for training, which is assumed to approximate the typical unseen data that the model will encounter in the wild, reducing the chance of overfitting.

Choosing the prediction model. A plethora of prediction models for supervised learning exist. Usually, some work better than others for a specific dataset or domain. In this paper, we experimented with several different prediction models, and select Random Forest [12] as the prediction engine due to its scalability, bias-variance trade-off, and overall good performance given all methods. We use 10-fold cross validation for all methods. We also provide a detailed discussion in Section 7 and examine how different models may impact the prediction results.

Hyperparameters. We use grid search on DS_1 to identify the best parameters. Experimentally, we set the embedding size d to 128, the high-order proximity threshold l to 4 (effectively, the random walk was truncated at length 6), and negative sampling times [15] to 50. For all baselines, we use the best parameters stated in the respective papers. We also set the number of trees in the random forest algorithm to 20. Note that hyperparameters optimization remains an active research area in the machine learning community. We refer the reader to [7] for an overview.

Evaluation metrics. We use the true positive rate (TPR) [58], the false positive rate (FPR) [58], the area

Dataset	Training					Test				
	Period	Ratio	# Events	# Dev	# Apps	Period	Ratio	# Events	# Devs	# Apps
DS_1	00:00 - 18:00 (Mar. 1)	0.73	844,531	644,823	63,650	18:00 - 24:00 (Mar. 1)	0.27	317,474	189,327	26,083
DS_2	March 1 - 6	0.86	2,050,865	1,272,505	99,464	March 7	0.14	334,383	237,594	32,961
DS_3	March 1 - 24	0.84	3,194,838	1,864,021	131,903	March 25 - 31	0.16	599,458	404,417	47,099

TABLE 2: Training/test data split. We deliberately eliminate the temporal bias [54] that may be introduced using traditional training/test split.

under the ROC curve (AUC) score [28], and the average precision (AP) [34], [44] to extensively evaluate AN-DRUSPEX's performance. TP, TN, FP, and FN in the rest of the section denote the number of true positives, true negatives, false positives, and false negatives, respectively. **Thresholds.** Note that predicting the exact PHA installations for end users is a challenging task in the real world. A higher false positive rate leads to worse user experience hence higher customer churn rate. For example, a 0.01 FPR for 317,474 events (see DS_1, Table 2) may have an undesirable impact on approximately 3,174 end users in a six hour window which is not acceptable in a real world deployment. We therefore fix the FPR at three scales, respectively 0.0001, 0.001 and 0.005 from real world usability perspective.

5.2. Comparison Study (RQ1)

Experimental setup. We use all three datasets (see Table 1) in this section to comparatively study prediction performance of baseline methods and ANDRUSPEX. Training and test data are split following the setup specified in Section 5.1.

Baselines. In this section we aim at understanding whether ANDRUSPEX is required for the task of predicting PHA installations, or whether existing baseline methods are sufficient for the task at hand. For comparison purposes, we implemented the following four baseline methods.

Preferential Attachment. Preferential attachment [5], [50] is a popular network growth model, in which vertices accumulate new edges in proportion to the number of the edges they already have. The rationale of using preferential attachment as a baseline is to verify if popular PHAs dominate the future PHA installations. In this paper, we use PA_score$(v_i, v_j) = \frac{|\mathcal{N}(v_i)| \times |\mathcal{N}(v_j)|}{2 \times |\mathcal{E}|}$ to compute the normalized preferential attachment score of all vertex pairs.

First-order and Second-order proximity. We use LINE [74] as the first-order and second-order proximity baseline. This method directly generates vertex embeddings by taking explicit relationships (first-order proximity) and indirect relationships (second-order proximity) into consideration. It is able to partially preserve the local and global structure of the graph by conducting edge sampling. The rationale of using LINE is to verify if a limited number of proximity can already offer good prediction results.

High-order proximity via matrix factorization. Chen *et al.* [14] provided a collaborative similarity embedding (CSE) framework to model high-order proximity by capturing the similarities of two vertex sets respectively (i.e., device-device and PHA-PHA similarities in this paper), and integrate them into a single objective function through shared embedding vectors. The rationale of using CSE as a baseline is to understand if high-order proximity matrix

factorization itself can offer superior prediction results for the real world deployment.

Excluded methods. We also experimented with BiNE [24], an embedding model specialized for bipartite graphs and Jodie [37], a coupled recurrent neural network based embedding model. However, BiNE does not finish the computation for DS_1 after 72 hours while Jodie failed to train due to the out-of-memory issue in a TITAN V 12GB graphics card. Therefore, we exclude them from the comparison study for practicality reasons.

Experimental Results. Table 3 shows the prediction performance of ANDRUSPEX compared to the baseline systems. As it can be seen, ANDRUSPEX outperforms the baseline methods in all three datasets regarding all evaluation metrics. We can also note that the high-order proximity method (i.e., CSE) performs better than first/second-order proximity methods, and first/second-order proximity methods perform better than preferential attachment. These results demonstrate that higher proximity between devices and PHAs is crucial to capture the implicit relations and improve prediction results. However, the high-order proximity baseline is less effective than ANDRUSPEX since it does not consider the diminishing coefficient with the increasing order (see Section 3.3). In general, ANDRUSPEX reports a TPR above 0.991 for a FPR of 0.0001 in all three datasets (e.g., given DS_1 and 0.001 FPR, ANDRUSPEX is able to correctly predict 314,616 out 317,474 PHA installation events with 2,857 FNs), while the high-order proximity baseline only reaches a TPR of 0.893 in the best case (i.e., given DS_2) for a FPR of 0.0001, representing a substantial 10% TPR difference comparing to ANDRUSPEX (i.e., high-order proximity leads to 283,504 TPs with 35,779 FNs which represent 10 times more FNs and 31,112 less TPs comparing to ANDRUSPEX). For different FPR values (0.0001 - 0.005), ANDRUSPEX consistently reports TPRs ranging from 0.991 to 0.998 for all three datasets. The high-order proximity baseline gets close to ANDRUSPEX's prediction performance given a FPR of 0.005. Yet, this represents a 50 times increase in FPR comparing to the optimal operational FPR of 0.0001. In addition, we can observe that the preferential attachment baseline performs the worst. This supports our hypothesis that popular PHAs are not the dominant factor when predicting future installations, but rather that rare PHAs also need to be factored in for effective predictions. Figure 7 shows the ROC curve of ANDRUSPEX compared to the baseline systems. As it can be seen, ANDRUSPEX outperforms the baseline methods in all three datasets, which further confirms ANDRUSPEX's operational capability in the real world deployment (see Section 5.6 for ANDRUSPEX's runtime performance).

Method	DS_1					DS_2					DS_3				
	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP
Pref. Attach.	0.072	0.268	0.512	0.977	0.974	0.099	0.310	0.593	0.980	0.978	0.094	0.338	0.584	0.981	0.980
1st-order prox.	0.782	0.898	0.936	0.983	0.986	0.837	0.927	0.950	0.982	0.986	0.844	0.927	0.965	0.990	0.990
2nd-order prox.	0.863	0.922	0.959	0.992	0.993	0.867	0.918	0.953	0.993	0.993	0.868	0.941	0.966	0.993	0.994
high-order prox.	0.873	0.969	0.985	0.997	0.997	0.893	0.957	0.977	0.996	0.996	0.879	0.951	0.978	0.995	0.996
ANDRUSPEX	**0.991**	**0.996**	**0.998**	**0.999**	**0.999**	**0.994**	**0.997**	**0.998**	**0.999**	**0.999**	**0.992**	**0.996**	**0.997**	**0.999**	**0.999**

TABLE 3: Prediction performance comparison study: ANDRUSPEX vs. baseline approaches.

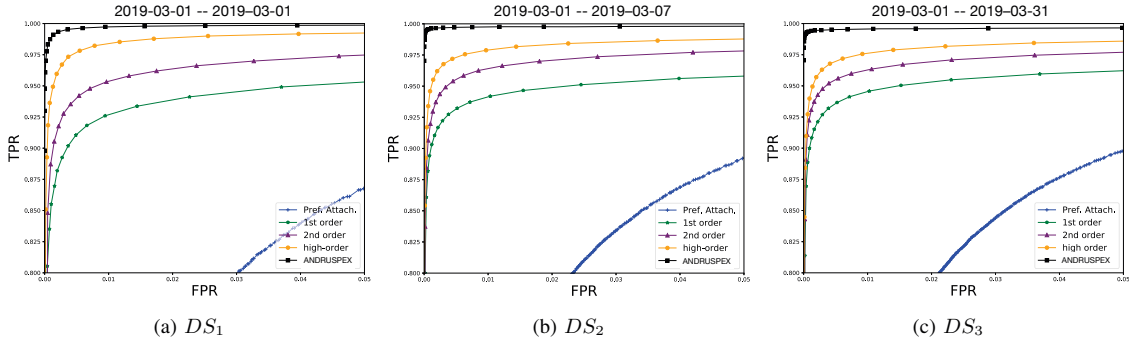(a) DS_1 (b) DS_2 (c) DS_3

Figure 7: ROC curves comparison: ANDRUSPEX vs. baseline approaches.

5.3. Interpretation of ANDRUSPEX Prediction Performance (RQ2)

In this section, we aim at understanding why AN-DRUSPEX performs better than the baseline methods (e.g., ruling out the potential overfitting, etc). To this end, we leverage LargeVis [73], an efficient high-dimensional data visualization system, to project and visualize both positive (i.e., observed) and negative (i.e., non-existent) PHA installation events (i.e., concatenated $2d$-dimensional vectors) into a 2-dimensional space. Such visualization can facilitate the straightforward description, exploration, and sense-making of the original data projected in the latent space by ANDRUSPEX (see Section 3). Our hypothesis is that, since we formulate the prediction task as a binary classification problem (i.e., a PHA installation will be observed or not), ANDRUSPEX's low dimensional representations should be able to separate these two categories of events (i.e., positive events should be apart from negative events) in the latent space.

The LargeVis visualization results from all methods (see Section 5.2) are shown in Figure 8. For illustration purposes, we show the baseline results using DS_1 and ANDRUSPEX's results using all three datasets. Note that preferential attachment does not produce embeddings hence is excluded from the figure. As we can see, AN-DRUSPEX can well separate positive (in yellow) from negative (in blue) PHA installation events (see Figure 8d, Figure 8e and Figure 8f) in the latent space without relying upon the prediction models (see Section 7). At the same time, the high-order proximity method (Figure 8c) can separate the two categories of events better than first/second-order proximity methods do (Figure 8a and 8b). This is consistent with the results obtained in the comparison study (see Section 5.2). Our visualization results demonstrate that ANDRUSPEX can offer a better PHA installation event representation in the latent space, providing us with a good explanation of why ANDRUS-PEX performs better than the baseline methods as shown

Dataset	Training ratio	Data latency ratio	Test ratio	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP
DS_2	0.86	0.00	0.14	0.994	0.997	0.998	0.9994	0.9995
	0.79	0.07	0.14	0.994	0.997	0.998	0.9994	0.9995
	0.70	0.16	0.14	0.993	0.997	0.998	0.9994	0.9995
	0.61	0.25	0.14	0.991	0.994	0.997	0.996	0.995
DS_3	0.839	0.00	0.161	0.992	0.995	0.997	0.9994	0.9995
	0.769	0.07	0.161	0.992	0.995	0.997	0.9992	0.9994
	0.679	0.16	0.161	0.991	0.994	0.995	0.9992	0.994
	0.589	0.25	0.161	0.990	0.992	0.994	0.996	0.997

TABLE 4: ANDRUSPEX's resilience to data latency.

in Section 5.2. In Section 7 we also show that the installation event representations obtained by ANDRUSPEX work well with different prediction models.

5.4. Resilience to Data Latency (RQ3)

The mobile security product from the major security company performs its detection periodically, and usually sends detection data to the backend when the devices are connected to a wireless network in order to minimize mobile data usage. From the data perspective, we observe the phenomenon that a device's detection data is less frequent and may be delayed for several hours or a couple of days. In light of this constraint, we expect ANDRUSPEX to cope with this limitation and remain operational. In this section, we look at how ANDRUSPEX performs when facing such data latency issue in a real world deployment.

Experimental setup. We use DS_2 and DS_3 (see Table 1) to study ANDRUSPEX's prediction performance when dealing with the real world data latency issue. We randomly remove 7%, 16% and 25% PHA installation events from the training data to simulate the data latency. Training, data latency and test ratio of these two datasets are summarized in Table 4. We use the same parameters and the evaluation metrics laid out in Section 5.1. Note that we fix the test data ratio so that the prediction results with the simulated data latency are comparable to those without (see Table 3).

Experimental results. Table 4 summarizes the results obtained using the aforementioned experimental setup.

(a) 1st-order proximity   (b) 2nd-order proximity   (c) high-order proximity

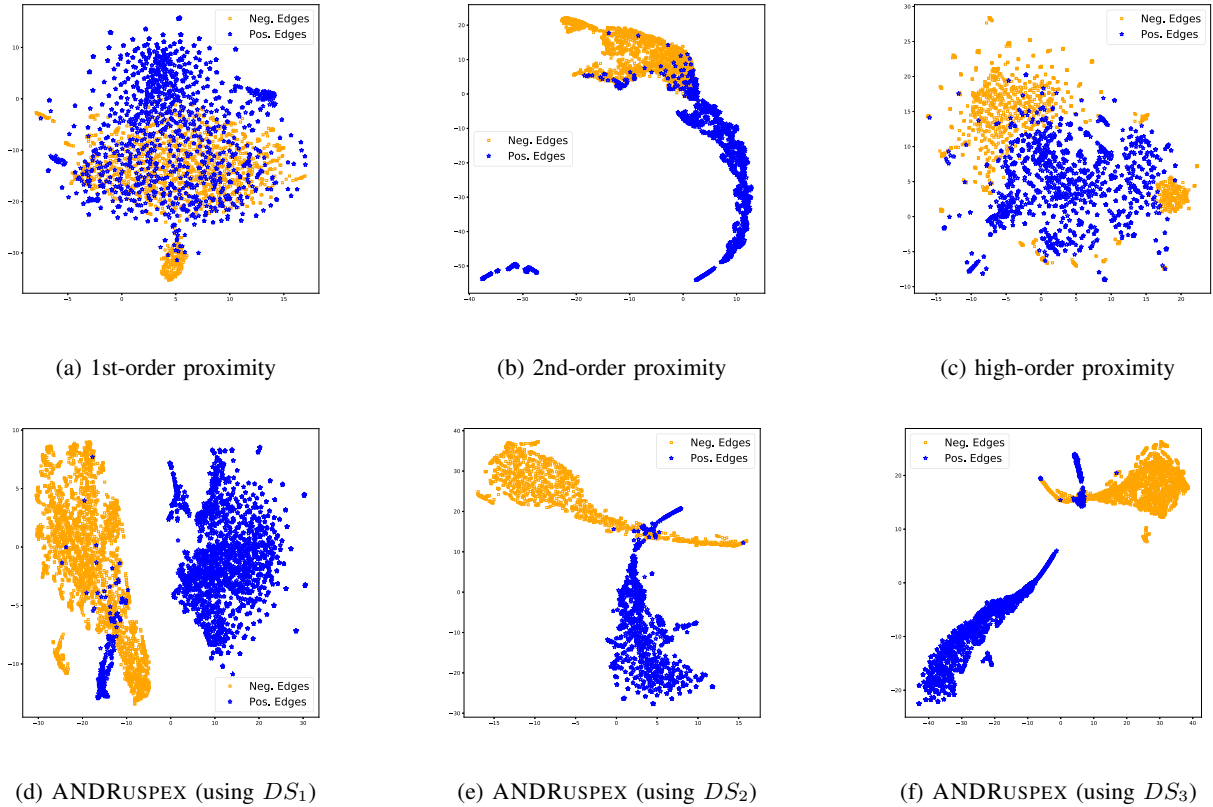(d) ANDRUSPEX (using DS_1)   (e) ANDRUSPEX (using DS_2)   (f) ANDRUSPEX (using DS_3)

Figure 8: LargeVis projections of positive (i.e., observed) and negative (i.e., non-existent) PHA installation events (i.e., concatenated $2d$-dimensional vectors) from all methods (see Section 5.2). The visualization results demonstrate that ANDRUSPEX can offer a good PHA installation event representation in the latent space. Note that preferential attachment does not produce embeddings hence is excluded from the visualization. This figure is better viewed in color.

For clarification purposes, we also include the prediction results from Table 4 in Section 5.2. Overall, ANDRUSPEX shows strong resilience to data latency. For example, AN-DRUSPEX's prediction performance is not affected by any of the evaluation metrics when data latency ratio is 7%. When data latency ratio increases to 16%, ANDRUSPEX's prediction drops by 0.001 - 0.002 TPR given various FPR thresholds. Even when the data latency ratio reaches 25%, ANDRUSPEX can still offer high accuracy with a small decrease of AUC and AP. For example, given 0.0001 FPR, ANDRUSPEX can reach 0.991 and 0.990 TPR in predicting PHA installations respectively for DS_2 and DS_3, dropping by 0.003 and 0.002 TPR respectively to those from full training data. These results confirm that the model can cope well with the data latency issue and predict PHA installations with high accuracy in a real world deployment. In Section 7 we provide a detailed discussion on how ANDRUSPEX's resilience to data latency can deal with the potential adversarial attacks [9].

5.5. Reliability over Time (RQ4)

In this section, we evaluate ANDRUSPEX's prediction accuracy when the system is deployed in the real world, where periodical retraining is required. Our goal is to evaluate the reliability of the model's prediction performance over time.
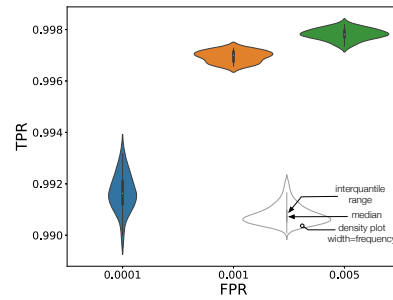

Figure 9: Violin plot [29] for ANDRUSPEX's reliability study using a 7-day rolling window over DS_3. The white dot in the middle is the median value, the thick black bar in the centre represents the interquartile range and the contour represents distribution shape of the data.

Experimental setup. We use DS_3 (see Table 1) in this section to study prediction performance of ANDRUSPEX when periodical retraining is required. Training and test data are split in a rolling one-week window following the setup specified in Section 5.1. That is, we use the first 6 days in the window to form our training data and predict the PHA installations on the 7th day. This setup reflects the rapidly changing mobile threat landscape whereas new

Dataset	Graph Construction (seconds)	ANDRUSPEX runtime (seconds)	Pred. Eng. training time (seconds)
DS_1	230.06	557.12	598.88
DS_2	778.27	702.15	1,311.65
DS_3	3,424.86	791.33	1,986.21

TABLE 5: ANDRUSPEX runtime performance

PHAs emerge daily. There are in total 24 evaluation points in this setup. We reuse the same parameters and the evaluation metrics as stated in Section 5.1.

Experimental results. We use violin plots to visualize the distribution of TPRs and its probability density per FPR threshold. Figure 9 shows the results obtained using the aforementioned experimental setup. In this setup where retraining is required frequently, ANDRUSPEX's prediction performance remains steady. For example, given a 0.0001 FPR, ANDRUSPEX shows a 0.9916 TPR on average with a 0.0006 standard deviation. Given 0.001 and 0.005 FPRs, ANDRUSPEX shows a 0.997 TPR on average with a 0.0002 standard deviation and a 0.998 TPR with a 0.0002 standard deviation respectively. These results demonstrate that ANDRUSPEX can work well when frequent retraining is required, and underpins its real world practicability.

5.6. Runtime Performance (RQ5)

Following our prediction performance analysis in the previous sections, we evaluate ANDRUSPEX's runtime performance to answer several key practical questions such as 1) *how long does* ANDRUSPEX *take to construct a PHA installation graph*, 2) *how long does* ANDRUSPEX *take to build low dimensional representations of PHA installations* and 3) *how long does* ANDRUSPEX *take to train the prediction engine*. To answer these questions, we use all three datasets (see Table 1). ANDRUSPEX's runtime performance is summarized in Table 5.

As we can see in Table 5, all three components (see Figure 5 in Section 3) of ANDRUSPEX scale linearly with regards to the input data size. In terms of graph reconstruction time, ANDRUSPEX takes 230.06 seconds to reconstruct the PHA installation graph given DS_1. Respectively, ANDRUSPEX takes 778.27 seconds given DS_2 (i.e., one week data) and 3424.86 seconds given DS_3 (i.e., one month data). This represents 3.5 and 14.9 times increase of time given roughly 6.7 and 29.2 times more raw input data. In terms of generating low dimensional edge representations, ANDRUSPEX takes 598.88 seconds to build the edge representations for DS_1. Respectively, ANDRUSPEX takes 702.15 and 791.33 seconds to build corresponding edge representations for DS_2 and DS_3. This represents 1.17 and 1.56 times increase of time given roughly 1.98 and 2.97 times increase of input edge sizes (see Table 1). In terms of training the prediction engine, ANDRUSPEX also shows similar patterns. This implies that the end-to-end operation of ANDRUSPEX takes approximately 46.5 minutes for a typical usage scenario (i.e., prediction for one week data using DS_2, see Section 5.5). In summary, ANDRUSPEX's empirical linear scalability exemplifies its computational advantage and practical deployment in the wild.
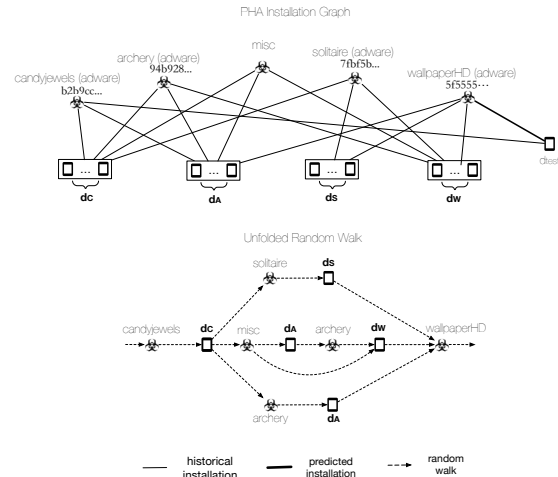

Figure 10: Case study: ANDRUSPEX predicts adware installation

6. Case Studies

In this section, we provide two interesting case studies that we encountered while operating ANDRUSPEX. They demonstrate how ANDRUSPEX could be used as an early warning tool, and how it may explain the implicit connections between PHAs.

6.1. Adware Installation Prediction

In this section, we carry out a detailed adware installation prediction case study. Figure 10 shows part of a PHA installation graph built by ANDRUSPEX on March 1, 2019. Vertices connected by thin solid lines are observed PHA installations and the thick solid line represents the prediction by ANDRUSPEX. There are four repackaged adware apps that aggressively use the MoPub library and consequently are not available from the Google Play store. Respectively they are candyjewels with SHA2 B2B9CC..., archery with SHA2 94B928..., solitaire with SHA2 7FBF5B... and wallpaperHD with SHA2 5F5555.... During the training, there are a number of random walks with various numbers of hits leading from candyjewels to wallpaperHD. The reason for this is likely that those apps were available on the same third party marketplace and suggested to populations of users with similar app installation history. We only provide a snippet of the unfolded random walks (dash lines) in Figure 10 for clarification purposes. These random walks exhibit various l-order proximities (see Section 3), and the accumulation of these walks enables ANDRUSPEX to capture the implicit relationship between candyjewels and wallpaperHD. At the test time, once a device d_{test} installs candyjewels, ANDRUSPEX is able to predict that d_{test} may install wallpaperHD with high confidence based upon the evidential information collected from these random walks. Following up with the prediction, ANDRUSPEX alerts the end user to be vigilant when installing wallpaperHD app by displaying a summarized description of the unfolded random walks. The end
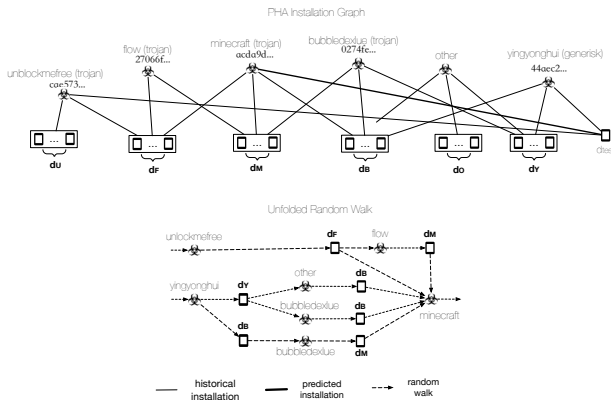
Figure 11: Case study: ANDRUSPEX predicts trojan installation

user can reach an informed decision to not install a PHA which might weaken Android's built-in security.

Additionally, we can observe that the unfolded random walks shown in Figure 10 reveal some interesting patterns that are not available in the detection data. For example, adware may explore a user's preference (e.g., `candyjewels`, `archery`, and `solitaire` are all games), and lead the users to install additional adware (e.g., `wallpaperHD`) which is favored by certain demographics group playing these games. ANDRUSPEX is able to capture these implicit patterns without collecting additional data and could warn targeted users ahead of time.

6.2. Trojan Installation Prediction

In this section, we carry out a detailed trojan installation prediction case study. Figure 11 shows part of a PHA installation graph built by ANDRUS-PEX on March 1, 2019. Thin solid lines represent observed PHA installations and the thick solid line represents the prediction by ANDRUSPEX. There are four trojans repackaged from popular games, respectively `unlockmefree` with SHA2 `CAE573...`, `flow` with SHA2 `27066F...`, `minecraft` with SHA2 `ACDA9D...` and `bubbledexlue` with SHA2 `0274FE...` from third party marketplaces. In addition, we have a third party marketplace application `yingyonghui` with SHA2 `44AEC2....` Note that `yingyonghui` functions as a gateway to applications, however it is itself associated with information leakage risk since it collects both network and SMS data without explicit notification to/consent from the users. The other applications with limited hits during the random surfing are classified as 'misc'. We also provide a snippet of the unfolded random walks (dash lines) in Figure 11 for illustration purpose as we did in Section 6.1. As we can see in Figure 11, there are multiple sources from different application categories (e.g., `unlockmefree` is a game and `yingyonghui` is a market application) that may lead to the installation of trojan application `minecraft`. For example, the trojan (e.g., `minecraft`) makers can target third party marketplaces (e.g., `yingyonghui`) which don't enforce code review, and leverage a user's inclinations to install this popular application for free.

Additionally, the trojan makers can target specific application categories (e.g., game), and tactically repackage popular titles (e.g., `bubbledexlue`, `unlockmefree` etc.) and spread additional trojans. ANDRUSPEX is able to capture these different strategies by leveraging the collective PHA installation graph. At the test time, if a device d_{test} installs `unlockmefree` and `yingyonghui`, ANDRUSPEX can predict the impending installation of the malicious version of `minecraft` with high confidence.

7. Discussion

Influence of different prediction models. In Section 5.3 we demonstrate that ANDRUSPEX can well separate positive (i.e., observed) from negative (i.e., non-existent) PHA installation events in the latent space without relying on the prediction model (see Section 5.1). For discussion purposes, we also illustrate how different prediction models may impact evaluation results. To this end, we select logistic regression [30] and gradient boosting [49], respectively a linear and an ensemble model, to demonstrate the impact of alternative prediction models. As we can see in Table 6, ANDRUSPEX is less affected by different models and attains reasonable prediction capability in both cases. For example, ANDRUSPEX is able to offer useful prediction performance (e.g., 0.900 TPR and 0.917 TPR given 0.0001 FPR respectively using logistic regression and gradient boosting machine). In contrast, all the baseline methods have shown substantial drops in terms of prediction accuracy. This gives us additional confidence that ANDRUSPEX can offer a better PHA installation event representation and consequent prediction in the latent space.

Influence of different edge representations. The core of ANDRUSPEX is a graph representation learning algorithm that learns the low dimensional vertex representations in the latent space. Other than the concatenation used in this paper, there are alternative methods to combine two vertex representations to form a PHA installation event representation. To this end, we used four other methods, respectively *average*, *Hadamard*, *weighted L1* and *weighted L2*, to combine two vertex representations. We train ANDRUSPEX using these edge-wise representations obtained with the methods shown in Table 7. As we can see in Table 7, all four alternative PHA installation event representation methods do not offer comparable prediction performance. ANDRUSPEX with the concatenation of vectors is consistently the best edge representation.

Evasion. ANDRUSPEX may be subject to evasion by attackers. At its core, ANDRUSPEX relies on graph representation learning to embed both device and PHA vertices into a latent space. Therefore, it may be subject to adversarial attacks [9] from an adversary who might influence the representation learning by adding/removing edges from the PHA installation graph. In terms of adding edges to influence prediction results, one technique that could be used by the adversaries is leveraging the strategy used by mimicry attacks, i.e., evading prediction by injecting many irrelevant applications into alternative markets and ensure the users to install them to cover the prediction of a real trojan application. We argue

Method	Logistic Regression					XGBOOST				
	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	ROC AUC	AP
1st-order prox.	0.151	0.172	0.188	0.446	0.551	0.500	0.552	0.603	0.894	0.914
2nd-order prox.	0.722	0.821	0.869	0.923	0.959	0.633	0.768	0.852	0.981	0.982
high-order prox.	0.782	0.832	0.875	0.96	0.97	0.684	0.850	0.901	0.991	0.993
ANDRUSPEX	**0.900**	**0.903**	**0.98**	**0.97**	**0.959**	**0.917**	**0.947**	**0.982**	**0.999**	**0.999**

TABLE 6: ANDRUSPEX's prediction performance (using DS_1) with alternative prediction models, respectively logistic regression and gradient boosting.

method	operation	TPR @ 0.0001	TPR @ 0.001	TPR @ 0.005	AUC	AP
Average	$(a+b)/2$	0.045	0.191	0.402	0.926	0.930
Hadamard	$(a_1 * b_1, ..., a_i * b_i)$	0.056	0.159	0.347	0.921	0.922
weighted L1	$(|a_1 - b_1|, ..., |a_i - b_i|)$	0.053	0.187	0.347	0.896	0.903
weighted L2	$((a_1 - b_1)^2, ..., (a_i - b_i)^2)$	0.052	0.184	0.347	0.895	0.903
concat	$[a_1, ..., a_i, b_1, ..., b_i]$	**0.991**	**0.996**	**0.998**	**0.999**	**0.999**

TABLE 7: ANDRUSPEX's prediction performance (using DS_1) with alternative edge representations.

that monetization is a key factor for mobile PHA makers. If they could install irrelevant applications in mobile devices, they would easily monetize from these applications. In other words, it would not make sense for the adversaries to distribute irrelevant applications since it would make PHA delivery much more expensive to prevent ANDRUSPEX from correctly predicting future installations. In terms of removing edges to influence prediction results, it is infeasible from a practicality perspective. Once a PHA installation is observed by the security product from this security company, the adversaries cannot remove the edges (i.e., install events) from the PHA installation graph. One possible strategy can be leveraged by the adversaries is to prevent the security product from sending back the telemetry data. We already show in Section 5.4 that ANDRUSPEX can well cope with data latency and predict PHA installations with high accuracy in the real world deployment. Additionally, an adversary may inject multiple similar PHAs slightly differing from each other. The PHAs would target a similar audience and their installation would be predicted with similar probabilities. From a performance perspective, such strategy would reduce ANDRUSPEX's TPR since the users will not install all the very similar PHAs. From a practical perspective, the users will be wrongly notified about a PHA not fitting their intention. This could consequently lead to an increased "customer churn rate." One plausible approach to mitigate this evasion attack is using in-app messages to ask the end user to be vigilant of the markets and the developers where they may install apps from (e.g., using official markets). The rationale of using in-app messages is to explain to the end user the potential risk associated with the known on-device PHAs. More importantly, the end users would have contextual information (e.g., which PHAs in what category) that may be more effective to raise their awareness as observed by [46]. This way, the end user would benefit from an informative explanation rather than a short notification via the status bar. Consequently, the device's owner can still be warned about the threat and reach an informed decision to not install a PHA from that category which might weaken their device security.

Bias. Our dataset is biased towards the end users of a single mobile security product, and therefore still presents

some biases. However, the distribution of devices used in this study is not heavily skewed towards any specific region. It is also challenging to ascertain the coverage of PHAs covered by our study since it is infeasible to determine the total number of all PHAs. ANDRUSPEX, however, is retrained frequently to accommodate newly discovered PHAs. That is, ANDRUSPEX accumulates its knowledge of known PHAs when it is retrained. For instance, if a PHA was not initially included to train ANDRUSPEX by the security company, this PHA will later be used to retrain ANDRUSPEX once its signature and associated detection events are made available to the security company. As such, we believe the bias initially introduced by ANDRUSPEX will diminish over time.

Limitations. As any prediction system, ANDRUSPEX has some limitations. The main limitation of ANDRUSPEX is that it can only predict PHAs that are already known to be malicious. This makes sense in our application case, since our goal is to solve the current limitations of the Android security model, which allow PHAs to act undetected on their victim devices until the user decides to uninstall them. One way was to reduce this limitation is to retrain ANDRUSPEX frequently, as new PHAs become known. As we showed in Section 5.5, ANDRUSPEX can work well on different observation window under frequent retraining, underpinning its real world practicability. We argue that the advantage of proactive moderation compared to existing Android anti-malware solutions significantly overcomes these limitations.

8. Related Work

Predictive security. Prior work studied the feasibility of forecasting future security incidents using observed historical information [71]. The core idea of these research work is to extract a set of pre-defined features from historical data, and train a machine learning model to predict, in a binary format, if a vulnerability is likely to be exploited [11], if a PoC of a vulnerability would be devised in the real world [64], if an enterprise would be breached using publicly available security incident datasets [43], the volume of actual malware infections in a country [33], the likelihood of endpoints at risk of infection in the future [8], etc. In recent years, researchers also leveraged deep learning techniques such as RNN [66] and CNN [65] to predict the *exact* upcoming security

events. Our approach focuses on learning low dimensional representations of global malware installation graph and predict emerging malware installations.

Applications of representation learning in security research. The core idea of these research work is to captures the posterior distribution of the underlying explanatory factors for the observed input in latent space. Lin *et al.* [41], [42] propose a function representation learning method to obtain the high-level and generalizable function representations from the abstract syntax tree (AST). Rhodeb *et al.* [62] use RNNs to learn the representations of their behavior profiles. Ding *et al.* [19] represents an assembly function as a control flow graph (CFG) and leveraging a customized PV-DM model [39] to learn the latent representations of assembly functions. Shen *et al.* [67] propose ATTACK2VEC model to understand the evolution of cyberattacks in latent space.

Graph-based malware detection. The core idea of graph-based malware detection is modeling the interactions among malware, endpoints and network servers as graphs, and leverage various machine learning models to understand the patterns and detect previous unknown malicious files or activities. For example, CAMP [60], Mastino [59], and Polonium [48] built graphs from binary activity data and detect malware. Similarly, Marmite [69], NAZCA [32], AESOP [72] and Kwon *et al.* [38] built graphs from binary download/distribution data and detect previous unknown malware. Different from these approaches, ANDRUSPEX predicts PHA installations in the future instead of detecting them.

Closest work. The closest work to this paper is highly predictive blacklisting (HPB) [84]. Essentially, HPB builds a victim correlation graph using Jaccard similarity over occurrences of overlapping attackers among victims. However, HPB cannot scale to large dataset due to its quadratic time complexity incurred by Jaccard similarity. From a theoretical perspective, HPB is equivalent to weighted one mode projection of a bipartite graph [85] which only considers explicit attack-victim relationship. In contrast to HPB, we demonstrate that ANDRUSPEX has an empirical linear scalability coping with millions of vertices, and considers both explicit and implicit relationships offering accurate PHA installation predictions.

9. Conclusion

We presented ANDRUSPEX, a system that learns latent relationships between PHAs and mobile devices and leverages them for prediction. We showed that it is possible to predict PHA installation on mobile devices with very accurate results (0.994 TPR with 0.0001 FPR) up to one week ahead of the real installations, and our system is performant and can account for delays in receiving data, which are typical in real world deployments. This prediction approach has the potential of successfully complementing current on-device anti-malware systems, which are inherently reactive. We plan to study how to devise effective warning that could pro-actively sway users into not installing the PHAs that they will encounter, for example while browsing third party marketplaces. To this end, we plan to work in the spirit of past research in usable security and warning design [20], [76].

Acknowledgments

References

[1] Yousra Aafer, Wenliang Du, and Heng Yin. Droidapiminer: Mining api-level features for robust malware detection in android. In *SecureComm*, 2013.

[2] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. Androzoo: Collecting millions of android apps for the research community. In *IEEE/ACM Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016.

[3] Nicoló Andronio, Stefano Zanero, and Federico Maggi. Heldroid: Dissecting and detecting mobile ransomware. In *RAID*, 2015.

[4] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*, 2014.

[5] Albert-Laszlo Barabâsi, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4), 2002.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE TPAMI*, 35(8), 2013.

[7] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

[8] Leyla Bilge, Yufei Han, and Matteo Dell'Amico. Riskteller: Predicting the risk of cyber incidents. In *ACM CCS*, 2017.

[9] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *ICML*, 2019.

[10] Yazan Boshmaf, Dionysios Logothetis, Georgos Siganos, Jorge Lería, Jose Lorenzo, Matei Ripeanu, and Konstantin Beznosov. Integro: Leveraging victim prediction for robust fake account detection in osns. In *NDSS*, 2015.

[11] Mehran Bozorgi, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Beyond heuristics: learning to classify vulnerabilities and predict exploits. In *KDD*, 2010.

[12] Leo Breiman. Random forests. *Machine learning*, 45(1), 2001.

[13] Saurabh Chakradeo, Bradley Reaves, Patrick Traynor, and William Enck. Mast: Triage for market-scale mobile malware analysis. In *WiSec*, 2013.

[14] Chih-Ming Chen, Chuan-Ju Wang, Ming-Feng Tsai, and Yi-Hsuan Yang. Collaborative similarity embedding for recommender systems. In *WWW*, 2019.

[15] Chih-Ming Chen, Yi-Hsuan Yang, Yian Chen, and Ming-Feng Tsai. Vertex-context sampling for weighted network embedding. *arXiv*, 2017.

[16] Jonathan Crussell, Ryan Stevens, and Hao Chen. Madfraud: Investigating ad fraud in android applications. In *MobiSys*, 2014.

[17] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1), 1998.

[18] George Danezis and Prateek Mittal. Sybilinfer: Detecting sybil nodes using social networks. In *NDSS*, 2009.

[19] Steven HH Ding, Benjamin CM Fung, and Philippe Charland. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In *IEEE S&p*, 2019.

[20] Serge Egelman, Lorrie Faith Cranor, and Jason Hong. You've been warned: an empirical study of the effectiveness of web browser phishing warnings. In *CHI*. ACM, 2008.

[21] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. 2009.

[22] Emre Erturk. A case study in open source software security and privacy: Android adware. In *World Congress on Internet Security (WorldCIS-2012)*, 2012.

[23] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. Android permissions: User attention, comprehension, and behavior. In *SOUPS*, 2012.

[24] Ming Gao, Leihui Chen, Xiangnan He, and Aoying Zhou. Bine: Bipartite network embedding. In *SIGIR*, 2018.

[25] Google. Android security & privacy 2018 year in review. 2019.

[26] Michael Grace, Yajin Zhou, Qiang Zhang, Shihong Zou, and Xuxian Jiang. Riskranker: scalable and accurate zero-day android malware detection. In *MobiSys*, 2012.

[27] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[28] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1), 1982.

[29] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2), 1998.

[30] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. 2013.

[31] Colin C Ife, Yun Shen, Steven J Murdoch, and Gianluca Stringhini. Waves of malice: A longitudinal measurement of the malicious file delivery ecosystem on the web. In *ASIACCS*, 2019.

[32] Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Christopher Kruegel, Sabyasachi Saha, Giovanni Vigna, Sung Ju Lee, and Marco Mellia. Nazca: Detecting malware distribution in large-scale networks. In *NDSS*, 2014.

[33] Chanhyun Kang, Noseong Park, B Aditya Prakash, Edoardo Serra, and VS Subrahmanian. Ensemble models for data-driven prediction of malware infections. In *WSDM*, 2016.

[34] Kazuaki Kishida. *Property of average precision and its generalization: An examination of evaluation indicator for information retrieval experiments*. 2005.

[35] Platon Kotzias, Leyla Bilge, and Juan Caballero. Measuring {PUP} prevalence and {PUP} distribution through pay-per-install services. In *USENIX Security*, 2016.

[36] Platon Kotzias, Juan Caballero, and Leyla Bilge. How did that get in my phone? unwanted app distribution on android devices. In *IEEE S&P*, 2021.

[37] Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*, 2019.

[38] Bum Jun Kwon, Jayanta Mondal, Jiyong Jang, Leyla Bilge, and Tudor Dumitraş. The dropper effect: Insights into malware distribution with downloader graph analytics. In *ACM CCS*, 2015.

[39] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, 2014.

[40] Charles Lever, Manos Antonakakis, Bradley Reaves, Patrick Traynor, and Wenke Lee. The core of the matter: Analyzing malicious traffic in cellular carriers. In *NDSS*, 2013.

[41] Guanjun Lin, Jun Zhang, Wei Luo, Lei Pan, and Yang Xiang. Poster: Vulnerability discovery with function representation learning from unlabeled projects. In *ACM CCS*, 2017.

[42] Guanjun Lin, Jun Zhang, Wei Luo, Lei Pan, Yang Xiang, Olivier De Vel, and Paul Montague. Cross-project transfer representation learning for vulnerable function discovery. *IEEE Transactions on Industrial Informatics*, 2018.

[43] Yang Liu, Armin Sarabi, Jing Zhang, Parinaz Naghizadeh, Manish Karir, Michael Bailey, and Mingyan Liu. Cloudy with a chance of breach: Forecasting cyber security incidents. In *USENIX Security*, 2015.

[44] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[45] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *NDSS*, 2017.

[46] Nicholas Micallef, Mike Just, Lynne Baillie, and Maher Alharby. Stop annoying me! an empirical investigation of the usability of app privacy notifications. In *OZCHI*, 2017.

[47] Omid Mirzaei, Guillermo Suarez-Tangil, Jose M de Fuentes, Juan Tapiador, and Gianluca Stringhini. Andrensemble: Leveraging api ensembles to characterize android malware families. In *ASIACCS*, 2019.

[48] Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. Polonium: Tera-scale graph mining and inference for malware detection. 2011.

[49] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 7, 2013.

[50] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2), 2001.

[51] James R Norris. *Markov chains*. Cambridge university press, 1998.

[52] Jon Oberheide and Charlie Miller. Dissecting the android bouncer. *SummerCon2012, New York*, 12, 2012.

[53] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, 1999.

[54] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. Tesseract: Eliminating experimental bias in malware classification across space and time. In *USENIX Security*, 2019.

[55] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. Using probabilistic generative models for ranking risks of android apps. In *ACM CCS*, 2012.

[56] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *ICCIS*, 2005.

[57] Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu, et al. The ghost in the browser: Analysis of web-based malware. *HotBots*, 2007.

[58] Foster Provost and R Kohavi. Glossary of terms. *Journal of Machine Learning*, 30(2-3), 1998.

[59] Babak Rahbarinia, Marco Balduzzi, and Roberto Perdisci. Real-time detection of malware downloads via large-scale url− > file− > machine graph mining. In *ASIACCS*, 2016.

[60] Moheeb Abu Rajab, Lucas Ballard, Noé Lutz, Panayiotis Mavrommatis, and Niels Provos. Camp: Content-agnostic malware protection. In *NDSS*, 2013.

[61] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[62] Matilda Rhode, Pete Burnap, and Kevin Jones. Early-stage malware prediction using recurrent neural networks. *computers & security*, 2018.

[63] Martin Rosvall and Carl T Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4), 2008.

[64] Carl Sabottke, Octavian Suciu, and Tudor Dumitras. Vulnerability disclosure in the age of social media: Exploiting twitter for predicting real-world exploits. In *USENIX Security*, 2015.

[65] Mahmood Sharif, Jumpei Urakawa, Nicolas Christin, Ayumu Kubota, and Akira Yamada. Predicting impending exposure to malicious content from user behavior. In *ACM CCS*, 2018.

[66] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *ACM CCS*, 2018.

[67] Yun Shen and Gianluca Stringhini. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *USENIX Security*, 2019.

[68] Gianluca Stringhini, Pierre Mourlanne, Gregoire Jacob, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. {EVILCOHORT}: Detecting communities of malicious accounts on online services. In *USENIX Security*, 2015.

[69] Gianluca Stringhini, Yun Shen, Yufei Han, and Xiangliang Zhang. Marmite: spreading malicious file reputation through download graphs. In *ACSAC*, 2017.

[70] Guillermo Suarez-Tangil and Gianluca Stringhini. Eight years of rider measurement in the android malware ecosystem: evolution and lessons learned. *arXiv preprint arXiv:1801.08115*, 2018.

[71] Nan Sun, Jun Zhang, Paul Rimba, Shang Gao, Leo Yu Zhang, and Yang Xiang. Data-driven cybersecurity incident prediction: A survey. *IEEE Communications Surveys & Tutorials*, 21(2), 2018.

[72] Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. Guilt by association: large scale malware detection by mining file-relation graphs. In *KDD*, 2014.

[73] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualizing large-scale and high-dimensional data. In *WWW*, 2016.

[74] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.

[75] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. Investigating commercial pay-per-install and the distribution of unwanted software. In *USENIX Security*, 2016.

[76] Christopher Thompson, Martin Shelton, Emily Stark, Maximilian Walker, Emily Schechter, and Adrienne Porter Felt. The web's identity crisis: understanding the effectiveness of website identity indicators. In *USENIX Security*, 2019.

[77] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond google play: A large-scale comparative study of chinese android app markets. In *IMC*, 2018.

[78] G.H. Weiss. *Aspects and Applications of the Random Walk (Random Materials and Processes)*. North Holland, 1994.

[79] Dominik Wermke, Nicolas Huaman, Yasemin Acar, Bradley Reaves, Patrick Traynor, and Sascha Fahl. A large scale investigation of obfuscation use in google play. In *ACSAC*, 2018.

[80] Chao Yang, Zhaoyan Xu, Guofei Gu, Vinod Yegneswaran, and Phillip Porras. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In *ESORICS*, 2014.

[81] Jheng-Hong Yang, Chih-Ming Chen, Chuan-Ju Wang, and Ming-Feng Tsai. Hop-rec: high-order proximity for implicit recommendation. In *RecSys*, 2018.

[82] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, 2014.

[83] Gianluca Stringhini Yun Shen, Pierre-Antoine Vervier. Understanding worldwide private information collection on android. In *NDSS*, 2021.

[84] Jian Zhang, Phillip A Porras, and Johannes Ullrich. Highly predictive blacklisting. In *USENIX Security*, 2008.

[85] Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. Bipartite network projection and personal recommendation. *Physical review E*, 76(4), 2007.

[86] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *IEEE S&P*, 2012.