MalAF : Malware Attack Foretelling From Run-time Behavior Graph Sequence

Anonymous Author(s)

ABSTRACT

Foretelling ongoing malware attacks in real-time is challenging due to the stealthy and polymorphic of their executive behavior patterns. In this paper, we present MalAF, a novel Malware Attack Foretelling framework utilizing run-time behavior (i.e., sequences of API events) of malware to foretell the attack that has not yet executed. MalAF first samples suspicious API events by assessing the sensitivity of the parameters of each API event, and divides them into multiple attak time slots by calculating the stong correlation. Following that, MalAF employs dynamic heterogeneous graph sequences to incrementally model contextual semantic for each attack time slot, generating malware state sequences in real time. Moreover, MalAF proposes a greedy adaptive dictionary (GAD)-optimized IRL preference learning method to automate the capture of families' intrinsic attack preferences, which achieves higher performance than the existing IRL. Additionally, with the guidance of families' attack preferences, MalAF trains an LSTM to foretell the future path of the target malware. Finally, MalAF matches the identified APIs' paths with a malicious capability base and reports the comprehensible attacks to an analyst. The experiments on real-world datasets demonstrate that our proposed MalAF outperforms the state-of-the-art methods, especially MalAF improves the baseline by 3.01%~4.73% of accuracy in terms of path foretell.

CCS CONCEPTS

• **Security and privacy → Malware attack foretelling**.

KEYWORDS

malware attack foretelling, API event, dynamic heterogeneous graph, inverse reinforcement learning, proactively preventing

ACM Reference Format:

Anonymous Author(s). 2018. MalAF : Malware Attack Foretelling From Run-time Behavior Graph Sequence. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the past years, cyber attacks have seriously damaged the Internet due to the proliferation and growth of malware, which perform several attack steps to reach their specific objectives [28]. Besides, attackers also frequently utilize novel techniques to obfuscate their actions [17]. To combat the threats imposed by malware, many efforts have been proposed in malware classification or detection [2, 8, 19, 23, 29, 31, 40, 44], but they are no longer effective because they fail to proactively defend against the upcoming threats in advance. Therefore, security researchers are increasingly perceiving that it is indispensable to predict potential malware attacks to proactively prevent unknown cyber risks [34, 35]. From that perspective, several works [1, 4, 5, 14, 17, 22] in recent years to predict malware attacks. For instance, some researchers [1, 14] applied Hidden Markov Models (HMMs) to reveal the multiple-step attacks of malware. However, their performance is strongly dependent on the collected alarms, and they are unable to handle the new types of attacks. Moreover, LightGBM [22], LSTM [5], and DeepAG [17] utilized machine learning or deep learning models to achieve the goal of predicting future malicious activity by extracting various behavioral features, such as API calls, permissions, or execution events, in system logs. However, the aforementioned methods can only make a binary prediction of whether the process includes an attack rather than what the process's future attack is likely to be. Recently, Alrawi *et al.* presented FORECAST [4], a symbolic analysis technique to forecast what capabilities are possible based on the attack memory images. Although FORECAST has a promising predictive performance, it requires the posterior probabilities of attacks occurring at each node, which is not easy to tackle as the prior knowledge for unexpected and sophisticated strategies taken by adversaries is difficult to obtain. To overcome these limitations, this paper strives to propose an effective prediction model to automatically foretell malware attacks to get ahead of attackers.

However, designing a productive malware attack foretelling model encounters three major challenges: (1) The entire run-time behavior of malware is a sequence, in which most of the paths are normal and several may be malicious. In that case, a fundamental challenge is to accurately identify the multiple stages of event paths that are most likely to lead to attacks from the raw event sequence and generalize their comprehensible malicious capability; (2) how to automatically model the malware behavior sequences rather than overly relying on expert prior knowledge to simulate all possible paths from a sophisticated environment; and (3) how to dynamically model malware nonlinear dependencies and construct semantic attack graphs to assist grasp malware states in real time.

To cope with the above challenges, we propose MalAF, a Malware Attack Foretelling framework capable of predicting malware attacks that have not yet executed utilizing dynamic inverse reinforcement learning on forensics parameter-sensitive API events. The major contributions of this paper are summarized below.

- We propose MalAF, a novel malware attack foretelling framework, which includes a set of automatic workflows for a dynamic heterogeneous graph-based state sequence generating phase, a greedy adaptive dictionary (GAD)-optimized IRL

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

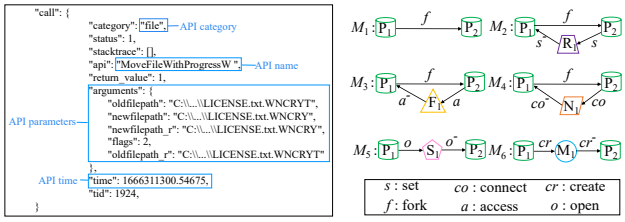


Figure 1: (a) An API event. (b) Meta-graphs of MalAF.

preference learning phase, and a family preference-guided LSTM path foretelling phase, thus mitigating the analytic cost of security experts and helping them make proactive prevention in real time. Afterward, MalAF can not only foretell the future attack path but also the comprehensible malicious capability, reducing the semantic gap.

- We devise a strong correlation calculation (SCC) method that effectively divides the malware event sequences into multiple attack time slots while leveraging the security semantics contained in run-time API parameters to filter irrelevant API events, thereby reducing path explosion.
- We investigate the dynamic heterogeneous graph sequence to model the contextual semantic of various stages of malware and incrementally learn the system state sequence of malware, which can capture the attack patterns of malware in real time, only costing at most 77% of the time of state-of-the-art static methods.
- We comprehensively evaluate the effectiveness and efficiency of MalAF on three real-world datasets. MalAF demystifies malware attack foretelling and provides a powerful counter to emerging cyber threats.

2 THREAT MODEL AND PROBLEM STATEMENT

2.1 Threat Model

Malware has the characteristics of being dynamic, polymorphic, and stealthy. To avoid detection, attackers typically stay in the target host for several days, employing novel techniques to obfuscate their actions, resulting in a complex attack environment. Indeed, malware performs several malicious API events (consisting of name, time, category, and parameters shown in Figure 1(a)) to reach their specific objectives, which would be recorded as a sequence. Analyzing the API sequence used by malware is useful for identifying its capabilities because malware's behavior stems from its API calls and data flow. Evidently, some paths in the "sequence" are malicious attacks. In this case, analyzing the API sequence used by malware regularly is useful for foretelling the future malicious path and identifying its capabilities in advance, security experts can perform proactive defenses before the network is compromised.

Figure 2 depicts the complete attack sequences of a ransomware called Prolock [3]. To begin, the attackers inject malicious ShellCode into an image file "WinMgr. BMP" in order to infiltrate the victim's host; When the process arrives, it begins a series of actions

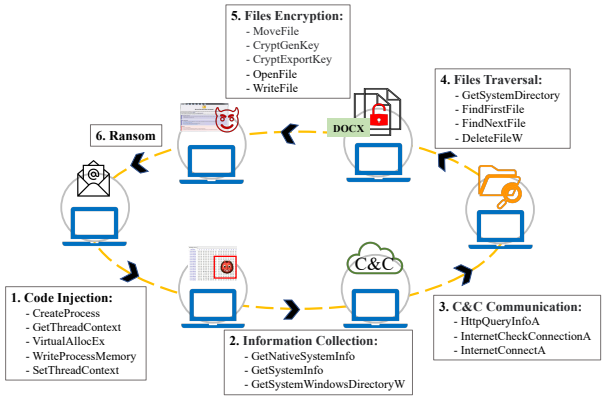


Figure 2: Six typical attack steps of Prolock.

in the host machine, including exploring the running environment and collecting information about the victim's device; Then, the malicious process contacts its Command and Control (C&C) server to get the encryption key; Next, it traverses files and searches for files with specific extensions; In the next stage, the malicious process moves the targeted files to another location and then encrypts them. Finally, attackers ransom the victim. As Figure 2 shows, the different stages of attacks are actually intrinsically related, through they seem seriously complex. Therefore, by forensically identifying and modeling these "fixed paths" involved in malicious processes as much as possible, we are capable of capturing the attacker's preferences and foretelling its next attack path, as well as the malicious capability. For example, by forensically examining the sensitive path of the previous two stages, we can foretell the malware attack "C&C Communication = (HttpQueryInfoA, InternetCheckConnectionA, InternetConnectA)" that has not yet occurred.

2.2 Problem Statement

MalAF is proposed in this paper to foretell the malware attacks ahead of attackers. MalAF considers each malware family (for example, Trojan.Rokrat) to be an "agent" and collects a certain amount of demonstrated behavior (i.e., expert API sequences) $D = (\zeta_1,\ldots,\zeta_{|D|})$ to train the family's intrinsic attack preference, where $\zeta_d = (e_1,\ldots,e_L)$ and e_l is a four-tuple consisting of $\langle name, time, category, parameter\rangle$. MalAF first employs statistic-analysis and clustering-analysis techniques to assess the sensitivity of each API event e_l based on the security semantics implicit in concrete API parameters, and then filters out the irrelevant API events based on the lowest degrees of sensitivities (i.e., 0). Then MalAF leverages the strong correlation calculation (SCC) method to organically divide the filtered sensitive API events into several combinations, forming the preprocessed multi-stage attack sequence $\zeta'_d = (C_1,\ldots,C_T)$, in which the APIs contained in C_t are invoked simultaneously to carry out the specific attack at time slot t. In this case, we input the attack sequences ζ' of each agent into MalAF for analysis. MalAF first employs dynamic heterogeneous graph sequences to model the interaction relationship of API parameters contained in $\zeta'_d = (C_1,\ldots,C_T)$ and then implements dynamic graph learning to generate semantically rich state sequences $(s_1,\ldots,s_T)$ in real time; subsequently, in order to reduce

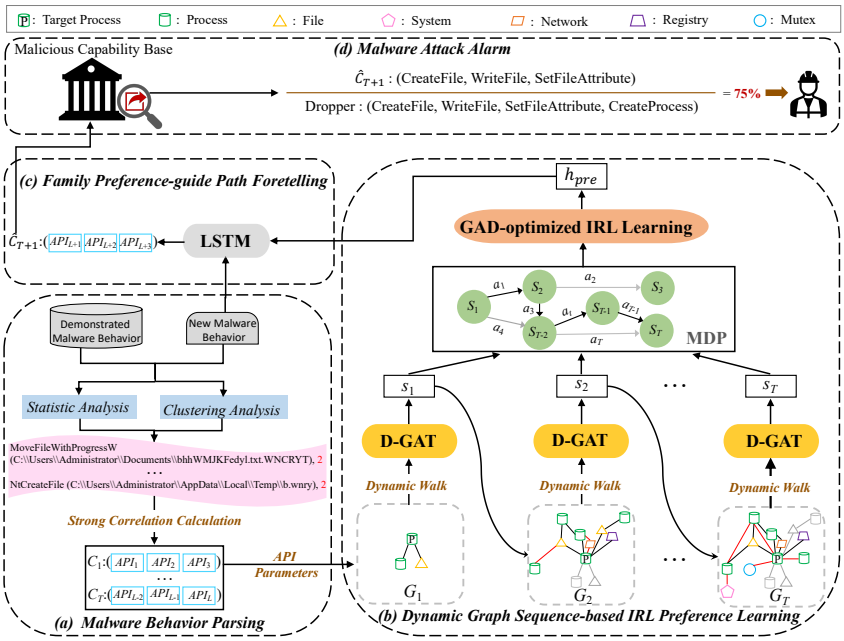


Figure 3: MalAF workflow. (a) Malware Behavior Parsing aims to preprocess the raw malware behavior into the multiple stages of attack paths by computing the sensitivity and the correlation of the basic API events. (b) Dynamic Graph Sequence-based IRL Preference Learning to automatically learn the intrinsic attack preferences h_{pre} of a malware "agent" by utilizing dynamic graph attention networks (D-GATs) and greedy adaptive dictionary learning based on the constructed malware heterogeneous graph sequences (in which the gray nodes and gray lines indicate the expired events and the red lines indicate newly created events). (c) Family Preference-guide Path Foretelling leverages an LSTM network to precisely foretell the future path $\hat{C}_{T+1}$ under the guidance of the learned family preference h_{pre}. (d) Malware Attack Alarm reports the matched malicious capability to an analyst in real time.

the computing overhead, MalAF explores greedy adaptive dictionary (GAD) learning to project the massive malware state space into a transformation domain to obtain a lower-dimensional state vector sequence $(s'_1,\ldots,s'_T)$. Ultimately, based on the constructed MDP of malware, MalAF utilizes IRL to learn the intrinsic family attack preference h_{pre} of each agent from the demonstration D.

In the foretelling phase, given the historical API events of the new malware, MalAF employs the family attack preference h_{pre} of the "agent" to guide the new malware to interact with the current environment and outputs the foretold attack path $\hat{C}_{T+1}$ = (CreateFile, WriteFile, SetFileAttribute) in the future. Finally, MalAF compares $\hat{C}_{T+1}$ to the malicious capability base to identify the most likely malicious capability (i.e., Dropper) and report it to security experts.

3 PRELIMINARIES

Definition 1. ***Malware Dynamic Heterogeneous Graphs Sequence (MDHGS).*** *A dynamic heterogeneous graphs sequence of malware is a graph set $\boldsymbol{G} = \{G_1, G_2,\ldots, G_T\}$, each $G_t = (V_t, E_t)$ with an entity type mapping $\phi: V_t \rightarrow A$ and a relation type mapping $\psi: E_t \rightarrow R$, where V_t and E_t denote the entity set and the relation set of G_t, respectively, and A and R denote the entity type and relation type, respectively. Among them, $|A| > 1$, $|R| > 1$.*

Definition 2. ***Meta-graph [39].*** *A meta-graph M is a directed acyclic graph with a single source node n_s (i.e., with in-degree 0) and a single target node n_t (i.e., with out-degree 0), defined on an MDHGS with schema $T_G = (A, R)$, then a meta-graph can be defined as $M = (V_M, E_M, A_M, R_M, n_s, n_t)$, where $V_M \in V$, $E_M \in E$ are constrained by $A_M \in A$ and $R_M \in R$, respectively.*

As shown in Figure 1(b), in this work, we investigate 6 real-world meta-graphs in MalAF. Different meta-graphs represent different semantic information.

Definition 3. ***Markov Decision Process (MDP) [21].*** *An MDP is formally defined as a tuple $X = (S, A, P, R, \gamma)$, where S is the malware agent's set of states and A is the set of actions. $P : S \times A \rightarrow P(S)$ is the conditional transition probability function. The transition function P models the uncertainty in the evolution of system states based on the agent's action. $R : S \times A \rightarrow \mathbb{R}$ is the reward function that helps the agent learn. $\gamma \in [0,1]$ is the long-term reward discount coefficient, which indicates the agent's emphasis on the future reward value.*

4 METHODOLOGY

Though the malware attack patterns are highly stealthy and complicated, we still find them through analyzing sequences of API events because a malware's behavior stems from its API calls and data flow [9]. Inspired by that, we propose MalAF, as illustrated

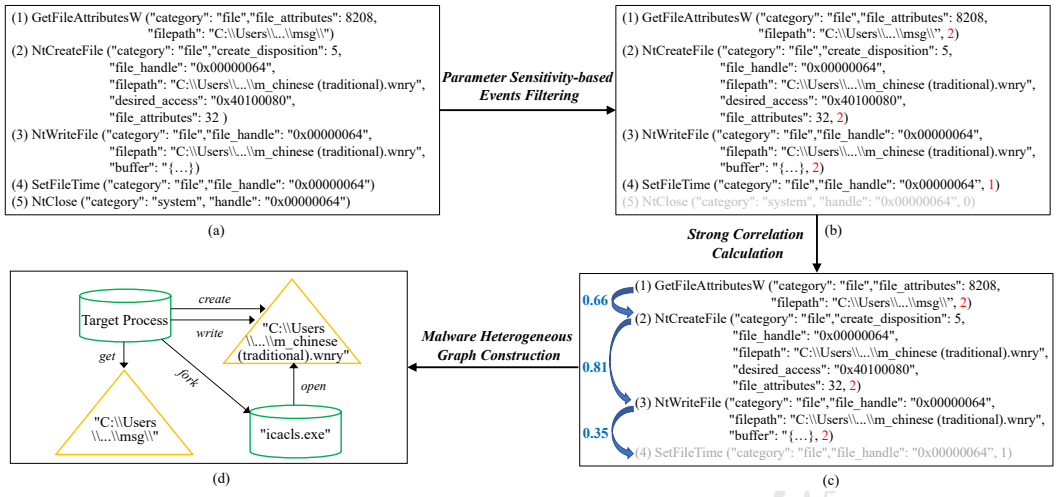


Figure 4: Malware behavior parsing.

in Figure 3, which is divided into four phases: malware behavior parsing (Figure 3(a)), dynamic graph sequence-based IRL preference learning (Figure 3(b)), family preference-guide path foretelling (Figure 3(c)), and malware attack alarm (Figure 3(d)).

4.1 Malware Behavior Parsing

This phase is in charge of converting raw malware behavior into multiple stages of attack paths as input to the MalAF training phase and foretelling phases. As stated before, the execution behavior of the malware is essentially composed of a sequence of run-time API call events, shown in Figure 4(a), whose length is extremely large, e.g., the vast majority of malware invokes more than 1,000 API events during the 5-minute execution, and most of the events are normal and several may be malicious, making it hard to automatically perform forensics from such a large-scale sequence.

To address this challenge, we first leverage a clever parameter sensitivity-based events filtering method to discard irrelevant events in each sequence by assessing their security semantics from concrete API parameters; Furthermore, we present a strong correlation calculation method to establish the multi-stage attack sequence $\zeta'_d = (C_1,\ldots,C_T)$ by aggregating the strongly correlated basic APIs.

4.1.1 Parameter Sensitivity-based Events Filtering. Actually, each event is represented as $\langle name, time, category, parameter\rangle$, and the run-time parameters passed to APIs contain security-sensitive information that is valuable for malware analysis. As a result, we propose a parameter sensitivity-based events filtering method that integrates statistic-analysis and clustering-analysis techniques to compute the degree of malice (i.e., sensitivity) of each API event, with benign events having a minimum sensitivity of 0, malicious events having a maximum sensitivity of 2, and the rest of the events having the default sensitivity of 1.

We first employ a statistic-analysis to determine whether a parameter is malicious or benign. Concretely, we implement term frequency inverse document frequency (TF-IDF) [12] to quantify the parameter sensitivity of each API event. Formally, the frequency can be expressed as:

$$TF_i = \frac{n_i}{\sum_k n_k}, \tag{1}$$

where n_i is the number of occurrences of parameter i in the sample set B, and $\sum_k n_k$ is the sum of the occurrences of all parameters in the sample set B.

Consequently, let TF_m refer to the frequency of a parameter in malicious software, while DF_m refers to the frequency of software containing the parameter. TD_m equals $TF_m \times DF_m$. Similarly, TF_b, DF_b, and TD_b denote the corresponding values in benign software. Thereby, for a parameter, if it has a higher TD_m but a lower TD_b may imply a malicious event with a high probability, while a parameter with a lower TD_m but a higher TD_b is more likely to be benign. In this case, we determine the sensitivity of a portion of API events (i.e., 40%) with high confidence, then we exploit a clustering-analysis technique to evaluate the uncovered events' sensitivity.

Notably, we formalize parameters clustering as short text clustering. Motivated by the promising performance of GSDMM [36] (collapsed Gibbs Sampling algorithm for the Dirichlet Multinomial Mixture) in short text clustering, we also adopt it to divide the entire events space into K (i.e., $K = 3$) categories following their run-time parameters, namely benign, sensitive, and malicious. Eventually, as shown in Figure 4(b), we are capable of filtering the irrelevant API events with varying degrees of sensitivity, which is beneficial to reducing extraneous path explosion.

4.1.2 Multi-stage Attack Sequence Establishing Based on Strong Correlation Calculation. Indeed, as illustrated in Figure 4(c), the sensitive APIs involved in malware do not appear individually, yet they are invoked simultaneously in a combined manner to carry out the specific attack [16]. Hence, it is reasonable to gather the highly related successive APIs into a combination by exploiting their Pearson correlation coefficients. For API_i and API_j, we compute $Pearson_{i,j}$ between them as below:

$$Pearson_{i,j} = cov(API_i, API_j) \big/ \delta_{API_i}\delta_{API_j}, \tag{2}$$

Algorithm 1 Multi-stage Attack Sequence Establishing

Input: Sensitive API events sequence $\zeta = \{API_1, API_2, \ldots, API_{L'}\}$ of the demonstrated malware;
Output: Multi-stages attacks sequence $\zeta' = \{C_1, \ldots, C_T\}$ of the demonstrated malware;

$\zeta' \leftarrow \{\} \in [1, L']$;
$T \leftarrow 0$;
$last \leftarrow$ combination index of API_{i-1};
for $i \in 1$ to L' **do**
  for $API_j \in C_{last}$ **do**
    $Pearson_{i,j} = \frac{cov(API_i, API_j)}{\delta_{API_i}\delta_{API_j}}$;
    if $Pearson_{i,j} < \tau$ **then**
      T++;
      $C_T \leftarrow API_i$;
    else
      $C_{last} \leftarrow C_{last} \cup API_i$;
    end if
  end for
end for
return $\zeta' = \{C_1, \ldots, C_T\}$

where $cov(API_i, API_j)$ represents the covariance between API_i and API_j, δ_{API_i} and δ_{API_j} are the standard deviations of API_i and API_j.

Ultimately, as shown in Algorithm 1, we obtain the multi-stage attack sequence $\zeta'_d = (C_1, \ldots, C_T)$ of the demonstrated malware, which dramatically reduces the learning cost of MalAF.

4.2 Dynamic Graph Sequence-based IRL Preference Learning

To model temporal sequences, inverse reinforcement learning (IRL) is a promising technology that can automatically learn the agent's goals or intentions from the observed demonstration, which has shown soaring performance in vehicle trajectory [24, 27, 33, 38], traffic prediction [42], autonomous driving [10, 37], and so on. Motivated by the strength of inverse reinforcement learning, in this subsection we attempt to implement IRL to automatically learn the malware attack preferences from the demonstrated API sequences.

Here, we will first discuss how to use MDP to model the decision-making processes of the multi-stage attack sequence $\zeta'_d = (C_1, \ldots, C_T)$. Concretely, we regard each malware family as an "agent" and then collect a certain amount of expert API sequences from each malware family. The goal of each agent is to evaluate the various actions associated with the current state using intrinsic preferences. It is universally acknowledged that there are five factors in MDP: State set S, Action set A, Transition probability function P_{sa}, Discount factor γ, and Reward R, however, the transition function P_{sa} and reward function R in our task are not available since malware environments are complex. To this end, we emphasize learning semantically rich state sequences and leveraging a model-free IRL without knowledge of the transition function to generate the intrinsic attack preferences for each malware family "agent". As depicted in Figure 3(b), this phase includes two advanced components (see subsection 4.2.1 and 4.2.2).

4.2.1 Dynamic Heterogeneous Graph-based State Learning. As stated before, we divide the entire malware's API sequence into multiple time slots, where each C_t corresponds to a state of the malware at one specific stage. To accurately catch the fine-grained state representation of malware, it is essential to fully take advantage of the contextual semantic information from the API parameters at C_t. Concretely, we first extract six types of system entities (i.e., process, file, network, system, mutex, and registry) and six types of relationships (i.e., process-process, process-file, process-network, process-system, process-registry, and process-mutex) among them, and then employ a dynamic heterogeneous graph sequence (e.g., Figure 4(d)) to model the sensitive parameter interactions at each time slot (e.g., Figure 4(c)). Different from constructing the heterogeneous graph from scratch in the existing static frameworks [32, 40], we efficiently utilize the overlapping information at adjacent stages (e.g., G_{t-1} and G_t) to receive richer contextual representations, which offers better efficiency. As shown in Figure 3(b), G_t at C_t can be updated from G_{t-1} as new events are joined and expired events are removed, as represented by the adjacency matrix A_t.

Subsequently, MalAF investigates the dynamic graph attention networks (D-GATs) with the guidance of the meta graph-based dynamic walk to timely update the evolving state representation s_t, which discriminatively aggregates the newly joined nodes in G_t. In particular, given the constructed malware heterogeneous graph G_t and the pre-defined meta-graphs $\boldsymbol{M}$, we capture the fine-grained state representation s_t through the following steps:

1) Search for meta-graph-based dynamic neighborhood $\Delta N_t^{(m)}$: To capture the semantic-unique malware attack patterns, as shown in Figure 1(b), MalAF leverages the pre-defined meta-graph set $\boldsymbol{M} = \{M_1, M_2, \ldots, M_{|M|}\}$ to guide the dynamic random walk, which only traverses the changed nodes rather than all nodes in the current graph. Formally, given G_t, the dynamic neighborhood $\Delta N_t^{(m)}$ can be generated by a meta-graph M_m is:

$$\Delta N_t^{(m)} = N_t^{(m)} \bigcup \Delta V_t, \tag{3}$$

where $N_t^{(m)}$ represents the meta-graph-based neighborhood of target process node p walks along with M_m, and ΔV_t denotes the dynamic node set (i.e., new nodes or nodes of a new connection edge) on the current heterogeneous graph G_t.

2) Aggregate node-level representation $\mathbf{h}_{t_p}^{(m)(k)}$: Particularly, we first obtain the node-level representations $\mathbf{h}_{t_p}^{(m)(k)}$ of the target process node p, which is iteratively updated by combining its own feature of the representations over its dynamic neighbors in $\Delta N_t^{(m)}$ since the known nodes in G_t have been aggregated in G_{t-1}. Thus, the k-th layer of the node-level aggregator [18] is:

$$\alpha_{t_u}^{(m)} = \frac{exp(LeakyReLU(\mathbf{W}^T[\mathbf{s}_{t-1}, \mathbf{h}_{t-1_u}]+\mathbf{b}))}{\sum_{u' \in \Delta N_t^{(m)}} exp(LeakyReLU(\mathbf{W}^T[\mathbf{s}_{t-1}, \mathbf{h}_{t-1_{u'}}]+\mathbf{b}))}, \tag{4}$$

$$\mathbf{h}_{t_p}^{(m)(k)} = MLP^{(k)}\left((1+\epsilon^{(k)})\mathbf{h}_{t_p}^{(m)(k-1)} + \sum_{u \in \Delta N_t^{(m)}} \alpha_{t_u}^{(m)} \mathbf{h}_{t_u}^{(m)(k-1)}\right), \tag{5}$$

where $\boldsymbol{h}_{t_p}^{(m)(k-1)}$ and $\boldsymbol{h}_{t_u}^{(m)(k-1)}$ are the node-level representations of the target process node p and its corresponding neighbor node u at the (k-1)-th layer, respectively, and they can be initialized

by the learned representations $\boldsymbol{s}_{t-1}$ and $\boldsymbol{h}_{t-1_u}$ at G_{t-1}. $\epsilon^{(k)}$ is a balance parameter. Finally, the overall node-level representation of the target process node p concerning the meta-graph M_m is:

$$\mathbf{h}_{t_p}^{(m)} = CONCAT([\mathbf{h}_{t_p}^{(m)(k)}]_{k=1}^{K}). \tag{6}$$

3) Aggregate graph-level state representation $\boldsymbol{s}_t$: Once the semantic-unique node representations are generated, we intend to aggregate them into graph embedding space. Formally,

$$\omega_t^{(m)} = \frac{exp(\sigma(\mathbf{b}[\mathbf{W}_b\mathbf{h}_{t_p}^{(m)} \| \mathbf{W}_b\mathbf{h}_{t_p}^{(n)}]))}{\sum_{g\in|M|} exp(\sigma(\mathbf{b}[\mathbf{W}_b\mathbf{h}_{t_p}^{(m)} \| \mathbf{W}_b\mathbf{h}_{t_p}^{(g)}]))}, \tag{7}$$

$$\mathbf{s}_t = \sum_{m=1}^{|M|} \omega_t^{(m)} \times \mathbf{h}_{t_p}^{(m)}, \tag{8}$$

where $m \neq n \in \{1,\ldots,|M|\}$, $\omega_t^{(m)}$ is the meta-graph attention weight of M_m. σ is the activation function, $\boldsymbol{b}$ is the weight vector from the input layer to the hidden layer of the neural network, and $\boldsymbol{W}_b$ is the corresponding weight matrix.

Consequently, as demonstrated in Figure 3(b), based on the semantically rich state sequence and various API actions, we construct the MDP of malware.

4.2.2 GAD-optimized IRL Learning. To further improve the real-time performance of IRL, we propose a greedy adaptive dictionary (GAD) learning algorithm to project the malware state space to a transform domain [43], where the coefficients are fewer and smaller than those of the time domain. Formally, the problem of GAD learning is to find an orthogonal matrix $\boldsymbol{H}$ such that the following equation holds:

$$\mathrm{S} = \mathrm{H}\,\mathrm{S}', \tag{9}$$

where $\boldsymbol{S'}$ is the coefficient with respect to the dictionary $\boldsymbol{H}$. $\boldsymbol{S}$ is the malware state matrix.

Dictionary learning constructs a dictionary by using an alternating optimization strategy. Hence, we design a minimum index to match the sparse representation and minimize coefficients, which is defined as:

$$\eta = \|s_n\|_0 + \|s_n\|_2, \tag{10}$$

where $\|\cdot\|_0$ and $\|\cdot\|_2$ are the l_0-norm and l_2-norm, respectively; among them, the former keeps the sparsity of the coefficients and the latter keeps the coefficients as small as possible. The vector s_n denotes the n-th column of the original state matrix $\boldsymbol{S}$.

We obtain the orthogonal matrix $\boldsymbol{H}$ through multiple iterations, during each iteration l, we define a residual term $\Phi^l = [\phi_1^l, \phi_2^l, \ldots, \phi_{N^2}^l]$, where $\phi_n^l \in R^{N^2}$ is the n-th column of the residual value. Hence, the optimal orthogonal vector with the minimum l_0-norm and l_2-norm can be obtained by:

$$n^l \leftarrow arg\ min_{n\notin N^l}\{\eta = \|\phi_n^l\|_0 + \|\phi_n^l\|_2\}, \tag{11}$$

where element n^l denotes the n-th column of the residual term at the l-th iteration. During each iteration, the column of the residual term with the minimum index is determined and normalized to gain an orthogonal basis.

$$\mathbf{d}^{\mathbf{l}} \leftarrow \phi_{n^l}^l / \|\phi_{n^l}^l\|_2, \tag{12}$$

$$\phi_n^{l+1} \leftarrow \phi_n^l - \mathbf{d}^{\mathbf{l}} < \mathbf{d}^{\mathbf{l}}, \phi_n^l > . \tag{13}$$

Eventually, through several greedy iterations, the proposed greedy adaptive dictionary learning algorithm is capable of finding the optimal dictionary in the sense that the coefficient matrix $\boldsymbol{S'}$ obeys the optimal sum of the l_0-norm and l_2-norm, which significantly reduces the computational complexity of the subsequent IRL.

Motivated by [15], we as well as utilize model-free Q-learning to capture the malware's intrinsic attack preference. Formally, given the preprocessed multi-stage attack sequence (i.e., a sequence of state-action pairs of length T) $\zeta' = \{(s'_1, a_1), (s'_2, a_2), \ldots, (s'_T, a_T)\}$ in D, the log-likelihood can be expressed as:

$$\begin{aligned} L(h_{pre}) &= log\ P(D; h_{pre}) \\ &= \sum_{\zeta'\in D} (log\ P(s'_1) + \sum_{i=0}^{T-1} log\ \pi_{h_{pre}}(s'_{i+1}, a_{i+1}) + \sum_{i=1}^{T-1} log\ T(s'_i, a_i, s'_{i+1})), \end{aligned} \tag{14}$$

where h_{pre} is the attack preference vector of the expert API sequences D of the malware "agent". π and T denote the malware agent's policy and transition function, respectively.

Considering the transition function in Eq. 14 is always excluded when computing the gradient in a model-free Q-learning, we have:

$$\frac{\partial L(h_{pre})}{\partial h_{pre}} = \sum_{\zeta'\in D} \sum_{i=1}^{T-1} \frac{1}{\pi_{h_{pre}}(s'_i, a_i)} \frac{\partial \pi_{h_{pre}}(s'_i, a_i)}{\partial h_{pre}}. \tag{15}$$

Additionally, the policy π can be explored by the parameterized Boltzmann exploration [15] as follows:

$$\pi_{h_{pre}}(s', a) = \frac{e^{\beta\, Q_{h_{pre}}(s', a)}}{\sum_{a'\in A} e^{\beta\, Q_{h_{pre}}(s', a')}} = \frac{e^{\beta\, Q_{h_{pre}}(s', a)}}{Z_{h_{pre}}(s')}, \tag{16}$$

where β is the Boltzmann temperature.

Given the exploration policy $\pi_{h_{pre}}$, we estimate the Q-function $Q_{h_{pre}}(s', a)$ of the demonstration D by leveraging the powerful model-free Q-learning, which is defined as:

$$\begin{aligned} Q_{h_{pre}}^t(s', a) \leftarrow (1-\mu) Q_{h_{pre}}^{t-1}(s', a) + \mu(R_{h_{pre}}(s', a) + \\ \gamma \sum_{a'\in A} \pi_{h_{pre}}^{t-1}(s', a') Q_{h_{pre}}^{t-1}(s', a')), \end{aligned} \tag{17}$$

where $\pi_{h_{pre}}^t(s', a)$ is obtained from the t-th iteration of the Q-function. μ is the learning schedule, γ is the discount factor, and $R_{h_{pre}}(s', a)$ is the reward for taking action a in state s'.

Next, we differentiate Eq. 17, and this operation may be performed recursively to update the attack preference vector h_{pre}.

4.3 Family Preference-guide Path Foretelling

In the foretelling phase, given the historical API events of the new malware, MalAF employs the family attack preference h_{pre} of the "agent" to guide the new malware to interact with the current environment and outputs the foretold attack path $\hat{C}_{T+1} = (API_{L+1}, API_{L+2}, API_{L+3})$ that has the highest probability. Subsequently, we first utilize a long short-term memory (LSTM [13]) network to output the temporal embedding of the new malware by inputting the preprocessed multi-stage attack sequence:

$$h_{C_t} = MLP(f_i \| f_{i+1} \| \ldots \| f_{i+w-1}), \tag{18}$$

$$h_T = LSTM(h_{C_1}, \ldots, h_{C_t}, \ldots, h_{C_T}), \tag{19}$$

Table 1: Description of 8 Malicious Capabilities in MalAF

Malicious Capability	Tracked APIs	Description of Behaviour
ATT_1 = File Exfiltration	(OpenFile, ReadFile, Socket, Send)	Track Socket creation, file access, and the parameters associated with each API.
ATT_2 = Code Injection	(CreateProcess, GetThreadContext, VirtualAllocEx, WriteProcessMemory, SetThreadContext)	Process hollowing, tracks from SetThreadContext to identify parameter constraints tying back to pContext, hProcess, and pHandle.
ATT_3 = Dropper	(CreateFile, WriteFile, SetFileAttribute, CreateProcess)	Download of a malicious piece of code, which may be disguised as legitimate software, an image or document.
ATT_4 = Screen Spying	(GetDesktopWindow, GetWindowDC, CreateCompatiableBitmap, CreateCompatiableDC, SelectObject)	Capture screen by identifying a handle to bitmap object. Then constraints the handle to a Windows handle object that is created by referencing the user Window.
ATT_5 = Persistence	(RegSetValueEx, RegOpenKeyEx, GetFullPathnameA)	Implement a persistent that relies on registry keys.
ATT_6 = C&C Communication	(HttpQueryInfoA, InternetCheckConnectionA, InternetConnect)	Exchanges with the discovered its C&C server, over IRC, HTTP, or in some cases, custom protocols.
ATT_7 = Network Scan	(Socket, listen)	As a server that listens on the socket.
ATT_8 = Information Collection	(GetNativeSystemInfo, GetSystemInfo, GetSystemWindowsDirectoryW)	Attempt to get detailed information about the operating system and hardware.

where f_i, f_{i+1}, and f_{i+w-1} are the feature vectors of the objects API_i, API_{i+1}, and API_{i+w-1} in C_t; $\|$ denotes the concatenation operation.

As illustrated in Figure 3(d), MalAF foretells $\hat{C}_{T+1}$ = (CreateFile, WriteFile, SetFileAttribute) by inputting h_T and h_{pre} through a fully connected decoder, which is formulated as:

$$\hat{C}_{T+1} = \sigma_d(w_d \cdot (h_T \| h_{pre}) + b_d), \tag{20}$$

where $\|$ denotes the concatenation operation. σ_d is the nonlinear activation function of the decoder, and w_d and b_d are the trainable weight parameters and bias of the decoder, respectively. Finally, the loss function l of our MalAF can be formulated as:

$$l = ||(C_{T+1} - \hat{C}_{T+1})||_F^2. \tag{21}$$

With the loss function l, we perform stochastic gradient descent to fine-tune all learnable parameters.

4.4 Malware Attack Alarm

To generalize high-level malicious capabilities, bridging the semantic gap, as shown in Figure 3(d), MalAF maintains a malicious capability base to match the foretold APIs' path to report comprehensible malicious capabilities to an analyst in real-time. According to the existing research [1, 4, 20], we raise eight typical capabilities in our malicious capability base: Code Injection, File Exfiltration, Dropper, Persistence, Screen Spying, C&C Communication, Network Scan, and Information Collection. As shown in Table 1, each malicious capability is defined in terms of the corresponding API path and their parameters, and they can easily be extended to capture additional capabilities based on the target system's APIs.

Concretely, as shown in Figure 3(d), the foretold path $\hat{C}_{T+1}$ matches the "Dropper" with a 75% probability. To that end, MalAF quickly and clearly reports the alarm to security experts, reducing analytic cost and enabling proactive prevention in real time.

5 EXPERIMENT

In this section, we assess MalAF's ability to foretell malware attacks. We first introduce the datasets, experiment setup, and baseline methods. Then, we report the experimental results, including the performance of path foretelling, the performance of malware attack foretelling, parameter sensitivity, ablation study, and the performance of packed malware.

5.1 Dataset

5.1.1 Statistics. We validated the performance of MalAF on three datasets, involving two public datasets (i.e., Kaggle Malware[1] and Mal-API-2019[2]) and our captured ACT-SANDBOX dataset. Kaggle Malware dataset [25, 26] is the most widely used in Kaggle competitions and recent malware research, which includes the first 100 non-repetitive API sequences and the labels of 37,784 malware and 1,079 benign software. Mal-API-2019 dataset [6, 7] involves a total of 7,107 malware, and each record is an ordered API sequence generated by the Cuckoo sandbox environment. Spanning from Mar 2020 to Dec 2021, the ACT-SANDBOX dataset collects 38,703 malware samples from 108 families and 9,719 benign samples from the authoritative VirusShare,[3] each of which contains a Cuckoo sandbox behavior report and a label generated by VirusTotal.[4] We assume that our dataset provides good coverage because of the sheer number of files submitted to the platform. The basic statistics of all datasets are presented in Table 2.

For all datasets, we randomly selected 50 known sample from each type of malware as the expert API sequences to learn families'

[1]https://www.kaggle.com/ang3loliveira/malware-analysis-datasets-api-call-sequences.
[2]https://github.com/ocatak/malware_api_class.
[3]https://virusshare.com.
[4]https://www.virustotal.com.

Table 2: Statistics of The Three Datasets

Dataset	Samples Distribution								
Kaggle Malware	Trojan	Downloader	Virus	Spyware	Adware	Dropper	Worm	Backdoor	Benign
	12,824	6,560	5,522	5,897	4,449	945	808	779	1,079
Mal-API-2019	Trojan	Downloader	Worms	Virus	Backdoor	Dropper	Spyware	Adware	-
	1,001	1,001	1,001	1,001	1,001	891	832	379	-
ACT-SANDBOX	Trojan	Downloader	Virus	Spyware	Ransom	Dropper	Worm	Backdoor	Benign
	21,572	573	5,272	864	3,124	697	4,283	2,318	9,719
# Families	47	2	13	5	11	3	17	10	1

Table 3: Performance Comparison of Malware Path Foretelling With Different Methods

Method	**ACT-SANDBOX**			**Kaggle Malware**			**Mal-API-2019**		
	ACC(%)	F1-score(%)	Foretell Time(s)	ACC(%)	F1-score(%)	Foresee Time(s)	ACC(%)	F1-score(%)	Foretell Time(s)
MA-HMM [14]	84.76	84.82	15.9	81.57	81.09	8.8	83.55	83.90	13.1
LightGBM [22]	79.41	78.66	83.2	75.65	75.71	57.5	78.32	78.29	80.5
FORECAST [4]	95.81	95.93	147.7	92.14	92.77	89.4	93.19	93.21	120.3
MA-GAIL [41]	93.11	93.47	28.1	90.27	90.35	21.8	90.66	90.58	25.4
Smell [5]	90.13	90.46	23.6	85.34	85.93	19.0	87.55	87.64	21.7
MalAF (ours)	**98.82**	**98.85**	**38.8**	**96.87**	**96.90**	**24.1**	**97.39**	**97.76**	**30.9**

attack preferences. Then, in the path foretelling phase, we used 60% of the rest samples as training data and 40% samples as test data.

5.1.2 Ground Truth. In our experiments, evaluating the foretell performance of MalAF involves 2 aspects: (1) the accuracy of the foretell path, and (2) the accuracy of the identified malicious capability. As a result, we manually analyze the attack stages based on Cuckoo reports and label ground truth C_{T+1} for samples collected. Concretely, we first located the typical attack point T+1 in the malware sequences by tracking APIs in Table 1, and then labeled the path and specific malicious capability for C_{T+1}.

5.2 Experiment Setup

We default the learning rate to 0.01, the dropout to 0.5, the embedding size to 128, the number of layers of GAT to 3, and the foretold path length to 3. Moreover, we use the experimental setup described in [15] for our model-free IRL experiments, with γ=0.01 and β=0.01. We train MalAF on a machine equipped with a 16 cores Intel(R) Core(TM) i7-6700 CPU @3.40 GHz with 64 GB RAM and 4 × NVIDIA Tesla K80 GPU. All of the experiments developed with Python 3.6 are executed on the TensorFlow-GPU framework supported by the Ubuntu 16.0.4 operating system.

5.3 Baseline Methods

We compare MalAF with five state-of-the-art prediction methods and briefly introduce these baselines as follows:

- MA-HMM [14] is presented to predict multi-step attacks by leveraging the hidden Markov model and IDS alerts.
- LightGBM [22] is a gradient boosting decision tree algorithm to predict future malware attacks by combining decision trees, random forests, and logistic regression.
- FORECAST [4] leverages the execution context from the malware's memory image to guide a symbolic analysis, then computes percentages for each discovered capability.
- Smell [5] relies on extracting various static and dynamic characteristics (e.g., API calls, system calls, and permissions) and training the LSTM model to predict the presence of malicious behavior in a process.
- MA-GAIL [41] is an imitation learning-based traffic prediction model that employs generative adversarial imitation learning to estimate the vehicle trajectory.

5.4 Path Foretelling Evaluation

To verify the performance of MalAF on path foretelling, we empirically compared MalAF with the state-of-the-art malware attack prediction methods in terms of effectiveness and efficiency.

5.4.1 Effectiveness Evaluation. The comparison results of MalAF and baseline methods are recorded in Table 3, and the following observations can be made:

First, MalAF outperforms the traditional deep learning models (such as LightGBM and Smell) with a significant performance gain in terms of all metrics. In particular, the proposed MalAF achieves improvements of at least 8.69%, 11.53%, and 9.84% on accuracy (i.e., ACC) on the ACT-SANDBOX dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively, which demonstrates that the model-free inverse reinforcement learning based on the dynamic heterogeneous graph state sequences has excellent ability to automatically capture the temporal attack preference, thus boosting the foretelling accuracy of potential (or upcoming) paths.

Second, FORECAST has the closest performance to MalAF; this mainly contributes to the fact that FORECAST not only employs symbolic analysis to manually simulate all possible paths based on ongoing API sequence but also screens the suspicious paths from

Table 4: Malware Attack Foretelling Performance of 12 Select Malware Families From ACT-SANDBOX Dataset (P_M : Avg. Matched Percentage, ACC : Foretelling Accuracy)

Malware		File Exfilt		Code Inject		Dropper		Screen Spy		Persistence		C&C Comm		Network Scan		Info Coll	
Type	**Family**	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)	P_M	ACC(%)
Trojan	Rokrat	75%	98.4	61%	96.6	-	-	50%	98.1	-	-	67%	100	-	-	-	-
Backdoor	Andromeda	-	-	80%	100	-	-	-	-	100%	100	84%	97.6	100%	100	50%	99.5
	AveMaria	92%	96.8	87%	100	-	-	56%	92.5	78%	97.9	54%	96.3	100%	100	73%	98.2
	Berbew	-	-	-	-	86%	97.2	78%	100	76%	96.3	85%	95.2	75%	100	91%	100
Virus	Floxif	79%	100	-	-	-	-	-	-	-	-	64%	99.0	-	-	67%	97.4
	Sality	65%	95.5	-	-	-	-	-	-	-	-	48%	95.4	100%	100	-	-
Ransom	CTBLocker	80%	99.2	-	-	-	-	-	-	-	-	83%	98.7	100%	100	78%	99.1
	WannaCrypt	83%	98.6	78%	98.3	-	-	-	-	84%	100	74%	98.5	90%	98.7	43%	100
Downloader	Marap	80%	96.1	93%	94.8	-	-	83%	97.9	-	-	-	-	-	-	68%	98.7
Spyware	AgentTesla	50%	95.8	60%	100	91%	98.0	94%	100	67%	94.4	94%	100	-	-	83%	100
Worm	Mydoom	88%	100	-	-	-	-	-	-	67%	99.3	67%	99.8	100%	100	-	-
Dropper	Dinwod	65%	97.7	-	-	100%	100	-	-	51%	95.2	-	-	88%	100	-	-

fine-grained API arguments, which can discover the malicious path with a high probability by leveraging extensive expert knowledge to calculate the "concreteness" score of each API operation.

Third, compared with the imitation learning prediction method, MA-GAIL, our proposed MalAF is superior to it in terms of foretelling effectiveness on all datasets. The main reason is that, ulike MA-GAIL, which naively investigates temporal states without considering interactive context, our proposed MalAF ingeniously utilizes a dynamic heterogeneous graph sequence to capture the comprehensive semantic information from various API parameters, which can better capture the evolutionary pattern of malware in real time to generate the fine-grained state representations.

5.4.2 Efficiency Evaluation. Here we also study the time efficiency of MalAF on three datasets. The comparison results are shown in Table 3. We make three crucial observations.

First, the traditional statistics-based and machine learning-based prediction models (i.e., MA-HMM and Smell) always consume less time than complex deep learning models or reinforcement learning models (i.e., MalAF, LightGBM, and MA-GAIL). These phenomena show that the simpler the model, the lower the time complexity.

Second, both are imitation learning; the foretelling time of MalAF is slightly greater than that of MA-GAIL. Notably, MalAF sacrifices a slight time consumption to learn a series of dynamic heterogeneous graph-based state representations from the API parameters, which dramatically boost the effectiveness of malware path foretelling.

Third, MalAF is significantly faster than FORECAST over all three datasets. The reasons can be attributed to the facts: first, different from FORECAST that consumes a large amount of time to manually simulate all possible paths, the proposed MalAF utilizes IRL to optimize the learn progress of attack preferences, which reduces the foretelling time. Second, MalAF leverages the security semantics contained in run-time API parameters to filter redundant API events, which is beneficial to reduce the API event space to boost the foretelling efficiency.

5.5 Malware Attack Foretelling Evaluation

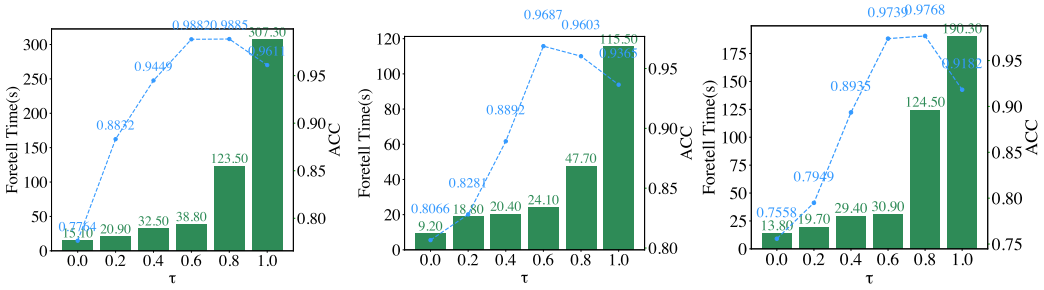
To verify the performance of MalAF on malware attack foretelling, we selected the most recent samples of 12 malicious families from the ACT-SANDBOX dataset, with a sample size ranging from 10 to 120, and manually analyzed ground truth for multiple stages of each malware family. As shown in Table 4, MalAF outputs 57 distinct malicious capabilities, and P_M represents the matched percentage of the foretold APIs' paths C_T to the relevant tracked APIs in Table 1.

File Exfiltration: File Exfiltration can be seen as an "end goal" malicious capability where malware sends stolen information from an infected host by uploading a file to its drop site. MalAF reported 10 File Exfiltrations, in which the *Floxif* and *Mydoom* families achieved 100% foretelling accuracy. Concretely, *Floxif* and *Mydoom* invoke OpenFile and ReadFile APIs to copy data into a buffer, followed by use of the send network API.

Code Injection: MalAF reports 6 Code Injections, which are done by the OpenProcess or CreateProcess API operations, followed by WriteProcessMemory (process hollowing). Explorer and Svchost are the most common Windows programs injected into. The *AveMaria*, *Andromeda*, and *AgentTesla* families achieved 100% foretelling accuracy. In particular, *AveMaria* hollows into Svchost by invoking CreateProcess, and then swaps the code pages with WriteProcessMemory and SetThreadContext. Dissimilarly, *TeslaAgent* and *Andromeda* inject into Explorer by invoking these API sequences.

Dropper: MalAF reports 3 Droppers, which write a file to disk and change its attributes for execution. *Berbew*, *AgentTesla*, and *Dinwod* families drop files in the AppData and ProgramData directories and manipulate their permissions by invoking SetFileAtrributes.

Screen Spying: MalAF reports 5 Screen spyings. We focused on detecting screens based on the GDI API toolkit. *Berbew* and *TeslaAgent* families achieved 100% foretelling accuracy, which checks if a device context handle returned by GetDC or GetWindowDC is passed to CreateCompatibleBitmap.



(a) ACT-SANDBOX dataset (b) Kaggle Malware dataset (c) Mal-API-2019 dataset

Figure 5: The performance of MalAF with different threshold τ.

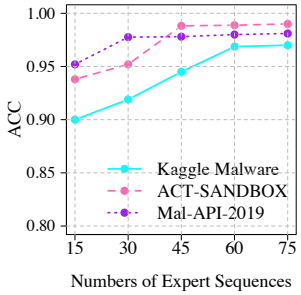


Figure 6: The performance of MalAF with different $|D|$.

Persistence: We find that most malware persists by infecting the registry. MalAF reports 7 Persistence attacks, which compare the registry key handle returned by RegOpenKey or RegSetValue with the input to RegSetValue. Specifically, as shown in Table 4, the foretelling accuracy of the *Andromeda* and *WannaCrypt* families reaches 100%, and the path matching percentage is also the highest.

C&C Communication: MalAF reports 10 C&C Communication attacks by tracking the WinINET APIs like InternetCheckConnectionA and InternetConnect. In particular, we concretized their domain and IP address parameters. MalAF foretells with 100% accuracy that the *Rokrat* and *AgentTesla* families are about to perform a C&C communication attack. Concretely, *Rokrat* uses dropbox.com to communicate, and *AgentTesla* uses a hardcoded IP address and a gmail account to communicate externally.

Network Scan: MalAF reports 8 Network Scan by monitoring the APIs such as Socket and listen. listen is a server that listens to the socket after invoking socket and bind. Specifically, as shown in Table 4, the path matching percentage and foretelling accuracy of *Andromeda*, *AveMaria*, *Sality*, and *Mydoom* families all reach 100%.

Information Collection: Most malware illegally collects detailed information about the victim host, including the operating system and hardware. Information Collection is easier to obtain and thus has the best foretelling. MalAF reports 8 Information Collections, in which the *Berbew*, *WannaCrypt*, and *AgentTesla* families achieved 100% accuracy. They invoke the GetSystemInfo and GetSystemWindowsDirectoryW APIs to get system information.

5.6 Parameter Sensitivity Analysis

In this subsection, we investigate the sensitivity of several essential parameters in MalAF. We mainly focus on these parameters, including the Pearson correlation coefficient threshold (i.e., τ) and the number of expert sequences (i.e., $|D|$).

Pearson Correlation Coefficient Threshold τ. The threshold of the Pearson correlation coefficient has a strongly influence on the division of each attack slot, because we organically combine the basis API events whose Pearson correlation coefficient is greater than τ. As shown in Figures 5(a-c), MalAF achieves the pleasant foretelling performance on all datasets when τ is set to 0.6. On the contrary, as τ becomes smaller, the foretelling accuracy becomes smaller. In the most extreme case, when τ is equal to 0, the foretelling accuracy is reduced by 22.18%, 16.21%, and 21.81% compared with when τ = 0.6. These results prove that when τ is too small, many weakly-correlated APIs are forced to be grouped together, and the real attack path of malware is seriously disturbed, resulting in a significant decline in foretelling performance. When τ is greater than 0.6 in Figures 5(a-c), it can be seen that MalAF's foretelling accuracy would decline, because the larger the threshold, the fewer API parameters interactions contain in C_t, and MalAF's advantage of capturing semantically rich state representations with D-GATs will gradually fade away, which limits the capability of MalAF.

Figures 5(a-c) also display that the larger τ, the more foretell time, especially if τ = 1, which is the case because our proposed MalAF does not contain strongly correlated combinations (SCC). As a result, we find that MalAF's foretell time at τ = 0.6 is 7.92×, 4.79×, and 6.16× faster than that at τ = 1 on the ACT-SANDBOX dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively. This shows that the SCC is important for boosting the time efficiency of MalAF, which can divide a large number of basis APIs into fewer time slots.

Number of Expert Sequences $|D|$. We also study the effect of varying the number of expert API sequences in MalAF. Demonstration D plays an important role in recovering the attack preference of each agent's malware. As shown in Figure 6, we find that ACC is basically stable even when the number of the expert sequence is greater than 45, 60, and 30 on the ACT-SANDBOX dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively. These findings demonstrate that an sufficient number of expert API

Table 5: A Summary of The Ablation Study (PSEF : parameter sensitivity-based events filtering, IRL : inverse reinforcement learning, DSR : dynamic state representations)

Method	Metric	ACT-SANDBOX	Kaggle Malware	Mal-API-2019
MalAF no PSEF	ACC	0.9651	0.9364	0.9408
MalAF no IRL	ACC	0.9145	0.8792	0.8966
MalAF no DSR	ACC	0.9466	0.9026	0.9174
MalAF	ACC	0.9882	0.9687	0.9739

sequences can fully capture the agent malware's intrinsic attack preference.

5.7 Ablation Study

Parameter Sensitivity-based Events Filtering. We first study the contribution of the parameter sensitivity-based events filtering on MalAF. As shown in Table 5, the foretelling performance of MalAF is slightly superior to its variant model, MalAF no PSEF, confirming the parameter sensitivity-based events filtering method emphasizes picking behavioral events that contain sensitive semantics, which is able to discard irrelevant noise events and effectively improve the foretelling accuracy of MalAF. Moreover, Figure 8 reveals that MalAF is 3×, 2.3×, and 2.7× faster than the variant model MalAF no PSEF on the ACT-SANDBOX dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively. The possible reason is that MalAF leverages the parameter sensitivity-based events filtering method to dramatically reduce the event space by assessing their security semantics from concrete API parameters, which productively boosts the foretell efficiency of MalAF.

IRL Preference Learning. Inverse reinforcement-based preference learning is a crucial component in MalAF; we investigate its contribution by comparing MalAF with the variant model MalAF no IRL. Concretely, as illustrated in Table 5, compared to MalAF no IRL, our proposed MalAF improves 7.37% on the ACT-SANDBOX dataset, 8.95% on the Kaggle Malware dataset, and 7.73% on the Mal-API-2019 dataset in terms of ACC, respectively. These results indicate that the Inverse Reinforcement-based Preference Learning component has the ability to automatically capture the intrinsic attack preferences of malware agents from the observed demonstrations, which is beneficial for accurately foretelling the upcoming attack of stealthy and polymorphic malware.

Dynamic Heterogeneous Graph-based State Representations. We then compare MalAF and its variant model, MalAF no DSR, to investigate how dynamic heterogeneous graph-based state learning can boost foretelling accuracy. From Table 5, we observe that MalAF outperforms its variant model, MalAF no DSR. Specifically, ACCs are increased by 4.16%, 6.61%, and 5.65% on the ACT-SANDBOX dataset, Kaggle Malware dataset, and Mal-API-2019 dataset, respectively, which evinces that the contextual interaction semantics of the underlying API parameters can provide valuable knowledge to help foretell a potential malware attack.

Meta-graph. We also assess the impact of different meta-graphs on malware attack foretelling performance by progressively adding meta-graphs to MalAF. In Figure 7, we can notice that by adding more meta-graphs, the ACCs of the MalAF on all datasets are higher,

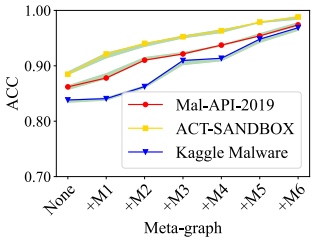


Figure 7: Performance change of MalAF when progressively adding meta-graphs in terms of ACC.

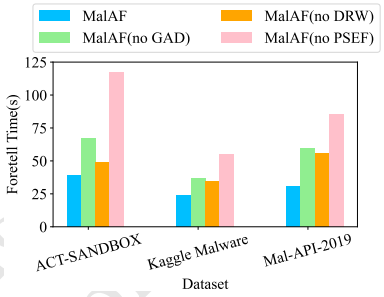


Figure 8: Efficiency comparison of MalAF with its variant models.

which visibly proves that each meta-graph is capable of capturing the unique semantic information and helping capture the fine-grained malware state representations. Specifically, we can see that when we consider the M_1 in the ACT-SANDBOX dataset, the M_3 and M_5 in the Kaggle Malware dataset, and the M_2 and M_6 in the Mal-API-2019 dataset, MalAF's performance improves significantly.

Greedy Adaptive Dictionary (GAD) Learning. We also examine the effect of greedy adaptive dictionary (GAD) learning in our MalAF framework by comparing MalAF with the variant model, MalAF no GAD. As Figure 8 shown, MalAF's foretell time is significantly less than that of MalAF no GAD in all datasets, because MalAF leverages the greedy adaptive dictionary learning algorithm to project the massive malware state space to a fewer and smaller transform domain, which can improve MalAF's training efficiency.

Dynamic Random Walk. We finally investigate the role of dynamic random walk when searching for meta-graph-based neighborhoods by comparing MalAF and its variant model, MalAF no DRW. As shown in Figure 8, MalAF incurs less overhead than MalAF no DRW due to the fact that the dynamic random walk solely searches the newly joined nodes in G_t instead of considering all known nodes in G_t like the variant model MalAF no DRW, which empirically proves that the dynamic random walk is more valuable to real-time foretelling than the existing static random walk.

5.8 Packed Malware

We evaluated MalAF's robustness against packers using the encountered packed malware in the ACT-SANDBOX dataset. As shown in Table 6, we consider five different types of packers, whose complexity ranges from Type-I to Type-VI [11]. Concretely, MalAF competently foretells Type-I and III packer variants (i.e., UPX 2.90,

Table 6: Performance on Packed Malware Foretelling

Packer	Packer Type	# Malware	ACC
UPX 2.90	Type-I	487	0.991
UPX 2.93	Type-I	315	0.982
BobSoft Mini Delphi	Type-I	334	0.987
ASPack	Type-III	381	0.974
Armadillo	Type-VI	630	0.895

UPX 2.93, BobSoft Mini Delphi, and ASPack), while it is inferior to analyze malware variants that use Type-VI Armadillo, judging from the foretelling accuracy of Armadillo being only 89.5% in Table 6. There are two reasons. First, Type-I through Type-III packers can fully unpack in the sandbox when executing the malicious code, thus MalAF has the ability to identify almost every malicious capability found in Table 1. Second, the unpacking routine of the Armadillo packer is interwoven with the malware payload, increasing the complexity, which especially affects the related paths of C&C communication and Persistence. However, MalAF cannot handle more complex packed techniques (e.g., virtualization and repackaging), which account for a tiny fraction of packed malware [30].

6 RELATED WORK

In this section, we review the works related to our study, including malware attack prediction and inverse reinforcement learning.

6.1 Malware Attack Prediction

Malware has the characteristics of being dynamic, polymorphic, and stealthy, making it very challenging to effectively predict future malware attacks. In recent years, few efforts have been made to predict malware attacks. Holgado *et al.* [14] proposed a hidden Markov model to predict multi-step attacks. MA-HMM consisted of a training module and a prediction module. The training phase computed the probability matrices based on the observable IDS alerts. The prediction module computes the best state sequence based on the state probability using the forward-backward algorithms. To further foretell the attack time, Abaid *et al.* [1] leveraged a semi-Markov chain model to foretell the time of upcoming attacks. However, the performance of the these methods is strongly dependent on the collected alarms, and they are weakly equipped to handle the new types of attacks. Moreover, Patel *et al.* [22] presented LightGBM, a gradient-boosting decision tree to predict future malware attacks on cloud computing systems. They integrated weak algorithms such as decision trees, random forests, binary classification, and logistic regression to predict the probability that a host would be infected with various types of malware. To improve the prediction performance of machine learning, Li *et al.* [17] proposed DeepAG, which utilizes Bi-directional deep learning to predict the attack paths of APT (Advanced Persistent Threats) based on system logs and achieves higher performance than traditional Bi-LSTM. Amer *et al.* [5] also proposed a robust deep learning early alarm prediction model, Smell, which extracted a variety of static and dynamic features (such as API calls, system calls, and permission sequences) and then trained multiple LSTMs to predict whether the malicious behavior exists in the target process based on the transformed features. Nevertheless, these aforementioned methods can only make a binary prediction of whether a process includes attack, instead of what the future attack of the process is likely to be.

Recently, Alrawi *et al.* [4] presented FORECAST, a symbolic analysis technique to forecast what capabilities are possible from memory images. Although FORECAST has a promising predictive performance, it requires the posterior probabilities of attacks occurring at each node, which is not easy to tackle as the prior knowledge for unexpected and sophisticated strategies taken by adversaries is difficult to obtain. To address these limitations, we propose MalAF, an effective malware attack foretelling model that can proactively foretell malware attacks in order to stay ahead of attackers. MalAF has the ability to automatically capture the malware agent's attack intentions from the observed demonstration, which separates environment-specific conditions and inaccessible prior knowledge.

6.2 Inverse Reinforcement Learning

Inverse reinforcement learning (IRL) has been widely used to predict future behavior, especially in vehicle trajectory prediction tasks. Sharifzadeh *et al.* [27] first leveraged Deep Q-Networks (DQNs) to learn the rewards, which addressed the exploding state-space problem. Saleh *et al.* [24] also utilized inverse reinforcement learning (IRL) and the bidirectional recurrent neural network (B-LSTM) to predict pedestrians' trajectories. Zhang *et al.* [38] designed a two-stage neural network model that considers motion and environment together to recover the reward function. Moreover, Zheng *et al.* [42] defined an IRL-based traffic prediction model that realized a parameter-sharing mechanism in a multi-agent context. Fernando *et al.* [10] proposed a D-IRL framework by taking into account various complex factors, which has proved a great success in the autonomous driving community.

Inspired by the soaring successes of IRL on various prediction tasks, this paper is the first attempt to introduce IRL into the security field to solve the malware attack foretelling problem. However, the existing IRL struggles to cope with the massive event and state spaces of malware, so our MalAF designs a parameter sensitivity-based events filtering, a dynamic heterogeneous graph-based state learning, and a greedy adaptive dictionary learning to achieve a smaller event space and a more semantically rich state space, assisting in foretelling the upcoming malware attacks in real-time.

7 CONCLUSION

In this paper, we investigate the problem of malware attack foretelling and design a viable framework, MalAF, which deploys a GAD-optimized IRL based on the semantically rich state sequence to efficiently capture malware attack preference from fine-grained API events. MalAF overcomes the challenge of exploring massively irrelevant paths as well as bridging the semantic gap between underlying API parameters and comprehensible malicious capability. Comprehensive experimental results verify that MalAF can significantly enhance the malware attack foretelling performance by at least 3.01%~4.73%, which can be used to proactively resist the upcoming attack risks.

REFERENCES

[1] Zainab Abaid, Dilip Sarkar, Mohamed Ali Kaafar, and Sanjay Jha. 2021. All Infections are Not Created Equal: Time-Sensitive Prediction of Malware Generated Network Attacks. *arXiv preprint arXiv:2102.01944* (2021).

[2] Sahar Abdelnabi, Katharina Krombholz, and Mario Fritz. 2020. VisualPhishNet: Zero-Day Phishing Website Detection by Visual Similarity. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. 1681–1698.

[3] Bander Ali Saleh Al-rimy, Mohd Aizaini Maarof, and Syed Zainudeen Mohd Shaid. 2018. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Computers & Security* 74 (2018), 144–166.

[4] Omar Alrawi, Moses Ike, Matthew Pruett, Ranjita Pai Kasturi, Srimanta Barua, Taleb Hirani, Brennan Hill, and Brendan Saltaformaggio. 2021. Forecasting Malware Capabilities From Cyber Attack Memory Images. In *30th USENIX Security Symposium (USENIX Security 21)*. 3523–3540.

[5] Eslam Amer and Shaker El-Sappagh. 2022. Robust deep learning early alarm prediction model based on the behavioural smell for android malware. *Computers & Security* 116 (2022), 102670.

[6] Ferhat Ozgur Catak, Javed Ahmed, Kevser Sahinbas, and Zahid Hussain Khand. 2021. Data augmentation based malware detection using convolutional neural networks. *PeerJ Computer Science* 7 (Jan. 2021), e346. https://doi.org/10.7717/peerj-cs.346

[7] Ferhat Ozgur Catak and Ahmet Faruk Yazı. 2019. A benchmark API call dataset for windows PE malware classification. *arXiv preprint arXiv:1905.01999* (2019).

[8] Tanmoy Chakraborty, Fabio Pierazzi, and VS Subrahmanian. 2017. Ec2: Ensemble clustering and classification for predicting android malware families. *IEEE Transactions on Dependable and Secure Computing* 17, 2 (2017), 262–277.

[9] Paolo Milani Comparetti, Guido Salvaneschi, Engin Kirda, Clemens Kolbitsch, Christopher Kruegel, and Stefano Zanero. 2010. Identifying dormant functionality in malware programs. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 61–76.

[10] Tharindu Fernando, Simon Denman, Sridha Sridharan, and Clinton Fookes. 2021. Deep Inverse Reinforcement Learning for Behavior Prediction in Autonomous Driving: Accurate Forecasts of Vehicle Motion. *IEEE Signal Processing Magazine* 38, 1 (2021), 87–96. https://doi.org/10.1109/MSP.2020.2988287

[11] Jonathan Fuller, Ranjita Pai Kasturi, Amit Sikder, Haichuan Xu, Berat Arik, Vivek Verma, Ehsan Asdar, and Brendan Saltaformaggio. 2021. C3PO: Large-Scale Study of Covert Monitoring of C&C Servers via Over-Permissioned Protocol Infiltration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3352–3365.

[12] Samujjwal Ghosh and Maunendra Sankar Desarkar. 2018. Class specific TF-IDF boosting for short-text classification: Application to short-texts generated during disasters. In *Companion Proceedings of the The Web Conference 2018*. 1629–1637.

[13] S Hochreiter and J Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[14] Pilar Holgado, Víctor A Villagrá, and Luis Vazquez. 2017. Real-time multistep attack prediction based on hidden markov models. *IEEE Transactions on Dependable and Secure Computing* 17, 1 (2017), 134–147.

[15] Vinamra Jain, Prashant Doshi, and Bikramjit Banerjee. 2019. Model-free IRL using maximum likelihood estimation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 3951–3958.

[16] Yuhei Kawakoya, Eitaro Shioji, Makoto Iwamura, and Jun Miyoshi. 2019. Api chaser: Taint-assisted sandbox for evasive malware analysis. *Journal of Information Processing* 27 (2019), 297–314.

[17] Teng Li, Ya Jiang, Chi Lin, Mohammad Obaidat, Yulong Shen, and Jianfeng Ma. 2022. DeepAG: Attack Graph Construction and Threats Prediction with Bi-directional Deep Learning. *IEEE Transactions on Dependable and Secure Computing* (2022).

[18] Xiang Ling, Lingfei Wu, Wei Deng, Zhenqing Qu, Jiangyu Zhang, Sheng Zhang, Tengfei Ma, Bin Wang, Chunming Wu, and Shouling Ji. 2022. MalGraph: Hierarchical Graph Neural Networks for Robust Windows Malware Detection. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 1998–2007.

[19] Chen Liu, Bo Li, Jun Zhao, Ming Su, and Xu-Dong Liu. 2021. MG-DVD: A Real-time Framework for Malware Variant Detection Based on Dynamic Heterogeneous Graph Learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. 1512–1519. https://doi.org/10.24963/ijcai.2021/209

[20] MITRE. 2022. MITRE ATT&CK. https://attack.mitre.org.

[21] Sindhu Padakandla. 2021. A Survey of Reinforcement Learning Algorithms for Dynamically Varying Environments. *ACM Comput. Surv.* 54, 6 (2021), 127:1–127:25. https://doi.org/10.1145/3459991

[22] Vrushang Patel, Seungho Choe, and Talal Halabi. 2020. Predicting future malware attacks on cloud systems using machine learning. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 151–156.

[23] Rachel Petrik, Berat Arik, and Jared M Smith. 2018. Towards architecture and OS-independent malware detection via memory forensics. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. 2267–2269.

[24] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. 2018. Long-term recurrent predictive model for intent prediction of pedestrians via inverse reinforcement learning. In *2018 Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, 1–8.

[25] Angelo Schranko de Oliveira and Renato José Sassi. 2019. Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks. (2019).

[26] Hermawan Setiawan, Prasetyo Adi Wibowo Putro, Yogha Restu Pramadi, et al. 2020. Comparison of LSTM Architecture for Malware Classification. In *2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS)*. IEEE, 93–97.

[27] Sahand Sharifzadeh, Ioannis Chiotellis, Rudolph Triebel, and Daniel Cremers. 2016. Learning to Drive using Inverse Reinforcement Learning and Deep Q-Networks. *CoRR* abs/1612.03653 (2016). http://arxiv.org/abs/1612.03653

[28] Luman Shi, Jiang Ming, Jianming Fu, Guojun Peng, Dongpeng Xu, Kun Gao, and Xuanchen Pan. 2020. Vahunt: Warding off new repackaged android malware in app-virtualization's clothing. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*. 535–549.

[29] Guosong Sun and Quan Qian. 2018. Deep learning and visualization for identifying malware families. *IEEE Transactions on Dependable and Secure Computing* (2018).

[30] Xabier Ugarte-Pedrero, Davide Balzarotti, Igor Santos, and Pablo G Bringas. 2015. SoK: Deep packer inspection: A longitudinal study of the complexity of run-time packers. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 659–673.

[31] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A Gunter, et al. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis.. In *NDSS*.

[32] Shen Wang and S Yu Philip. 2019. Heterogeneous Graph Matching Networks: Application to Unknown Malware Detection. In *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 5401–5408.

[33] Markus Wulfmeier, Dushyant Rao, Dominic Zeng Wang, Peter Ondruska, and Ingmar Posner. 2017. Large-scale cost function learning for path planning using deep inverse reinforcement learning. *The International Journal of Robotics Research* 36, 10 (2017), 1073–1087. https://doi.org/10.1177/0278364917722396

[34] Kaiming Xiao, Cheng Zhu, Junjie Xie, Yun Zhou, Xianqiang Zhu, and Weiming Zhang. 2018. Dynamic defense strategy against stealth malware propagation in cyber-physical systems. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 1790–1798.

[35] Chunlin Xiong, Tiantian Zhu, Weihao Dong, Linqi Ruan, Runqing Yang, Yan Chen, Yueqiang Cheng, Shuai Cheng, and Xutong Chen. 2020. CONAN: A Practical Real-time APT Detection System with High Accuracy and Efficiency. *IEEE Transactions on Dependable and Secure Computing* (2020).

[36] Jianhua Yin and Jianyong Wang. 2014. A dirichlet multinomial mixture model-based approach for short text clustering. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 233–242.

[37] Changxi You, Jianbo Lu, Dimitar Filev, and Panagiotis Tsiotras. 2019. Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning. *Robotics and Autonomous Systems* 114 (2019), 1–18.

[38] Yanfu Zhang, Wenshan Wang, Rogerio Bonatti, Daniel Maturana, and Sebastian Scherer. 2018. Integrating kinematics and environment context into deep inverse reinforcement learning for predicting off-road vehicle trajectories. *arXiv preprint arXiv:1810.07225* (2018).

[39] Huan Zhao, Quanming Yao, Jianda Li, Yangqiu Song, and Dik Lun Lee. 2017. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 635–644.

[40] Kaifa Zhao, Hao Zhou, Yulin Zhu, Xian Zhan, Kai Zhou, Jianfeng Li, Le Yu, Wei Yuan, and Xiapu Luo. 2021. Structural Attack against Graph Based Android Malware Detection. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21)*. 3218–3235.

[41] Guanjie Zheng, Hanyang Liu, Kai Xu, and Zhenhui Li. 2020. Learning to simulate vehicle trajectories from demonstrations. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1822–1825.

[42] Guanjie Zheng, Hanyang Liu, Kai Xu, and Zhenhui Li. 2021. Objective-aware traffic simulation via inverse reinforcement learning. *arXiv preprint arXiv:2105.09560* (2021).

[43] Yuhui Zheng, Xilong Wang, Guoqing Zhang, Baihua Xiao, Fu Xiao, and Jianwei Zhang. 2019. Multi-Kernel Coupled Projections for Domain Adaptive Dictionary Learning. *IEEE Trans. Multim.* 21, 9 (2019), 2292–2304. https://doi.org/10.1109/TMM.2019.2900166

[44] Şerif Bahtiyar, Mehmet Barış Yaman, and Can Yılmaz Altınığne. 2019. A multi-dimensional machine learning approach to predict advanced malware. *Computer Networks* 160 (2019), 118–129. https://doi.org/10.1016/j.comnet.2019.06.015