Learning Fast and Slow: PROPEDEUTICA for Real-time Malware Detection

Ruimin Sun[†*], Xiaoyong Yuan[‡*], Pan He[§], Qile Zhu[§], Aokun Chen[§], Andre Gregio[¶], Daniela Oliveira[§], and Xiaolin Li[‡‡]
[†]Northeastern University [‡]Michigan Technological University [§]University of Florida
[¶]Federal University of Parana [‡‡]Cognization Lab
[*]Equal Contribution

***Abstract*—Existing malware detectors on safety-critical devices have difficulties in runtime detection due to the performance overhead. In this paper, we introduce PROPEDEUTICA[1], a framework for efficient and effective real-time malware detection, leveraging the best of conventional machine learning (ML) and deep learning (DL) techniques. In PROPEDEUTICA, all software start execution are considered as benign and monitored by a conventional ML classifier for fast detection. If the software receives a borderline classification from the ML detector (e.g. the software is 50% likely to be benign and 50% likely to be malicious), the software will be transferred to a more accurate, yet performance demanding DL detector. To address spatial-temporal dynamics and software execution heterogeneity, we introduce a novel DL architecture (DEEPMALWARE) for PROPEDEUTICA with multi-stream inputs. We evaluated PROPEDEUTICA with 9,115 malware samples and 1,338 benign software from various categories for the Windows OS. With a borderline interval of [30%-70%], PROPEDEUTICA achieves an accuracy of 94.34% and a false-positive rate of 8.75%, with 41.45% of the samples moved for DEEPMALWARE analysis. Even using only CPU, PROPEDEUTICA can detect malware within less than 0.1 seconds.**

***Index Terms*—deep learning, malware detection, spatial-temporal analysis, multi-stage classification**

I. INTRODUCTION

Malware is continuously evolving [1], and existing protection mechanisms have not been coping with their sophistication [2]. The industry still heavily relies on signature-based technology for malware detection [3], but these methods have many limitations: (i) they are effective only for malware with known signatures; (ii) they are not sustainable, given the massive amount of samples released daily; and (iii) they can be evaded by zero-day and polymorphic/metamorphic malware (practical detection 25%-50%) [4].

Behavior-based approaches attempt to identify malware behaviors using instruction sequences [5], [6], computation trace logic [7], and system or API call sequences [8]–[10]. These solutions have been mostly based on conventional machine learning (ML) models, such as K-nearest neighbor, SVM, neural networks, and decision tree algorithms [11]–[14]. However, current solutions based on ML still suffer from high false-positive rates, mainly because of (i) the complexity and diversity of modern benign software and malware [1], [9], [15], [16], which are hard to capture during the learning phase of the algorithms; (ii) sub-optimal feature extraction; (iii) limited training/testing datasets, and (iv) the challenge of concept drift [17], which makes it hard to generalize learning models to reflect future malware behavior.

The accuracy of malware classification depends on gaining sufficient context information about software execution and on extracting meaningful abstraction of software behavior. For system/API-call based malware classification, longer sequences of calls likely contain more information. However, conventional ML-based detectors (e.g., Random Forest [18]) often use short windows of system calls during the training phase to avoid the curse of dimensionality (when the dimension increases, the classification needs more data to support and becomes harder to solve [19]), and may not be able to extract useful features for accurate detection. Thus, the main drawback of current behavioral-based approaches is that they might lead to low accuracy and many false-positives because it is hard to analyze complex and longer sequences of malicious behaviors with limited window sizes, especially when malicious and benign behaviors are interposed. For instance, the n-grams of system calls are widely used as features of ML detection, which, however, are prone to bring blind spots in the detection.

In contrast, emerging deep learning (DL) models [20] are capable of analyzing longer sequences of system calls and making more accurate classification through higher level information extraction, while circumventing the curse of dimensionality. However, DL requires more time to gain enough information for classification and to predict the probability of detection. Moreover, the state-of-the-art DL models consist of deep layers and a huge amount of parameters, which results in slow calculation in the prediction. Such slow process makes DL models infeasible to provide predictions on malware in real time. This trade-off is challenging: fast and perhaps not-so-accurate (ML methods) *vs.* time-consuming and more accurate classification (DL methods). Applying a single method only can be either inefficient or ineffective, which becomes impractical for real-time malware detection.

In this paper, we introduce and evaluate PROPEDEUTICA, a framework for efficient and effective real-time malware detection, considering both prediction performance and efficacy in the detection phase. PROPEDEUTICA is designed to combine

[1]*Propedeutics* refers to diagnosing a patient condition by first performing initial non-specialized, low-cost exams, and then proceeding to specialized, possibly expensive diagnostic procedures if preliminary exams are inconclusive.

the best of ML and DL methods, i.e., speed in prediction from the ML methods and accuracy from the DL methods. In PROPEDEUTICA, all software in the system is subjected to ML for fast classification. If a piece of software receives a borderline malware classification probability, it is then subjected to additional analysis with a more performance expensive and more accurate DL classification. The DL classification is performed via our proposed algorithm, DEEPMALWARE. Compared to existing DL methods [21]–[23], DEEPMALWARE can learn spatially local and long-term features and handle software heterogeneity via the application of both ResNext blocks and recurrent neural networks with multi-stream inputs. By leveraging such multi-stage design, PROPEDEUTICA is capable of providing precise and real-time detection.

We evaluated PROPEDEUTICA with a set of 9,115 malware samples and 1,338 common benign software from various categories for the Windows OS. PROPEDEUTICA leveraged sliding windows of system calls for malware detection. Our proposed deep learning algorithm, DEEPMALWARE, achieved a 97.03% accuracy and a 2.43% false positive rate. By combining conventional ML and DL, PROPEDEUTICA substantially improves the prediction performance while keeping the detection time less than 0.1 seconds, which makes DL feasible for the real-time malware detection.

In this paper, we present the following contributions: (i) PROPEDEUTICA, a new framework for efficient and effective real-time malware detection for Windows OS, (ii) an evaluation of PROPEDEUTICA with a collection of 9,115 malware and 1,338 benign software, and (iii) DEEPMALWARE, a novel DL algorithm, learning multi-scale spatial-temporal system call features with multi-stream inputs that can handle software heterogeneity.

II. METHODOLOGY

A. Threat Model

PROPEDEUTICA is designed to provide precise, real-time malware detection, i.e., a high accuracy rate, few false-positives, and less processing time. PROPEDEUTICA is capable of defending against the most prevalent malware threats, and not against directed attacks, such as specifically-crafted, advanced, and persistent threats. Hence, we assume that malware will eventually get in if an organization is targeted by a well-motivated attacker. Therefore, attacks either targeting PROPEDEUTICA's software implementation, or the machine learning engine (adversarial machine learning [24], [25]) are outside of PROPEDEUTICA's scope. We discuss the adversarial attacks against malware detection in Section V-B.

PROPEDEUTICA operation assumes that, before it loads, the system is on a pristine state. Thus, PROPEDEUTICA will not scan all running processes at startup, which limits its checking procedure to each new process created.

B. Overall Design

PROPEDEUTICA (Figure 1) consists of: (i) a system call reconstruction module, (ii) an ML classifier, and (iii) our newly proposed DEEPMALWARE classifier.

The workflow of PROPEDEUTICA is outlined as follows.

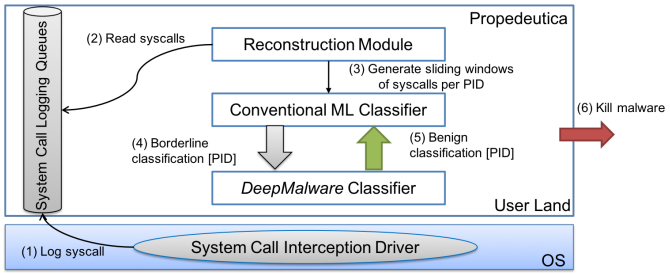


Fig. 1: Architecture and workflow of PROPEDEUTICA for multi-stage malware detection.

STEP 1: PROPEDEUTICA leverages a System Call Interception Driver to intercept and log all system calls invoked in the system, and associate them with the PID of the invoking process.

STEP 2: The Reconstruction Module reads and parses logged system calls, compressing the system calls for input to the two classifiers.

STEP 3: The classifiers generate sliding windows of system call traces with the PID of the invoking process.

STEP 4: Conventional ML Classifier introduces a configurable borderline probability interval [lower bound, upper bound], which determines the range of classification that is considered inconclusive for detection. For example, let's consider that the borderline interval is [30%-70%]. If the ML classifier assigns the label "malware" for a piece of software whose classification range is within the predefined borderline interval, PROPEDEUTICA considers its classification result inconclusive. If the software receives a "malware" classification probability less than 30%, PROPEDEUTICA considers that the software is not malicious. If the classification probability is greater than 70%, PROPEDEUTICA considers the software malicious. For the inconclusive case ([30%-70%] interval), the software continues to run, but is subjected to analysis by DeepMalware Classifier for a definitive classification.

STEP 5: If the software is classified as benign by DeepMalware Classifier, it continues running under the monitoring of the Conventional ML classifier.

STEP 6: If the software is considered malicious, PROPEDEUTICA kills the malicious software.

By leveraging the fast speed in ML methods and high accuracy in DL methods, PROPEDEUTICA aims to achieve precise and real-time detection. The main goal of this paper using PROPEDEUTICA is to present the concept of a multi-stage malware detector that combines two distinct classification models—ML and DL. Therefore, any presented implementation should be understood as a proof-of-concept (PoC) to accomplish such a goal, and not as a platform-specific solution. We intend that this PoC leads to a PROPEDEUTICA version that will be implemented in the future by OS vendors to be integrated into their products. Moreover, PROPEDEUTICA does not rely on a GPU-powered system to speed up its detection procedures, which broaden the application of PROPEDEUTICA.

For the sake of simplicity, our PoC was implemented in user land, i.e., as a user process in a non-privileged ring of execution. Thus, PROPEDEUTICA's trusted computing base

includes the OS kernel, the learning models running in user land, and the hardware stack. We modeled PROPEDEUTICA's PoC for Microsoft Windows, since it is the most targeted OS by malware writers [26]. We limited our PoC to the 32-bit Microsoft Windows version, which currently is the most popular Windows architecture.

III. IMPLEMENTATION DETAILS

This section discusses implementation details of three main aspects of PROPEDEUTICA's operation: the system call interception driver, the reconstruction module, and our newly proposed DEEPMALWARE algorithm.

A. The System Call Interception Driver

Intercepting system calls is a key step for PROPEDEUTICA operation as it allows collecting system call invocation data to be input to ML and DL classifiers. Despite many existing tools for process monitoring, such as Process Monitor [27], drstrace library [28], Core OS Tool [29], and WinDbg's Logger [30], they suffer from various challenges: Process Monitor provides only coarse-grained file and registry activity. drstrace works at the application level, and can be bypassed by malware. WinDbg's Logger starts logging only at the entry point of the execution, so system calls executed by the initialization code in statically imported shared libraries will not be seen. Core OS tools trace coarse-grained events, such as interrupts, memory events, thread creation and termination, and etc. Therefore, we opted to implement our own solution to have more flexibility for deploying a real-time detection mechanism. PROPEDEUTICA collection driver was implemented for Windows 7 SP1 32-bit. The driver hooks into the System Service Dispatch Table (SSDT), which contains an array of function pointers to important system service routines, including system call routines. In Windows 7 32-bit system, there are 400 entries of system calls [31], of which 155 system calls are interposed by our driver. We selected a comprehensive set of system calls to be able to monitor multiple, distinct subsystems, which includes network-related system calls (e.g., *NtListenPort* and *NtAlpcConnectPort*), file-related system calls (e.g., *NtCreateFile* and *NtReadFile*), memory-related system calls (e.g., *NtReadVirtualMemory* and *NtWriteVirtualMemory*), process-related system calls (e.g., *NtTerminateProcess* and *NtResumeProcess*), and other types (e.g., *NtOpenSemaphore* and *NtGetPlugPlayEvent*).

This system call interception driver is publicly available at [32]. We open sourced our system call interceptor so that future studies can generate real-time software execution traces, and evaluate the performance of their ML/DL algorithms in malware detection. Our solution can also be used as a baseline for comparison in future work.

System call logs (timestamp, PID, syscall) were collected using DbgPrint Logger [33], which enables real-time kernel message logging to a specified IP address (localhost or remote IP), thus allowing the logging pool and the PROPEDEUTICA to reside on different hosts for scalability and performance. The driver monitors all processes added to the *Borderline_PIDs* list, which keeps track of processes which received borderline classification from the conventional ML classifier. The time to collect system calls varies among different software. The time depends on the functionalities of the software (benign or malicious), and the types of system calls invoked, and the frequency of system call invocations. Hence, the time to collect system calls is software dependent.

B. The Reconstruction Module

In PROPEDEUTICA, both the ML and the DEEPMALWARE classifiers use the RECONSTRUCTION MODULE to preprocess the input system call sequences and obtain the same preprocessed data. The RECONSTRUCTION MODULE splits system calls sequences according to the PIDs of processes invoking them and parses them into three types of sequential data: process-wide n-gram sequence, process-wide density sequence, and system-wide frequency feature vector. Then, the RECONSTRUCTION MODULE converts the sequential data into windows using the sliding window method, which is usually used to translate a sequential classification problem into a classical one for every sliding window [34].

Process-wide n-gram sequence and density sequence. We use the n-gram model, widely used in natural language processing (NLP) applications [35], to compress system call sequences. N-gram is defined as a combination of n contiguous system calls. The n-gram model encodes low-level features with simplicity and scalability and compresses the data by reducing the length of each sequence with encoded information. The process activity in a Windows system can be intensive depending on the workload, e.g., more than 1,000 system calls per second, resulting in very large sliding windows. Therefore, for PROPEDEUTICA, we decided to further compress the system call sequences and translate them into two-stream sequences: n-gram sequences and density sequences. n-gram sequence is a list of n-gram units, while density sequence is the corresponding frequency statistics of repeated n-gram units. There are many possible n-gram variants such as n-tuple, n-bag, and other non-trivial and hierarchical combinations (e.g., tuples of bags, bags of bags, and bags of grams) [9]. PROPEDEUTICA uses 2-gram because (i) compared with n-bag and n-tuple, the n-gram model is considered the most appropriate for malware system call-based classification [9] and (ii) the embedding layer and the first few convolutional neural layers can automatically extract high-level features from neighbor grams, which can be seen as hierarchical combinations of multiple n-grams. Once n-gram sequences fill up a sliding window, the RECONSTRUCTION MODULE delivers the window of sequences to ML classifier and then DL classifier (if in the inconclusive case) redirects the incoming system calls to the new n-gram sequences.

System-wide frequency feature vector. Our learning models leverage system calls as features from all processes running in the system. This holistic, opposite to process-specific approach is more effective for malware detection compared to current approaches because modern malware can be multi-threaded [36], [37]. System-wide information helps the models learn the interactions among all processes in the system. To gain whole system information, the RECONSTRUCTION MODULE collects the frequency of different types of n-grams

from all processes during the sliding window and extracts them as a frequency feature vector. Each element of the vector represents the frequency of an n-gram unit in the system call sequence.

C. DEEPMALWARE *Classifier*

PROPEDEUTICA aims to address spatial-temporal dynamics and software execution heterogeneity. Spatially, advanced malware leverages multiple processes to work together to achieve a long-term common goal. Temporally, a malicious process may demonstrate different types of behaviors (benign or malicious) during the lifetime of execution, such as keep dormant or benign at the beginning of the execution. To thoroughly consider available behavior data in space and time, we introduced a novel DL model, DEEPMALWARE. First, to capture the spatial connections between multiple processes, we feed both process-wide features and system-wide features into DEEPMALWARE. The process-wide and system-wide inputs provide detailed information about how the target software (malware or benign software) interacts with the rest software in the system. Second, to capture the temporal features, we first introduced the bi-directional LSTM [38] to capture the connection of n-grams. However, due to the gradient vanishing problem of LSTM in processing long sequences, we further introduce multi-scale convolutional neural networks to extract high-level representation of n-gram system calls so as to reduce the input length of LSTM and avoid blind spots. To facilitate harmonious coordination, DEEPMALWARE stacks spatial and temporal models for joint training.

System call sequences are taken as input for all processes subjected to DEEPMALWARE analysis (see Figure 2 for a workflow of the classification approach). DEEPMALWARE leverages n-gram sequences of processes and frequency feature vectors of the system. First, two streams (process-wide n-gram sequence and density sequence) model the sequence and density of n-gram system calls of the process. The third stream represents the global frequency feature vector of the whole system. DEEPMALWARE consists of four main components: (i) N-gram Embedding, (ii) ResNext blocks, (iii) Long Short-Term Memory (LSTM) Layers, and (iv) Fully Connected Layers (Figure 3).

N-gram Embedding. DEEPMALWARE adopts an encoding scheme called N-gram Embedding, which converts sparse n-gram units into a dense representation. DEEPMALWARE treats n-gram units as discrete atomic inputs. N-gram Embedding helps to understand the relationship between functionally correlated system calls and provides meaningful information of each n-gram to the learning system. Unlike vector representations such as one-hot vectors (which may cause data sparsity), N-gram Embedding mitigates sparsity and reduces the dimension of input n-gram units. The embedding layer maps sparse system call sequences to a dense vector. This layer also helps to extract semantic knowledge from low-level features (system call sequences) and largely reduces feature dimension. The embedding layer uses 256 neurons, which reduce the dimension of the n-gram model (number of unique n-grams in the evaluation) from 3,526 to 256.

ResNext blocks. DEEPMALWARE uses six ResNext blocks to extract different sizes of n-gram system calls in the sliding window. Conventional sliding window methods leverage small windows of system call sequences and, therefore, experience challenges when modeling long sequences. These conventional methods represent features in a rather simple way (e.g., only counting the total number of system calls in the windows without local information of each system calls), which may be inadequate for a classification task. The design of ResNext follows the implementation in [39]. Each ResNext block aggregates multiple sets of convolutional neural networks with the same topology. ResNext blocks extract different n-grams to avoid blind spots in detection. Residual blocks perform short cuts between blocks to improve the training of deep layers. Batch normalization is applied to speed up the training process after convolutional layers and a non-linear activation function, ReLU to avoid saturation.

Long Short-Term Memory (LSTM) Layers. The internal dependencies between system calls include meaningful context information or unknown patterns for identifying malicious behaviors. To leverage this information, DEEPMALWARE adopts one of the recurrent architectures, LSTM [38], for its strong capability of learning meaningful temporal/sequential patterns in a sequence and reflecting internal state changes modulated by the recurrent weights. LSTM networks can avoid gradient vanishing and learn long-term dependencies by using forget gates. LSTM layers gather information from the first two streams: process-wide n-gram sequence and density sequence.

Fully Connected Layers. DEEPMALWARE deploys a fully connected layer is deployed to encode system-wide frequency. Then, it is concatenated with the output of bi-directional LSTMs to gather both sequence-based process information and frequency-based system-wide information. The last fully-connected layer with the softmax function outputs the prediction with probabilities.

Moreover, we adopt three popular techniques to avoid overtraining in DEEPMALWARE. 1) We add a Dropout layer [40] with a dropout rate of 0.5 following the fully connected layer; 2) We include batch normalization [41] in the ResNext blocks after the convolutional layers; 3) We use 20% training data as a validation set and apply early stopping when the loss on the validation set does not decrease.

D. Conventional ML Classifier

We consider three pervasively used conventional ML algorithms: Random Forest (RF), eXtreme Gradient Boosting (XGBoost) [42], and Adaptive Boosting (AdaBoost) [43]. Random Forest and boosted trees have been considered as the best supervised models on high-dimensional data [44]. We use AdaBoost and XGBoost as representatives of boosted trees.

To select the best features used in machine learning classifiers, we consider both efficacy and effectiveness. In PROPEDEUTICA, we take the frequency of n-gram units in a sequence as input features and estimate the probabilities of a sequence of system calls invoked by a malware. Specifically, a feature vector will be created for each sequence, and each element in the vector represents the frequency of the corresponding

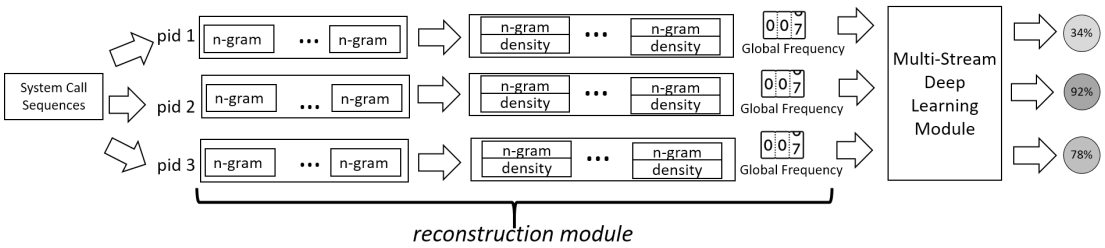


Fig. 2: Workflow of DEEPMALWARE classification approach.

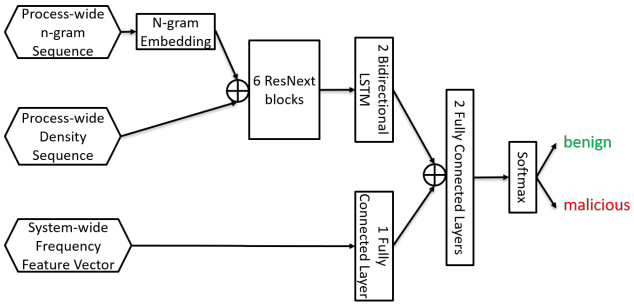


Fig. 3: DEEPMALWARE architecture.

n-gram unit. We discuss the details of ML classifiers in Section IV.

Notice that ML classifiers in PROPEDEUTICA can be generalized to other ML algorithms, when ML classifiers can output classification probabilities for borderline classification (i.e., probabilistic ML models). For Non-probabilistic ML classifiers, we will adopt a conversion function to provide class probabilities. For example, the classification scores provided by Support Vector Machine can be mapped into the class probabilities via a logistic transformation with trained parameters [45]. K Nearest Neighbor (kNN) methods can be extended with probabilistic methods for estimating the posterior probabilities using distances from neighbors [46]. The classification probabilities of ML classifiers are used for the borderline classification (Fig. 1 Step 4). It is worth to note that PROPEDEUTICA is designed to integrate with any machine learning and deep learning methods in malware detection with minimal efforts. Therefore, we select commonly used and representative algorithms to show PROPEDEUTICA's capability in plugging in these algorithms.

IV. EVALUATION

The main goal of our evaluation is to determine PROPEDEUTICA's effectiveness for real-time malware detection. More specifically, we sought to discover to what extent PROPEDEUTICA outperforms ML on prediction performance and DL on classification time.

We first describe the dataset used in our evaluation. Next, we evaluate the performance of our newly proposed DEEPMALWARE algorithm in comparison with the most relevant ML classifiers from the literature. Then, we present the results of our experiments on the proposed PROPEDEUTICA. In our evaluation, we compare the performance of classifiers when operating on a CPU and a GPU. GPUs not only reduce the training time for DL models but also help these models achieve classification time several orders of magnitude less than conventional CPUs in the deployment.

We performed our test on a server running Ubuntu 12.04 with 2 CPUs, 4GB Memory, 60GB Disk. DEEPMALWARE was implemented using Pytorch [47]. The training and testing procedures ran on Nvidia Tesla M40 GPUs.

A. Software Collection Method

For the malicious part, the dataset consisted of 9,115 Microsoft Windows PE32 format malware samples collected between 2014 and 2018 from threats detected by the incident response team of a major financial institution (who chose to remain anonymous) in the corporate email and Internet access links of the institution's employees, and 7 APTs collected from Rekings [48]. Figure 4 shows the categorization of our malware samples using AVClass [49][2]. The goal of malware collection is to be as broad as possible. The distribution of our collected malware family (Figure 4) correlates to the distribution of malware detected by AV companies. For the benign part, our dataset was composed of 1,338 samples, including 866 Windows benchmark software, 50 system software, 11 commonly used GUI-based software, and 409 other benign software. GUI-based software (e.g., Chrome, Microsoft Office, Video Player) had keyboard and mouse operations simulated using WinAutomation [50].

B. System Call Dataset

For each experiment, we run malware and benign software and collect system-wide system calls for five minutes. The experiments were carried out under three different scenarios: (1) running one general malware, one benchmark/GUI-based/other benign software, and dozens of system software. (2) running two general malware, several benchmark/GUI-based/other benign software, and dozens of system software. (3) running one APT, one benchmark/GUI-based/other benign software, and dozens of system software.

We randomly sampled half of the malware and benign software for training and the other half for testing, to avoid overfitting [51]. We collected 493,095 traces in total. During the running of one malware sample, multiple benign processes (especially system processes) would run concurrently. Therefore, the number of benign process executions we collected would be much larger than that of malicious process executions. Our classifier would end up handling imbalanced datasets,

[2]AVClass is an automatic, vendor-agnostic malware labeling tool. It provides a fine-grained family name rather than a generic label (e.g., Trojan, Virus, etc.), which contains scarce meaning for the malware sample.

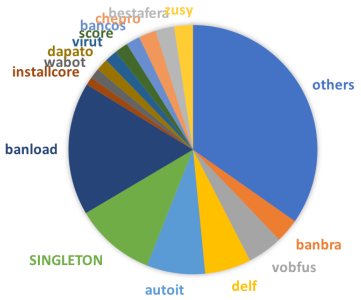


Fig. 4: Malware families in our dataset classified by AVClass.

which could adversely affect the training procedure, because the classifier would be prone to learn from the majority class. Hence, we under-sampled the dataset by reducing the number of sliding windows from benign processes to the same as malware processes, so that the ML and DL detectors can learn from the same number of positive and negative samples.

Window size and stride are two important hyper-parameters in sliding window methods, indicating the total and the shifting number of n-gram system calls in each process. In real-time detection, large window sizes need more time to fill up the sliding window, provide more context for DL models, and therefore achieve higher accuracy. We aim to choose the proper two hyper-parameters to ensure malware are detected accurately and in time (before completing its malicious behavior). In our experiments, we observed that malware can successfully infect the systems in one to two seconds, and 1,000 system calls can be invoked in one second. Therefore, we selected and compared three window size and stride pairs: (100, 50), (200, 100), (500, 250). The results showed that with pair (500, 250), DEEPMALWARE needs 1 to 2 seconds to fill up the window and analyze. With pair (200, 100), DEEPMALWARE needs 0.5 to 1 second. While with pair (100, 50), DEEPMALWARE needs only 0.1 to 0.5 seconds. The F1-scores for DEEPMALWARE are 97.02% , 95.08% and 94.97% respectively.

Although the detection performance is increased with the increase of window sizes, large window sizes indicate longer wait time for the detector to allow the software to be executed and fill up the windows. During this long wait time, the malware are more likely to conduct malicious behaviors. Therefore, a proper window size needs to be carefully selected so as to balance the prediction performance and the wait time. Since the performance gain with respect to the increased window size is marginal (around 2% increase in terms of F1 score from 500 to 100), while the waiting time can be reduced to 10%-20%, we set the window size and stride as 100 and 50, respectively, for the following evaluation. In practice, the value of window sizes can be altered in our PROPEDEUTICA framework to meet the required detection performance (e.g., a required accuracy, or an acceptable FP rate).

C. Performance Analysis of standalone ML and DL classifiers

We started by comparing the performance of DEEPMALWARE with relevant ML and DL classifiers. Our metrics for model performance were accuracy, precision, recall, F1 score, and false positive (FP) rate.

Table I shows the detection performance and execution time of using a single ML or DL model. DEEPMALWARE achieves the best performance among all the models. With window size 500 and stride 250, DEEPMALWARE achieved an accuracy of 97.03% and a false positive rate of 2.43%. The performance of DEEPMALWARE increased with the increase in window size and stride. Random Forest performs the second best with an accuracy of 89.05% at window size 100 and stride 50, and an accuracy of 93.94% at window size 500 and stride 250, approximately 10% better than AdaBoost and 5% better than XGBoost. Random Forest also achieves the fastest speed among all the machine learning methods. While DEEPMALWARE model is 3.09% more accurate than Random Forest, its classification time is approximately 100 times slower on CPU and 3 to 5 times slower on GPU.

We did not apply GPU devices to ML models because their classification time with conventional CPUs is much smaller (at least one order of magnitude) than those measured for DL algorithms. We denote 'N/A' as not applicable in Table I. In real life, GPU devices, especially specific DL GPUs for accelerating DL training/testing are still not sufficiently widespread in end-user or corporate devices. In addition, the preprocessing time of Reconstruction Module is around 1.098 seconds per run or 0.0003 seconds per window (when using 100 window size), which is almost negligible compared with ML (0.0088 seconds per window) or DL execution time (0.0383 seconds per window using GPU or 1.4104 seconds per window using CPU).

We observed that malware could successfully infect the target system in one to two seconds, and 1,000 system calls can be invoked in one second. Moreover, traces from one process may take a significant amount of time to fill up one sliding window because the process might be invoking system calls at a low rate. Thus, in the remainder of this section, we chose to compare all the algorithms using window sizes of 100 and strides of 50 system calls. For all window sizes and strides, the best ML model was Random Forest. Therefore, for PROPEDEUTICA, we chose Random Forest for ML classification and DEEPMALWARE for DL classification.

In addition, we evaluated PROPEDEUTICA's performance on seven APTs with various borderline intervals. PROPEDEUTICA successfully detected all the APTs and six of them were subjected to DEEPMALWARE classification. For different borderline intervals, the false positive rate was approximately 10%. Random Forest in isolation detected one APT with a 52.5% false positive rate.

D. Performance Analysis of PROPEDEUTICA, combining ML and DL classifiers

Next, we evaluated PROPEDEUTICA's performance for different borderline intervals. Based on the results from Table I, we chose Random Forest as ML classifier and DEEPMALWARE as DL classifier with window size 100 and stride 50. PROPEDEUTICA is evaluated based on various borderline intervals. As discussed before, in PROPEDEUTICA, if a piece of software receives a malware classification probability from Random Forest smaller than the lower bound, it is considered

TABLE I: Detection performance of standalone ML or DL classifiers. DEEPMALWARE achieves the best performance among all the models. Random Forest is approximately 8% more accurate and much faster than the other machine learning models.

Models	Window Size	Stride	Accuracy	Precision	Recall	F1 Score	FP Rate	Detection Time with GPU (s)	Detection Time with CPU (s)
AdaBoost	100	50	79.25%	78.11%	81.31%	79.67%	22.80%	N/A	0.0187
Random Forest			**89.05%**	**84.63%**	**95.44%**	**89.71%**	**17.35%**	**N/A**	**0.0089**
XGBoost			84.78%	90.99%	77.22%	83.54%	7.66%	N/A	0.0116
DEEPMALWARE			**94.84%**	**92.63%**	**97.43%**	**94.97%**	**7.76%**	**0.0383**	**1.4104**
AdaBoost	200	100	78.27%	72.84%	90.19%	80.59%	33.64%	N/A	0.0132
Random Forest			**93.49%**	**89.99%**	**97.90%**	**93.77%**	**10.91%**	**N/A**	**0.0063**
XGBoost			70.81%	63.65%	97.08%	76.89%	70.81%	N/A	0.0073
DEEPMALWARE			**94.96%**	**93.05%**	**97.20%**	**95.08%**	**7.27%**	**0.0543**	**1.3784**
AdaBoost	500	250	81.81%	79.44%	85.86%	82.52%	22.24%	N/A	0.0108
Random Forest			**93.94%**	**97.16%**	**90.54%**	**93.73%**	**2.65%**	**N/A**	**0.0048**
XGBoost			79.19%	94.14%	62.27%	74.95%	3.88%	N/A	0.0062
DEEPMALWARE			**97.03%**	**97.54%**	**96.50%**	**97.02%**	**2.43%**	**0.0797**	**1.3344**

TABLE II: Comparison on different borderline policies for the PROPEDEUTICA combining ML and DL classifiers. Borderline intervals are described with a lower bound and an upper bound. The move percentage represents the percentage of software in the system that received a borderline classification with Random Forest (according to the borderline interval) and was subjected to further analysis with DEEPMALWARE.

Lower Bound	Upper Bound	Accuracy	Precision	Recall	F1 Score	FP Rate	Detection Time with GPU (s)	Detection Time with CPU (s)	Move Percentage
10%	90%	94.71%	92.41%	97.43%	94.85%	8.00%	0.0223	0.381	55.95%
20%	80%	94.61%	92.23%	97.43%	94.76%	8.21%	0.0173	0.1543	48.32%
30%	70%	94.34%	91.77%	97.43%	94.51%	8.75%	0.0146	0.0884	41.45%
40%	60%	93.72%	90.79%	97.37%	93.96%	9.92%	0.0123	0.0497	34.96%

benign software. If the probability is greater than the upper bound, it is considered malicious. However, if the classification falls within the borderline interval, the software is subjected to DEEPMALWARE. Table II shows the performance of PROPEDEUTICA using various (configurable) borderline intervals when combining ML and DL methods. We chose lower bound as 10%, 20%, 30%, and 40%, upper bound as 60%, 70%, 80%, and 90%. We observe that when a small borderline interval is configured (e.g., 30%-70%, 40%-60%), the detection time is reduced to less than 0.1 seconds even using CPU, with minimal degradation of prediction performance. With a borderline interval of [30%-70%], PROPEDEUTICA achieved an accuracy of 94.34% and a false positive rate of 8.75%, with 41.45% of the samples moved for DEEPMALWARE analysis. This highlights the potential of PROPEDEUTICA: approximately 60% of the samples were quickly classified with high accuracy as malware or benign software using an initial triage with faster Random Forest. Only 40% of the samples needed to be subjected to a more expensive analysis using DEEPMALWARE. PROPEDEUTICA achieves high prediction performance while making it feasible to use expensive DL algorithms in real-time malware detection.

E. Comparison with Other DL Malware Detectors

We compared PROPEDEUTICA with other DL-based malware detectors in the literature leveraging system/API calls as features. The features used in our work and [23] are Windows system calls. The features used in [21] and [22] are Windows API calls. System calls provide more insights on the software at the kernel level, compared with API calls collected at the user level. Moreover, we evaluated the detection performance on a much larger dataset with more traces, which provides more accurate evaluation results.

In Table III, we summarize the obtained results. Compared with the existing state-of-the-art DL-based malware detectors, our approaches achieve much better detection performance in terms of accuracy, precision, and recall. More importantly, current literature mainly focuses on offline analysis [21]–[23] and lacks analysis of the detection time in real-time. We evaluated DEEPMALWARE on a much larger dataset with better prediction performance and detection time guarantee.

V. DISCUSSION

We proposed a real-time malware detection framework, PROPEDEUTICA, accomplishing the fast speed of ML and the high accuracy of emerging DL models. Only software receiving borderline classification from an ML detector needed further analysis by DEEPMALWARE, which saved computational resources, shortened the detection time, and improved accuracy. The key contribution of PROPEDEUTICA is the intelligent combination of ML with emerging modern DL methods in an effective real-time detector, which can meet the needs of high accuracy, low false-positive rate, and short detection time required to counter the next generation of malware. In this section, we discuss several challenges in the current design and potential directions of improving PROPEDEUTICA in future work.

A. Recurrent Loop in PROPEDEUTICA

PROPEDEUTICA can run into a worst-case scenario, which is having a process continuously loop between Random Forest

TABLE III: Comparison among DEEPMALWARE and other DL-based malware detection in the literature.

Work	Model	Dataset	Performance
DMIN'16 [21]	DL4MD	45,000 traces	Accuracy 95.65%, Precision 95.46%, Recall 95.8%
KDD'17 [22]	DNN	26,078 traces	Accuracy 93.99% [3]
LNCS'16 [23]	LSTM	4,753 traces	Accuracy 89.4%, Precision 85.6%, Recall 89.4%
PROPEDEUTICA	DEEPMALWARE	493,095 traces	Accuracy 97.0%, Precision 97.5%, Recall 97.0%

and DEEPMALWARE. For example, consider a process receiving a borderline classification from Random Forest, and then being moved to DEEPMALWARE. Then DEEPMALWARE classifies it as benign, and the software would continue being analyzed by Random Forest, which again provides a borderline classification for the process, and so on. We plan to mitigate such recurrent processes by combining the previous prediction of DEEPMALWARE with the prediction of Random Forest in the first stage.

B. Adversarial Attacks against Malware Detection

Despite initial successes on malware classification, a resourceful and motivated adversary can bypass the protection mechanisms using evasion attacks. A plethora of recent studies have demonstrated that ML and DL models are vulnerable to evasion attacks, such as *adversarial examples* [24], [25], [52], [53]. By adding a small perturbation to the input features of machine learning models, adversaries can mislead the detection system to classify a malware as a benign software [54]–[61]. However, most adversarial attacks target at static malware detection: malware is not executed and machine learning conducts detection on the static code. For example, Park etal. injected dummy code into malware source code using an obfuscation technique [57]. Liu etal. manipulated the malware images generated from malware binaries using adversarial attacks [56]. Only [62] generated adversarial sequences to attack dynamic behavior-based malware detection systems. However, their victim detection model is much simpler than our proposed model. In our work, we propose a framework for dynamic malware detection, where malware is investigated during execution, which makes the adversarial attacks much easier to detect, since it is challenging to generate a sequence of system/API calls with normal behaviors. In the future, we plan to thoroughly investigate the robustness of PROPEDEUTICA against adversarial attacks.

C. Weakly Supervised Anomaly Detection

In our work, we assume that the malware samples are well annotated and ML and DL models are trained on these samples, i.e., fully supervised anomaly detection. However, the malware samples have been generated and evolved rapidly in recent years. It becomes impractical to analyze and identify all the malware samples for model training. Therefore, a weakly-supervised learning paradigm [63] is urgently needed for malware detection, where only a limited number of labeled positive samples are provided and a large number of positive samples are not labeled. Although unsupervised learning approaches leveraging Autoencoders and generative adversarial networks (GANs) can be used in weakly supervised learning by learning features that are robust to small deviations in negative samples [64]–[69], the labeled positive samples are not fully utilized. Recently, Pang et al. proposed DevNet and PRO to learn anomaly scores using deep learning models for better use of labeled positive data [70], [71]. The proposed framework in PROPEDEUTICA can be further extended with the capabilities of learning from weakly labeled samples using these advanced weakly-supervised approaches.

[3]The performance in terms of precision and recall is not provided in [22].

VI. RELATED WORK

Our work pertains to fields related to behavior-based malware detection. In this section, we summarize the state-of-the-art in these areas and highlights topics currently under-studied.

A. Behavior-based Malware Detection

Dynamic behavior-based malware detection [5], [15] evolved from Forrest et al.'s seminal work [34] on detecting anomalies using system calls as features. Christodorescu et al. [5], [6] extract high-level and unique behavior from the disassembled binary to detect malware and its variants. The detection is based on predefined templates covering potentially malicious behaviors, such as mass-mailing and unpacking. Willems et al. proposed CWSandbox, a dynamic analysis system that monitors malware API calls in a virtual environment [72]. Rieck et al. [73] leveraged API calls as features to classify malware into families using Support Vector Machines (SVM), and processed system calls into q-grams representations using an algorithm similar to the k-nearest neighbors (kNN) [14]. Mohaisen et al. [74] introduced AMAL, a framework to dynamically analyze and classify malware using SVM, linear regression, classification trees, and kNN. Kolosnjaji et al. [75] proposed maximum-a-posteriori (MAP) to classify malware families using Cuckoo's Sandbox [76]. Although these techniques were successfully applied for malware classification, they did not consider benign samples [77], and were limited to labeling unknown malware pertaining to one of the existing clusters.

Wressnegger et al. [78] proposed Gordon for detecting Flash-based malware. Gordon considered execution behavior from benign and malicious Flash applications and generated *n-grams* for an SVM model. Bayer et al. used a modified version of QEMU to monitor API calls and breakpoints [10]. This approach was later used to build Anubis [79]. Anubis is better suited for offline malware analysis (by providing a detailed execution report). PROPEDEUTICA, on the other hand, focuses on real-time detection and monitors a more diverse set of system calls. Kirat et al. introduced BareBox, which hooked system calls directly from SSDT [80]. Barebox runs in bare metal, and can potentially obtain behavioral traces from malware equipped with anti-analysis techniques. Barebox aims

in live system restoring and analyzed on only 42 malware samples. PROPEDEUTICA also uses system call hooking to monitor software behavior, but is larger scale and uses a testbed to run malware automatically. Anubis [79] works best for offline malware analyzers by providing a detailed execution report while PROPEDEUTICA delivers on-thy-fly detection results to end users. PROPEDEUTICA improves the experiment of Anubis by monitoring more kinds of system calls, running a longer time for each experiment, and using cutting-edge DL models to help with the detection.

Some lines of work focus on developing tools to detect evasive malware [81]. PROPEDEUTICA is resilient to anti-analysis because its monitoring mechanism operates at the kernel level. Other works using tainting techniques, e.g., VMScope [82], TTAnalyze [10], and Panorama [36], emulate environments with QEMU, and trace the data-flow from whole-system processes. PROPEDEUTICA differs from such works by monitoring software behavior through system-wide system call hooking instead of data tainting, while maintaining the interactions among different processes in a lightweight manner. Contrary to PROPEDEUTICA, such tools might encounter challenges when deployed in practice. For example, Panorama [36] relies on a human to manually label address and control dependency tags that should be propagated. Canali et al. evaluated different types of models for malware detection. Their findings confirm that the accuracy of some widely used learning models is poor, independently of the values of their parameters, and high-level atoms with arguments model performs the best, which corroborates our experimental results. Further, the paper points out that on-the-fly detection suffers from even higher false-positive rates because of the diversity of applications and the system calls invoked [15]. All these findings corroborate the results of our paper.

B. ML-based Malware Detection

Xie et al. proposed a one-class SVM model with different kernel functions [83] to classify anomalous system calls in the ADFA-LD dataset [84]. Ye et al. proposed a semi-parametric classification model for combining file content and file relation information to improve the performance of file sample classification [85]. Abed et al. used bags of system calls to detect malicious applications in Linux containers [86]. Kumar et al. used K-means clustering [87] to differentiate legitimate and malicious behaviors based on the NSL-KDD dataset. Fan et al. used a sequence mining algorithm to discover malicious sequential patterns and trained an All-Nearest-Neighbor (ANN) classifier based on these discovered patterns for malware detection [88].

The Random Forest algorithm has been applied to classification problems as diverse as offensive tweets, malware detection, de-anonymization, suspicious Web pages, and unsolicited email messages [89], [90]. Conventional ML-based malware detectors suffer, however, from high false-positive rates because of the diverse nature of system calls invoked by applications, as well as the diversity of applications [15].

C. DL-based Malware Detection

There are recent efforts to apply DL for malware detection. Deep learning has made great successes in speech recognition [91], language translation [92], speech synthesis [93], and other sequential data [94]. Li et al. proposed a hybrid malicious code classification model based on AutoEncoder and DBN and acquired a relatively high classification rate on the part of the now outdated KDD99 dataset [95]. Pascanu et al. first applied deep neural networks (recurrent neural networks and echo state networks) to model the sequential behaviors of malware. They collected API system call sequences and C run-time library calls and detected malware as a binary classification problem [96]. David et al. [97] used a deep belief network with denoising autoencoders to automatically generate malware signatures for classification. Saxe and Berlin [98] proposed a DL-based malware detection technique with two-dimensional binary program features and provided a Bayesian model to calibrate detection. Huang and Stokes [99] designed a neural network to extract high-level features and classify them into both benign/malicious and also 100 malware families based on shared features. Dahl et al. detected malware files by neural networks and used random projections to reduce the dimension of sparse input features [100]. Wang [101] extracted features from Android Manifest and API functions to be the input of the deep learning classifier. Hou et al. collected the system calls from the kernel and then constructed the weighted directed graphs and use DL framework to make dimension reduction [102]. All these DL-based methods rely on handcrafted features. Recently, Kolosnjaji et al. [23] proposed a DL method to detect and predict malware families based on system call sequences. Our proposed DEEPMALWARE has shown superior performance over their approaches by coping the spatial-temporal features with ResNext designs. To cope with the fast-evolving nature of malware, Transcend [103] addresses concept drift in malware classification. Transcend can detect aging machine learning models before their degradation. As future work, we plan to leverage Transcend's concept drift model in PROPEDEUTICA for re-training.

Our work addressed the performance of DL-based malware detection algorithms running in a real system. Compared with the recent work ([21]–[23]), our experiments show that deep learning performs fewer false-positive samples compared with conventional machine learning models. Meanwhile, conventional machine learning algorithms run about 2 to 3 orders of magnitude faster than deep learning ones.

VII. CONCLUSION

In this paper, we introduced PROPEDEUTICA, a novel paradigm and framework for real-time malware detection, which combines the best of conventional machine learning and deep learning techniques using multi-stage classification. In PROPEDEUTICA, all software in the system start execution subjected to a conventional machine learning detector for fast classification. If a piece of software receives a borderline classification, it is subjected to further analysis via more performance expensive and more accurate deep learning methods, via our novel algorithm DEEPMALWARE. DEEPMALWARE utilizes both process-wide and system-wide system calls and predicts malware in both short and long-term ways. With a borderline interval of [30%-70%], PROPEDEUTICA achieved

an accuracy of 94.34% and a false positive rate of 8.75%, with 41.45% of the samples moved for DEEPMALWARE analysis. The detection time using GPU is 0.0146 seconds on average. Even using CPU, the detection time is less than 0.1 seconds.

On one hand, our work provided evidence that conventional machine learning and emerging deep learning methods in isolation might be inadequate to provide both high accuracy and short detection time required for real-time malware detection. On the other hand, our proposed PROPEDEUTICA combines the best of ML and DL methods and has the potential to change the design of the next generation of practical real-time malware detectors.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No 1801599. This material is based upon work supported by (while serving at) the National Science Foundation.

REFERENCES

[1] A. Calleja, J. Tapiador, and J. Caballero, "A look into 30 years of malware development from a software metrics perspective," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 325–345.

[2] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "Triggerscope: Towards detecting logic bombs in android applications," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 377–396.

[3] G. Vigna and R. A. Kemmerer, "NetSTAT: A Network-Based Intrusion Detection Approach," 1998.

[4] BROMIUM, INC. (2010) Bromium end point protection. [Online]. Available: https://www.bromium.com

[5] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantics-aware malware detection," in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, ser. SP '05, 2005.

[6] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC-FSE '07, 2007, pp. 5–14.

[7] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Detecting malicious code by model checking," *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 3548, pp. 174–187, 2005.

[8] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proceedings of the 18th Conference on USENIX Security Symposium*, ser. SSYM'09, 2009, pp. 351–366.

[9] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "A quantitative study of accuracy in system call-based malware detection," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA 2012, 2012, pp. 122–132.

[10] U. Bayer, C. Kruegel, and E. Kirda, "Ttanalyze: A tool for analyzing malware," in *15th European Institute for Computer Antivirus Research (EICAR)*, 2006.

[11] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of computer security*, vol. 6, no. 3, pp. 151–180, 1998.

[12] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering." in *NDSS*, vol. 9, 2009, pp. 8–11.

[13] S. Revathi and A. Malathi, "A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research and Technology. ESRSA Publications*, 2013.

[14] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[15] A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, "Accessminer: using system-centric models for malware protection," in *Proceedings of the 17th ACM conference on Computer and communications security*, 2010, pp. 399–412.

[16] Palo Alto Networks. (2013, March) The modern malware review. [Online]. Available: https://media.paloaltonetworks.com/documents/The-Modern-Malware-Review-March-2013.pdf

[17] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, no. 1, pp. 69–101, 1996.

[18] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[19] Y. Bengio, Y. LeCun *et al.*, "Scaling learning algorithms towards ai," *Large-scale kernel machines*, vol. 34, no. 5, pp. 1–41, 2007.

[20] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[21] W. Hardy, L. Chen, S. Hou, Y. Ye, and X. Li, "Dl4md: A deep learning framework for intelligent malware detection," in *Proceedings of the International Conference on Data Mining (DMIN)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2016, p. 61.

[22] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia II, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1145–1153.

[23] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[24] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[25] J. Zhang and C. Li, "Adversarial examples: Opportunities and challenges," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 7, pp. 2578–2593, 2019.

[26] I. Arghire, "Windows 7 most hit by wannacry ransomware," http://www.securityweek.com/windows-7-most-hit-wannacry-ransomware, 2017.

[27] M. Russinovich, "Process monitor v3.40," https://technet.microsoft.com/en-us/sysinternals/bb896645.aspx, 2017.

[28] drmemory, "drstrace," http://drmemory.org/strace_for_windows.html, 2017.

[29] A. B. Insung Park, "Core os tools," https://msdn.microsoft.com/en-us/magazine/ee412263.aspx, 2009.

[30] Microsoft, "Windbg logger," https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/logger-and-logviewer, 2017.

[31] M. Jurczyk. (2017) Ntapi. [Online]. Available: http://j00ru.vexillium.org/syscalls/nt/32/

[32] anonymity. (2018) Propedeutica driver publicly available at <https://github.com/gracesrm/windows-system-call-hook>.

[33] A. A. Telyatnikov. (2016) DbgPrint Logger. [Online]. Available: https://alter.org.ua/soft/win/dbgdump/

[34] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff., "A sense of self for Unix processes," in *IEEE Security and Privacy*, 1996, pp. 120–128.

[35] C. D. Manning and H. Schütze, *Foundations of statistical natural language processing*. MIT press, 1999.

[36] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *ACM conference on computer and communications security*, 2007, pp. 116–127.

[37] B. Caillat, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna, "Prison: Tracking process interactions to contain malware," in *IEE 17th High Performance Computing and Communications (HPCC), IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), and IEEE 12th International Conference on Embedded Software and Systems (ICESS)*. IEEE, 2015, pp. 1282–1291.

[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[39] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1492–1500.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[42] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, pp. 785–794.

[43] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European conference on computational learning theory*. Springer, 1995, pp. 23–37.

[44] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 96–103.

[45] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, 1999, pp. 61–74.

[46] C. Holmes and N. Adams, "A probabilistic nearest neighbour method for statistical pattern recognition," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 64, no. 2, pp. 295–306, 2002.

[47] PyTorch. (2018). [Online]. Available: http://pytorch.org

[48] Rekings. (2018) Security through insecurity. [Online]. Available: https://rekings.org/

[49] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.

[50] Winautomation. (2017) Powerful desktop automation software. [Online]. Available: http://www.winautomation.com

[51] M. B. Christopher, *PATTERN RECOGNITION AND MACHINE LEARNING*. Springer-Verlag New York, 2016.

[52] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP)*. IEEE, 2017, pp. 39–57.

[53] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[54] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[55] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[56] X. Liu, J. Zhang, Y. Lin, and H. Li, "Atmpa: Attacking machine learning-based malware visualization detection methods via adversarial examples," in *2019 IEEE/ACM 27th International Symposium on Quality of Service (IWQoS)*. IEEE, 2019, pp. 1–10.

[57] D. Park, H. Khan, and B. Yener, "Generation & evaluation of adversarial examples for malware obfuscation," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 1283–1290.

[58] B. Chen, Z. Ren, C. Yu, I. Hussain, and J. Liu, "Adversarial examples for cnn-based malware detectors," *IEEE Access*, vol. 7, pp. 54 360–54 371, 2019.

[59] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2019, pp. 8–14.

[60] M. Ebrahimi, N. Zhang, J. Hu, M. T. Raza, and H. Chen, "Binary black-box evasion attacks against deep learning-based static malware detectors with adversarial byte-level language model," *arXiv preprint arXiv:2012.07994*, 2020.

[61] J. Yuan, S. Zhou, L. Lin, F. Wang, and J. Cui, "Black-box adversarial attacks against deep learning based malware binaries detection with gan," in *ECAI 2020*. IOS Press, 2020, pp. 2536–2542.

[62] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Adversarial examples on discrete sequences for beating whole-binary malware detection," *arXiv preprint arXiv:1802.04528*, 2018.

[63] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National science review*, vol. 5, no. 1, pp. 44–53, 2018.

[64] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[65] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 2019, pp. 594–602.

[66] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *International conference on information processing in medical imaging*. Springer, 2017, pp. 146–157.

[67] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Asian conference on computer vision*. Springer, 2018, pp. 622–637.

[68] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar, "Adversarially learned anomaly detection," in *2018 IEEE International conference on data mining (ICDM)*. IEEE, 2018, pp. 727–736.

[69] Y. Zhou, X. Song, Y. Zhang, F. Liu, C. Zhu, and L. Liu, "Feature encoding with autoencoders for weakly supervised anomaly detection," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2021.

[70] G. Pang, C. Shen, H. Jin, and A. v. d. Hengel, "Deep weakly-supervised anomaly detection," *arXiv preprint arXiv:1910.13601*, 2019.

[71] G. Pang, C. Shen, and A. van den Hengel, "Deep anomaly detection with deviation networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 353–362.

[72] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[73] K. Rieck, T. Holz, C. Willems, P. Düssel, and P. Laskov, "Learning and classification of malware behavior," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 108–125.

[74] A. Mohaisen, O. Alrawi, and M. Mohaisen, "Amal: High-fidelity, behavior-based automated malware analysis and classification," *Computers & Security*, vol. 52, pp. 251 – 266, 2015.

[75] B. Kolosnjaji, A. Zarras, T. Lengyel, G. Webster, and C. Eckert, "Adaptive semantics-aware malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 419–439.

[76] C. Guarnieri, "Cuckoo sandbox," https://www.cuckoosandbox.org/, 2017.

[77] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *International Conference on Recent Advances in Intrusion Detection (RAID)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 178–197.

[78] C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Comprehensive analysis and detection of flash-based malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment DIMVA*, 2016, pp. 101–121.

[79] Anubis, "Analyzing unknown binaries," http://anubis.iseclab.org., 2010.

[80] D. Kirat, G. Vigna, and C. Kruegel, "Barebox: efficient malware analysis on bare-metal," in *Proceedings of the 27th Annual Computer Security Applications Conference*. ACM, 2011, pp. 403–412.

[81] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna, "Efficient detection of split personalities in malware." in *Network and Distributed System Security Symposium*, 2010.

[82] N. M. Johnson, J. Caballero, K. Z. Chen, S. McCamant, P. Poosankam, D. Reynaud, and D. Song, "Differential slicing: Identifying causal execution differences for security applications," in *Security and Privacy (SP)*. IEEE, 2011, pp. 347–362.

[83] M. Xie, J. Hu, and J. Slay, "Evaluating host-based anomaly detection systems: Application of the one-class svm algorithm to adfa-ld," in *Fuzzy Systems and Knowledge Discovery (FSKD)*. IEEE, 2014, pp. 978–982.

[84] G. Creech and J. Hu, "Generation of a new ids test dataset: Time to retire the kdd collection," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2013, pp. 4487–4492.

[85] Y. Ye, T. Li, S. Zhu, W. Zhuang, E. Tas, U. Gupta, and M. Abdulhayoglu, "Combining file content and file relations for cloud based malware detection," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 222–230.

[86] A. S. Abed, T. C. Clancy, and D. S. Levy, "Applying bag of system calls for anomalous behavior detection of applications in linux containers," in *2015 IEEE Globecom Workshops*. IEEE, 2015, pp. 1–5.

[87] V. Kumar, H. Chauhan, and D. Panwar, "K-means clustering approach to analyze nsl-kdd intrusion detection dataset," *International Journal of Soft*, 2013.

[88] Y. Fan, Y. Ye, and L. Chen, "Malicious sequential pattern mining for automatic malware detection," *Expert Systems with Applications*, vol. 52, pp. 16–25, 2016.

[89] D. Chatzakou, N. Kourtellis, J. Blackburn, E. De Cristofaro, G. Stringhini, and A. Vakali, "Mean birds: Detecting aggression and bullying on twitter," 2017.

[90] E. Mariconti, L. Onwuzurike, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *Network and Distributed System Security Symposium*, 2017.

[91] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International Conference on Machine Learning*, 2016, pp. 173–182.

[92] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[93] S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[94] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[95] Y. Li, R. Ma, and R. Jiao, "A hybrid malicious code detection method based on deep learning," *methods*, vol. 9, no. 5, 2015.

[96] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 1916–1920.

[97] O. E. David and N. S. Netanyahu, "Deepsign: Deep learning for automatic malware signature generation and classification," in *Neural Networks (IJCNN)*, 2015, pp. 1–8.

[98] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE, 2015, pp. 11–20.

[99] W. Huang and J. W. Stokes, "Mtnet: a multi-task neural network for dynamic malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 399–418.

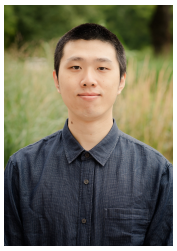
[100] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, 2013, pp. 3422–3426.

[101] Z. Wang, J. Cai, S. Cheng, and W. Li, "Droiddeeplearner: Identifying android malware using deep learning," in *Sarnoff Symposium, 2016 IEEE 37th*. IEEE, 2016, pp. 160–165.

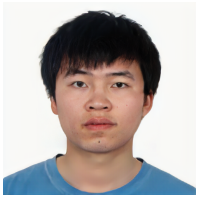
[102] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," in *Web Intelligence Workshops (WIW), IEEE/WIC/ACM International Conference on*. IEEE, 2016, pp. 104–111.

[103] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," 2017.

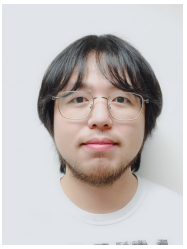
Ruimin Sun is a Postdoctoral Research Associate at Northeastern University. She received her Ph.D. from the University of Florida. She works on secure, reliable systems, and machine learning model protection.



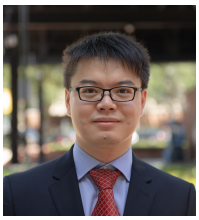
Xiaoyong Yuan is an Assistant Professor at the College of Computing, Michigan Technological University. He received the B.S. degree from Fudan University, in 2012, the M.E. degree from Peking University, in 2015, and the Ph.D. degree from the University of Florida, in 2020. His research interests lie in deep learning, security & privacy, and cloud computing.



Pan He received his B.E. in Software Engineering from Sichuan University in 2015. He is pursuing a Ph.D. degree in Computer Science at the University of Florida. His current research is focused on understanding the three-dimensional motion and structure of a dynamic scene via developing deep learning methods in various environments.



Qile Zhu received Ph.D. from the Department of Computer & Information Science & Engineering at the University of Florida in 2020. His research interests include deep learning and natural language processing. He received a B.E. degree from Zhejiang University in 2013 and a master degree from the University of Florida in 2015.



Aokun Chen is a Postdoctoral Researcher under the supervise of Dr. Guo at the University of Florida (UF). He received his Ph.D. degree in Computer Engineering on machine learning for security systems from UF. His current research interests include machine learning for bioinformatics and cybersecurity.

André Grégio is an Assistant Professor at the Federal University of Paraná (UFPR). His research intersects computer security and data science, e.g., effective malware analysis and attack detection systems.

Daniela Oliveira is an Associate Professor at University of Florida (UF). Her research includes human factors security and IoT security, especially the application of dynamic information flow to thwart attacks.

Xiaolin (Andy) Li is Partner of Tongdun Technology, heading the AI Institute; Chair Professor and Chief Scientist of IBMC, Chinese Academy of Sciences. He was a Professor at the University of Florida and founding center director of NSF Center for Big Learning. His research interests include deep learning, cloud computing, security & privacy, IoT, and intelligent medicine.