Multi-Layer Mapping of Cyberspace for Intrusion Detection

Sicong Shao, Pratik Satam, Shalaka Satam,
Khalid AI-Awady, Gregory Ditzler and Salim Hariri
Dept. of Electrical & Computer Engr.
University of Arizona
Tucson, AZ 85721 USA

Cihan Tunc
Dept. of Computer Science & Engr.
University of North Texas
Denton, TX 76203 USA

***Abstract*—The ubiquity and vulnerability of computer applications make them ideal places for intrusion attacks increasing in intensity and complexity. Computer applications have a relationship with various networks, physical components, host devices, and users with different roles and requirements. Therefore, securing computer applications in such a complex and dynamic cyberspace is urgent and challenging. This paper attempt to tackle the challenges by proposing a Multi-Layer Abnormal Behaviors Analysis (MLABA) framework for intrusion detection associated with tri-layer (i.e., system, process, and network layers) in cyberspace for characterizing their normal operations and detect any abnormal behavior that might be triggered by malicious activities. The proposed technique is evaluated on several popular applications (i.e., Firefox, Opera, Chrome, and Ruby). The experimental results demonstrate the feasibility of MLABA framework that can detect the intrusion and abuse for applications.**

I. Introduction

The advancement of modern applications and software have enabled the revolutionary capabilities that has served many fields, such as healthcare systems, smart devices, and the emerging of the Internet of Things (IoT). Users use applications on different platforms to perform day-to-day tasks including, accessing the webpage and emails through web browsers, editing document through office suites, and shopping online by e-banking. However, attackers have also infected applications by performing malicious behaviors such as breaching health data [1], attacking medical devices [1], stealing privacy information [2], injecting malicious code, exploiting vulnerabilities of executable file, and redirecting the control-flow of program execution [3]. In addition, modern applications cannot be bug-free due to their sheer size and complexity [4]. As a result, many existences of faults and bugs may create security vulnerability with potential risk exploited by attackers to produce significant security concerns and financial losses [2]. Further, the attackers can be insiders who are trusted and have full access to the computing system, rendering applications susceptible to abuse and insider threats. Therefore, computer application security is a critically important and challenging task. Intrusion detection systems (IDS) are one of the promising security solutions, which can be used to detect intrusion attacks and insider threats for protecting computer systems and applications.

Corresponding Author: Salim Hariri [hariri@arizona.edu]

Traditional application IDS solutions have suffered problems such as (1) many of them are signature-based IDS that result in assumptions by assuming that they know the signatures and the attack patterns that cyber attackers might leverage; however, signature-based detection cannot recognize zero-day attacks and unknown attacks, i.e., attacks created by exploiting unidentified or unknown vulnerabilities that no patch is provided [5], [6]. (2) To imporve the IDS performance, machine learning-based IDS has been proposed; however, many solutions (i.e., that use supervised learning for training classification model to identify the classes appearing in the training data) with strong close-set assumption that needs security experts to collect and label both normal and attack data for training detection model. Since it is unrealistic to collect all the possible categories of attack data for training, it might be better if the detection model can only use normal data to be trained. Our previous work also suggests that a model that learns the normal data distribution strictly is more robust than a supervised model in IDS [7]. (3) Further, many of the current efforts of application-level IDS using the single application data source for anomaly detection may not enough for successfully detecting planned and motivated attacks that leverage vulnerabilities from multiple layers of the host machine [6], [8], [9]. In order to tackle the limitations mentioned above of traditional techniques, we propose MLABA methodology that is based on a holistic approach that will continuously monitor, analyze, and diagnose the operations of three cyberspace layers (i.e., system layer, process layer, and network layer) through denoising autoencoder (DAE) neural network in an integrated manner.

The main contributions of this paper are listed as follows: (1) We propose a novel MLABA framework that integrates process layer, system layer, and network layer to achieve decent application-level intrusion detection results based on training detection model only with normal data. (2) We implement the proposed MLABA framework and conduct the case studies on popular web browser and software development tool including Firefox, Opera, Chrome, and Ruby. (3) We benchmark the performance of MLABA on real-world datasets containing multiple attacks (i.e., Fork Bomb, Infinite Loop,

Infinite Array, Ransomware, Brute Force Password Cracking, and Rainbow Table Attack) targeting the normal opearations of web browsers and software development tool. Through the experimental datasets, we address the effectiveness of the MLABA framework.

This manuscript is organized as follows: Section II covers related work in machine learning for IDS, Section III presents the multi-Layer mapping of cyberspace, Section IV presents the experiments and the key findings, and Section V provides a discussion of the work.

II. Related Work

In this section, we briefly introduce the IDS and the related work of IDS with machine learning.

A. Intrusion Detection Systems

According to the types of detection method, IDS can be classified into signature-based IDS, anomaly-based IDS, specification-based IDS, and hybrid-based IDS [10] [11]. The signature-based detection uses the predefined library of attack signatures to model the attack of the monitored object. An alarm is generated whenever those signatures are matched and detected. A match indicates a possible known attack for signature-based detection solutions while other operations are classified as normal. Hence, the signature-based detection techniques have a low false positive alarm rate. However, its major drawback is that it cannot detect unknown or zero-day attacks [11]. Anomaly-based detection does not need data of known attacks and recognizes abnormal behaviors through modeling normal behaviors. Therefore, it can identify unknown and zero-day attacks. However, one downside is that anomaly-based detection suffers from high false alarms if the model is not fine-coarse-grained [12]. The specification-based IDS also called stateful protocol analysis (SPA), since this mechanism allows the IDS to know the state of the protocol. It uses the mechanism by setting thresholds and rules regulated by behavior of protocols, nodes, and routing tables [13]. According to the deviation of system behavior, this type of IDS alarm intrusion behaviors, similar to anomaly detection. However, the significant difference between them is that the rules are manually set for each specification by security expert. Therefore, this type of IDS creates lower false positives than anomaly detection-based IDS due to manually setting specifications. However, this approach suffered time-consuming problems and created significant delays [14]. The hybrid-based IDS usually combines two or three IDS such as signature-based, anomaly-based, and specification-based IDS [15]. With combing different IDS, several things need to be considered. First, how to solve the conflicts between different predictions of IDS. Second, the architecture that is either layered or parallel has to be preliminarily determined. Moreover, selecting the correct order of multiple IDS components for processing is also a challenge [16].

B. IDS with Machine Learning

The research of IDS with machine learning has been extensively studied in the security domain [2], [17]. Paulauskas et al. [18] present a network intrusion detection system (NIDS) to detect the network flow anomalies using Local Outlier Factor (LOF). LOF is an anomaly detection method computing the local density deviation of a given sample concerning its local neighborhood.In Bhattacharjee et al. [19], an intrusion detection scheme based on LOF is proposed to detect the spectrum sensing data falsification attack. Montero et al. [20] propose an IDS called self-protection agent whose core algorithms are error correcting output codes and Support Vector Machine (SVM). The work of Khreich et al. [21] presented an One-Class SVM (OCSVM) model-based IDS to detect system abnormal behaviors at the host level. They performed substantial feature engineering for OCSVM by segmenting the system call traces into multiple n-grams of variable length and mapping them to fixed-size sparse feature vectors. Hess et al. [7] developed an IDS that can identify malicious HTML files. In the experiment, different classifiers, including AdaBoost, GentleBoost, RobustBoost, RusBoost, random forest, and isolation forest (iForest), were compared on a high imbalance dataset. The finding suggests that it might be better to use detection model that can only learn the data distribution of the majority class, such as iForest that is an anomaly detection model by isolating anomalies. Tao et al. combine isolation forest and Spark to implement a NIDS that can leverage Spark's benefit and run iForest in a parallel manner [22]. In recent years, deep learning is also used to perform intrusion detection. One of the superiority of deep learning is that it can learn complicated concepts well. Ferrag et al. [23] perform a general assessment for a number of deep learning models using two datasets: Bot-IoT and CSE-CIC-IDS2018. Regarding the studies of using the autoencoder-based deep learning model for IDS, Farahnakian and Heikkonen [24] develop a deep autoencoder approach for intrusion detection. The experiments show that deep autoencoder is superior to deep belief network on the KDD-Cup'99 dataset. Mirsky et al. [25] develop an anomaly detection-based NIDS using autoencoder ensembles on the local network. Shone et al. [26] propose a stacked nonsymmetric deep autoencoder for NIDS and evaluated it by KDD-Cup'99 and NSL-KDD datasets.

III. Multi-Layer Mapping of Cyberspace

The structure of MLABA framework is shown in Figure 1. First, using our continuous monitoring tool, we monitor the cyber infrastructure consist of system, data, application, and users. The cyberspace footprint (i.e., selected measurements from cyber infrastructure) extractor then extracts the system footprint, application footprint, and network footprint. Here, the system footprint, application footprint, and network footprint refer to the selected measurements that can respectively record system, application, and network behaviors in the appropriate and informative manner. After that, the footprints of three layers are transformed to feature vectors and combined to form the MLABA feature representation model which is used to learn DAE for anomaly behaviors analysis.

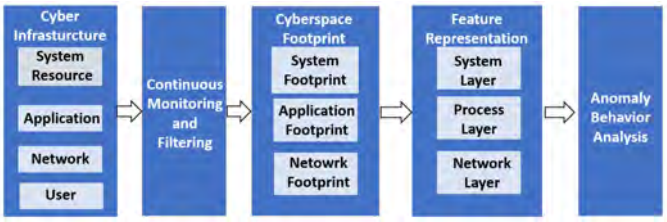

Fig. 1. The structure of the MLABA framework.

A. Data Collection and Feature Representation Model

Modern applications are complicated, highly customized, and many of them facilitate the interaction between application sessions and user sessions. Thus, the attack surface consists of a wide range of components, including logical components, network components, and physical components [27]. Therefore, we aim to create an application IDS for detecting suspicious activities using a comprehensive feature representation rather than the representation designed to only focus on the application level in many previous work [6], [8], [9]. For example, the malicious behavior of stolen SSH credentials from application is hard to identify only from the application data, but abnormal traffic such as unusual interaction between user and computer in the network traffic might expose this malicious activity. Therefore, we aim to combine data from multiple layers of monitored hosts to develop an IDS that is capable of detecting sophisticated attacks having characteristics across different layers. To this end, the MLABA feature representation model built on the data from multiple sources using process data, system data, and network data, is developed. To profile the tri-layer feature representation model, our framework collects the system footprint, application footprint, and network footprint, then convert to the feature vectors. Table I lists the cyberspace footprint from three layers for designing the feature representation model.

The system layer maps hardware and physical objects onto the cyberspace through the system footprint that contains system-wide measurements of CPU time, disk usage statistics, disk I/O statistics, CPU utilization, CPU frequency, CPU percent, CPU statistics, and memory information. These system footprints are a numeric system activity measurements that represents states, resource consumption, behaviors of the entire host, and benefits for understanding its interaction with other layers. In addition, these measurements change frequently, even imperceptible activities in the host system are reflected by them. Therefore, it is possible to identify abnormal behaviors of the host fast with system footprint.

The process layer maps the target running processes using the application footprint including the process measurements such as the process CPU-related measurements, the process memory-related measurements, the process I/O-related measurements, the process threads, the process handles, and so on (see Table I). For an application that has multiple processes running simultaneously, we average the measurements of its processes recorded at the same time. The CPU-related measurements were chosen since abnormal CPU measurements can occur when encountering anomalous conditions. For example,the process may consume unusual amounts of CPU power when it fell in with impossible termination conditions due to malicious code injection, resulting in infinite loop and deadlock. The memory-related information can reflect the abnormal memory consumption of a poisoned process whose allocated chunks of memory could not be freed. The anomalous high number of the process I/O within a short amount of time may reveal the system call of the process often fail. This may also be brought about by application abuse or malicious code injection. Therefore, these collected application footprint can be utilized for identifying the abnormal states and behaviors of the process.

The network layer maps the data communication into cyberspace using the network I/O measurements from the network footprint. The activity of applications such as web browsers significantly related to Internet usage for diverse purposes, e.g., for watching streaming video, using social network platforms, and viewing news. In addition, the traffic density of network I/O reflects how active the background processes connecting to other hosts. Therefore, the network layer could establish the connection between the application and the network resources, providing the capability to capture the application behaviors and preferences reflected by network I/O. We partition complicated network I/O information into different measurements related to the transmission of bytes, packets, and errors.

The richness of cyberspace footprints from three layers captures a wide spectrum of possible behaviors of applications since application-level cyber-attacks perhaps reflect their impact by the monitored measurements at each layer. Therefore, we convert cyberspace footprints from three layers into system layer feature vectors, process layer feature vectors, and network layer feature vectors. For each feature obtained from the cyberspace footprints including system disk I/O count, system disk I/O time, system disk I/O merged count, system disk usage, system CPU statistics, system CPU times, system memory, process CPU times, process I/O count, various process memory information, Network I/O bytes, and Network I/O packets, we calculate the difference of the feature value compared with the feature value in the previous sample. We then combine the feature vectors from three layers and used as input for anomaly detection model.

B. Intrusion Detection Framework

We denote the monitored samples represented by feature representation model as $X = \{x_1, x_2, x_3, \ldots, x_i\}$, where x_i represents i^{th} sample. Then the problem of the intrusion detection can be defined as follows:

$$Score_i = F(x_i) \tag{1}$$

where F is an evaluation function that uses anomaly detector to evaluate anomaly score of sample x_i, and $Score_i$ denotes

TABLE I
THE DESCRIPTION OF MLABA CYBERSPACE FOOTPRINTS.

Cyberspace Footprints	Description
System Layer	
System disk I/O count	The measurements of the system's disk I/O, measured by the count numbers of reads and writes, and the bytes number of read and written, respectively.
System disk I/O time	The time has spent on reading from disk and writing to disk, respectively. The time has spent on performing actual I/Os.
System disk I/O merged count	The respective numbers of merged reads and writes.
System disk usage	The measurements of the system's hard disk usage, measured by the bytes of total space, used space, free space, and percentage usage, respectively.
System CPU utilization	The percentage of system's whole CPU utilization. The percentage of system's each CPU utilization. The number of logical CPU in the system. The number of physical CPU in the system.
System CPU frequency	The current CPU frequencies. The maximum and minimum CPU frequencies since boot.
System CPU statistics	The number of context switches. The number of interrupts. The number of software interrupts. The number of system calls.
System CPU times	The CPU times has spent from the given mode including user, system, and idle. The CPU time has spent by fields including nice, iowait, irq, softirq, steal, guest, and guest nice, respectively.
System Memory	The percentage of system's virtual memory utilization. The system's used virtual memory. The system's used swap memory.
Process Layer	
Process CPU times	The process times in the given mode including user and system. The user time and the system time of all child processes. The process time of waiting for the completion of blocking I/O.
Process I/O count	The respective numbers of bytes read and written. The respective numbers of operations of read and write. The respective numbers of bytes which process transmit to the system calls of read function and pread function. The respective numbers of bytes which process transmit to the system calls of write function and pwrite function.
Process CPU utilization	The percentage of process CPU utilization.
Various process memory information	The process memory information of RSS (resident set size), VMS (virtual memory size), shared (memory shared with other processes), TRS (text resident set), DRS (data resident set), lib (memory of shared libraries), dirty (number of dirty pages).
Process memory percent	The percentage of the process memory utilization.
Process file descriptors	The number of file descriptors currently opened by the process.
Process threads	The number of threads currently used by the process.
Process handles	The number of handles currently used by the process.
Process niceness	The priority number of the process.
Network Layer	
Network I/O bytes	The numbers of bytes has sent and received, respectively.
Network I/O packets	The numbers of packets has sent and received, respectively.
Network I/O error	The numbers of errors while receiving and sending.
Network I/O drop	The numbers of dropped incoming packets and dropped outgoing packets.

the anomaly score of the monitored application's observation. An abnormal sample of the monitored application is one whose anomaly score exceeds a threshold, while a normal sample is one whose anomaly score is less than a threshold. For getting a function F, not only do we analyze the monitored process mapped at the process layer for extracting features for the application, but we also consider the behaviors of the system layer and network layer. With tri-layer (i.e., system layer, process layer, and network layer) features, we can evaluate the impact of monitored application across three layers. For example, consider that web browser application's memory utilization and system-wide memory utilization are dramatically increased while the network traffic is not increased or even decrease. This may indicate the web browser application gets stuck in a Fork-Bomb (i.e., an attack makes the browser constantly open tabs and thus attempts to exhaust the system's resources). According to the above considerations, MLABA, a knowledge-based multi-layer mapping of cyberspace intrusion detection framework, is proposed. The overview of the machine learning framework of MLABA is shown in Figure 2. On the collected cyber footprint, the objective is to extract the features for data mining. Then, the extracted feature from three layers are concatenated to form tri-layer feature representation stored in database for training anomaly detection model. The machine learning pipeline leverages anomaly detection model for the anomaly behaviors analysis. The stage for training anomaly detection model requires that MLABA's machine learning pipeline observes previous normal behaviors represented by tri-layer feature representation model. This allows the anomaly detection model to model normal behavior. The anomaly detection model is only trained on the normal sample. After finish the training stage, the anomaly detection model can distinguish the behaviors of normal and abnormal behaviors.

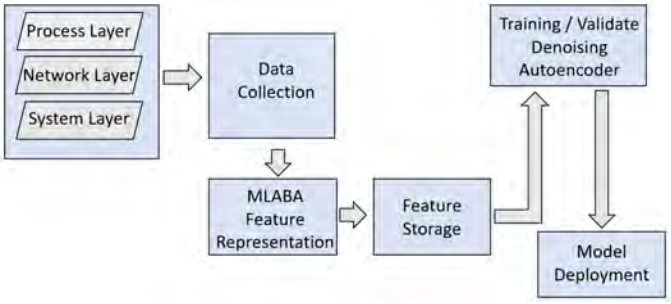

Fig. 2. Overview of the machine learning framework for MLABA.

C. Automated Intrusion Detection with Denoising Autoencoders

The DAE is selected as the anomaly detection algorithm. A basic autoencoder is a feedforward artificial neural net that learns a map from the input to itself through a pair of encoding and decoding phases. A DAE is a variant form from a basic autoencoder, that adds noise to the input. The DAE can be trained to reconstruct an uncorrupted input $x \in \mathcal{R}^n$ according to $\hat{x} \in \mathcal{R}^n$ (i.e., a corrupted version of $x \in \mathcal{R}^n$). The corrupted input $\hat{x}$ is mapped into the hidden layer $h \in \mathcal{R}^m$ by the encoding phase, as given by:

$$h = f(W\hat{x} + b) \tag{2}$$

where $f(\cdot)$ is an activation function, $W \in \mathcal{R}^{m \times n}$ is a weight matrix, and $b \in \mathcal{R}^n$ is a bias matrix. The hidden layer h maps the coding information into the output layer by a decoding phase to the output representation $y \in \mathcal{R}^n$, which are given by:

$$y = f^{'}(W^{'}h + b') \tag{3}$$

where $f^{'}(\cdot)$ is another activation function, $W^{'} \in \mathcal{R}^{m \times n}$ is a weight matrix, and $b^{'} \in \mathcal{R}^n$ is a bias matrix in the decoding phase. Unlike basic autoencoder that minimizes the reconstruction error between the input $\hat{x}$ and output y, a basic DAE is trained to minimize the reconstruction error between uncorrupted input x and output y. Therefore, the root mean square error (RMSE) between uncorrupted input x and output representation y is used as the objective function defined as:

$$RMSE(x, y) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{4}$$

A deeper DAE used as anomaly detector for MLABA can be formed by adding more hidden layers, which allows DAE to learn complex concept through layer-by-layer processing. Mini-batch gradient descent is applied to learn the parameters of the weight matrices and bias matrices. We use a DAE with three hidden layers. The detailed parameters of DAE are shown in Table II. Concretely, the dimensionality of MLABA input feature space is 87 after concatenating the features from three layers. The dimensionality of the output layer is also 87 since the purpose is to reconstruct the input. The compression network structure of DAE is set as 64-32-64. The first dimension *B* of the output shape is the size of mini-batch, and the second dimension of the output shape is the number of features output by this layer. The dense (fully-connected) layer is used to construct the hidden layer and output layer. ReLU is used as the activation function for each hidden layer, and linear function is used at the output layer. We train our DAE by adding dropout regularization with probability 0.1 to reduce the overfitting in anomaly detector learning. The learned free parameters of DAE for each dense layer are 5,632, 2,080, 2,112, 5,655, respectively. After completing the autoencoder learning phase with denoising Gaussian noise, our DAE can detect the anomalies. An observation that belongs to normal or abnormal is determined by RMSE. During the test phase, an observation is normal if it has a low RMSE while it is abnormal if its RMSE is larger than the anomaly threshold.

TABLE II
THE DETAILED PARAMETERS OF THE DENOISING AUTOENCODER

Layer Type	Output Shape	#Parameters
Input	(*B*, 87)	0
Dense	(*B*, 64)	5,632
Dropout	(*B*, 64)	0
Dense	(*B*, 32)	2,080
Dropout	(*B*, 32)	0
Dense	(*B*, 64)	2,112
Dropout	(*B*, 64)	0
Dense	(*B*, 87)	5,655

IV. EXPERIMENTAL RESULTS

A. Experiment Setup and Parameterization

To demonstrate the feasibility of MLABA, we evaluated it through web browser applications including Firefox, Chrome, Opera, and software development application Ruby. They are conducted as use cases due to three reasons: (1) All of these web browsers are widely used computer applications. (2) The increasing number of web attacks has made web application security one of the foremost issues in cybersecurity [27] (3) Ruby is a popular development tool and considered as one of the most used programming languages for building web applications [28].In preparation for experiments, we run these four applications on an Ubuntu 18.04 OS PC machine using i7-3770 CPU (4 cores 8 threads) and 16 GB memory. The total hours of collected data from Firefox, Chrome, Opera, and Ruby are 29 hours, 38 hours, 39 hours, and 7 hours, respectively. For the web browser applications, we collected normal operational data when the user has visited regular websites and perform normal operations. We collected the web browser's abnormal operational data for several web browser application attacks that were performing. For the dataset of Ruby application, we collect normal data by performing regular developing behaviors such as running machine learning tests and experiments through Ruby. We also collect its abnormal behavior through running several Ruby scripts for abuse and malicious purpose. The following types of attacks are used in our experimental results and evaluation:

TABLE III
DESCRIPTION OF THE EXPERIMENTAL DATASETS

Operation	Firefox-AD	Chrome-AD	Opera-AD	Ruby-AD
Normal	40,717	54,314	55,399	9,270
Fork Bomb	1,074	1,095	1,031	0
Infinite Loop	1,226	1,215	1,100	0
Infinite Array	1,047	1,546	1,041	0
Ransomware	0	0	0	685
Brute Force Password Cracking	0	0	0	554
Rainbow Table Attack	0	0	0	648

- Fork Bomb Attack: This attack opens tab continuously, results in a very high number of tabs and consumption of the hardware resources for that web browser.
- Infinite Loop Attack: This attack runs an infinite loop, making the web browser running the script hang.
- Infinite Array Attack: It creates a dynamic array and adds random data to the end of the array during the execution. Hence, until crashes, the web browser application aims consuming all memory in the system.
- Ransomware: A Ruby script encrypts files on a computer, rendering files unusable, and affecting the application relying on the encrypted files.
- Brute Force Password Cracking: A Ruby script involves trying an exhaustive search to find the plain text of the hash text for cracking password.
- Rainbow Table Attack: A Ruby script uses the time-memory trade-off technique by loading a huge table filled with hash values, attempting to crack passwords quickly.

One of the important reasons for choosing these attacks because none of them are detectable by web browsers, Ruby, and Ubuntu operating systems. The descriptions of four AD (anomaly detection) datasets including Firefox-AD, Chrome-AD, Opera-AD, and Ruby-AD are summarized in Table III.

We use Isolation Forest (iForest), One-Class SVM (OCSVM), Local Outlier Factor (LOF), and Elliptic Envelope (EE) as baselines of anomaly detection model compared to DAE: (1) iForest: An ensemble-based anomaly detection method of detecting anomalies using the mean path length of trees within forest [29]. (2) OCSVM: A kernel-based anomaly detection method of estimating the support of a high-dimensional distribution [30]. (3) LOF: A density-based anomaly detection method by comparing the local deviations of density [31]. (4) EE: An anomaly detector based on estimating covariance on Gaussian distribution data [32].

Training methodology. Since evaluating our approach through the one-class learning principle [33], we use the clean normal training data for training anomaly detectors. More specifically, we perform random sampling on normal data to extract 60% of normal samples as the training data, and the remaining 40% of normal samples are mixed with all abnormal samples for testing. We repeat the random sampling 15 times to generate 15 independent rounds of different combinations of training set and test set for performance evaluation. The min-max normalization is applied for feature scaling. [1].

For the hyper-parameters of OCSVM with RBF (radial basis function) kernel, we select the ν from $\nu \in \{0.5, 0.1, 0.05, 0.01\}$ and RBF kernal coefficient γ from $\gamma \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$. For the LOF, we select the number of nearest neighbors from $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$. The novelty parameter is set to a boolean value of True. For the iForest, we use the number of trees t from $t \in \{50, 100, 3000\}$. The bootstrap parameter is set to a boolean value of True. As the sub-sampling size, we select it from $\psi \in \{2^8, 2^9, 2^{10}, 2^{11}, 2^{12}, 2^{13}, 2^{14}\}$. We also test the iForest using all the training samples without sub-sampling. For the EE, we select the support fraction μ from $\mu \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. All other parameters of iForest, LOF ,OCSVM and EE are taken from the default setting of sklearn [1]. The structure of DAE with MLABA feature representation model is described in Table II. As for the other hyper-parameters, Adam [34] is leveraged to optimize the model since DAE is neural networks requiring iterative updating of the free parameters. The learning rate η of Adam is selected from $\eta \in \{10^{-3}, 10^{-4}\}$. We train DAE 64 epochs, and 64 mini-batch size is used.

Evaluation metric. During the test stage, the labels of the test data are used to measure the performance. The receiver operator characteristic (ROC) curve and precision-recall (PR) curve are used to perform a comprehensive evaluation for all the anomaly detectors. ROC curve presents the tradeoff between the true positive rate (TPR) and the false positive rate (FPR), while PR curve plots the recall against precision. ROC curve and PR curve are two important and complementary evaluation metrics. ROC curve is commonly used because it is easy to interpret. However, it is possible to give an overly optimistic evaluation of the method's performance when dealing with highly class-imbalance datasets such as datasets for anomaly detection [35]. A popular alternative is to use PR curve for evaluation classification performance since it tends to obtain a precise evaluation in the anomaly detection tasks, though it also turns out to be a highly imprecise metric in the tasks that encounter class imbalance problems with small instances of negative instances [36]. Furthermore, we also consider the evaluation metrics including Area Under ROC curve (AUC-ROC) and Average precision (AP). AUC-ROC can summarize the entire location of the ROC curve. AP can summarize the PR curve as the weighted mean of precisions achieved at each threshold. [1].

B. Results

Figure 3 presents visualizations of the mean ROC curves and the mean PR curves for all the anomaly detectors. The standard deviation information is also included in these curves. For the Firefox-AD dataset, the mean ROC curve and the mean PR curve of DAE cover other anomaly detectors, showing the superiors of DAE. For the Chrome-AD dataset, the mean ROC curves and the mean PR curves of LOF and DAE present very

[1]https://scikit-learn.org/stable/index.html

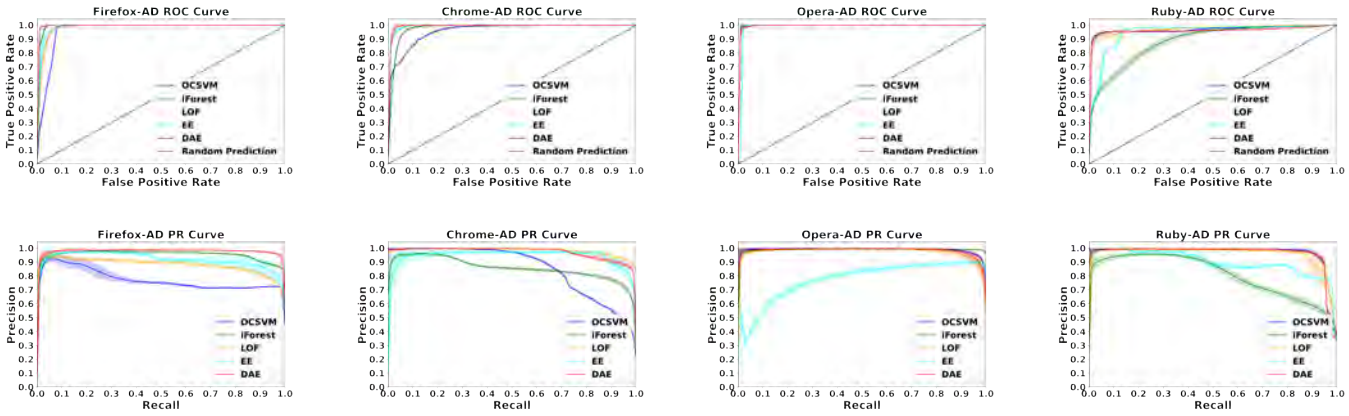

Fig. 3. ROC curves and PR curves for different anomaly detection models with MLABA feature representation.

TABLE IV
RESULTS OF MEAN AUC-ROC OF MLABA FEATURE REPRESENTATION MODEL WITH DIFFERENT ANOMALY DETECTORS

Dataset	Layer	#Sample	#Anomalies	OCSVM	iForest	LOF	EE	DAE
Firefox-AD	Tri-layer	44,064	3,347	96.29 ± 0.15	99.39 ± 0.05	98.21 ± 0.19	98.77 ± 0.36	$\mathbf{99.77} \pm 0.04$
Chrome-AD	Tri-layer	58,170	3,856	96.16 ± 0.16	97.89 ± 0.15	$\mathbf{99.73} \pm 0.03$	99.36 ± 0.15	99.55 ± 0.07
Opera-AD	Tri-layer	58,571	3,172	99.89 ± 0.02	$\mathbf{99.94} \pm 0.01$	99.75 ± 0.13	98.62 ± 0.20	99.82 ± 0.09
Ruby-AD	Tri-layer	11,157	1,887	97.11 ± 0.20	88.41 ± 1.03	$\mathbf{97.29} \pm 0.68$	94.89 ± 0.43	96.43 ± 0.16
Average				97.36 ± 0.13	96.41 ± 0.31	98.75 ± 0.26	97.91 ± 0.29	$\mathbf{98.89} \pm 0.09$

TABLE V
RESULTS OF MEAN AP OF MLABA FEATURE REPRESENTATION MODEL WITH DIFFERENT ANOMALY DETECTORS

Dataset	Layer	#Sample	#Anomalies	OCSVM	iForest	LOF	EE	DAE
Firefox-AD	Tri-layer	44,064	3,347	77.24 ± 1.47	95.37 ± 0.40	88.32 ± 1.17	91.79 ± 1.90	$\mathbf{97.87} \pm 0.43$
Chrome-AD	Tri-layer	58,170	3,856	85.83 ± 0.33	85.74 ± 0.86	$\mathbf{98.14} \pm 0.27$	94.28 ± 1.92	97.32 ± 0.32
Opera-AD	Tri-layer	58,571	3,172	$\mathbf{99.08} \pm 0.37$	99.07 ± 0.24	97.68 ± 0.92	77.78 ± 2.15	98.57 ± 0.68
Ruby-AD	Tri-layer	11,157	1,887	$\mathbf{96.60} \pm 0.41$	80.56 ± 1.56	95.81 ± 0.87	89.99 ± 1.49	96.41 ± 0.39
Average				89.69 ± 0.65	90.19 ± 0.77	94.99 ± 0.81	88.46 ± 1.87	$\mathbf{97.54} \pm 0.46$

close performance and cover other anomaly detectors. For the Opera-AD, all the anomaly detectors attain similar shapes on the mean ROC curves. However, the mean PR curve of EE reveals that it has a significant recall drop when precision is over 0.9. As for the mean ROC curve of Ruby-AD dataset, LOF obtains the best coverage while OCSVM and DAE obtain better coverage on the mean PR curve than other detectors. However, LOF also shows higher standard deviations of true positive rate and precision when the false positive rate is low and precision is high, making its stability of prediction somewhat unsatisfactory compared with OCSVM and DAE.

Tables IV and V show the results of all the anomaly detectors in terms of mean AUC and mean AP of anomaly detectors using MLABA feature representation model. In most cases, they show good performance for anomaly detection, validating the effectiveness of MLABA feature representation model. Concretely, DAE attains the best performance on the Firefox-AD dataset. LOF achieves the best results on the Chrome-AD dataset. OCSVM and iForest perform better than other anomaly detection methods on the Opera-AD dataset. OCSVM approaches the best performance on the Ruby-AD dataset. As for the average performance, DAE achieves the best results in both tables. Therefore, the experimental results validate the superiors of using DAE as the anomaly detector for MLABA framework deployment since it shows satisfactory performance on both measurements with respect to all the datasets. We also observe that OCSVM and EE respectively perform unsatisfactorily on the mean AP evaluations of Firefox-AD and Opera-AD, such that they are unsuitable for deployment. We also find that AUC-ROC scores present more optimistic evaluations than AP scores for all the anomaly detectors in the experimental datasets, especially for OCSVM and EE. It is due to the fact that AUC-ROC depends on both the performance of normal and abnormal classes. Therefore, AUC-ROC can be biased by the normal class performance because of the high class imbalance nature in our intrusion detection task. However, AP summarizes the PR curve, which assesses the proportions of positive predictions that are truly positive and the proportion

of truly positive samples that are correctly detected. As a result, AP cares more on the performance of the abnormal class and does not concern too much about the normal class's performance compared to AUC-ROC.

V. Conclusion

To secure applications in the complex and dynamic cyberspaces, this paper presents a knowledge-based Multi-Layer Abnormal Behaviors Analysis (MLABA) intrusion detection framework. MLABA creates a novel feature representation model through integrating the process data, system data, and network data for application intrusion detection. Besides, MLABA framework enables decent anomaly detection performance only utilizing normal training data. Through the experiments, we address the feasibility of our framework.

Acknowledgment

This work was supported by grants from the Department of Energy #DE-NA0003946, National Science Foundation CAREER #1943552, Army Research Lab W56KGU-20-C-0002, Office of Naval Research N6833518C0416, Air Force Office of Scientific Research (AFOSR) Dynamic Data-Driven Application Systems (DDDAS) award number FA9550-18-1-0427, National Science Foundation (NSF) research projects NSF-1624668 and NSF-1849113, and National Institute of Standards and Technology (NIST) 70NANB18H263.

References

[1] J. Sametinger, J. Rozenblit, R. Lysecky, and P. Ott, "Security challenges for medical devices," *Communications of the ACM*, 2015.

[2] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[3] S. A. Musavi and M. R. Hashemi, "HPCgnature: A hardware-based application-level intrusion detection system," *IET Information Security*, vol. 13, no. 1, pp. 19–26, 2019.

[4] E. Dinella, D. Hanjun, Z. Li, M. Naik, L. Song, and K. Wang, "Hoppity: Learning graph transformations to detect and fix bugs in programs," in *International Conference on Learning Representations*, 2020.

[5] A. Blaiseab, M. Boueta, V. Conana, and S. Secci, "Detection of zero-day attacks: An unsupervised port-based approach," *Computer Networks*, vol. 180, no. January, 2020.

[6] R. Bronte, H. Shahriar, and H. M. Haddad, "A signature-based intrusion detection system for web applications based on genetic algorithm," in *International Conference on Security of Information and Networks*, 2016.

[7] S. Hess, P. Satam, S. Hariri, and G. Ditzler, "Malicious html file prediction: A detection and classification perspective," in *ACS/IEEE International Conference on Computer Systems and Applications*, 2018.

[8] M. K. Gupta, M. C. Govil, G. Singh, and P. Sharma, "Xssdm: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2010–2015, 2015.

[9] Y. Gao, Y. Ma, and D. Li, "Anomaly detection of malicious users' behaviors for web applications based on web logs," in *International Conference on Communication Technology (ICCT)*, 2017.

[10] R. A. Bridges, T. R. Glass-Vanderlan, M. D. Iannacone, M. S. Vincent, and Q. Chen, "A survey of intrusion detection systems leveraging host data," *ACM Computing Surveys*, vol. 52, no. 6, 2019.

[11] P. Satam and S. Hariri, "WIDS: An Anomaly Based Intrusion Detection System for Wi-Fi (IEEE 802.11) Protocol," *IEEE Transactions on Network and Service Management*, no. c, pp. 1–16, 2020.

[12] H. Alipour, Y. B. Al-Nashif, P. Satam, and S. Hariri, "Wireless Anomaly Detection Based on IEEE 802.11 Behavior Analysis," *IEEE Transactions on Information Forensics and Security*, 2015.

[13] K. Khan, A. Mehmood, S. Khan, M. A. Khan, Z. Iqbal, and W. K. Mashwani, "A survey on intrusion detection and prevention in wireless ad-hoc networks," *Journal of Systems Architecture*, 2020.

[14] M. Zeeshan, H. Javed, and S. Ullah, "Discrete R-Contiguous bit matching mechanism appropriateness for anomaly detection in Wireless Sensor Networks," *International Journal of Communication Networks and Information Security*, 2017.

[15] M. A. Aydin, A. H. Zaim, and K. G. Ceylan, "A hybrid intrusion detection system design for computer network security," *Computers and Electrical Engineering*, 2009.

[16] E. Tombini, H. Debar, L. Mé, and M. Ducassé, "A serial combination of anomaly and misuse IDSes applied to HTTP traffic," in *Proceedings - Annual Computer Security Applications Conference, ACSAC*, 2004.

[17] G. Ditzler, A. Akoglu, and S. Hariri, "Framework to support DDDAS decision support systems: Design overview," in *Workshop on InfoSymbiotics: DDDAS Dynamic Data Driven Applications Systems*, 2017.

[18] N. Paulauskas and A. F. Bagdonas, "Local outlier factor use for the network flow anomaly detection," *Security and Communication Networks*, 2015.

[19] S. Bhattacharjee and N. Marchang, "Malicious user detection with local outlier factor during spectrum sensing in cognitive radio network," *International Journal of Ad Hoc and Ubiquitous Computing*, 2019.

[20] F. de la Pěna Montero, S. Hariri, and G. Ditzler, "A self-protection agent using error correcting output codes to secure computers and applications," in *IEEE International Conference on Cloud and Autonomic Computing*, pp. 58–68, 2017.

[21] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, "An anomaly detection system based on variable N-gram features and one-class SVM," *Information and Software Technology*, 2017.

[22] X. Tao, Y. Peng, F. Zhao, P. Zhao, and Y. Wang, "A parallel algorithm for network traffic anomaly detection based on Isolation Forest," *International Journal of Distributed Sensor Networks*, vol. 14, no. 11, 2018.

[23] M. A. Ferrag, L. Maglaras, S. Moschoyiannis, and H. Janicke, "Deep learning for cyber security intrusion detection: Approaches, datasets, and comparative study," *Journal of Information Security and Applications*, vol. 50, 2020.

[24] F. Farahnakian and J. Heikkonen, "A deep auto-encoder based approach for intrusion detection system," *International Conference on Advanced Communication Technology*, 2018.

[25] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: An ensemble of autoencoders for online network intrusion detection," *arXiv*, no. February, pp. 18–21, 2018.

[26] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, no. 1, pp. 41–50, 2018.

[27] N. Agarwal and S. Z. Hussain, "A closer look at intrusion detection system for web applications," *arXiv*, vol. 2018, 2018.

[28] P. Lubartowicz and B. Pańczyk, "Performance comparison of web services using Symfony, Spring, and Rails examples," *Journal of Computer Sciences Institute*, vol. 17, no. November, pp. 384–389, 2020.

[29] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, 2012.

[30] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Piatt, "Support vector method for novelty detection," *Advances in Neural Information Processing Systems*, pp. 582–588, 2000.

[31] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," vol. 29, p. 93–104, May 2000.

[32] M. Ashrafuzzaman, S. Das, A. A. Jillepalli, Y. Chakhchoukh, and F. T. Sheldon, "Elliptic Envelope Based Detection of Stealthy False Data Injection Attacks in Smart Grid Control Systems," *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020.

[33] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International Conference on Machine Learning*, 2018.

[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[35] J. Davis and M. Goadricht, "The relationship between precision-recall and roc curves," in *International Conference on Machine Learning*, 2006.

[36] K. H. Brodersen, C. S. Ong, K. E. Stephany, and J. M. Buhmann, "The binormal assumption on precision-recall curves," *International Conference on Pattern Recognition*, 2010.