Hindawi
Wireless Communications and Mobile Computing
Volume 2020, Article ID 8838571, 13 pages
https://doi.org/10.1155/2020/8838571

Research Article

A Host-Based Anomaly Detection Framework Using XGBoost and LSTM for IoT Devices

Xiali Wang[1,2] **and Xiang Lu**[1,2]

[1]*Institute of Information Engineering, CAS, 100093, China*
[2]*School of Cyber Security, UCAS, 100049, China*

Correspondence should be addressed to Xiang Lu; luxiang@iie.ac.cn

Received 28 March 2020; Revised 16 June 2020; Accepted 9 July 2020; Published 5 October 2020

Academic Editor: Ximeng Liu

The Internet of Things (IoT) is rapidly spreading in various application scenarios through its salient features in ubiquitous device connections, ranging from agriculture and industry to transportation and other fields. As the increasing spread of IoT applications, IoT security is gradually becoming one of the most significant issues to guard IoT devices against various cybersecurity threats. Usually, IoT devices are the main components responsible for sensing, computing, and transmitting; in this case, how to efficiently protect the IoT device itself away from cyber attacks, like malware, virus, and worm, becomes the vital point in IoT security. This paper presents a brand new architecture of intrusion detection system (IDS) for IoT devices, which is designed to identify device- or host-oriented attacks in a lightweight manner in consideration of limited computation resources on IoT devices. To this end, in this paper, we propose a stacking model to couple the Extreme Gradient Boosting (XGBoost) model and the Long Short-Term Memory (LSTM) model together for the abnormal state analysis on the IoT devices. More specifically, we adopt the system call sequence as the indicators of abnormal behaviors. The collected system call sequences are firstly processed by the famous n-gram model, which is a common method used for host-based intrusion detections. Then, the proposed stacking model is used to identify abnormal behaviors hidden in the system call sequences. To evaluate the performance of the proposed model, we establish a real-setting IP camera system and place several typical IoT attacks on the victim IP camera. Extensive experimental evaluations show that the stacking model has outperformed other existing anomaly detection solutions, and we are able to achieve a 0.983 AUC score in real-world data. Numerical testing demonstrates that the XGBoost-LSTM stacking model has excellent performance, stability, and the ability of generalization.

1. Introduction

As the commercial 5G rolls out, innumerable studies [1–3] predict that a bump from 5G technology will spur IoT innovations to be an integral part of our economy and lifestyle [4], by virtue of its technical advantages in greater transmission speed, lower latency, and a greater number of device connections [5]. In such an exciting 5G era, the full potential of IoT will burst out in the way of Industrial IoT (IIoT) [6], Internet of Medical Things (IoMT) [7], and so on, by planting various sensors into cars, IP cameras, actuators, cardiac pacemaker, and insulin pumps. Gartner, Inc. reported that about 8.4 billion IoT endpoints were deployed in 2017 and will reach 20.4 billion by 2020 [8]. By connecting objects, IoT is bringing tremendous changes to the world, from significant productivity improvement, accurate predictive analysis, and rapid response to the innovation of business models.

Yet, along with all kinds of benefits brought by IoT, it also represents a growing attack surface and exposes its operators, managers, and customers to high-profile cyber threats [9], which are frequently inducing shocking IoT security breach incidents. Kaspersky said that the number of IoT attacks in the first half of 2019 is 9 times more than that in the same period of 2018 [10]. In such attacks, IoT devices, especially those exposed directly on the Internet, like IP cameras, Wi-Fi routers, and smart home facilities, are always the

principal victims. According to the latest released report by Palo Alto Networks [11], 57% of IoT devices are vulnerable to medium- or high-severity attacks, making IoT the low-hanging fruit of attackers. For example, the most notorious malware targeting IoT devices, Mirai, was first found in August 2016, which launched historically distributed denial-of-service (DDoS) attacks globally in 2016 by compromising IoT devices with weak passwords towards a large-scale IoT botnet. Since Mirai's source code was published online [12], variants of Mirai with customized functionalities, as well as new exploits, keep evolving to be a continuing threat [13].

We can see that, in almost every IoT security incidents, the fragile IoT terminal devices are always the Achilles' heel suffering from unencrypted traffics, outdated software, and neglected security designs. Therefore, the security issues of IoT endpoints, in terms of exploits, password guessing, IoT worms, and so on, should be one of the top considerations in the IoT threat landscape. How to secure massively deployed IoT devices should be the primary question to be answered by cybersecurity researchers.

To protect IoT endpoint devices against all kinds of orchestrated attacks, researchers from both academia and industry designed a large number of mechanisms in recent years. In this paper, we propose an abnormal state detection framework to indicate ongoing exploit-compromised or malware-infected attack behaviors happening on IoT devices, thereby identifying those eccentric endpoints. The basic idea of our framework is on the assumption that an IoT device's behaviors are commonly predictive, repetitive, and periodic in statistics [14]. If attacks occur, the compromised IoT devices are prone to showing some unusual behaviors in different manners, like unexpected traffics and strange device actions. Based on this idea, we focus on abnormal behaviors of embedded operating systems to identify malicious IoT device attacks and leverage the statistical analysis of system calls to indicate abnormal system behaviors from device attacks. To this end, we firstly set up an abnormal system call dataset captured in different attack scenarios. Then, to achieve more accurate results of attack indications, we then develop a fusion model for system call sequence analysis by coupling a machine learning model (XGBoost) and a deep learning model (LSTM neural network) together using stacking theory. Our experimental results demonstrate that the system call trace is a good indicator to identify abnormal behaviors from normal behaviors, and the performance of our anomaly detection framework exceeds the other excellent machine learning models.

From the above, we describe our contributions in this paper as follows:

(1) We choose system call as the indicators and deploy several types of attacks to collect normal behaviors and abnormal behaviors in IoT devices

(2) We propose an anomaly detection framework and design a fusion model using the stacking method. As for the base models, we select the XGBoost algorithm and LSTM neural network. Moreover, the metamodel is the logistic regression model

(3) We evaluate the performance of our anomaly detection framework, and the results show that it is valid and stable; moreover, it has good generalization

The following paper is organized as follows. Section 2 introduces related works regarding the intrusion detection of IoT devices. Section 3 illustrates the proposed anomaly-based intrusion detection framework, as well as methods designed for data processing and analysis. In Section 4, we detail the experiment setup and discuss the experimental results. Section 5 concludes our works in this paper.

2. Related Works

As a classical cybersecurity research area, anomaly detection-based intrusion detection system designs keep evolving with fruitful literatures. In general, according to the information used for analysis and employed techniques for the decision-making of deviation away from normal behavior, the intrusion detection can be divided into five groups [15], that is, statistical methods, rule-based methods, distance-based methods, profiling methods, and model-based approaches. In these years, IoT device-oriented IDS is becoming the new hot topic in the classical area. With the rapid advance of artificial intelligence (AI) technologies, we can see an obvious trend; more and more IoT device-oriented IDS mechanisms are established on various AI models to achieve efficient feature extraction and accurate behavior patterns. Although many mechanisms turn to AI models, there also exist many differences between them, which mainly originate from data types used for anomaly indication and models used for data analysis.

[16] developed a model for intrusion detection in IoT microservices by using clustering models like K-means and BIRCH, based on the traffic dataset captured from 4 different emulated IoT sites. Similar to captured network traffics in [16, 17] adopted a deep learning model, named the Dense Random Neural Network, to detect security breaches in a smart home application scenario. In [18], the authors proposed a host-based intrusion detection system on the basis of a machine learning approach. In their designs, the immunity-inspired algorithms are involved to distinguish whether the current behavior patterns are matching with desired ones; if not, it means that the attack is happening. Also, for the smart home IoT scenario, [19] implemented a supervised intrusion detection system. For the sake of performance evaluations of the implemented system, the authors deployed 12 attacks, which belong to 4 threat types, including denial of service (DoS) attack, man-in-the-middle (MITM)/-spoofing attack, reconnaissance attack, and replay attack.

Despite the fact that AI-based IoT IDS schemes are the dominant techniques, there is also a lot of work to do towards efficient and accurate anomaly detection on IoT endpoints. In existing literatures, network traffic data is the commonly used indicators to identify device attacks, since it is easy to be collected. Therefore, they are actually the so-called network-based IDS, not the host-based IDS [20]. However, in the real-setting IoT IDS scenarios, the salient features of IoT devices on massive proprietary protocols, uninterrupted

running, and sometimes encrypted traffics will significantly deteriorate the network-based IDS. Thus, the host-based IDS framework, as proposed in this paper, is indispensable for IoT security.

3. System Framework and Scheme Design

In this section, we firstly introduce the outline of the proposed anomaly-based intrusion detection framework, including explanations of the main building blocks. Then, we illustrate critical approach design used in data processing and model analysis.

3.1. Methodology Outline. As presented above, this work is aimed at proposing an anomaly-based intrusion detection framework to identify ongoing attacks on IoT devices. To achieve this objective, we leverage system call sequences as the anomaly indicator to capture malicious system behaviors brought by vulnerability exploitations or malware infections. Figure 1 shows the main building blocks of the proposed anomaly-based IoT IDS framework, which is also the basic idea outline of this paper.

To make the framework implemented as the running intrusion detection module embedded in the IoT device, we have to finish several critical scheme designs, regarding the construction of training data set, the system call data preprocessing, the data analysis model design, and performance evaluations, as shown in Figure 1, all of which will be illustrated in details in the following subsections.

3.2. Training Dataset Constructions. Different from network-based IoT IDS, which mainly depends on the captured network traffic data for data analysis and modelling, in our framework design, we leverage system call sequences as the anomaly indicator for intrusion detection inside the IoT device, also called as host-based IDS. Therefore, how to collect system call data, especially in different attack scenarios, becomes the primary issue in our framework. To construct such a training dataset, we tried to look for IoT system call sequence dataset under attack conditions, but there is no existing one. In this case, we turn to embedded Linux for system call sequence data collection, which is the most popular operating system used in IoT devices [21], taking about 21% market shares globally. More specifically, we program shell scripts by leveraging the "strace" command [22] and the "crond" mechanism [23] for system call sequence collection when different attacks are loaded. Among these two, the "strace" command is usually used as the system call tracer, while the "crond" mechanism is designed to execute scheduled commands. With the shell script, we are able to capture traces of all processes (except for "strace" its own process) and stored them in the file while the attack is deployed.

In terms of typical IoT attacks, we choose a HiSilicon Hi3516CV300-based IP camera as the example to deploy attacks. For the dataset construction, we collect system call sequence data in 5 classes, as shown in Table 1. Class 0 data is captured in the normal state used to indicate expected system behaviors when the IP camera is running, whose main functions include scene sensing, video coding, video transmission, and even web access. Class 1 data is acquired when a vulnerability exploit is executed. The vulnerability is from CVE-2016-5195 [24], also known as "Dirty COW," which allows local users to gain privileges by leveraging incorrect handling of a copy-on-write feature to write to a read-only memory mapping. Class 2 data is collected when a notorious malware, BASHLITE [25], is infecting devices. Class 3 data is from an emulated abnormal (malicious) operation on the IP camera device; that is, we add a user and set the password on the device. Class 4 data is from a memory leak attack, in which we keep creating threads to request connections to the embedded Real-Time Streaming Protocol (RTSP) stack and lead to memory leak and a fatal firmware error.

As the way in Table 1, we totally collect about 5.9 million system calls in different classes. We then use the dataset to verify the effect and efficiency of the framework proposed in this paper.

3.3. Feature Extraction and Selection. The objective of data preprocessing in this work is to extract features from seemingly clueless system call sequences for the following anomaly decision-making. Figure 2 shows an example of a system call sequence. To achieve efficient feature extractions, we take 2 key steps for dataset sorting.

The first step is to digitalize the dataset by employing the system call table to replace the system call function with a unique system call number, while ignoring impacts of parameters. For example, after the digitalization, the sequence shown in Figure 2 turns to the notation of a trace as follows:

$$\text{Trace} = (5, 114, 78, 174). \tag{1}$$

After the digitalization of dataset, in the second step for classification of the behavior from system call traces, we should extract features from the system call traces. Usually, data representation techniques can be used to convert the system call trace into a feature vector. We adopt the famous n-gram model [26] from the Natural Language Processing area, which is also a common data processing method in intrusion detection scheme design [27].

N-gram is a contiguous of n items from a given sample of text or speech; it is an algorithm based on the statistical language model. Its basic idea is to slide the contents of the text into n-size Windows according to bytes, forming a sequence of n-length byte fragments. Each byte fragment is called gram. The occurrence frequency of all grams is counted and filtered according to the threshold set in advance to form the key gram list, namely, the vector eigenspace of the text. Each gram in the list is an eigenvector dimension.

To explain this language model, let us consider the following trace:

$$\text{Trace} = (6, 6, 63, 6, 42, 120, 6, 195, 120, 6, 6, 114, 114). \tag{2}$$

We can set n to different values, such as 3, 5, and 7. If we set n as 5, we can get the following set of sequences:

$$\text{Trace} = (6, 6, 63, 6,120), (6, 63, 6, 120, 6), \cdots, (120, 6, 6, 114, 114). \tag{3}$$

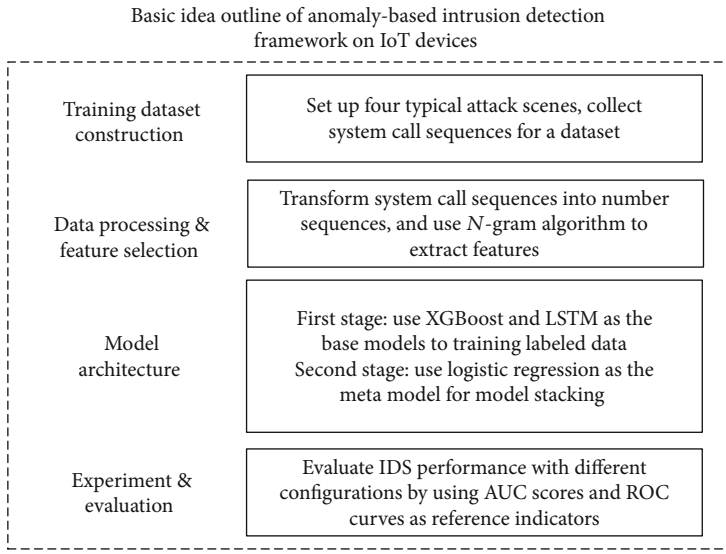

Figure 1: Methodology outline of this paper.

Table 1: Five classes of collected system call sequence dataset.

No.	Class name	State	Notes
1	Class 0	Normal state	Syscall sequence data in normal state
2	Class 1	Vulnerability exploiting	Syscall sequence data in CVE-2016-5195
3	Class 2	Malware infection	Syscall sequence data in BASHLITE malware
4	Class 3	Abnormal operation	Syscall sequence data in user add operation
5	Class 4	Memory leak	Syscall sequence data in RTSP memory leak

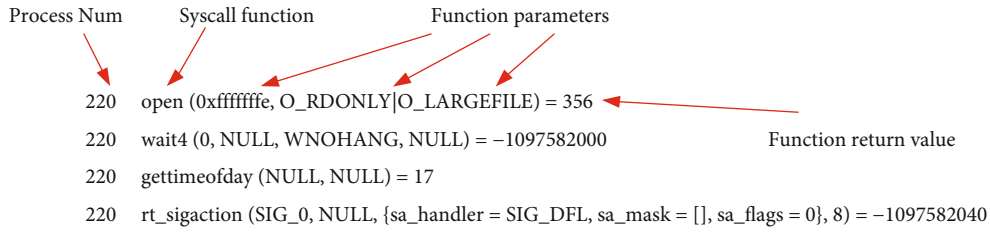

Figure 2: An example of a system call sequence.

By considering collected system call traces as a set of document and system calls as words, we can apply classical data representation and classification techniques used in the area of natural language processing (NLP) and information retrieval (IR).

Let us consider a system call trace S with L system calls, the n-gram is defined as tuple $(g_1, g_2, g_3, \cdots, g_n)$ of n terms where n is generally a small positive integer. For a given gram, its weight $\omega(g)$ in trace S is given by

$$\omega(g) = \frac{f(g)}{|L|}, \tag{4}$$

where $f(g)$ is the frequency of g in S and $|L|$ is the length of one trace.

Through using the n-gram language model, the syscall number sequences are processed into many subsequence features and build a dictionary which key is the grams and the value is the frequency. These features from the system call traces can be used for normal or abnormal behavior classification.

After we extract the n-gram features and compute the frequency of the weight, it is easy to find that, as n increases, the feature extraction work will become a time-consuming task. So, we must select some important features as the model training features. We set a parameter m; it represents the top m features that we pick based on frequency. We use the top m features to sketch the system call behaviors in IoT devices and construct models based on these features.

3.4. Extreme Gradient Boosting. In Sections 3.4 and 3.5, we will introduce the machine learning methods we used in

our anomaly detection system. In our anomaly detection framework, we chose the XGBoost (Extreme Gradient Boosting) [26] classifier as one base model of the anomaly detection framework.

XGBoost is a decision-tree-based ensemble machine learning algorithm that uses a gradient boosting algorithm to a known dataset and then classifies the data accordingly. The two reason to use XGBoost are also the two goals of a machine learning project: execution speed and model performance. XGBoost is really fast when compared to other implementations of gradient boosting. From Szilard Pafka's experiment [28], he proved that XGBoost is fast, memory efficient, and of high accuracy. So, we use XGBoost as a part of our anomaly detection architecture to obtain perfect performance.

Here are XGBoost's mathematical explanation. XGBoost is composed of multiple CARTs. CART is known that a basic decision tree can be established on the concept of entropy. The object of CART is Gini coefficient:

$$\begin{aligned} \text{Obj} : \min \text{Gini}_{\text{index}}(D, a) &= \sum_{v=1}^{V} \frac{|D^v|}{|D|} \text{Gini}(D^v), \\ \text{Gini}(D) &= \sum_{k=1}^{K} \sum_{k' \neq k} p_k p_k' = 1 - \sum_{k=1}^{K} p_k^2, \end{aligned} \tag{5}$$

where a is one of the attributes we selected, V is the scale of a, v is one of the values of the attribute a, and K is the scale of labels. Intuitively, the Gini coefficient reflects the probability of two samples in the dataset that the labels are different. Moreover, this is the principle to construct one tree.

The goal of XGBoost is to fit the residual. Residual means the difference between the real value and predicted value. XGBoost is defined as an addition model:

$$F(X, w) = \sum_{k=0}^{K} \alpha_k h_k(X; w_k) = \sum_{k=0}^{K} f_k(X, w_k), \tag{6}$$

where X is the input data, $F(X, w)$ is the model that we finally get, h_k means one single tree, w is the parameter of the tree, and α_k is the weight of the tree. By minimising the loss function, we can get the optimal model $F(X, w)$. The loss function is defined as

$$\begin{aligned} F^* &= \arg \min_F \sum_{k=0}^{K} L(Y_i, F(X_i; w_k)), \\ \text{loss} &= \sum_i l\left(\hat{Y}_i, Y_i\right) + \sum_k \Omega(f_k), \\ \Omega(f_k) &= \gamma N_{\text{leaf}} + \frac{1}{2} \lambda \|w_k\|^2, \\ l\left(\hat{Y}_i, Y_i\right) &= (Y \wedge_i - Y_i)^2, \end{aligned} \tag{7}$$

where N_{leaf} means the number of leaf nodes in the decision tree. This is a published measurement to restrain the complexity of our model. Y_i is the real value and $\hat{Y}_i$ is the predicted value. γ and λ are the parameters.

3.5. Long Short-Term Memory. In our anomaly detection architecture, we also use the LSTM (Long Short-Term Memory) neural network as the other base model in the model stacking algorithm. In this part, we introduce the LSTM neural network and explain why we use this model to make classifications in our anomaly detection architecture.

The LSTM model is an improved version of the RNN model, which overcomes back-propagation gradient dispersion or gradient explosion and is more suitable for processing sequence data with long-term dependence. So, we choose the LSTM model as the other base model to combine with the XGBoost model in order to get a valid and stable anomaly detection system for IoT devices.

Here is the basic theory of the LSTM model. Figure 3 shows a single LSTM cell. And we describe the gate mechanism which is the special characters of the LSTM model and explain the equations to compute the values of three gates and cell state.

Compared with the RNN neural network, the LSTM neural network adds the gate mechanism: input gate, forget gate, and output gate in each neuron cell. It is this mechanism that enables LSTM to solve the gradient disappearance and gradient explosion problems in the long sequence training. Each forget gate reads the output value x_t and h_{t-1}, splices the two values, and retains information through the control of the forget gate. LSTM internal operations mainly include sigmoid, tanh, addition, and multiplication, among which sigmoid and tanh are two different activation functions reserved for the next memory unit. The formula is

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{8}$$

The tanh function is a hyperbolic tangent function that maps any value to the interval [-1, 1]. The formula is

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{9}$$

As shown in Figure 2, state preservation is very important in the LSTM network model. C_{t-1} and C_t in Figure 2 represent states that are used for information that needs to be retained over time. The specific training process of LSTM is as follows.

In the first step, LSTM uses the forget gate to determine the retention degree f_t of information. In formula (10), W_f is the weight matrix of the forget gate and b_f is the offset value of the forget gate. This step integrates the previous output value with the current input value and passes it to the next memory unit through the forget gate. If the forget gate outputs zero, the previous memory will be cleared. If the output is one, all the previous content is credited to the memory unit:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right). \tag{10}$$

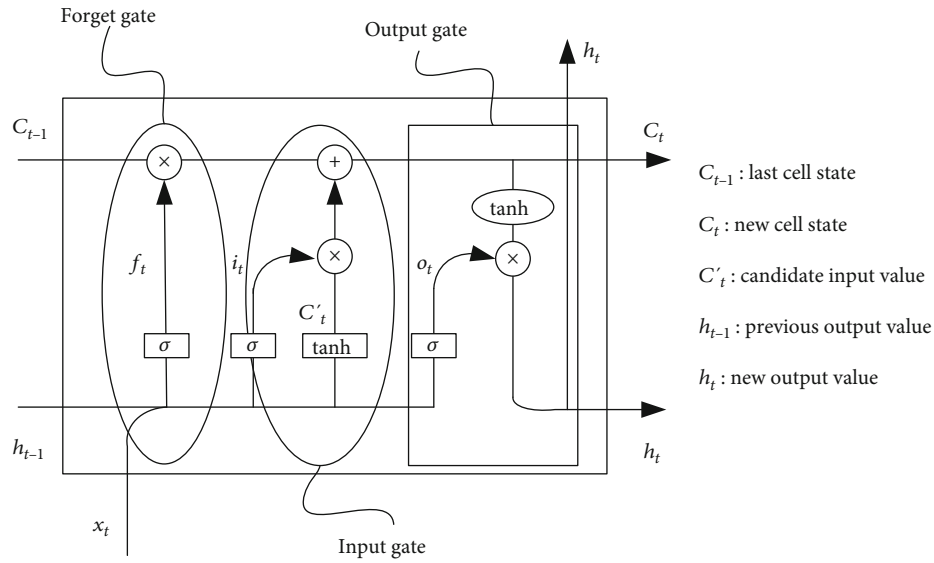

Figure 3: LSTM algorithm structure diagram.

In the second step, select memory. This step is used to determine the information held in the cell state. It is mainly divided into the generating temporary new state and updating old state. Assume that $c_t^{'}$ is a temporary new state and i_t is the input gate output to determine which data from the previous step needs to be updated. The old state c_{t-1} multiplies f_t to decide which information to forget and which information to keep. Finally, the new temporary state $c_t^{'}$ multiplies i_t to get the new state c_t. Among these phases,

$$\begin{aligned} c_t^{'} &= \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right), \\ i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right). \end{aligned} \tag{11}$$

Finally, output. LSTM firstly determines the output information through the sigmoid function and then maps the unit state to [-1, 1] through the tanh function. The output value is obtained by multiplying the sigmoid threshold:

$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right). \tag{12}$$

In the above steps, W_c, W_i, and W_o are the weight matrixes. The matrix is randomly generated at the beginning of the iteration, and in each process of backward transmission, the information of the previous unit is retained and updated after obtaining the accurate matrix.

3.6. Anomaly Detection Architecture. In this part, we introduce the anomaly detection architecture which is applied in the model building stage of our anomaly detection framework.

So far, we have used the n-gram language model to segment the system call number sequences, selected the top max features in order to solve the problem of too many feature dimensions, and introduced the core principles and main advantages of XGBoost and LSTM neural network. Finally, we use these two models to construct an anomaly detection architecture which is combined with feature selection of the tree model and sequence memory of the LSTM neural network.

In our anomaly detection architecture, we use the XGBoost and LSTM neural network to integrate as base models. The algorithm of model integration is stacking. It is an ensemble machine learning algorithm and uses a meta-learning algorithm to learn how to best the predictions from two or more base machine learning algorithms and base deep learning algorithms. The benefit of stacking is that it can harness the capabilities of a range of well-performing models on a classification task and make predictions that have better performance than any single model in the ensemble.

Let us simply introduce the stacking theory. The architecture of a stacking model involves two or more base models, often referred to as level 0 levels (base models), and a metamodel that combines the predictions of the base models, referred to as a level 1 model (metamodel). The metamodel is trained on the predictions made by base models on out-of-sample data. That is, data not used to train the base models is fed to the base models, predictions are made, and these predictions, along with the expected outputs, provide the input and output pairs of the training dataset used to fit the metamodel.

The main idea of constructing a predictive model by combining different models can be schematically illustrated in Figure 4.

Figure 4 describes the key information for model stacking. At first, initial training data (X) has m samples and n features. There are M different models that are trained on X (by some method of training, like crossvalidation). Each model provides predictions for the outcome (y) which are then cast into a second level training data X^{l2} which is now $m * M$. Namely, the M predictions become features for this second level data. A second level model (or models) can then be trained on this data to produce the final outcomes which will be used for predictions.

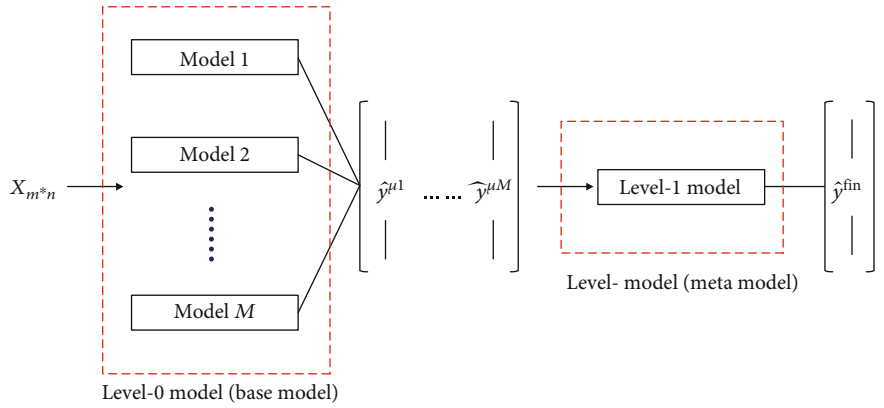

Figure 4: The main idea of constructing a predictive model by combining different models.

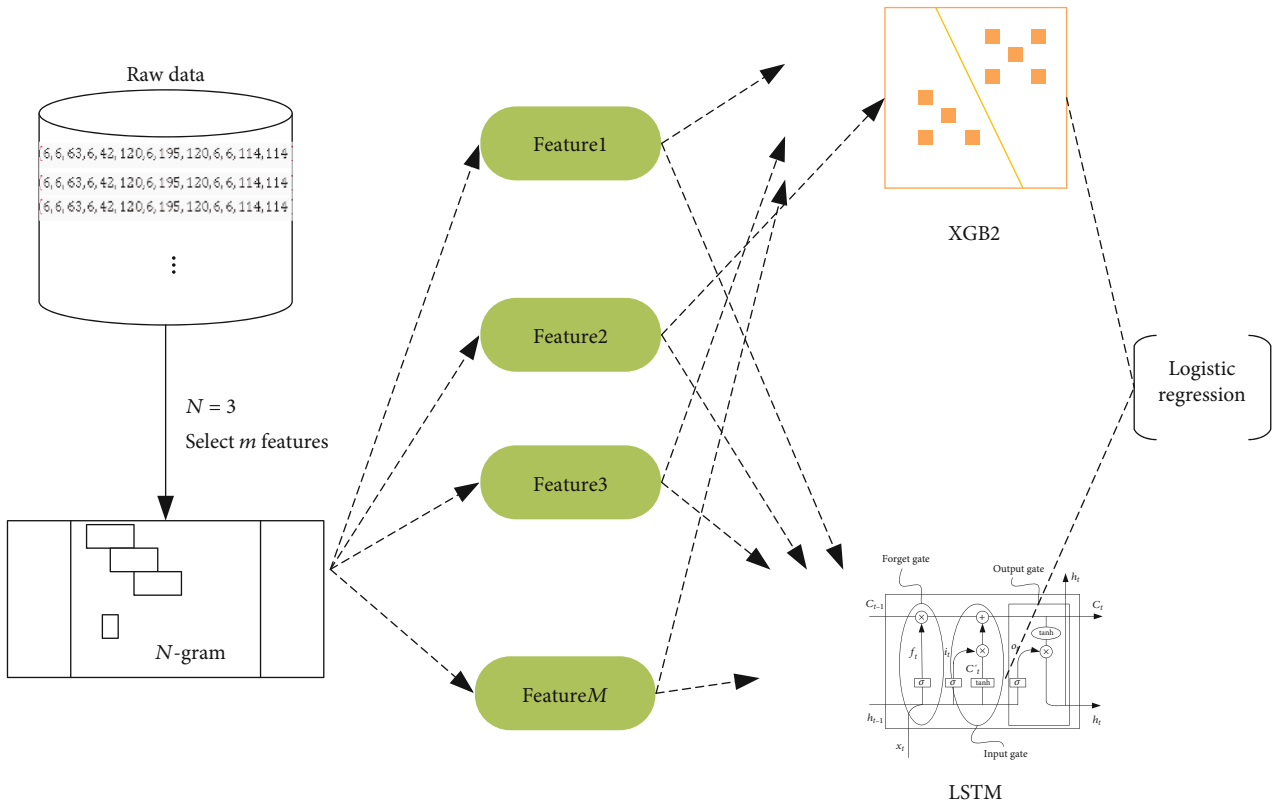

Figure 5: Model stacking with XGBoost and LSTM.

Based on the stacking theory, we design a procedure to classify abnormal behavior and normal behavior. The architecture of the procedure is shown in Figure 5, and a detailed description is as follows.

Step 1. (Data collection). We deploy several types of attacks in IoT devices and leverage the "strace" command and the "crond" mechanism for system call sequence collection.

Step 2. (Feature extraction and important feature selection). We use the n-gram model to extract features, and after that, we select a set of features with top frequencies as the input for the model construction in our framework.

Step 3. (Base model training). Use the selected features; we labelled the dataset: "0" is used to indicate normal behaviors, and "1" is used to indicate abnormal behaviors. In model construction, we use stacking theory to build a fusion model. The base models are XGBoost and LSTM models.

Step 4. (Model stacking using logistic regression). This step uses the logistic regression model to train the predictions

from the above base models; the prediction from the logistic regression model will be considered the final predictions. We can use the prediction to predict the unknown system call trace to tell it normal or abnormal and test the performance of the anomaly detection framework.

Step 5. (Model evaluation and model analysis). We test the performance of the anomaly detection framework with some baselines. After that, we further analyze the model and discuss how some changes affect the performance of our anomaly detection framework.

4. Evaluation

4.1. Experimental Settings. In our experiment, as mentioned before, we used a HiSilicon Hi3516CV300-based IP camera as an example, to deploy the system call sequence data collection program, as well as 5 different device states illustrated in Table 1, for the real-setting data collection.

According to Section 3.2, we construct our training dataset for our anomaly detection framework. In the experiment of the model construction stage, we split the dataset to training data and test data. We use just training normal data in training parse with a normal label, and at test parse, we use all attack data and partial data to measure the performance evaluation. Table 2 summarizes the training data and test data statistics.

After the settings of the experimental environment, we also set the basic model parameters of XGBoost and LSTM. As Table 3 shows, booster is used to determine what model to adopt. We choose "gbtree" for the tree booster. Max_depth is the maximum tree depth for base learners. An objective specifies the learning task, and we use binary:logistic for our framework. Learning_rate makes the model robust by shrinking the weights on each step.

In the LSTM model, there are several parameters which effect the performance of the LSTM model. We choose the parameters as Table 4 shows. The drop rate can slow down with regularization methods. We set three hidden layers to explore additional hierarchical learning capacity. As for the epochs of the training parse, we set it between 50 and 6000. Batch size controls how often to update the weights of the network; we select this parameter from 32 to 256. Last but not least, we set the learning rate to 0.001.

4.2. The Experiment Indicators. As for the classification algorithm, the evaluation indexes are accuracy (ACC), precision-recall curve (PRC), characteristic curve (receiver operating characteristic, ROC), and AUC (Area Under the Curve) of the curve area. Compared with PRC, the advantage of ROC curves and AUC is that they can remain basically unchanged when the positive and negative sample distribution changes in the test data.

The ROC curve is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. Let us define the true positive rate and the false positive rate:

$$\begin{aligned} \mathrm{TPR} &= \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \\ \mathrm{FPR} &= \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}}. \end{aligned} \tag{13}$$

In the above computational formula, we consider TP as true positive, FP as false negative, FN as false negative, and TN as true negative. We use the ROC curve and AUC as the evaluation metrics for our anomaly detection framework.

Moreover, in Section 4.4, we also analyze the performance of our approach by using the precision, recall, and F-measure in order to perform a comprehensive performance evaluation. Precision = TP/(TP + FP) shows the percentage of true anomalies among all anomalies detected, Recall = TP/(TP + FN) measures the percentage of anomalies in the datasets being detected, and F-measure = 2 ∗ Precision ∗ Recall/(Precision + Recall) is the harmonic mean of the two.

Table 2: The training data and the test data statistics.

Dataset	Samples	Numbers of syscall	Class of traces
Training data	4233	4,729,365	Normal
Test data	1048	1,171,359	Attack

Table 3: The parameters of XGBoost classifier.

Parameters	Value
Booster	gbtree
Max_depth	4
Objective	Binary:logistic
Eval_metric	AUC
Learning_rate	0.01

Table 4: The parameters of LSTM.

Parameters	Value
Dropout rate	0.2
Hidden layers	3
Epoch	50~6000
Batch size	32~256
Learning_rate	0.001

Table 5: The comparison with other models.

Method	AUC
SVM	0.924
Isolation forest	0.931
XGBoost	0.966
LSTM	0.978
Fusion model	0.983

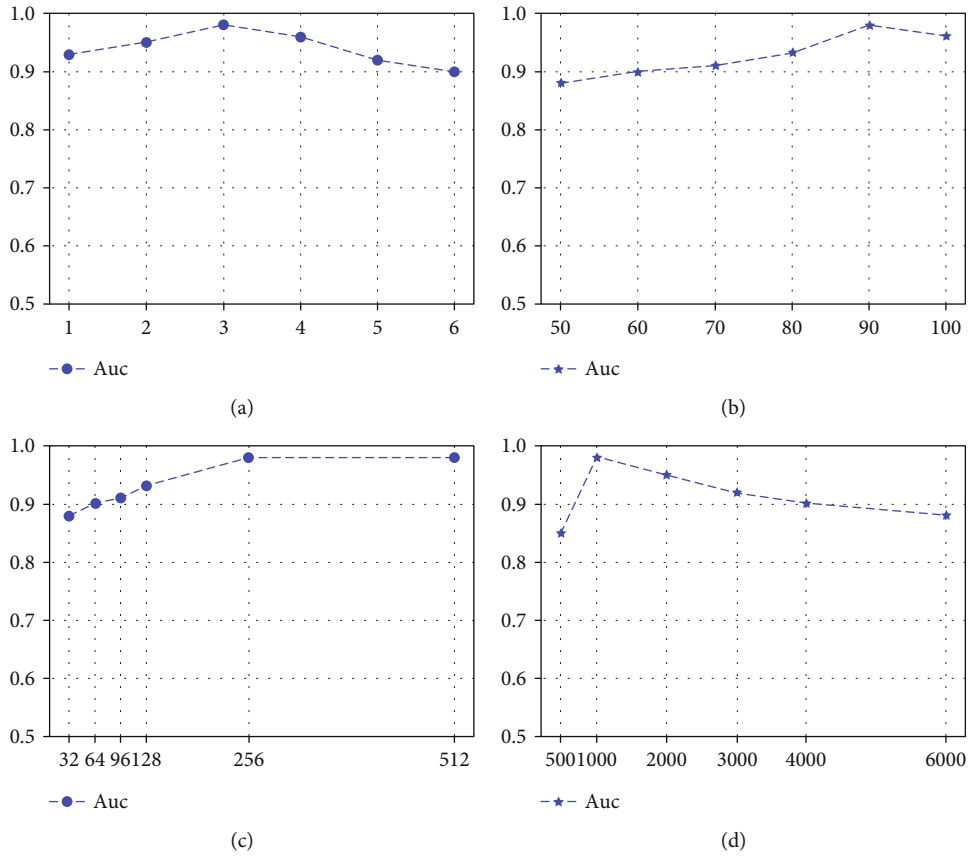

Figure 6: The performance with different parameters.

4.3. Comparison with Baselines. For baselines, the general solution for anomaly detection is to use a single machine learning model or approach. So, we select the most popular/-representative anomaly-based approaches which have been used in these research domains. We consider the XGBoost and LSTM as two baselines and also consider the SVM [29] and isolation forest [30] as two baselines. SVM are good at classification problems [31]. The isolation forest is an unsupervised learning algorithm for anomaly detection that works on the principle of isolating anomalies. For our framework, we set the numbers of n-gram in the feature extraction stage (n : between 2 and 7), the numbers of features selecting from the first layers (m: between 50 and 100), the number of batch size (b : between 32 and 512), and the number of training epochs (e: between 500 and 6000). By default, our anomaly detection framework use $n = 3$, $m = 80$, $b = 256$, and $e = 1000$ for evaluation and investigate the impacts of these parameters in our experiments.

Table 5 shows the performance of our anomaly detection framework and other popular approaches. The result is shown in Table 5; we can find that XGBoost and LSTM neural network can perform well on our dataset. Clearly, our anomaly framework has achieved the best overall performance by which the AUC score is 0.983.

4.4. Model Analysis. To have a clear understanding of our anomaly detection framework, we conduct a model ablation to observe the effectiveness of the crucial structure and parameters for performance. By choosing the parameters and using the model stacking, we can achieve excellent performance. The comparative results are shown mainly in the following subsections.

4.5. Discussion of the Performance Impact of Different Parameters in Data Preprocessing Parse. We study the impact of different parameters, including n, the numbers of features in data preprocessing before training in our anomaly detection framework. Figure 6 shows an in-depth comparison using AUC. We change the values of one parameter per time while the other parameter values are default in each experiment. The parameters and their settings are as follows.

For the parameter of the N-gram language model, we take 2 as the initial configuration of the parameter of the n-gram method; then, we use from 3 to 7 to construct a comparative trial. As shown in Figure 1, we compare these n values on our model. It can be summarized that the best method is to use the 3-gram method; the following experiments will use a 3-gram language model to prepare the input data.

The parameter of the number of important features is selected by frequency. We give a range value of l between 50 and 100. We find that when we control the value around 80, the performance of our anomaly detection framework can achieve the best result. It can be seen from the experiment that, when l is less than a certain threshold, the more features, the better result.

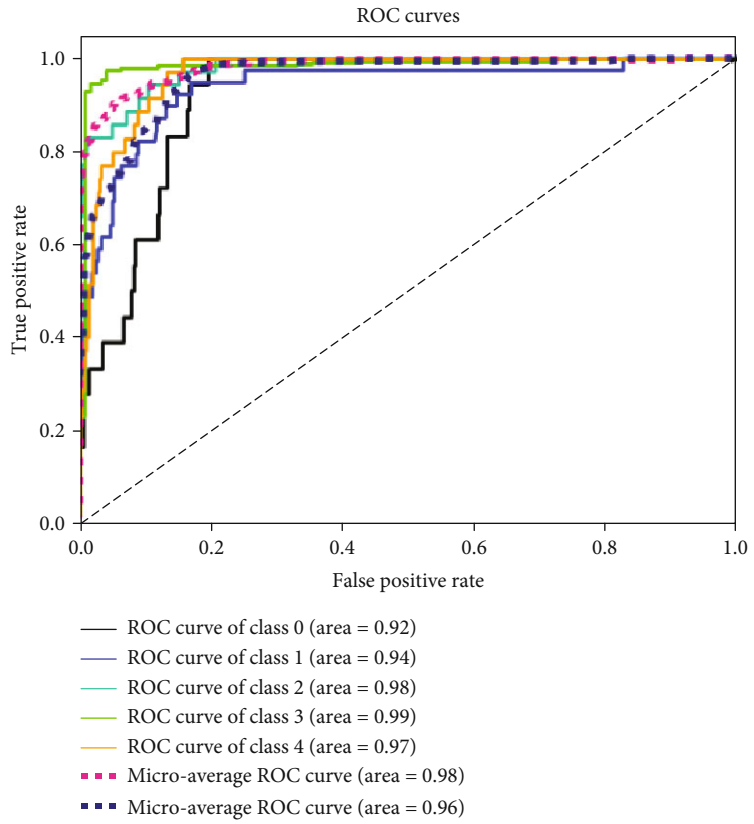

Figure 7: The ROC curves of different types of data classes.

In Figures 6(c) and 6(d), we design a systematic test for evaluating LSTM model configurations. We set different values of batch size and epochs in the LSTM classifier to evaluate the impact of these parameters. The result shows that when the batch size is 256, model performance peaked. And when the number of epochs is 1000, our framework can get best performance on the anomaly detection task.

As for the other parameters of our anomaly detection framework, we also tune them for better performance, for example, the k value in k-fold crossvalidation. In the third level of our anomaly detection framework, we use 10 to observe the results because a value of $k = 10$ is very common in the field of machine learning and is recommended by specialists and scholars.

In general, the performance of our anomaly detection framework is relatively stable that the results of experiment are relatively stable when adjusting the parameter values. Also, we get better performance by tuning the key parameters to improve our anomaly detection framework.

4.6. Discussion of the Performance Impact of Different Types of Attacks. To have a clear understanding of how our anomaly detection framework performs on each class of abnormal events, we use one class per time to our fusion model while the other events are null. Figure 7 shows the results of ROC curves of our anomaly detection model on different attacks. We find that some attacks are easy to detect from others, such as the class 2 and class 3 attacks, while some attacks have low probabilities to be detected using our anomaly detection framework, such as the class 0 (that is normal state) and class 1 attacks.

What is more, from the overall attacks' anomaly detection task and the single attack anomaly detection task, we find that the detection including the overall attacks performs better than all detections on a single abnormal state.

4.7. Discussion of the Performance of Different Datasets. Our anomaly detection framework for IoT devices includes the dataset obtained by ourself. In order to verify the validity of the dataset and our anomaly detection framework, we use the Australian Defence Force Academy-Linux Dataset (ADFA-LD) for a comparison. ADFA-LD is a similar dataset collecting system call traces on the Linux system for PCs. Figures 8 and 9, respectively, represent the detection performance of our framework on the two datasets. To be specific, Figure 8 shows the result of the dataset constructed by ourselves, and Figure 9 shows the result on the ADFA-LD dataset. From the comparison of performance, our framework can be proved to be effective and accurate. What is more, our framework has the ability of generalization so that it can be used in a similar dataset.

4.8. Discussion of the Performance of Different Metrics. A classifier is only as good as the metric used to evaluate it. A wrong metric to evaluate models will lead us to miss the expected performance of our anomaly detection framework. We compare four metrics which are mainly used for classification tasks, as shown in Table 6. The result shows that different

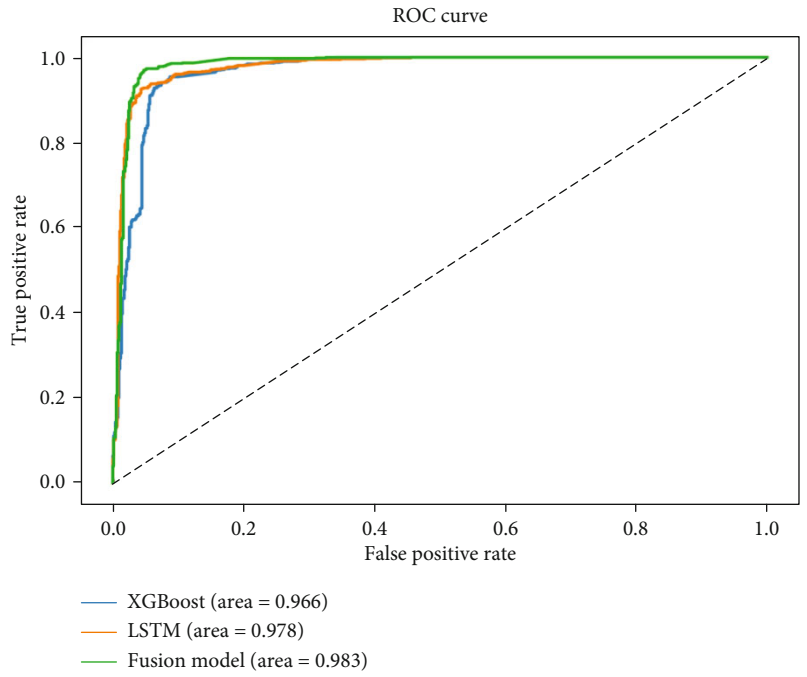

Figure 8: Performance on our datasets.

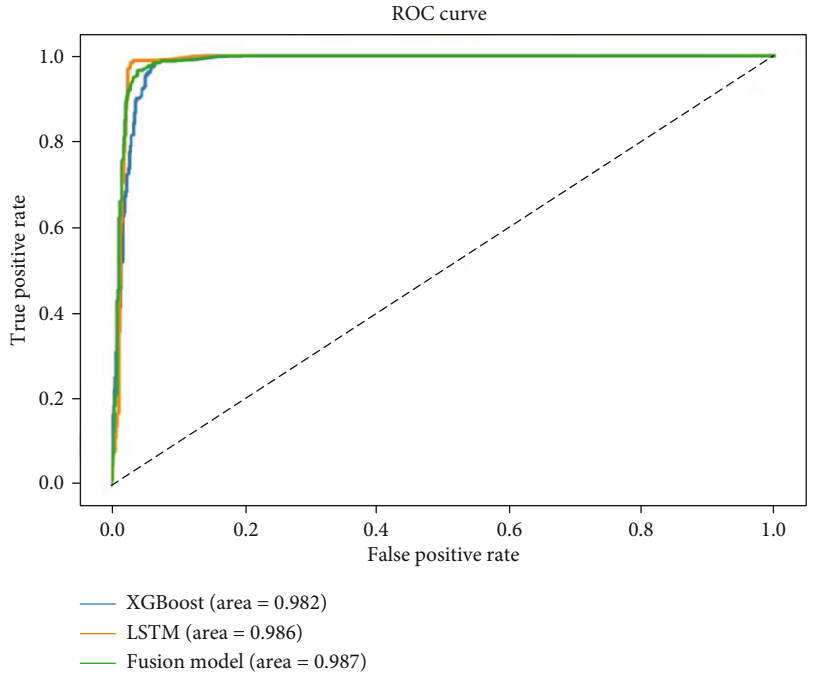

Figure 9: Performance on ADFA-LD datasets.

Table 6: Different metrics for anomaly detection framework.

Metrics	AUC	Precision	Recall	*F*-measure
Fusion model	98.3%	97.2%	98.1%	97.9%

evaluation metrics have little influence on performance. It also indicates the robustness of our anomaly detection framework.

5. Conclusion

In this paper, we propose a new anomaly-based intrusion detection framework for IoT devices. In the proposed

framework, we collect system call sequences as a dataset and use a machine learning method XGBoost and deep learning method LSTM to construct a model using stacking idea to detect the abnormal behavior from the normal behavior. The experiments prove that our anomaly detection framework is valid and has better classification performance. We propose a complete anomaly-based intrusion detection process for IoT devices, and it can be used in follow-up studies for scholars in this research field. In the future work, we can also improve the model from the following aspects: adjusting the parameters of this model and improving the diversity of the dataset.

Data Availability

The experiment data used to support the findings of this study were divided into two parts. One part of the data is constructed by authors. These data were designed for evaluation by system call anomaly-based IDS for IoT devices. Requests for access to these data should be made to Xia-Li Wang, xialiwang4@gmail.com. The other part of the data is the ADFA-LD dataset. It is designed for evaluation by system call base HIDS. The dataset can be contacted at https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

References

[1] S. Baghdassarian, *Report Highlight for Market Trends: 5G Opportunities in IoT for Communications Service Providers*, Gartner Research, 2019, https://www.gartner.com/en/documents/3964479/report-highlight-for-market-trends-5g-opportunities-in-i.

[2] M. Rouse, *Internet of Things (IoT), IOT Agenda*, 2019.

[3] J. Kaplan, *These 3 transformative technologies will have the biggest impact on 2020*, Digital Trends, 2019, https://www.digitaltrends.com/cool-tech/2020-trends-to-watch-ai-5g-iot/.

[4] P. Collela, *5G and IoT: ushering in a new era*, 2020, https://www.ericsson.com/en/about-us/company-facts/ericsson-worldwide/india/authored-articles/5g-and-iot-ushering-in-a-new-era.

[5] IoT Solutions World Congress, *Advantages of 5G and how will benefit IoT*, 2019, https://www.iotsworldcongress.com/advantatges-of-5g-and-how-will-benefit-iot/.

[6] World Economic Forum, *Accelerating the impact of industrial IoT in small and medium-sized enterprises: a protocol for action*, 2020, https://www.weforum.org/whitepapers/accelerating-the-impact-of-industrial-iot-in-small-and-medium-sized-enterprises-a-protocol-for-action/.

[7] A Whitepaper by Intersog, *What is the Internet of medical things, in the Journal of mHealth*, 2018.

[8] Gartner Inc, *Gartner says 8.4 billion connected "things" will be in use in 2017*, 2016, https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016.

[9] 5G Americas White Paper, *The future of IoT*, 2019.

[10] P. Muncaster, *Over 100 million IoT attacks detected in 1H 2019*, 2019, https://www.infosecurity-magazine.com/news/over-100-million-iot-attacks/.

[11] Unit 42, *2020 Unit 42 IoT Threat Report*, Palo Alto Networks, 2020, https://start.paloaltonetworks.com/unit-42-iot-threat-report.

[12] Krebs On Security, *Source code for IoT botnet 'Mirai' Released*, 2016, https://krebsonsecurity.com/2016/10/source-code-for-iot-botnet-mirai-released/.

[13] P. Paganini, *Mirai – the evolving IoT threat*, 2018, https://resources.infosecinstitute.com/mirai-botnet-evolution-since-its-source-code-is-available-online/#gref.

[14] M. Kuniavsky, *User experience and predictive device behaviour in the Internet of Things*, 2015, https://conferences.oreilly.com/experience-design-iot/public/schedule/detail/42519.

[15] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," *Proceedings of the 2003 SIAM International Conference on Data Mining*, pp. 25–36, 2003.

[16] M. Pahl and F. Aubet, "All eyes on you: distributed multi-dimensional IoT microservice anomaly detection," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 72–80, Rome, Italy, November 2018.

[17] E. Gelenbe and Y. Yin, "Deep learning with dense random neural networks," in *International Conference on Man–Machine Interactions*, pp. 3–18, Springer, Cham, 2017.

[18] A. Briana, B. LiEsa, R. Rahmira, and A. Esterline, "Behavioral modelling intrusion detection system (BMIDS) using Internet of Things (IoT) behavior-based anomaly detection via immunity-inspired algorithms," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, Waikoloa, HI, USA, August 2016.

[19] E. Anthi, L. Williams, M. Slowinska, G. Theodorakopoulos, and P. Burnap, "A supervised intrusion detection system for smart home IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 9042–9053, 2019.

[20] SANS, *Host- vs. Network-Based Intrusion Detection Systems*, SANS Penetration Testing, 2005, https://cyber-defense.sans.org/resources/papers/gsec/host-vs-network-based-intrusion-detection-systems-102574.

[21] ASPENCORE, *2019 Embedded Markets Study*, 2019, https://www.embedded.com/wp-content/uploads/2019/11/EETimes_Embedded_2019_Embedded_Markets_Study.pdf.

[22] Strace, linux syscall tracerhttps://strace.io.

[23] Linux man pagehttps://linux.die.net/man/8/crond.

[24] Common vulnerabilities and exposures, CVE-2016-5195https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2016-5195.

[25] A. Marzano, D. Alexander, O. Fonseca et al., "The evolution of Bashlite and Mirai IoT botnets," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 813–818, Natal, Brazil, June 2018.

[26] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016.

[27] A. Fabrizio, A. Luciano, and F. Angelo, "Exploiting N-gram location for intrusion detection," in *IEEE International Conference on Tools with Artificial Intelligence*, Vietri sul Mare, Italy, November 2015.

[28] B. Candice, C. Anna, and M. M. Gonzalo, *A Comparative Analysis of XGBoost*, 2019.

[29] J. A. Suvkens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[30] F. T. Liu, M. T. Kai, and Z. H. Zhou, "Isolation-based anomaly detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 6, no. 1, pp. 1–39, 2012.

[31] K. P. Bennett and E. J. Bredensteiner, "Duality and geometry in SVM classifiers," *ICML*, vol. 2000, pp. 57–64, 2000.