An Artificial Intelligence Method to Predict Malicious Behavior

Der-Chen Huang[1], Chun-Fang Hsiao[1], Bo-Kai Liu[1], Yu-Yi Chen[2]*

[1] Department of Computer Science and Engineering, National Chung Hsing University, Taiwan
[2] Department of Management Information Systems, National Chung-Hsing University, Taiwan
huangdc@nchu.edu.tw, {d108056003, g108056161}@mail.nchu.edu.tw, chenyuyi@nchu.edu.tw

Abstract

With the advancement of technology, more and more information equipment appear in people's lives. Up to date, with the improvement of network technology, the transmission of information between devices has become more convenient and faster, and the worries of information security follow. Although discussion of the information security of terminal equipment can be an issue, the information of terminal equipment will eventually be sent back to the server. Therefore, the research of the intrusion detection of servers is more fundamental. It is well known that the appearance of malicious behavior often means that the system may have been attacked by hackers. Thus, early detection of malicious behavior plays a vital role in preventing hackers from intrusion. However, most of the current known researches tend to focus only on how the system recognizes the malicious behavior when it is occurred, but the system cannot predict the occurrence in advance when the malicious behavior has not been completed. This research hopes to propose a method that can predict the appearance of malicious behavior before the malicious behavior is completed. We propose a method for predicting malicious behavior, which can determine whether the behavior is malicious before it is completed. The method of this research is to construct a malicious behavior prediction model by GAN (Generative Adversarial Network). It is based on the malicious behavior detection model established by the LSTM (Long Short-Term Memory) model. The experimental results show that the prediction accuracy of the model is about 83%.

Keywords: Host-Based Intrusion Detection System, Long Short-Term Memory, Generative Adversarial Network

1 Introduction

In recent years, with the advancement of technology, people's lives have become inseparable from the internet. The global reliance on the internet has intensified due to the impact of the pandemic. According to statistics from the International Telecommunication Union (ITU) [1], the number of individuals using the internet has rapidly increased from 4.1 billion in 2019 to 5.3 billion users in the past few years. As society's dependence on internet devices continues to grow, information security issues have emerged in tandem.

According to the Check Point Software 2023 Cyber Security Report [2], global cyber-attacks increased by 38% in 2022, with organizations facing an average of 1168 attacks per week. It is anticipated that the number of incidents will continue to rise in 2023.

In the past, most defense approaches involved collecting data and analyzing semantic vulnerabilities after an attack occurred [3], and then developing various anomaly detectors once the dataset was established. Even many methods have been proposed to improve detection accuracy [4] and to prevent potential future attacks, the damage has already been done. Moreover, with the gradual maturity of artificial intelligence technologies like ChatGPT, hackers will be able to automatically generate malicious code and spread it faster on the internet, even utilizing botnets for dissemination [5]. As a result, effective prevention measures will become more crucial than detection.

Traditional Host-Based Intrusion Detection Systems (HIDS) or Host-Based Intrusion Prevention Systems (HIPS) tend to focus on post-incident defense. The difference lies in the fact that traditional HIDS only alerts users to the occurrence of malicious behavior without actively preventing it, whereas HIPS not only alerts users but also takes further action to block the execution of malicious behavior. Research in accurately detecting whether a behavior is malicious is crucial, but this research goes a step further by exploring the accurate prediction of whether a behavior will develop into a malicious one in the future. This predictive model aims to discover and notify users before malicious behavior is completed. This research draws inspiration from literature [6], which made contributions to intrusion detection systems (IDS) regarding dataset imbalances. The referenced method utilizes the CIC-IDS2017 dataset, addressing the issue of imbalances in training data. However, the experiments conducted in [7] with the CIC-IDS2017 dataset often yielded suboptimal results due to data imbalance. The attacks in this dataset target the Linux operating system. To address cross-platform issues, [7] developed an Intrusion Detection System that works across both Linux and Windows platforms, specifically focusing on detecting zero-day attacks.

This research contributes by proposing a Host-Based Intrusion Prediction System (HIPRS). The system builds

*Corresponding Author: Yu-Yi Chen; E-mail: chenyuyi@nchu.edu.tw
DOI: https://doi.org/10.70003/160792642025032602007

upon host-based intrusion detection and introduces a predictive capability for anticipating malicious behavior. The architecture of this system is based on Generative Adversarial Networks (GAN) and employs Long Short-Term Memory (LSTM) models to establish a malicious behavior detection system. Through dynamic learning and training within the GAN framework, the system achieves the capability to accurately predict malicious behavior. Training and testing were conducted using the ADFA-LD dataset, resulting in an average prediction accuracy of 83%.

2 Related Work

2.1 Host-Based Intrusion Detection System (HIDS)

Traditional IDSs are often limited to monitoring and recording without the capability of active defense and response. IDSs are situated between the attacking end and the firewall, enabling monitoring and providing logs for review and post-incident tracking but lacking real-time defense capabilities.

The Host-Based Intrusion Detection System (HIDS) is also a subject of investigation in this research. It primarily examines and audits system log files for any malicious behavior. Its advantages lie in its ability to confirm whether a hacker has intruded and monitor the activities of specific host systems. However, HIDS has limitations as it can only view information received by the host that is deployed on, it is unable to access data from other hosts. Additionally, HIDS typically provides post-event notifications and actions, it is lacking real-time prevention of malicious behavior. It is commonly placed on large hosts within network switches to better monitor host performance and promptly detect potential intrusions.

2.2 Australian Defence Force Academy Linux Dataset

ADFA-LD stands for Australian Defence Force Academy Linux Dataset, which is a dataset collected from the Australian Defence Force Academy based on the Linux operating system. This dataset comprises both malicious and normal behavior data. It contains a total of 5950 records, with 746 records representing malicious behavior and 5204 records representing normal behavior [8]. The malicious behavior data is further categorized into six types of malicious activities: Password bruteforce FTP by Hydra, Password bruteforce SSH by Hydra, Add new superuser, Java Based Meterpreter, Linux Meterpreter Payload, and C100 Webshell.

The main purpose of designing ADFA-LD was the dissatisfaction of the authors with the existing KDD-99 dataset and UNM dataset [8-9], leading them to create a new dataset. The KDD-99 dataset had several issues that have been pointed out in various literature [10-15], with its age being the most commonly criticized aspect. KDD-99, short for Knowledge Discovery and Data Mining, was the dataset used in the 1999 KDD CUP competition, and it was pre-processed based on the DARPA 1998 DataSet. This dataset was compiled in 1998 and 1999, it is no longer adequate to address the current cybersecurity concerns in the modern network world. Other criticisms of the KDD-99 dataset include its lack of realism and unsuitability for training artificial intelligence models. On the other hand, the UNM dataset's drawback is its extremely limited scope.

2.3 Long Short-Term Memory (LSTM)

The architecture of Long Short-Term Memory (LSTM) model is illustrated in Figure 1. LSTM is a training model under the umbrella of Deep Learning, commonly used for training time-related sequential data. It addresses the issues of vanishing gradient and exploding gradient problems that occur during the training of Recurrent Neural Networks (RNN) with long sequences. LSTM is frequently applied in research related to natural language recognition and other artificial intelligence fields.

In ADFA-LD, individual actions may appear normal, but when various actions are combined to form behaviors, potential threats may arise. The use of LSTM is intended to detect malicious behaviors that are seemingly harmless when combined in specific patterns.

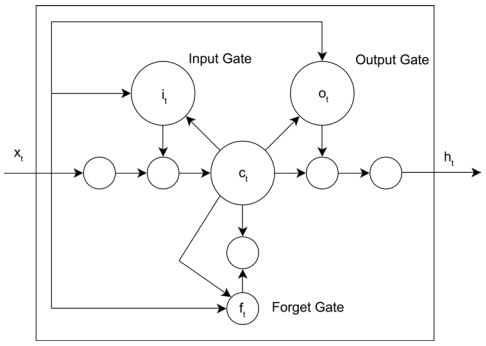

Figure 1. LSTM architecture

2.4 Generative Adversarial Network (GAN)

Generative Adversarial Network (GAN) is a type of unsupervised learning method that involves training two neural networks in a competitive manner. This approach was introduced by Ian Goodfellow and his colleagues in 2014. The two networks, namely the generator and the discriminator, compete against each other during the training process. The generator generates fake samples by sampling from noise, imitating real samples. On the other hand, the discriminator's task is to distinguish between fake samples generated by the generator and real samples. The training process aims to improve both models' accuracy by continuously attempting to deceive the discriminator and identify the generated fake samples accurately. Eventually, it results in a generator capable of creating realistic-looking fake samples and a discriminator with the ability to discern between real and generated samples. The typical architecture of a GAN is shown in Figure 2.

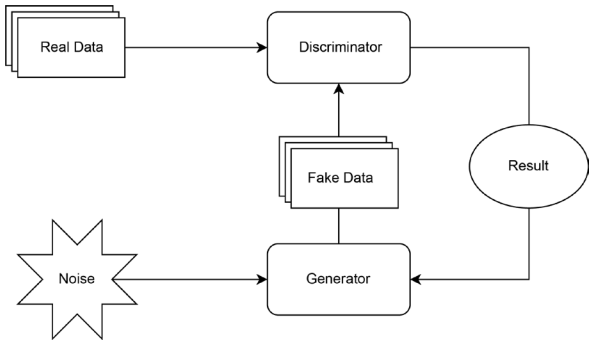

Figure 2. GAN architecture

3 Host-Based Intrusion Prediction System (HIPRS)

3.1 Method

This section provides a macro perspective of how the system proceeds from inputting one or multiple behaviors to predicting whether they are malicious or normal, as shown in Figure 3. The behaviors enter the HIPRS and are fed into the pre-trained prediction model. The prediction model will then generate a risk level prediction for the ongoing behavior, while the behavior is still in progress. The predicted risk result is compared to a predefined threshold value, and if the risk level is higher than the threshold, it will be classified as "high risk," prompting the user to terminate the behavior in advance. If the risk level is lower than the threshold, it will be classified as "medium risk" or "low risk," and the behavior will continue until it is detected as "high risk" or until the execution is completed.

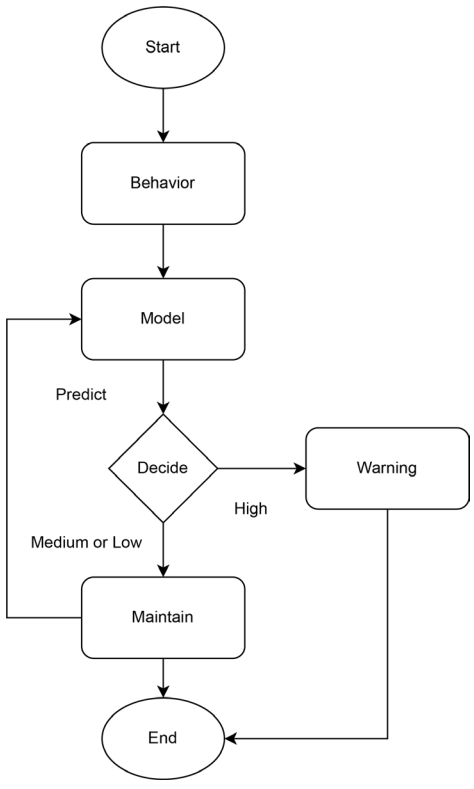

Figure 3. Method workflow

3.2 Structure of the System

In this section, we introduce the architecture of the malicious behavior prediction model. The conceptual design of this architecture is to first build a model that can accurately detect malicious behavior and then establish a model that can accurately predict malicious behavior based on it. We first use the Long Short-Term Memory (LSTM) method to train a malicious behavior detection model. Then, we incorporate this detection model into the Generative Adversarial Network (GANs) framework to establish a malicious behavior prediction model, as shown in Figure 4.

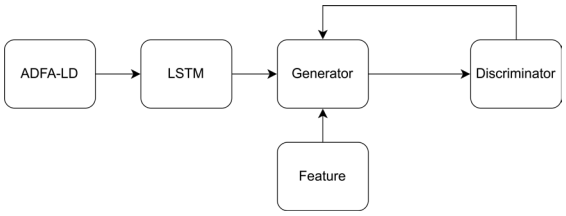

Figure 4. Architecture of the Host-Based intrusion prediction system

The architecture in Figure 4 is the core framework of this research. Data is obtained from ADFA-LD and fed into LSTM to build a malicious behavior detection model. Using this detection model, danger coefficient is continuously generated and passed into the generator model. The generation process will be detailed in the next section.

After the generator model obtains the danger coefficient generated by the LSTM model, it continuously generates prediction results using a proposed algorithm. These prediction results are then fed into the discriminator model. The discriminator model compares the prediction results from the generator model with the actual labels of ADFA-LD. If the prediction is incorrect, the weights and parameters in the model are adjusted and sent back to the generator. If the prediction is correct, further detection is performed to check if the prediction is earlier than the previous one. If it is not, the weights and parameters are changed and sent back to the generator. If it is earlier, the weights and parameters remain unchanged. It's essential to note that this step is specific to one behavior. When the next behavior enters, the process is repeated, and the changed weights and parameters will influence the subsequent predictions.

The inclusion of the "features" input in the generator part of Figure 4 serves a specific purpose in this research. It aims to provide an initial understanding of the behavior entering this model at the beginning of the model training. In this research, setting up this input is intended to help the model have a basic awareness of behaviors at the start of training, giving it a higher starting point for subsequent training. The model will undergo repetitive training, continuously improving the generator model's accuracy in predicting malicious behavior, and gradually converging the weight parameters of the discriminator model. When all behaviors in ADFA-LD have been processed, the training will be completed.

3.3 Train HIDS by LSTM

Before obtaining the malicious behavior prediction model, it is necessary to first obtain an accurate malicious behavior detection model. The difference between the two lies in that the malicious behavior detection model needs to run the entire behavior to conclude whether it is a malicious behavior, while the malicious behavior prediction model can predict whether future behavior will form malicious behavior without running the entire behavior. The purpose of setting LSTM is to find the hidden connections among seemingly unrelated and normal actions. LSTM helps us to find the relationships between actions that are not visible to the naked eye. Inputting a sequence of seemingly unrelated actions, the system determines whether this combination of actions is dangerous, and then combines these combinations to obtain a danger coefficient.

The process of training the malicious behavior detection model using LSTM is shown in Figure 5. The first step of the process is to read and split the dataset, where ADFA-LD is segmented. This research divides the dataset into a training set and a testing set, with 1409 and 4542 data instances, respectively. Among them, the training set contains 576 instances of malicious behavior and 833 instances of normal behavior, while the testing set contains 170 instances of malicious behavior and 4372 instances of normal behavior. It can be observed that the testing set and training set do not have a 2:8 ratio in total instances, which was intentionally designed. The characteristic of ADFA-LD is to have a small number of malicious behavior instances and a large number of normal behavior instances. If a 2:8 ratio is used to split the data into testing and training sets, the number of malicious behavior instances in the training set would be too small, significantly affecting the model's quality. To ensure the accuracy of the trained model, it is essential to increase the number of malicious behavior instances in the training set. Therefore, the ratio of malicious behavior instances in the testing set to the training set was chosen to be close to 2:8, which was also confirmed by the subsequent results.

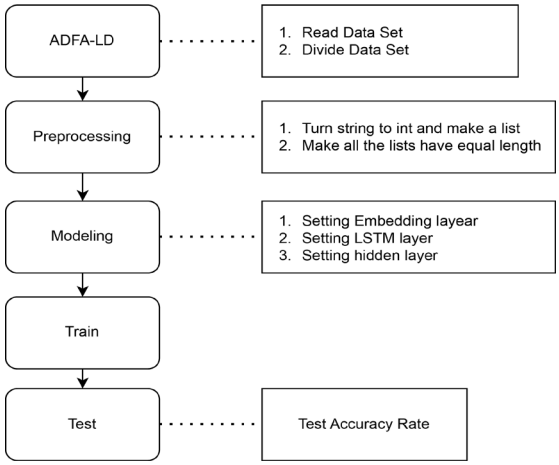

Figure 5. The workflow of train HIDS

The third step in Figure 5 is model building, which involves using Multilayer Perceptron (MLP) to assist in constructing the training model, and the specific parameter settings are shown in Figure 6. An Embedding layer is added with 500 neurons in this step, and the output has 32 neurons. To prevent overfitting, a dropout mechanism is specially introduced in this layer, randomly dropping 20% of neurons in the neural network during each training iteration. Next, a LSTM layer with 32 neurons is added, connecting to the output of the previous layer. A hidden layer with 256 neurons is added, and its activation function is defined as Relu. Similarly, to prevent overfitting, a dropout mechanism is set in this hidden layer, randomly dropping 20% of neurons during each training iteration. Finally, an output layer with only one neuron is added, and its activation function is defined as Sigmoid.

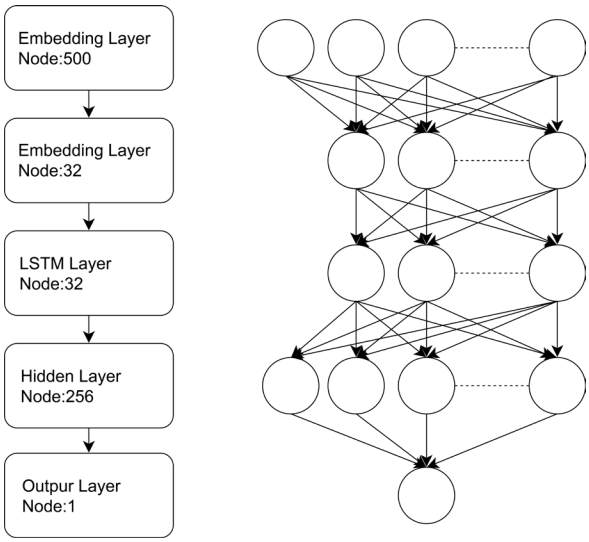

Figure 6. MLP parameters setting

After configuring the MLP parameters, the data can be fed into the model for training. Figure 7 shows the training progress at 10 epochs. As seen in the right plot of Figure 7, the loss steadily decreases, while the left plot of Figure 7 shows a steady increase in accuracy.

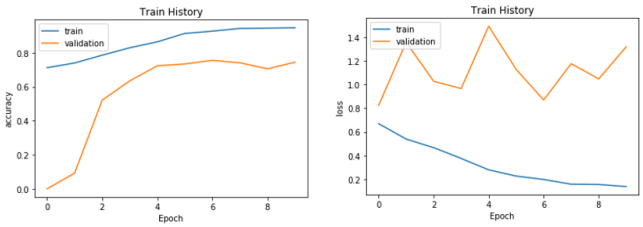

Figure 7. Trend chart of loss rate and accuracy rate with epoch

In summary, we have completed the training of Malicious behavior prediction model using LSTM. Next, we will proceed to the establishment of the generator model, which will continue from the detection model and create a predictive model.

3.4 Generator Model

The generator model is the primary contribution of this

research. It follows the previous LSTM model training to predict whether a behavior is malicious based on the danger coefficient output by the detection model.

The reason for incorporating the training of danger coefficient on top of the existing LSTM instead of solely using LSTM's predictive ability is twofold. Firstly, considering the usage of danger coefficient in the prediction model involves incorporating reference to feature values, while the discriminator model using LSTM does not have this reference. Secondly, the aim is to utilize the unique mechanism of mutual reinforcement learning between the generator and discriminator models. By continuously generating predictions through the generator and making judgments and feedback through the discriminator, the accuracy of the generator model's predictions can be improved, which cannot be achieved with LSTM's architecture alone.

In this section, three variables alpha (α), beta (β), and gamma (γ) will be introduced, known as weight parameters. They adjust the mathematical formulation of the generator model under the GAN framework to improve prediction accuracy. These weight parameters will be adjusted in the discriminator and then fed back to the generator.

Figure 8 depicts Generator algorithm. This algorithm encompasses the entire process of the generator model generating predictions, with external algorithm functions called in lines 15, 16, 18, and 19. Initially, the model will be given alpha (α), beta (β), and gamma (γ) as weight parameters. At this point, since the model starts with the first behavior, α, β, and γ have not undergone adjustments by the discriminator model.

Generator Algorithm

behavior: behavior in ADFA-LD
lengths: the length of behavior
n: number of windows in one behavior
α, β, γ: the weight parameter of Discriminator
i: the numbering of acts
j: the numbering of windows
k: the numbering of behaviors
act_i*: the number i acts*
$window_j$*: the number j windows*
$behavior_k$*: the number k behavior*
Window_size: the size of window (the number of acts in one window)
S_j*: the dangerous degree from j = 1to j*
D_j*: the Risk factor at j*
N_j*: the normalization of* S_j
LSTM, Past, Feature, Normalize, Decide: function

```
1.  Def Generator (α, β, γ):
2.      if lengths <= 100 then
3.          Window_size = 50
4.      else if lengths > 100 && lengths <= 500 then
5.          Window_size = 100
6.      else then
7.          Window_size = 500
8.      D_j ← LSTM(behavior_k)
9.      Def Recursive (S_{j++}):
10.         if j == 1 then
11.             S_j ← D_j
12.         else then
13.             S_j += D_j
14.             S_j += α * ((S_{j-1}) * (j - 1)) / n
15.             S_j += β * ((D_j - D_Past(j)) * (j - Past(j))) / n
16.             S_j += γ * Feature (window_j)
17.         return S_j
18.     N_j = Normalize (S_j)
19.     return Decide (N_j)
```

Figure 8. Generator algorithm

Next, based on the length of the behavior, the window size is set. The purpose of setting the window is that a behavior can be lengthy and needs to be divided into smaller parts. By looking at the danger coefficient of a specific time point within a window and continuously shifting the window along the behavior, we can observe the danger coefficient within the next time point's window. This continuous movement of the window allows us to observe the variations in the danger coefficient. By studying the patterns of these variations, we can predict whether a behavior is malicious before it is fully executed.

The design of the window size depends on the size of a behavior and is divided into three categories: windows with 50 actions, 100 actions and 500 actions. This design is based on the statistical findings of ADFA-LD in this research. There are 320 behaviors with action counts ranging from 1 to 100, 4248 behaviors with action counts ranging from 100 to 500, and 1382 behaviors with action counts larger than 500. The majority of behaviors have action counts between 100 and 500, so a window size of 100 is designed for this range. The other two ranges, with smaller numbers of actions, have window sizes of 50 and 500.

Once the window sizes are set, the Generator algorithm incorporates the danger coefficients obtained by the LSTM model from the previous section, as shown in formula (1). As the malicious behavior detection model has been successfully established in the previous section, the Generator algorithm uses the output of the LSTM model at this stage.

$$D_j \leftarrow LSTM(behavior_k) \tag{1}$$

In formula (1), it represents the danger coefficient at window number j, which refers to the position of the window as it moves along the behavior, also known as the current execution time, is the k-th behavior that has been executed.

Next, we enter a recursive function called Recursive. In this function, a variable is passed as an argument in the recursive calls and is passed again with each recursive generation, as shown in lines 9 to 17 in Figure 8.

The variable represents the cumulative risk level at time j, which calculates the sum of past, present, and future risk levels. The logic behind this variable is that to predict future malicious behavior, we only need to look back at past risk level data. By incorporating past risk levels, the present risk level at time j, and predictions of future risk level trends, along with adjustments and optimizations based on features, the model can calculate the current risk level, as shown in formulas (2), (3), (4), and (5). By comparing the calculated value with a threshold, we can predict whether future behavior will evolve into malicious behavior. Therefore, the overall implication is that the prediction of whether a behavior will evolve into malicious behavior in the future is determined by the sum of now risk level, past risk level, future risk level and behavior feature.

$$S_j += D_j \tag{2}$$

The first step of the recursive function "Recursive" is to detect whether the entered behavior is the first one, as shown in line 10 of Figure 8. If it is the first behavior, the value is directly assigned because there are no previous "past risk level" and "future risk level" at this point, and hence we cannot discuss formulas (3), (4), and (5). If it is not the first behavior, the value is updated by adding the current risk level's impact on the overall risk level, as shown in formula (2).

Next, we add the influence of "past risk level", as detailed in formula (3). It is important to note that in formula (3), the past risk level is calculated recursively by calling the risk level value at the previous time point, i.e., *j*-1, and so on. At the same time, this value is multiplied by the proportion of the current execution time to the total behavior length and then further multiplied by the weight parameter α, which determines the importance of this value in the overall risk level.

$$S_j += \alpha * \frac{(S_{j-1}) * (j-1)}{n} \tag{3}$$

Then, we incorporate the influence of future on the mathematical model, as shown in formula (4). The weight parameter β determines the importance of considering future impact on the current risk level. Future refers to whether the latest obtained danger coefficient value in the line chart is trending downward or upward. This research believes that the latest trend represents the potential direction of future danger coefficient development. Hence, increasing the weight parameter β can help the mathematical model predict the upcoming malicious behavior at an earlier stage.

$$S_j += \beta * \frac{(D_j - D_{past(j)}) * (j - Past(j))}{n} \tag{4}$$

However, increasing the weight parameter β is not always better, as excessively raising this weight may lead to classifying slightly unusual actions in the behavior as malicious behavior, which should be avoided.

Formula (4) calculates the trend by first searching for the nearest local maximum or local minimum of the current danger coefficient in the danger coefficient curve. Local maximum or minimum values refer to the extremum values within a limited range, rather than across the entire range. The function *Past*(*j*) calls the Past algorithm, which finds the value and position of the local maximum or minimum.

After finding the local maximum and minimum values in the region, the algorithm then calculates the difference between the current danger coefficient value and the found local maximum or minimum, as shown in formula (4) with D_j- $D_{Past(j)}$ and *j*- *Past*(*j*) respectively. The result is then multiplied by β, the weight parameter.

In formula (4), the total number of windows, *n*, is introduced to calculate the proportion of the difference between the position *Past*(*j*) of the extremum and the current position *j* in the entire window count. This proportion is then multiplied by the difference between the extremum value ($D_{Past(j)}$) and the current danger coefficient value (D_j). This approach is similar to formula (3) and is intended to limit the influence of attention to trends within their respective regions. The rationale behind this is that the impact of a trend should be constrained by its duration, rather than affecting the overall risk level of the behavior.

After completing the previous steps, the next step is to incorporate feature values into the model. During the training phase, the research often encounters situations where the model's prediction accuracy curve experiences significant fluctuations in the early stages of training. Upon investigation, it was found that the model lacked a basic understanding of actions within behaviors, leading to excessively high or low prediction accuracy values. Therefore, this research aims to be vigilant towards actions that frequently appear in malicious behaviors from the beginning of training, distinguishing their risk levels from other actions and avoiding training with similar risk levels. The purpose of formula (5) is to help the model incorporate such features into the training process.

$$S_j += \gamma * Feature(window_j) \tag{5}$$

In formula (5), an external function called Feature algorithm is invoked. The Feature algorithm conducts a statistical analysis of all actions occurring within malicious behaviors based on their frequency of occurrence. When a window enters this algorithm, it uses a table-like approach to retrieve the number of occurrences for each action within the window from the statistical records. Then, it calculates the ratio of occurrences for each action by dividing the retrieved values by the total number of actions in malicious behaviors. By summing up these ratios for all actions within the window, it derives a danger coefficient for the window.

The calculated danger coefficient is returned and then multiplied by the weight parameter γ before being added back to the overall risk level. This process enables the model to incorporate the risk contribution of actions frequently appearing in malicious behaviors during the training phase and improves the model's ability to distinguish between different risk levels among actions.

Finally, the Recursive function either proceeds with the next recursive calculation or concludes the computation by returning a value. If the Recursive function has finished its iterations, the Generator algorithm executes line 18, normalizing the value to obtain the normalized result as indicated in formula (6).

$$N_j = \text{Normalize}(S_j) \tag{6}$$

Normalization refers to scaling all values to the range of floating-point numbers between 0 and 1. This makes it easier to set a unified threshold for the risk level because the risk level may vary based on the size and length of each behavior. To have a consistent comparison, a unified threshold is necessary.

Table 1. Risk level threshold

Level	Low	Medium	High
Normalize danger	0-0.33	0.33-0.66	0.66-1

The Decide algorithm determines whether the normalized risk level obtained from the previous normalize step corresponds to malicious behavior. Since the risk level has been normalized, detection is based on the thresholds specified in Table 1. If the normalized risk level is greater than 0.66, it is detected as a high-risk behavior. If the risk level falls between 0.33 and 0.66, it is classified as a moderate threat. If the risk level is less than 0.33, it is identified as a low threat.

In the Decide algorithm, the prediction of whether a behavior will become malicious is based on whether the normalized risk level exceeds the threshold of 0.66. If the normalized risk level is higher than 0.66, it will be classified as malicious behavior. However, if the risk level is below 0.33, it will be considered normal behavior, and execution will continue.

It is important to note that if a behavior initially has a risk level higher than 0.66 but later decreases below the threshold, it will still be classified as normal behavior and continue to execute. On the other hand, if a behavior starts with a risk level below 0.66 but later increases and crosses the threshold of 0.66, it will be considered malicious behavior, and the execution will be terminated at the first occurrence of such situation.

Since no one can predict how a future behavior will unfold, waiting until the end of the behavior to determine whether it crossed the threshold of 0.66 would result in the loss of predictive capability. True prediction involves making judgments before the behavior is completed, not after it has already occurred. Waiting until the end of the behavior and then determining whether it was malicious or not does not qualify as prediction, as there is no anticipation involved.

For accurate predictions, the model needs to assess the risk and classify the behavior as malicious or normal during its execution, based on the evolving risk levels and predefined thresholds, rather than relying solely on the final outcome. This way, the model can provide timely predictions before the behavior is fully executed.

3.5 Discriminator Model

This section introduces the discriminator model, which is designed to determine whether the predictions generated by the generator model are correct and provide feedback to adjust the weight parameters. Additionally, to enhance the accuracy of the generator in predicting malicious behavior, the discriminator model offers corrections when the generator's predictions are inaccurate or not perfect enough.

The internal construction logic of the discriminator model is illustrated in the tree structure diagram shown in Figure 9. The top-level node, labeled Predict, represents the predictions passed in from the generator model. The second-level nodes detect whether these predictions are correct. If they are correct, the process proceeds to the third-level nodes. If the predictions are incorrect, the weight parameters are updated. In the second-level nodes, the weight parameters are adjusted by increasing the β parameter and decreasing the α. The reason for this adjustment is that prediction errors can occur due to two possibilities: predicting malicious behavior when it is actually normal behavior, or predicting normal behavior when it is actually malicious behavior. Both cases result from errors in detecting future trends. To improve prediction accuracy logically, the model needs to prioritize future trend predictions more while reducing emphasis on past historical behavior.

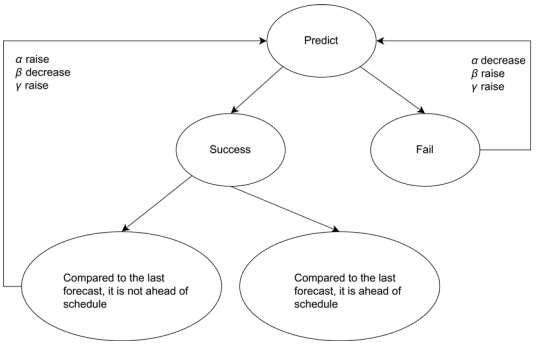

Figure 9. The structure of discriminator

Discriminator **Algorithm**

behavior: behavior in ADFA-LD
lengths: the length of behavior
n: number of windows in one behavior
α, β, γ: the weight parameter of Discriminator
j: the numbering of windows
k: the numbering of behaviors
$behavior_k$: the number k behavior
N_j: the normalization of S_j
LSTM, Past, Regularization, Normalize, Decide: function

1. **Def** *Discriminator* (*Decide* (N_j)):
2. **if** *Decide* (N_j) == *abnormal* **then**
3. decrease α
4. raise β, γ
5. **return** α, β, γ
6. **else then**
7. **if** $\frac{lengths_{behavior_k} - j_{behavior_k}}{lengths_{behavior_k}} < \frac{lengths_{behavior_{k-1}} - j_{behavior_{k-1}}}{lengths_{behavior_{k-1}}}$ **then**
8. raise α, γ
9. decrease β
10. **return** α, β, γ
11. **else then**
12. **return** α, β, γ

Figure 10. Discriminator algorithm

The Discriminator algorithm is shown in the Figure 10. In lines 3 to 5 and 8 to 10, it explains how to increase or decrease the α, β, and γ values. Additionally, in line 7, it elaborates on the method of comparing the prediction lead time of the current prediction with the previous one. This comparison is not based on absolute time duration but on the relative proportion of the prediction lead time to the total behavior duration. Since not all behaviors have the same duration, directly comparing the prediction lead time of different behaviors would be unfair.

After completing all the aforementioned processes, the Discriminator algorithm returns three weight parameters, α, β, and γ, to the Generator algorithm for the next round of training.

4 Experimental Results

4.1 Training Process

The malicious behavior prediction model is trained using the ADFA-LD dataset. The training results will be examined upon completing the last entry of the dataset. The training process is demonstrated in a specific sequence. Let's start with the typical process diagram where malicious behavior is predicted as malicious, as shown in Figure 11.

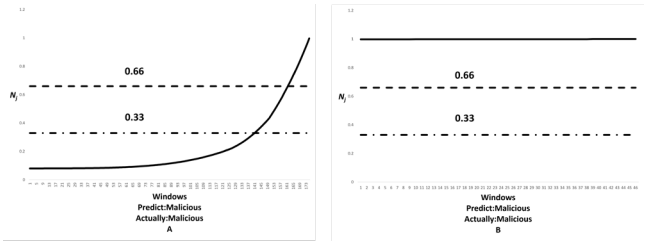

Figure 11. A typical process diagram for malicious behavior prediction

Figure 11 depicts typical process diagrams encountered during the training process where malicious behavior is predicted as malicious. In the left diagram of Figure 11, the risk level continuously and rapidly increases as the window *j* progresses. On the other hand, the right diagram of Figure 11 shows a scenario where the risk level starts very high, with values close to 1, from the beginning. In the left diagram, the behavior's value exceeds the specified threshold of 0.66, leading to its prediction as malicious behavior. In the right diagram, the behavior's value consistently remains above the threshold of 0.66, resulting in it being predicted as malicious behavior as well.

Figure 12 represents a typical process diagram where normal behavior is predicted as normal. Both behaviors start with a value of 1 but rapidly decrease over time. Afterward, their values remain constant, staying close to 0. Since they touch the threshold of 0.66 downwards, both behaviors are correctly predicted as normal behavior.

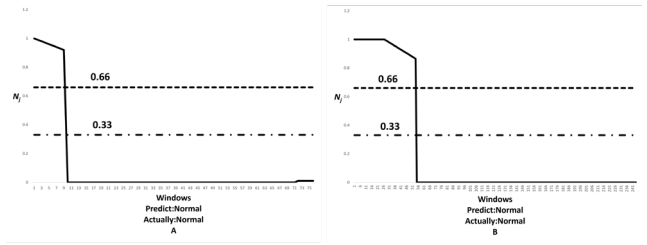

Figure 12. A typical process diagram for normal behavior prediction

Figure 13 is an atypical training process diagram where the prediction is correct, but the curve shape is relatively uncommon. The value continuously decreases with the increase of *j* until it falls below the threshold of 0.66. Therefore, the model detects it as normal behavior, which is confirmed to be correct. However, when the value of *j* is approximately 15, the value rises again to around 0.5, indicating that the model detects the behavior executing some risky actions. Nonetheless, the increase in value is not significant enough to cross the threshold of 0.66 again.

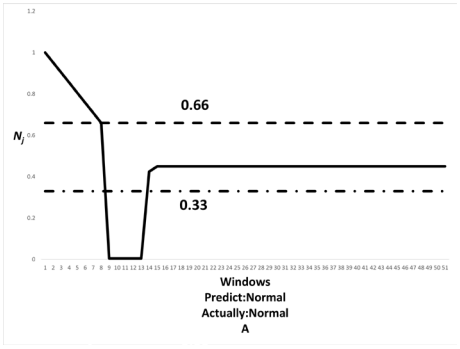

Figure 13. A typical training process diagram

Figure 14 shows a process diagram where behavior prediction fails. The values exhibit a relatively large and complex fluctuation. At time point A in Figure 14, the value crosses the threshold of 0.66, causing the model to detect it as normal behavior. However, at time point B, the value suddenly rises again, crossing the threshold of 0.66, and ultimately, this behavior is detected as normal behavior. The model captures the first instance of crossing the threshold of 0.66 but cannot anticipate the uncertain future event of crossing again. However, this behavior actually belongs to malicious behavior, as demonstrated after time point B. Therefore, this prediction is incorrect. In summary, the training process diagram was presented. Next, we introduce the overall prediction accuracy when the training is completed.

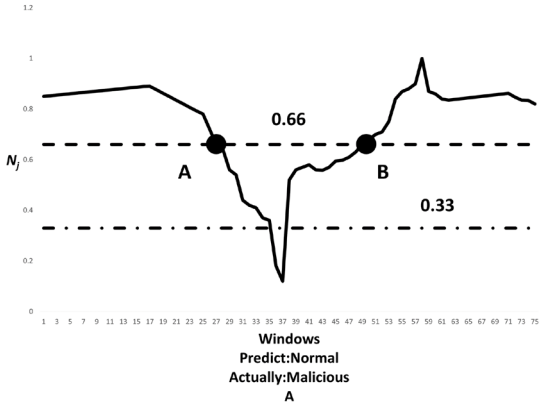

Figure 14. Process diagram of behavior prediction failure

It can be observed that the average accuracy initially undergoes significant fluctuations in Figure 15. The reason is that the model's training data is not yet sufficient. When there is a prediction error, the average accuracy drops significantly, and when there is a correct prediction, the average accuracy increases substantially. Therefore, when calculating the average accuracy, we only consider the accuracy values when there is no significant change, which means discarding the accuracy values during the early training stages, as shown in Figure 16. At this point, the

average accuracy is 83.64%. It can be observed that the model's average accuracy has converged and stabilizes around 83%, with a slight upward trend.

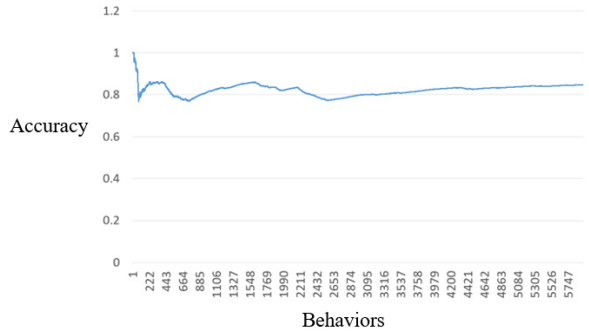

Figure 15. Average prediction accuracy

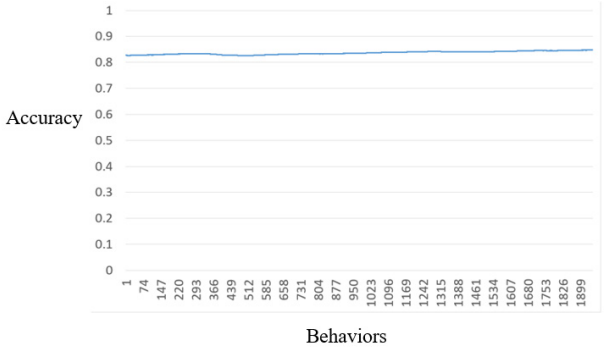

Figure 16. Discard the average accuracy map that has not yet converged

4.2 Performance Discussion

This section is to evaluate and test the performance of our research method. The metrics involves using the Accuracy, Precision, Recall, and F1 score indicators for evaluation. Accuracy refers to the proportion of correctly detected data in the dataset, including correctly predicting normal behavior as normal and malicious behavior as malicious. Precision measures the proportion of truly malicious data among those detected as malicious. Recall, on the other hand, is the complement of Precision. It describes the proportion of truly detected malicious data among all the malicious data in the dataset. Finally, the F1 score is a harmonic mean of Precision and Recall. Since both Precision and Recall are desirable to be high, a higher F1 score indicates better overall performance of the model in terms of Precision and Recall.

We calculated the impact of each formula on the overall model performance. First, we evaluated the contribution of formula (3) to the model's performance, as shown in Table 2. Similarly, effect of adding formulas (4) and (5) to the model's performance are shown in this table. We can observe that the Table 2 represents the incremental addition of formula (3) to formula (5), and the model's performance gradually improves. Notably, adding formula (4) significantly improves the performance compared to only having formula (3) in the model. This improvement can be attributed to the model incorporating formula (4) for predicting future trend. As the primary objective of this model is to predict malicious behavior, formula (4) plays a crucial role in aiding the model's predictions of future trend.

However, when comparing these results, the overall improvement in model performance by adding formula (5) is not substantial. This is because formula (5) mainly serves to provide guidance to the model in the initial phase for predicting the direction of behavior, whether it is more inclined towards malicious or normal behavior. From the perspective of the entire dataset with all behaviors, the impact of this formula on overall predictions is minor. Therefore, when assessing the influence of formulas on the model's prediction performance, there is no significant increase in the values of the four performance metrics. Finally, when considering all the formulas, the overall performance values of the model for the four metrics are: Accuracy (0.836), Precision (0.963), Recall (0.860), and F1 (0.909).

Table 2. Add formulas (3), (4) and (5) performance test

Metrics / Formulas	Accuracy	Precision	Recall	F1 score
Formulas (2) and (3)	0.785	0.966	0.781	0.864
Formulas (2), (3) and (4)	0.836	0.963	0.860	0.909
Formulas (2), (3), (4) and (5)	0.836	0.963	0.860	0.909

4.3 Literature Comparison and Analysis

Although a series of performance metrics have been obtained, it is through comparison with other methods that the effectiveness of this research can be demonstrated. In this section, several literatures similar to this research were compared, which may involve similar methodologies or the use of the same dataset to enhance comparability among methods. However, these proposed methods don't touch the prediction issue. Thus, the following discussion are based on our method prediction results with comparing the detection results from other methods.

[16] proposed a method using Cycle-GAN to augment the dataset, and training an intrusion detection system with the augmented dataset can increase the accuracy of detecting malicious behavior. The similarity between [16] and the method proposed in this research lies in the use of GAN and the utilization of the ADFA-LD dataset. However, the difference is that [16] employed Cycle-GAN for training and used data augmentation to enhance detection accuracy, unlike this research that utilizes GAN architecture for training. Additionally, the main contribution of [16] is to propose a method to augment the dataset size without improving the existing host-based malicious intrusion detection system. Therefore, the accuracy of detecting malicious behavior in [16]

is relatively low, at only 64%, which is lower than the accuracy achieved in this research, which is 83.64%.

In [17], an improvement approach for host-based intrusion detection systems is proposed. The method introduced in [17] utilizes semantic analysis to identify malicious behavior. It analyzes the semantics of malicious and normal behaviors to find their distinctive features, which are then stored in a database. When a new behavior enters the system, it uses the established semantic feature database to perform a table lookup and determine whether the behavior is closer to malicious or normal behavior.

In [18], a method utilizing both Generative Adversarial Network (GAN) and Variational Auto Encoder (VAE) to augment the dataset is proposed to enhance the accuracy of detecting malicious behavior. This approach is similar to the method in [16], as both use dataset augmentation to improve the accuracy of identifying malicious behavior. However, the difference lies in the fact that while [16] employs Cycle-GAN, [18] uses both GAN and VAE. After training the models using GAN and VAE, [18] compares the performance of both methods.

Table 3 compares the performance of the GAN method in [18] with the performance of this research. It can be observed that this research outperforms [18] in terms of Precision, indicating that our proposed method is less likely to predict normal behavior as malicious behavior. This means that our method can effectively protect the host system while maintaining system performance. The F1 score is similar for both methods, but this research is relatively weaker in terms of accuracy. The reason for this is that our research is a malicious behavior prediction system, and predicting behaviors is inherently more challenging, which may lead to lower accuracy.

In [19], a host-based intrusion detection and defense system for Android devices was proposed. This system utilizes the k-means method to classify behaviors into malicious and normal categories. Table 3 shows the comparison between the performance of our proposed method and the method in [19].

It can be observed that our research performs lower in terms of Accuracy compared to [19], which is consistent with the comparison with [18] as previously described. However, our research outperforms [19] in terms of Recall, indicating better ability to recognize malicious behaviors compared to [19].

When comparing the performance of our proposed method with other researches and summarizing the results in Table 3, it is evident that our research achieves the best Precision and F1 score. This demonstrates that our proposed method outperforms other compared studies in overall performance, with F1 score confirming this observation. However, it's worth noting that the other researches in the table also highlights specific advantages exhibited by other studies, such as higher Recall value in [18], indicating its advantage in accurately identifying malicious behaviors, which is a crucial indicator for host-based intrusion detection systems.

Table 3. Comparison between this research and other literatures

Metrics / Research	Accuracy	Precision	Recall	F1 score
Our research	0.8364	0.9630	0.8607	0.9090
Research [16]	0.64	0.2809	0.8049	0.4164
Research [18]	0.9	0.9	0.9	0.9
Research [19]	0.9087	0.9765	0.8367	0.8938

Nevertheless, these compared literature do not possess the ability to predict malicious behavior. Therefore, even though they might have relative advantages in some performance metrics, they still cannot compare with the contribution of our research in early prediction.

5 Conclusions

This research proposes an effective system for predicting malicious behavior attacks. The system focuses on host-based intrusion detection and utilizes ADFA-LD for training and testing. The primary methodology employed in this research includes LSTM and GAN. LSTM focuses on establishing an accurate model for host-based intrusion detection, and based on this model, the system is developed using the architecture of GAN to predict the occurrence of malicious behavior. Experimental results demonstrate that the system achieves an impressive average prediction accuracy of 83%.

References

[1] International Telecommunication Union, *ITU-D ICT Statistics*, ITU Statistics, https://www.itu.int/en/ITU-D/Statistics/Pages/stat /default.aspx.

[2] Check Point Research Team, *2023 Cyber Security Report*, February, 2023.

[3] S.-T. Ha, S.-S. Hong, M.-M. Han, Malware Detection Using Semantic Features and Improved Chi-square, *Journal of Internet Technology*, Vol. 19, No. 3, pp. 879-887, May, 2018.

[4] L. Zheng, J. Li, H. Wang, X. Zeng, Improving Accuracy and Automation of Anomaly Detectors Based on Self-Correlation, *Journal of Internet Technology*, Vol. 17, No. 1, pp. 39-51, January, 2016.

[5] Z. Wang, M. Tian, X. Zhang, J. Wang, Z. Liu, C. Jia, I. You, A hybrid learning system to mitigate botnet concept drift attacks, *Journal of Internet Technology*, Vol. 18, No. 6, pp. 1419-1428, November, 2017.

[6] I. Sharafaldin, A.-H. Lashkari, A.-A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization, *Proceedings of the 4th International Conference on Information Systems Security and Privacy ICISSp*, Funchal Madeira, Portugal, 2018, pp. 108-116.

[7] G. Creech, *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*, Ph. D. Thesis, The University of New South Wales, Sydney, Australia, 2014.

[8] G. Creech, J. Hu, Generation of a new IDS test dataset: Time to retire the KDD collection, *2013 IEEE wireless communications and networking conference (WCNC)*, Shanghai, P.R. China, 2013, pp. 4487-4492.

[9] G. Creech, J. Hu, A semantic approach to host-based intrusion detection systems using contiguousand discontiguous system call patterns, *IEEE Transactions on Computers*, Vol. 64, No. 4, pp. 807-819, April, 2014.
[10] C. Brown, A. Cowperthwaite, A. Hijazi, A. Somayaji, Analysis of the 1999 darpa/lincoln laboratory ids evaluation data with netadhict, *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Ottawa, Canada, 2009, pp. 1-7.
[11] P. Owezarski, A database of anomalous traffic for assessing profile based IDS, *Traffic Monitoring and Analysis: Second International Workshop, TMA 2010, Zurich, Switzerland, April 7, 2010. Proceedings 2*, Switzerland, Zurich, 2010, pp. 59-72.
[12] V. Engen, J. Vincent, K. Phalp, Exploring discrepancies in findings obtained with the KDD Cup'99 data set, *Intelligent Data Analysis*, Vol. 15, No. 2, pp. 251-276, April, 2011.
[13] S. Petrovic, G. Alvarez, A. Orfila, J. Carbo, Labelling clusters in an intrusion detection system using a combination of clustering evaluation techniques, *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)*, Kauai, Hawaii, USA, 2006, pp. 129b-129b.
[14] M.-V. Mahoney, P.-K. Chan, An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection, *International Workshop on Recent Advances in Intrusion Detection*, Pittsburgh, PA, USA, 2003, pp. 220-237.
[15] J. McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, *ACM Transactions on Information and System Security (TISSEC)*, Vol. 3, No. 4, pp. 262-294, November, 2000.
[16] M. Salem, S. Taheri, J.-S. Yuan, Anomaly generation using generative adversarial networks in host-based intrusion detection, *2018 9th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, New York, USA, 2018, pp. 683-687.
[17] M. Anandapriya, B. Lakshmanan, Anomaly based host intrusion detection system using semantic based system call patterns, *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, India, 2015, pp. 1-4.
[18] Y. Lu, J. Li, Generative adversarial network for improving deep learning based malware classification, *2019 Winter Simulation Conference (WSC)*, National Harbor, MD, USA, 2019, pp. 584-593.
[19] J. Ribeiro, F.-B. Saghezchi, G. Mantas, J. Rodriguez, R.-A. Abd-Alhameed, Hidroid: prototyping a behavioral host-based intrusion detection and prevention system for android, *IEEE Access*, Vol. 8, pp. 23154-23168, January, 2020.

Biographies

Der-Chen Huang received the BS degree in electronic engineering from Fung Chia University, Taiwan, in 1983, the MS degree in computer engineering from Florida Institute of Technology, U.S.A., in 1991, and the PhD degree in computer engineering from the Department of Computer Science and Information Engineering, Chung- Cheng University, Chiayi, Taiwan, R.O.C. in 2000. From 1983 to 1989, he worked as a design engineer with the Computer Communication Lab. (CCL)/Industrial Technology Research Institute (ITRI) and Chung-Shan Institute and Science of Technology (CSIST) when he was assigned to a partnership project at General Dynamics, Fort Worth, Texas, U.S.A. He was an associate professor with the Department of Electronic Engineering, National Chinyi Institute of Technology, Taichung, Taiwan, R.O.C. from 1991 to 2004. In 2004, he joined the Department of Computer Science and Engineering, National Chung Hsing University, Taichung, Taiwan, R.O.C. He was a director of Computer and Information Center of Chung Hsing University from 2007 to 2011. Currently, he is a professor of Chung Hsing University. Dr. Huang served as a reviewer for various technical journal and conferences and a member of editorial board of Journal of Internet Technology. He received the Best Paper Award from the 5th International Conference on Future Information Technology, Korea, in 2010. His research interests include VLSI design for testability and diagnosis, Artificial Intelligence, Communication and Medical Image.

Chun-Fang Hsiao received the M.S. degree in Computer Science and Engineering from National Chung Hsing University, Taichung, Taiwan, in 2018. He is currently pursuing the Ph.D. degree in Computer Science and Engineering at National Chung Hsing University, Taichung, Taiwan. Mr. Hsiao currently works as an engineer at the Computer Network Center of National Chung Hsing University, Taichung, Taiwan. His research interests include Artificial Intelligence, network security, and embedded system.

Bo-Kai Liu received B.S. degree in Department of Computer Science from National Taichung University of Education, Taichung, Taiwan, and M.S. degree in Computer Science and Engineering from National Chung Hsing University, Taichung, Taiwan, in 2021. Mr. Liu currently work as an IT manager at Taiwan Judicial Yuan. His research interests include network security, machine learning and Artificial Intelligence.

Yu-Yi Chen was born in Kaohsiung, Taiwan, in 1969. He received the B.S., M.S., and Ph.D. in Applied Mathematics from the National Chung Hsing University in 1991, 1993, and 1998, respectively. He is presently a professor of the Department of Management Information Systems, National Chung Hsing University. From 2015 to 2023, he was the Director of the Computer and Information Network Center, National Chung Hsing University. He is the principal investigator of the project "Information Security Certification Body"

supported from the Ministry of Education, Taiwan. His research interests include computer cryptography, network security, and e-commerce.