Bulletin of Networking, Computing, Systems, and Software – www.bncss.org, ISSN 2186-5140
Volume 14, Number 1, pages 85–93, January 2025

An Implementation of Lightweight AI-Based Network Intrusion Detection Systems

Ryuma Taira and Chikatoshi Yamada
Dept. of Inf. dnd Comm. Syst. Eng.
National Inst. of Tech. Okinawa College
905 Henoko, Nago, Okinawa, Japan
{ic201227@edu., cyamada@}okinawa-ct.ac.jp

Shuichi Ichikawa
Dept. Electrical and Electronic information Engineering
Toyohashi University of Technology
1-1 Hibarigaoka, Tempaku-cho, Toyohashi 441-8580, Japan
ichikawa@tut.jp

***Abstract*—Internet of Things (IoT) devices are often equipped with only the minimum necessary functions due to limited resources such as memory, which has led to security vulnerabilities. This paper examines efficient artificial intelligence (AI)-based implementations of network intrusion detection systems (NIDS) to solve this problem and presents prospects for future research.**

Keywords—Internet of Things, Edge AI, NIDS, Security

I. Introduction

In recent years, the rapid development of IoT technology has led to an explosion in the use of IoT devices in the home and industrial sectors. These devices are used in a wide variety of applications such as communication, monitoring, and control, and are deeply embedded in our daily lives and business processes. However, IoT devices are typically low-cost and small in size, which means that they have limited resources (especially memory and processing power) and often lack sufficient security features. As a result, these devices are easy targets for cyberattacks, especially network intrusion threats.

Conventional software-based intrusion detection systems are often designed for devices with powerful processing power and sufficient memory, which may be difficult to apply to resource-constrained IoT devices. Therefore, approaches to enhance security with low resources are needed.

The objective of this study is to investigate an efficient AI based implementation of a network intrusion detection systems (NIDS) and to provide a means to improve the security of IoT devices. Specifically, we aim to improve the performance of the network intrusion detection AI called Kitsune by measuring changes in its execution speed when its architecture is changed. As an outlook for future research, we will also present a path toward the development of lightweight, high-performance security solutions suitable for resource-constrained environments.

II. Methods

A. Kitsune's Algorithm

Kitsune [1][2] is one of the AI algorithms for software-based network intrusion detection and is available as open source; the architecture of Kitsune is shown in Figure 1. The three main features of this algorithm are summarized as follows.

- The algorithm is easy to compute because of its simple AI structure.
- Unsupervised learning.
- Real-time intrusion detection.

The first is that the AI part is composed of an ensemble of autoencoders, making it easy to compute. By extracting the features of the packets sent to the system and inputting them into individual autoencoders, the complexity of the calculation is reduced compared to a single large neural network.

Second, it is unsupervised learning, which does not require labeling the input data. This has the advantage of reducing the amount of effort used for learning, and of being able to respond to new malware, for example.

Third, packets arriving at the device can be monitored in real time. This is a great advantage because it allows for early detection and response when a device is in danger.

B. Packet Feature Extraction

The packet feature extraction function is implemented using tshark. The tshark is a command line version of the network packet analysis tool Wireshark. Packet data is converted from pcap files to tsv files using tshark, and information such as source and destination IP addresses, MAC addresses, port numbers, TCP/UDP flags, and packet size is extracted from the data. The data is then input to a feature extractor (FE), which calculates incremental statistics. Specifically, for each feature (flow size, packet interval, byte count, etc.), the statistics (mean, variance, covariance, etc.) are updated each time a packet is added. The extracted features are input to each autoencoder by the feature mapper (FM).

C. Anomaly Detector (AD)

Kitsune's anomaly detector consists of two layers: an ensemble layer and an output layer. After computing the root mean square error (RMSE) of the extracted features in the ensemble layer, the output layer outputs the final root mean square error. The anomaly detector also operates in the manner that it learns the first arbitrary number of input packets and detects whether the remaining packets are anomalous data or not. In other words, it is assumed that the packet data in the learning phase are safe packets with no anomalies.

D. Simulation

An environment was created to run Kitsune, and actual packet data was input for the simulation. For the input packets, network capture files of the “Mirai” malware were prepared and input. Mirai's network capture file contains 764138 packets of data, and the data after 370000 contains many compromised packets. Therefore, Kitsune's default program trains an autoencoder ensemble on the first 5000

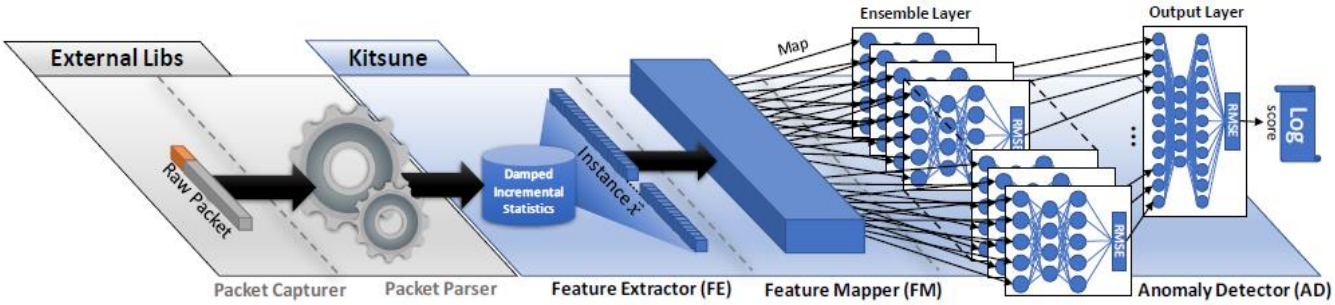

Fig. 1. Kitsune's Architecture [1]

packet data and trains an output layer autoencoder on packet data from 5001 to 50000. After training, the program goes into execute mode and outputs the RMSE values of the incoming packet data to check for dangerous data.

III. Experimental Results

A. *Simulation Results of the Sample Program*

Kitsune is written in python and released as open source. The implementation environment is shown in Table I.

TABLE I. Implementation Environment

Implementation Environment	**Details**
CPU	13th Gen Intel (R) Core (TM) i5-13400F 2.50 GHz
RAM	16 GB
Operating System	Windows11Home, 64bit, 23H2
Programing Language	Python 3.11.7

Figure 2 shows the execution results of this publicly available program.

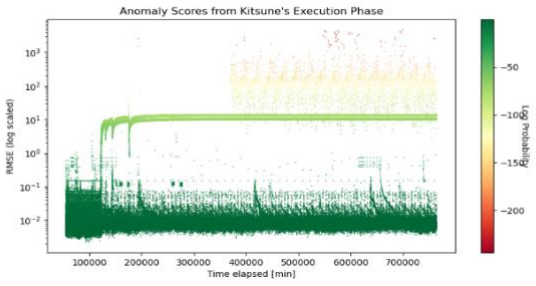

Fig. 2. Simulation results of Kitsune, where Maximum size of Autoencoder is 10.

In the Figure 2, the horizontal axis is the number of packets (time) and the vertical axis is the log scale of RMSE values. As can be seen from the graph, packets with high RMSE values are mostly found in packets after approximately 370000.

Therefore, we modified the program to focus on packets in the 350000 to 400000 range, and the output was as shown in Figure 3. As can be seen from Figure 3, which narrows down the range to be plotted, many packets with high RMSE values are found after approximately 370000 packets.

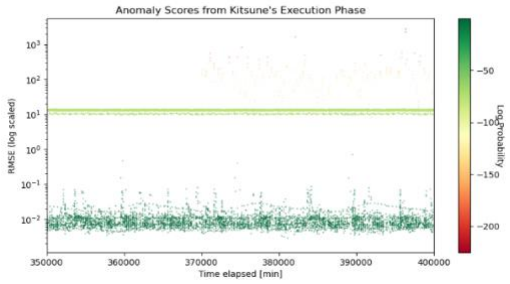

Fig. 3. Simulation results between 350000 and 400000

B. *Change the Number of Autoencoder Ensembles*

The results of changing the maximum size of the autoencoder within the ensemble layer are shown in Figures 4 and 5.

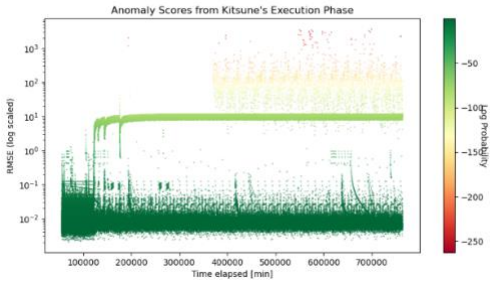

Fig. 4. Maximum Size of Autoencoder was set to 5

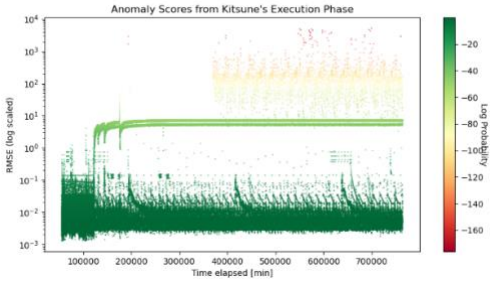

Fig. 5. Maximum Size of Autoencoder was set to 20

Figures 4 and 5 show the results when the maximum size of the autoencoders in the ensemble layer is changed from 10 to 5 and 20. The graphs indicate that there was little difference in the output results. However, changing the maximum size resulted in a difference in execution time, as shown in Table II.

TABLE II. DIFFERENCE IN EXECUTION TIME

Maximum Size of the Autoencoders	Input Parameter m	Execution Time (sec)
5	31	2541.56
10	19	2106.52
20	15	1944.87

It can be observed that as the maximum size of autoencoders is increased, the input parameter m is decreased, and the execution time is reduced.

IV. CONCLUSIONS

In this paper, we evaluated and discussed the operation of an open source implementation of Kitsune, an AI-based NIDS. In the experiment, we used network capture files from the Mirai botnet as input data, and checked the system's operation by adjusting the time range of the output graph and changing the maximum size of the autoencoder in the ensemble layer. As the experimental results, we were able to observe that malicious packets were detected appropriately, and that the execution time increased and decreased as expected when changing the maximum size of the autoencoder. Moreover, the Kitsune adopts an anomaly detection model using an autoencoder, but by replacing it with other neural network models such as convolutional neural networks (CNN) and variational autoencoders (VAE), the model can be improved. Furthermore, memory efficiency can be further improved by reevaluating the incremental statistics used as input data and reducing unnecessary data. In the future works, we aim to improve Kitsune's performance.

REFERENCES

[1] Mirsky, Y., Doitshman, T., Elovici, Y. and Shabtai, A.: ”Kitsune: an ensemble of autoencoders for online network intrusion detection,” Network and Distributed System Security Symposium 2018 (2018).

[2] K. Miyamoto, D. Goto, R. Ishibashi, C. Han, T. Ban, T.Takahashi, J. Takeuchi, "Packet Classification of Malicious Communications Using Kitsune Features." Proceedings of the Computer Security Symposium of the Information Processing Society of Japan, 2021; 2021(0): 1–8.