IJITCE

ISSN 2347- 3657

International Journal of Information Technology & Computer Engineering

Email : ijitce.editor@gmail.com or editor@ijitce.com

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

Health-Related Sensor Data Infection Detection Using Machine Learning

Dr. M V Rathnamma[1,] Dr. M. Sreenivasulu2, Dr. N. Ramanjaneya Reddy[3], Dr. V. Lokeswara Reddy[4]

Abstract—

Small modifications in the virus code are easily detected by conventional signature-based malware detection techniques. The majority of malware programs nowadays are modifications of other applications. They thus have various signatures yet have certain similar patterns. Instead than only seeing slight changes, it's important to recognize the virus pattern in order to protect sensor data. However, we suggest a quick detection technique to find patterns in the code using machine learning-based approaches in order to quickly discover these health sensor data in malware programs. To evaluate the code using health sensor data, XGBoost, LightGBM, and Random Forests will be specifically used. The codes are either supplied into them as single bytes or tokens or as sequences of bytes or tokens (e.g. 1-, 2-, 3-, or 4- grams).

I. INTRODUCTION

All types of sensors are being used to gather health sensor data as we enter the Internet of Things Era. Eventually, malicious software or programs that are hidden in health sensor data and are regarded as intrusions in the target host computer are executed in accordance with a hacker's predetermined logic. Computer viruses, worms, Trojan horses, bonnets, ransom ware, and other harmful codes fall under the category of hazardous codes in health sensor data [1]. Malware assaults may harm computer networks and systems while stealing sensitive data and core data. One of the biggest risks to modern computer security is it [2, 3].There are typically two kinds of malware analysis techniques. [4-7]. (1) Static analysis is often carried out by examining each component of a binary file and displaying its many resources without actually using it. A disassemble may also be used to disassemble (or redesign) binary files (such as IDA). Humans are able to read and comprehend assembly code, which may sometimes be converted from machine code. Malware analysts are able to decipher assembly instructions and see the program's intended behavior.

[1,3] Associate Professor, Department of CSE ,K.S.R.M College of Engineering(A), Kadapa
[2,4]Professor, Department of CSE, K.S.R.M College of Engineering(A), Kadapa

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

Some contemporary malware is developed utilizing unclear methods to thwart this kind of examination, such introducing grammatical flaws in the code. Although these mistakes might be perplexing to the disassemble, they are nonetheless functional during execution. (2) Dynamic analysis is carried out by analyzing the behavior of the virus while it is running on the host 1. The Qatar National Research Fund, a subsidiary of the Qatar Foundation, provided funding for this study under Grant NPRP10-1205-160012.

The writers alone bear full responsibility for the assertions stated in this article. System. Modern malware may employ a number of misleading strategies to evade dynamic analysis, such as testing active debuggers or virtual environments, delaying the execution of harmful payloads, or requesting interactive user input [8–10].

In this work, static code analysis is the major topic. The primary feature matching or broad-spectrum signature scanning techniques used in early static code analysis. Broad-spectrum scanning examines the feature code and employs masked bytes to separate the portions that need to be compared from those that do not, whereas feature matching simply uses feature string matching to complete the detection. The hysteresis issue is critical since both approaches must get malware samples and extract characteristics before they can be identified. In addition, when malware technology advances, the number of malware variants suddenly rises and malware starts to change during transmission in an effort to escape being detected and eliminated. It is challenging to extract a fragment of code to serve as a virus signature since the form of the variations varies greatly.

II. REALTED WORK

Given this circumstance, it makes sense to use machine learning-based techniques that analyze unfamiliar binary code using static code analysis while drawing on prior experience and expertise to automatically categorize malware. In accordance with the instructions, this work makes use of relevant machine learning-based technologies and investigates how to apply this strategy to the categorization of malware [11–14].

Malware detection essentially boils down to a classification issue that determines whether a sample is genuine software or malicious software. Therefore, the key processes of a machine learning algorithm drive host malware detection technology, and the primary research steps of this study are as follows:

Gather enough samples of both genuine software and malicious code.

Process the sample's data effectively and extract the characteristics. Select the classification's primary characteristics further.

By combining the training with machine learning techniques, a categorization model is created.

Using the learned classification model, find unknown samples. Finding the most useful characteristics and models for this practical endeavor is the final objective. The primary research issues and fundamental concepts are presented in this chapter. What people typically utilize in this field, how to get experimental data, and what we do with them in this study are all described below.

A detection model built on top of machine learning techniques and the model we use in the tests, followed by a summary and analysis of the outcomes.

III. MALWARE CODE ANALYSIS

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

A. Malware Sample Collection

The foundation for code analysis is the efficient acquisition of malware samples. The classification model may perform more accurate detection tasks when integrated with machine learning methods, but only after proper training using the sample data [38, 40]. Malware samples may be obtained in a variety of methods.

1) User-side sampling: The majority of anti-virus software providers use this as their primary technique. Antivirus software users that transmit malware samples to providers. This strategy performs well in real-time, but it is challenging to get the data directly since security providers often decide not to release their data in an open manner.

2) Open network databases; examples are VX Heavens, Open Malware, and Virus Bulletin. The open online sample systems are currently constrained in comparison to the pace at which malicious code is updated, and the websites have issues such being subject to assaults. Therefore, the development of a malware sharing mechanism has shown its significance more and more.

3) Additional technological methods a particularly fragile system is created to entice attackers to attack in order for the system to get malware samples via collection utilizing a capture tool like a honey pot (such as the Nepenthes honey pot). Additionally, certain Trojans and Internet backdoors may be acquired through spam traps or security discussion forums. The size of the capture sample for the aforementioned technological techniques is limited, however. Fortunately, Secure Age provided the raw data for this research, which we can utilize directly without further processing. Then, with regard to feature extraction, it is often essential to first disassemble the code in order to extract the static features of the malware. IDAPro, Hopper, OllyDbg, and other popular tools are included.

One of them, IDA Pro, is an interactive disassemble that can produce malware assembly code in addition to doing other tasks including locating functional blocks, obtaining input functions, and outlining functional flow diagrams. These are likewise used in this essay.

B. Feature Selection

Three primary categories of characteristics are as follows:

1) The majority of sample features are extracted using sequence-based feature types. The N-gram is a representative piece of technology. The N-gram, where N is the length of a feature sequence, presupposes that a word's N instances are only connected to its N1 preceding instances. The N-gram model breaks a phrase into LN+1 feature sequences using sliding windows if the phrase set has length L. For instance, when a 3-gram is applied to the word set PUSH SUB SAL, SUB SAL AND, SAL AND DIV, AND DIV LDS, and DIV LDS POP (L=7 at this time), five distinctive sequences are produced: PUSH SUB SAL, SAL AND DIV, AND DIV LDS, and DIV LDS POP. Each string consists of three words.

In order to successfully identify malware in the choice of lemmas, Abou-Assaleh [15] first proposed a feature extraction framework based on byte sequences and applied the K closest neighbor classification approach. Opcodes are employed to choose words in a unique manner. Henchiri published

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

an unique method for extracting n-gram characteristics [16]. We can better define malware by extracting opcode features. According to Moskovitch [17], who examined five different classifiers based on opcode sequences using a test set of more than 3 104 files, the detection accuracy for malware was as high as 99%. to increase classification precision in the presence of incomplete and noisy data. Abualsaud [39] published a new noise-aware signal combination (NSC) ensemble classifier. Using feature extraction, NSC Five different types of characteristics are selected for this paper:

The byte count function. As far as we are aware, a computer's files are entirely composed of binary and hexadecimal code. It makes sense to count the numbers in raw exe files. The first 4096 number strings of exe files are obtained using the PE header as shown below.

Fig. 1. Number string of exe files.

This is a series of strings from 0-255, and a label 0/1 is at the beginning. We count the number of 0-255s in all strings and make labs files using them.

Fig. 2. Labs files

A labs file is a common data format in machine learning. Each line in it starts with a label and some data such as x:y, which means that dimension x's value is y. In the labs, a label of 0 represents malware and 1 represents safe software, while x: y means that in this exe file, the number x occurs y times.

C. Model Selection

The characteristic data that are obtained by the static and dynamic analysis of the malicious code can be used as inputs into the machine learning algorithm training in order to generate a corresponding malicious code classifier.

The Naive Bays is a simple method to build a classifier. In many practical applications, the Naive Bayesian model

Parameter estimation uses the maximum likelihood estimation method. In other words, the Naïve Bayesian model

Can work without the Bayesian probability or any Bayesian model [39].

The KNN algorithm is one of the most intuitive machine learning algorithms. One of the KNN's strengths is to support "enhanced learning," where new samples of the training set can be trained as being incremental without having to retrain the model [38, 40].

The SVM algorithm tries to find a linear hyper plane for binary classification. The SVM and KNN algorithms are

More efficient when dealing with smaller samples, but as the data sets increase in size, the SVM and KNN become

Computationally expensive [39].

Random Forests are a kind of bagging model, which is a joint prediction model that is composed of multiple decision trees. If the model we train in the process is a decision tree, then we will get a Random

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

Forest. For a wide variety of data, it can produce high-accuracy classifiers. It can handle a large number of input variables. It can assess the importance of variables when determining categories. Additionally, the learning process is very fast.

The naïve Bayes, SVM, KNN [39] and Random Forest [25] are 4 traditional learning models. There are also some new machine learning models that were invented in recent years. XGBoost is an open-source software library that provides the gradient boosting framework for C++, Java, Python, R, and Julia. The biggest feature of XGBoost is that it can automatically use the CPU's multithreading to improve the algorithm's accuracy. It has gained much popularity and attention recently as it was the algorithm of choice for many winning teams of a number of machine learning competitions. (Wikipedia) CART (regression tree) is the most basic part of XGBoost. It builds a classification tree based on training characteristics and training data, and determines the prediction result of each

Piece of data. The construction of the tree uses the gin index to calculate the gain and select the features of the tree. The formula of the gin index is given as formula (1), and the gain formula of the gin index is given as formula (2)

$$Gini(D) = \sum_{k=1}^{K} p_k(1-p_k) \qquad (1)$$

Pk represents type k's probability in dataset D, and a large K means a large number of types in D.

$$Gini(D,A) = \frac{D_1}{D} Gini(D_1) + \frac{D_2}{D} Gini(D_2) \qquad (2)$$

D represents the entire dataset; D1 and D2 represent the datasets with feature A in the dataset and the datasets with

Feature non-A, respectively; and Gini(D1) denotes the gini index of the datasets with feature A.

In XGBoost [19], the objective function is approximated by a second-order Taylor expansion and the time complexity significantly decreases. It also defines the complexity of the tree and applies it to the target function, which can dynamic grow the tree through splitting and segment evaluations at the split nodes. These are the advantage of XGBoost that traditional boosting algorithms do not have.

LightGBM [20] is another kind of boosting model. It is a fast, distributed, high performance gradient framework based on decision tree algorithms, which is used for ranking, classification and many other machine learning tasks. It is under the umbrella of the DMTK project of Microsoft. The shortcomings of XGBoost are as follows: (1) each iteration requires traversing the entire training data multiple times. (2) When traversing each split node, a split-gain calculation is needed, which consumes a lot of time too. In this paper, the models we use are XGBoost, LightGBM

And Random Forest. We tested the SVM (Support vector machine) before, but the running speed is too slow and the

Performance is not good enough, and so we ultimately decided to use these 3 models.

IV. EXPERIMENTAL ANALYSIS

In order to illustrate the performance differences between the models, between the features and even between different dimensions in the same feature, we extract 27 subdivided features, including the byte

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

count (256d, where d represents dimensions), opcode 1-gram (150d), opcode 2-4-grams (150, 450, and 750d), daf 1-gram (150d), daf 2-4-grams (150, 450, and 750d), segment (150, 450, and 750d) and dll (150, 450, and 750d), and conduct 81 experiments (we run each feature's libsvm file in XGBoost, LightGBM and Random Forest). The training set is from Secure Age's malware sample from 04/2017 and the test set is the sample from 05/2017. The experiments include 4 parts:

1. Testing the effect of each feature and model in this practical dataset.
2. Testing which model performs the best for a certain feature.
3. Testing which feature performs the best on the whole.
4. Testing which dimension leads to the best results with respect to a certain feature.

For opcode and daf features, we assess the evolutionary trend from 1-gram to 4-grams and whether opcode or daf is better. For a certain model, we also test which kind of feature that model prefers to use.

Three measures are adopted to evaluate them.

1. AUC (area under the curve): If a positive sample and a negative sample are randomly selected, the AUC gives the

Probability that the classifier correctly gives the positive sample a higher score than the negative sample [21]. The

Higher the AUC value is, the stronger the sorting ability of the model [22].

2. Precision and Recall: In this dataset, we define the number of positive samples as P and the number of negative samples as *N* (malware or legitimate software). In the first case, for a sample, if the prediction is positive and it is actually positive, we call it a true positive (*TP*). In the second case, if the prediction is positive and the actual value is negative, we call it a false positive (*FP*). In the third case, if the prediction is negative and the actual value is positive, it is called a false negative (*FN*). In the last case, if the prediction is negative and the actual value is negative, it is called a true negative (*TN*) [39, 40]. Each sample can only belong to one of these four cases. There is no other possibility. Then, we have the following: *P=TP+FN*, *N=TN+FP*, Precision-*P=TP/TP+FP*, *Precision-N=TN/TN+FN*, *Recall-P=TP/P*, and *Recall -N=TN/N*. Recall reflects the ability of the classification model to identify P/N samples. The higher the recall is, the stronger the model's ability to identify *P/N* samples. The precision [23] [41, 42] reflects the ability of the model to discriminate *N/P* samples.

3. Accuracy: *TP+TN/P+N*. It reflects the accuracy of the classifier on the whole, that is, the proportion of correct

Predictions. First, compare the three models from the scale of features. The comparison for the byte count feature is shown in Fig. 4.

Byte count has only one dimension, 256, since there are 256 possible numbers in the byte count (0-255). LightGBM

Performs the best for byte count. The comparison for the instruction 1-4-grams features is shown in Fig. 5. For the

Instruction 1-gram, there are only 160+ kinds of instructions in the X86 assembly language, and so it has only 1 dimensions:

International Journal of
Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

150. For the 2-4-grams, we give it the 3 dimensions of 150, 450, and 750, which mean that we count the 150, 450, and 750 most common instructions, respectively. We also test 3 dimensions in the remaining features except for daf 1-gram since we want to compare it with the instruction 1-gram.

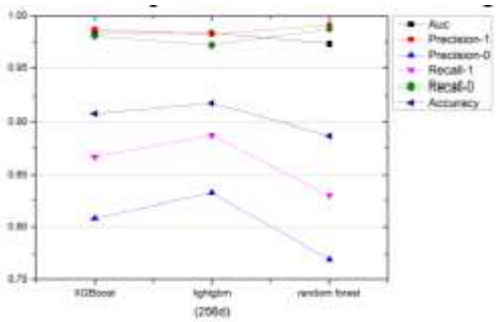

Fig. 4. Comparison for the byte count feature

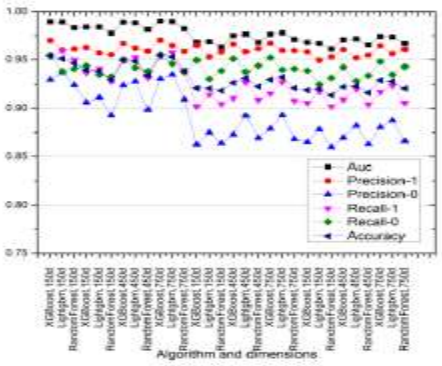

Fig. 5. Comparison for instruction 1-4-grams features for 1-gram and 2-grams, XGBoost is a little better than LightGBM, while for 3-grams and 4-grams, the result is just the opposite. However, in regard to the dimensions, the conclusion is the same, regardless of the model. 750d is better than 450d, and 450d is better than 150d. If we compare the best accuracy of 1-4-grams, we can find that 2-grams >1- gram >3-grams >4-grams. The comparison for the daf 1-4-grams features is shown in Fig. 6.

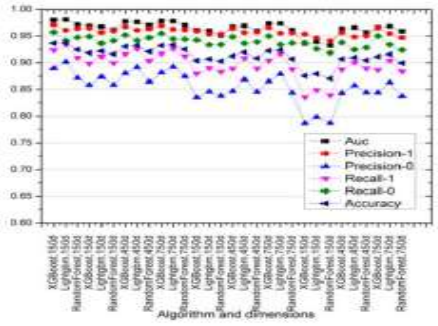

Fig. 6. Comparison for daf 1-4-grams features LightGBM is always the best in 1-4-grams. Furthermore,

We have same dimensional comparison result of 750d>450d>150d and the almost same result for the different grams of 1-gram>2-grams>3-grams>4-grams. Compared with the simple opcode feature [24], the opcode is better than the daf feature by 2.5%. The comparison for the segment feature is shown in Fig. 7.

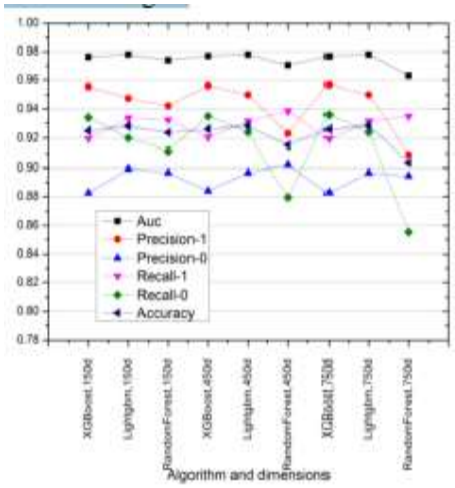

Fig. 7. Comparison for segment feature

LightGBM is the best in this feature, while to my surprise, 450d is better than 750d and 750d is better than 150d. The comparison for the Dll function feature is shown in Fig. 8.

International Journal of Information Technology & Computer Engineering

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

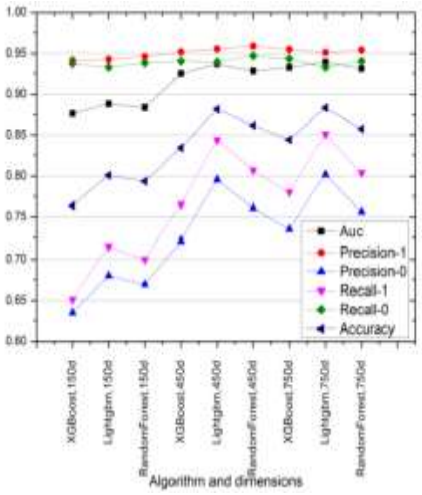

Fig. 8. Comparison for Dll function feature

In this feature, all of the 3 models do not perform well. If we must evaluate them, LightGBM is still the best, and

750d>450d>150d.

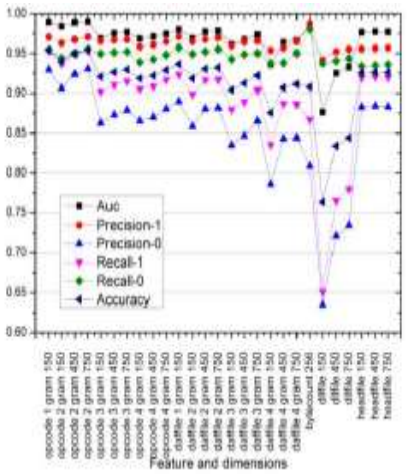

Fig. 9. XGBoost comparison

Second, we compare the three models from the scale of models. The comparison for XGBoost, LightGBM and Random Forest [25] are shown in Fig. 9, Fig. 10, and Fig. 11 respectively.

As is shown, XGBoost and LightGBM prefer the 750d feature while Random Forest prefers 450d feature. They all prefer the Sequence-based feature types rather than the API Call-based feature types.

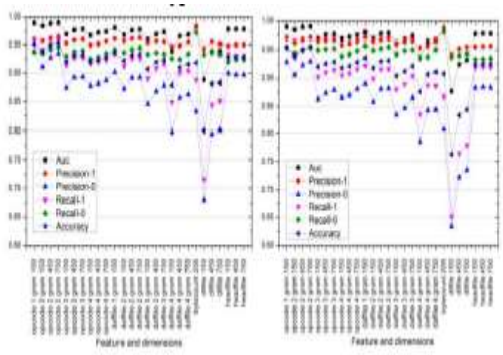

Fig. 10. LightGBM Comparation Fig. 11. Random Forest Comparison In general, we can say that opcode1-gram, 2-grams, daf 1- gram and segment count are the 4 most effective features. For the grams, 1 and 2 are better than 3 and 4, perhaps because 1-gram and 2-grams have more identifiable grammar [26]. For the dimension, the greater the dimension is, the better the result but the Random Forest may not be able to deal with too much noise in high dimensions, while XGBoost and LightGBM perform well. For the models, Random Forest is significantly worse than the other 2 models. In some features, XGBoost has better performance, while for most features, LightGBM is better. LightGBM is indeed an optimized version of XGBoost.

CONCLUSION

The use of machine learning techniques in the identification of dangerous code has been increasingly appreciated by the academic community and various security companies due to the complexity of malware codes hidden in health sensor data [27-30, 38, 40]. This study, which is founded on the idea of machine learning, integrates the benefits of many models [31–33, 36–37] and addresses static code

ISSN 2347–3657

Volume 9,Issue 4,Dec 2021

analysis using various machine learning methods and code attributes. For the design and implementation of malware detection technologies for machine learning in the future, this study may serve as a useful reference [34]. This section is still in the developing stage, nevertheless. Below is a list of all the duties and difficulties that still need to be accomplished.

1. A lack of useful data: To create an efficient model, a machine learning algorithm often has to be trained on tens of thousands of data points [35]. These fundamental data must often be acquired manually, and the speed cannot be guaranteed [36, 37].

2. Interpretable outcomes are lacking: Internally, this is due to the fact that we just know that numerous features are beneficial without understanding why. The biggest hurdle in the future will be how this issue is interpreted.

REFERENCES

[1] L. Wu, X. Du, W. Wang, B. Lin, "An Out-of-band Authentication Scheme for Internet of Things Using Block chain Technology," in Proc. of IEEE ICNC 2018, Maui, Hawaii, USA, March 2018.

[2] M. Sheen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hub, "Cloud-Based Approximate Constrained Shortest Distance Queries over Encrypted Graphs with Privacy Protection", IEEE Transactions on Information Forensics & Security, Volume: 13, Issue: 4, Page(s): 940 – 953, April 2018, DOI: 10.1109/TIFS.2017.2774451.

[3] P. Dong, X. Du, H. Zhang, and T. CSU, "A Detection Method for a Novel Dodos Attack against SDN Controllers by Vast New Low-Traffic Flows," in Proc. of the IEEE ICC 2016, Kuala Lumpur, Malaysia, 2016.

[4] Z. Tian, Y. Cui, L. An, S. Su, X. Yin, L. Yin and X. Cui. A Real-Time Correlation of Host-Level Events in Cyber Range Service for Smart Campus. IEEE Access. vol. 6, pp. 35355-35364, 2018. DOI: 10.1109/ACCESS.2018.2846590.

[5] Q. Tan, Y. Gao, J. Shi, X. Wang, B. Fang, and Z. Tian. Towards a Comprehensive Insight into the Eclipse Attacks of Tor Hidden Services. IEEE Internet of Things Journal. 2018. DOI: 10.1109/JIOT.2018.2846624.

[6] Z. Wang, C. Liu, J. Qiu, Z. Tian, C., Y. Dong, S. Su Automatically Trace back RDP-based Targeted Ransom ware Attacks. Wireless Communications and Mobile Computing. 2018. https://doi.org/10.1155/2018/7943586.

[7] L. Xiao, Y. Li, X. Huang, X. Du, "Cloud-based Malware Detection Game for Mobile Devices with Offloading", IEEE Transactions on Mobile Computing, Volume: 16, Issue: 10, Pages: 2742 – 2750, Oct. 2017. DOI: 10.1109/TMC.2017.2687918.

[8] https://en.wikipedia.org/wiki/Malware_analysis

[9] Z. Tian, W. Shi, Y. Wang, C. Zhu, X. Du, et al., "Real-Time Lateral Movement Detection Based on Evidence Reasoning Network for Edge Computing Environment", IEEE Transactions on Industrial Informatics, Volume: 15, Issue: 7, Page(s): 4285 – 4294, March 2019.

[10] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, M. Guizani, "Security in mobile edge caching with reinforcement learning", IEEE Wireless Communications Volume: 25, Issue: 3, pp. 116-122, June 2018, DOI: 10.1109/MWC.2018.1700291.