Journal of Theoretical and Applied Information Technology
31st August 2021. Vol.99. No 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A NOVEL APPROACH TO DETECT IOT MALWARE BY SYSTEM CALLS AND LONG SHORT-TERM MEMORY MODEL

[1]TOAN NGUYEN NGOC, [2]DUNG LUONG THE, [3]PHU TRAN NGHI

[1]Lecturers, People's Security Academy, and Academy of Cryptography Techniques, Hanoi, Vietnam

[1]PhD Student, Academy of Cryptography Techniques, Hanoi, Vietnam

[2]Associate Professor, Academy of Cryptography Techniques, Hanoi, Vietnam

[3]Lecturers, People's Security Academy, Hanoi, Vietnam

E-mail: [1]ngoctoan.hvan@gmail.com, [2]thedungluong1@gmail.com, [3]tnphvan@gmail.com

ABSTRACT

As the Internet of Things (IoT) devices become voguish, malware detection on IoT devices is crucial today. In this paper, a novel approach to detect IoT malware based on dynamic analysis and deep learning is proposed. Our method combines an IoT-sandbox to extract system call sequences that are considered as sentences in natural language, then two Long Short-Term Memory (LSTM) model are used to classify. In our approach, a program is determined whether malware or benign by two representative values which are the results of LSTM models. Experiment results show that our proposed method outperforms other based-line machine learning models using similar system call feature in terms of accuracy, F1-Weight and the length of system call sequence. Our method uses quite short system call sequence of 150, but the highest accuracy 98.37 per cent and F1-Weight achieves 98.38 per cent. Therefore, the method can be used in early IoT malware detection solutions.

Keywords: *IoT malware, malware detection, system calls, LSTM model.*

1. INTRODUCTION

Today, there are more and more IoT devices that are connected to the Internet. According to Statista predicted that 30.9 billion units are installed by 2025 in the worldwide [38]. So, malware on IoT devices also grows accordingly every year. According to [1] mobile malware metamorphosis increased by 54% in 2017 and IoT attacks increased by 600%, in which the Mirai malware and its variants created some of the most potent DDoS attacks in history. As a result, malware detection on IoT devices is very significant and is interested by researchers in recent years.

Primarily, there are 2 types of malware analysis include: static analysis and dynamic analysis. While static analysis involves inspecting an executable program without execution, dynamic analysis involves examining behavior of the executable program by running it. Both methods have their advantages and disadvantages. The static analysis [3], [4], [6] relied on extracting numerous features from executable programs such header section, String, Function Length Frequency (FLF), Printable String Information (PSI), Operational codes (Opcodes), etc. If malware uses encryption and obfuscation techniques or complex code, it could break away from detection programs. Therefore, it is necessary to use dynamic analysis methods to solve this problem. Dynamic approach on [2], [7]-[10] used various taxonomy such as network traffic, API call traces, instruction traces, memory and register's usage, system calls, … In dynamic analysis, n-gram methods have been used to extract and select features from system call sequences as investigated in Phu et al. [2] and Raymond et al. [11]. Using the combination of n-gram and system calls can be effective in the IoT malware detection. However, the n-gram feature selection methods only consider mutual connection between objects in a limited case, leading to an unsatisfied result. While most of the previously malware researches are focusing on traditional computing devices with the Intel architecture (i386), it has switched to develop methods to detect IoT malware, especially with the ARM or MIPS architecture in recent times [2].

The MIPS processor architecture is used in many popular IoT devices such as routers, switches, access points, and IP cameras [19], [20]. If an application is

run on different processor architectures and operating systems, their behaviors are dissimilar. Therefore, it is necessary to researching about malware detection on devices that are used the Embedded Linux OS and MIPS processor. Many researchers had positive results on malware detection on Android applications based on system call behavior [12]-[16], [21], but to our knowledge, researches to detect malware via system call are limited in MIPS ELF. Phu et al. [2] proposed a sandbox that automatically induct the suitable environment to activate MIPS ELF files and using machine learning classifiers based on n-gram feature selection method. However, the n-gram method and feature extraction methods based on frequency of occurrence of the features have many limitations for example, it will cause difficulty and reduce efficiency in processing malware detection models with large feature sets. While some feature selection methods, such as compressing and reducing features, cannot eliminate completely interfering features and can cause loss of important information, other methods such as LSTM bring high efficiency in natural language processing and miss data prediction. In this study, a dynamic analysis approach is proposed base on the LSTM language model and system call sequences in order to make the MIPS malware detection result better.

System call sequences have the same structure as sentences in natural languages. LSTM model can extract hidden semantic information in the natural language model, so a system call (syscall) is considered as one word and a syscall sequence as one sentence in the natural language model. Our research used two different LSTM language models, which are trained by syscall sequences from malware and benign dataset at first, then a feasibility probability is calculated for one sequence with a model. Based on the probabilities, a representative value is defined to performances the extent of the program belong to two models. The two representative values are compared to classify MIPS ELF programs. Our method is evaluated from several elements, including the maximum length of syscall sequences and the structure of model. Experiments demonstrated that our approach achieves elevated efficiency and the highest accuracy 98.37% with length of system call sequences 150, which is better than that of n-gram methods in [2].

In summary, this research has main contributions as follows:

- An approach based on the Long Short-Term Memory language model and dynamic analysis to detect IoT malware in MIPS architecture-based devices.
- A novel criterion for detection whether a program is malware based on its behaviors.
- Experimental results demonstrated that our proposed model had better results than other machine learning models using n-gram method and can detect malware early in real analysis systems.

The rest of the paper is structured as follows. Related works on malware detection based on system calls are discussed in section 2. LSTM model are introduced in section 3. Section 4 describes the proposed malware detection paradigm. Experiments and evaluations are introduced in section 5. Finally, conclusion and future works are discussed.

2. RELATED WORKS

According to Helenius et al. [17], malicious code is a program designed with the purpose of unwanted users. Ed Skoudis et al. [5] suggest that malware (malicious code) is a set of commands are infected on a user's computer to control the computer to carry out malicious actions. Similar to malware on mobile devices and Linux malware, IoT malware is considered as malicious code infecting IoT devices or IoT networks. However, novel malware detection on Embedded Linux operating system of IoT devices is a huge challenge because of the extensive range of application, dissimilarity of category and increasing processing capability of IoT devices [26].

Malware analysis is a process of determining malicious behavior of a program. Malware analysis is often based on static and dynamic features [27]. Static features have been used such as strings [28], bytes n-gram [29], opcode [30], [31], function call graph [32], entropy-based [33], etc. This method allows for detailed analysis of programs and supply activation capability information of malware. However, static analysis is ineffective in malware detection using complex techniques such as code encryption, obfuscation, polymorphic, ... The effectiveness of static analysis depends heavily on decompilation and disassembler tools.

On the other hand, to be able to analyze complex malicious codes, dynamic feature-based analysis is recommended. In malware analysis, common dynamic features include memory usage [34], instruction traces [8], network traffic [36], API call trace [10], [37]. The effectiveness of dynamic analysis is highly dependent on malware execution

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

environment. One of the most popular methods nowadays is using machine learning and deep learning to collect relevant data during malware execution. In dynamic feature collection, an adequate sandbox is required to monitor behaviors of executable programs. Collected behavior data plays an essential role in the accuracy of malware detection, so sandbox is a suitable environment for collecting malicious behaviors. There have been many proposed sandboxes for collecting application's behavior on the IoT devices, but the most popular are the IoT devices running Android OS [25]. There are some popular sandboxes for IoT devices such as IoTBox, IoTPOT [23], Linux Sandbox LiSa [24], V-Sandbox [25], F-Sandbox [2], Detux[1]. IoTBox which is a sandbox to collect network behaviors of IoT malware program, supports eight processor architectures such as ARM, MIPS, PowerPC, etc. IoTBOX has only focused on collecting network behaviors, not collecting other malware analysis high-meaning behaviors such as system behaviors. Detux is based on QEMU. It supports collecting traffic behaviors in five CPU architectures include x86, x64, ARM, MIPS, and MIPSEL. However, Detux sandbox did not virtualize network peripheral and consider the interaction with the OS. F-Sandbox collected diverse behaviors of ELF file on IoT devices, including both system calls and network association behaviors. In addition to that, adaptive environments are automatically configured for activating ELF files. So, F-sandbox is a suitable sandbox to collect system call sequences on IoT platforms.

A syscall is one mechanism for an application to request one service from underlying OS's kernel [14]. Malicious programs and benign programs have different behaviors, such as the malware request more internal connection or access sensitive resources more frequently. Each single individual system call cannot describe a program's behavior. Therefore, several syscalls sequentially should be considered to determine a program's code-level behavior characterize the program. Grasping the dependencies between the system calls is helpful for classifying the normal and malicious behavior of a program. System call sequence logs can be collected and analyzed with different tools. The use of the system call feature has brought about a lot of efficiency in detecting and classifying malware. Therefore, we use F-Sandbox [2] to extract system call sequences for malicious code detection on MIPS architectural platform.

Using system call feature has brought a lot of efficiency in detecting malware in general and IoT malware in particular. There are quite many studies in this aspect, such as Phu et al. [2] used machine learning algorithms to determine system call sequences generated by malware in a sandbox. Their experimental results have shown good results for system call-based IoT malware detection. Canzanese Raymond et al. [11] used the n-gram method to detect system calls of malicious processes. Marko Dimjašević et al. [14] used machine learning and system calls to classify malware on the Android platform. The research shown that system-call based techniques are viable to be used in practice to detect malware in Android applications. Nikolopoulos et al. [18] used a graphical model based on system calls to detect malicious code. Their model can detect malware with true positives over 94% but false positives 13.1%.

On the one hand, abnormality detection often uses machine learning (ML) and deep learning (DL) on data mining for malware detection. Moreover, machine learning and deep learning methods are effective in predicting novel and metamorphic malware that have never appeared before.

On the other hand, the forecast problem in sequential data could be resolved by establishing a statistical language model. Basically, the models based on n -gram method cannot predict a program's pattern which do not appear in the training dataset because of the limitation of dimensions. If n value is too small such as 1-gram, the frequency of single system call occurrences is shown, so it is difficult to be effective in creating malware detection models. Contrarily, if n value is huge, the quantity of the features is very big. Hence, some neural probabilistic language models are used to improve effective of n-gram method since they can analysis longer context. A neural network is a kind of mathematical model consisting of many layers of neurons. Recurrent neural network is a distinctive structure of neural network, and it can retain state information of previously hidden layer based on a special memory unit. Recurrent neural network (RNN) is used in various fields such as malware detection, speech recognition, and natural language processing. However, it is difficult for standard RNN to learning

[1] https://github.com/detuxsandbox/detux

long-term dependencies by the stochastic gradient going down. A special type of RNN is Long Short-Term Memory (LSTM) [22] that can significantly reduce the disappearing and detonating gradient problems. The LSTM structure includes a set of recurrently connected subnets (memory blocks). These connected subnets are considered as a distinguishable category of memory chips in one computer. The LSTM network can be used to resolve model text sequences forecasting problems.

3. BACKGROUND

The LSTM model contains a series of gates. Each LSTM cell has one structure of longer-term memory in the structure of one cell state that is updated into and out of time. A forget gate considered at the hidden state and new input. It also decides information which can be safely forgotten. Then, the input gate determines what information from the new input will be put on the cell state to remember. Finally, the output gate takes information from the input, cell state, and hidden state to create the output for the present step.

The key of LSTM is cell state. The cell state runs straight down the whole sequence and with only some inconsiderable linear interchanges. Therefore, information to just flow along it does not change. The information about cell state can be added or removed by structures called gates. The gates are made up of one layer of sigmoid neural network and a pointwise multiplication operation, which are ways to optionally let information through optional ways for information to pass through. The sigmoid neural network layer returns numbers between [0,1], expressing how much of each element should be let through. A value of zero means "let nothing through," while a value of one means "let everything through". Three of these gates are created to secure and dominate the cell state.

$$f_t = \sigma\,(W_f . [h_{t-1}, x_t] + b_f) \quad (1)$$

Next, new information which will be stored in the cell state is decided, include two section. A sigmoid layer decides which values will be updated called the "input gate layer". A vector is created with new candidate values by a tanh layer. The tanh function can be calculated as follows:

$$g(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2)$$

where value $g(x)$ in [-1,1].

In the following step, these two are incorporated to create an upgrade to the state.

$$i_t = \sigma\,(W_i . [h_{t-1}, x_t] + b_i) \quad (3)$$

$$\tilde{C}_t = \tanh(W_C . [h_{t-1}, x_t] + b_C) \quad (4)$$

Thereafter, a new cell state C_t is upgraded from C_{t-1} while old state is multiplied by f_t, forgetting the things are decided to forget earlier. Then, $(i_t * \tilde{C}_t)$ value is added. The new candidate values are scaled by how much to upgrade each state value.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (5)$$

Finally, an output is based on the cell state that be a filtered version. A layer of sigmoid neural network is given to decide what parts of the cell state that will be output. Thereafter, the cell state is moved through tanh value [-1,1], then it is multiplied by the output of the sigmoid gate.

$$o_t = \sigma\,(W_o . [h_{t-1}, x_t] + b_o) \quad (6)$$

$$h_t = o_t * \tanh(C_t) \quad (7)$$

LSTM language model can be used to foretell the next word in one sentence. A traditional neural network unit *i* incorporate an input activation a_i and an output activation b_i which is related when a tanh activation function is used by formula:

$$b_i = \tanh(a_i) \quad (8)$$

The LSTM unit adds several intermediate steps: After executing the activation function to a_i, the result is multiplied by factor b_i. Then, the inner activation value of the previous step is multiplied by the quantity b_φ that is added due to the repeated self-connection. Ultimately, the result is scaled by b_ω and is moved fed to another activation function to yielding b_i. The factors b_i, b_φ, $b_\omega \in (0, 1)$ are controlled by additional units called input, output, and forget gate. While the inner of the LSTM unit is activated, the gate units are aggregated from the activations of the previously hidden layer and the activations of the current layer from the previous step. The result value is squashed by a logistic sigmoid function which is set to b_i, b_φ, or b_ω, respectively.

Journal of Theoretical and Applied Information Technology
31st August 2021. Vol.99. No 16

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

4. PROPOSED MODEL

Firstly, system call sequences of MIPS ELF files are collected by executing them in F-Sandbox [2]. Each system call can be considered a word in the natural language. A syscall sequence is considered as one sentence in the natural language. The dataset of system call sequences of benign files is train to build Benign model (BM), and the system call sequences from malware ELF files to build Malware model (MM), a LSTM classifier model is approved to classify the syscall sequences. Finally, system call that is extracted from the malicious file can be identified according to the classification results. Our paradigm method is elucidated in Figure 1. The process consists two parts include system call sequences collection, and LSTM classifier.

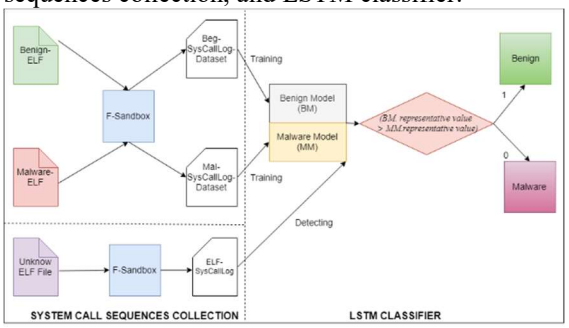

Figure 1: Paradigm of our work

4.1. System call sequences collection

To get the most complete system call sequences, feature vectors are constructed by all the system calls on the MIPS architecture Linux OS with 345 different syscalls. However, they are not used entirely in ELF file data sets. F-Sandbox is used to extracting system call logs of MIPS ELF files. F-Sandbox uses Kprobe to collect the system calls generated from the running program and its child processes. The samples are executed in the F-Sandbox with same configuration, simulating the network environment for an equal amount of time.

After extracting system call logs of the samples, the system call logs of the samples that executable failed or too short are removed. A sample can create multiple processes and one process will generate a system call log, so the system call logs generated from a sample are concatenated. Therefore, each ELF file in the dataset will collect a corresponding system call log file. As the result of this process, two datasets are collected include Mal-SysCallLog dataset which is system call sequences of malware ELF files and Beg-SysCallLog dataset which is system call sequences of benign ELF files.

4.2. LSTM classifier

Unlike n-gram method, the LSTM language model uses strength of all system calls before current call in a system call sequence to forecast the next syscall. More context information is picked up from the syscall sequences than some other language model such as language model using n-gram method. Thus, an LSTM language model classifier is designed by our research. The classifier includes two models: A model is used to train malware system call sequences while the other model is used to train benign system call sequences from the ELF file. Both models use the same LSTM neural network architecture, but parameters are different.

The LSTM network includes three parts: input layer, output layer, and hidden layers. The input gate is a vector encoded by (1-k) coding and the output gate is a vector of the likelihood distribution. In the LSTM classifier, one syscall is treated as a word in the language model. A syscall chain is taken as a sentence in the language model. For a syscall sequence, each next syscall is predicted according to all the preceding system calls in the chain. The likelihood of the syscall sequences can be calculated as follows:

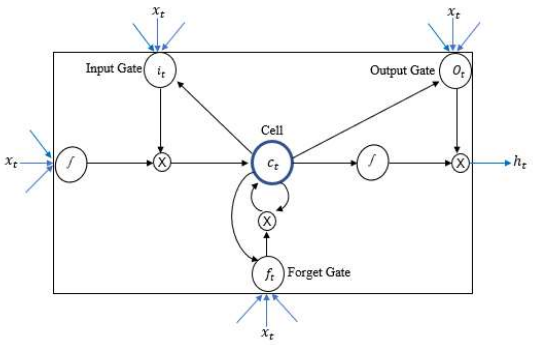

Figure 2: Memory block construction

The likelihood of syscall sequence S_1^N is calculated as follows:

$$p(S_1^N) = \prod_{m=2}^{N} p(S_m \mid S_1^{m-1}) \quad (9)$$

where, S_1^N is one syscall sequence with the length of N, in which S_m is the m^{th} syscall. S_1^{m-1} is subsequence from S_1 to S_1^{m-1}.

For each system call sequence of benign and malware dataset, we train two models which are thus call the Benign Model (BM) and the Malware Model (MM) are trained. Then, two representative values

are calculated based on two models. Then, two representative values used to classify an application by comparing them.

In training phare, two LSTM networks are used to train. As a matter of fact, the LSTM networks are trained at first. Syscalls encoded as vectors are fed into the LSTM network. After feeding one vector into the LSTM network, an output vector is created based on probability distribution V_p [6]. A multiplicative input gate protects memory contents stored from perturbation by irrelevant inputs. A multiplicative output gate protects other units from perturbation by currently irrelevant memory contents stored. A multiplicative forget gate protects other units from perturbation by previous irrelevant memory contents. Finally, next syscall is predicted by the formula:

$$\text{Predicted\_syscall}=(s_i|v_i=\max(V_p),1\le i\le n) \quad (10)$$

where: v_i represents i^{th} element in V_p, which is equal to $p(s_i|\text{subsequence\_before\_}s_i)$. n value is the number of different syscalls or length of V_p

In testing phase, a representative value is defined to depict representative degree between detected application and corresponding file category. The representative value in our method is defined as

$$\text{Representative\_value} = \exp(\log p_i) \quad (11)$$

in which, p_i is probability of i^{th} syscall sequence.

Finally, system call sequences are classified based on representative values. If representative value of an application from Malware model is greater than representative value of Benign model, application is determined as malware. Otherwise, application is determined as benign.

5. EXPERIMENTS AND EVULUATIONS

In our experiments, malware detection framework is based on Tensorflow framework[2] and used GPU on Google Colaboratory[3] to speed up.

5.1. Data collection

An IoT dataset used for testing includes 1,224 MIPS ELF samples (928 malware and 296 benign). The malware dataset is collected by Phu et al. [2] from different sources on the Internet and available programs on Embedded Linux. In addition, our dataset has added utility programs on MIPS platforms from vendors.

Then, system call sequence logs are collected from MIPS ELF samples based on F-Sandbox [2]. Each sample is executed on the sandbox in 30 seconds. The average length of the system call sequences in the two datasets is similar. The system call logs result collected are shown in Table 1.

Table 1: System call logs results are collected by F-Sandbox

Label	Malware	Begin
Number of samples	928	296
Average length of system call logs	327	305

After that, system call sequences that is shorter than 50 will be removed. Besides, concatenates system calls are generated by an application into a system call sequence that representing this application. After this processing, the system call dataset results are shown in Table 2.

Table 2: Syscall dataset results

Min (Length)	50	100	200	300	400	500	1000
Number of malware samples	928	904	858	844	63	61	43
Number of benign samples	296	268	149	128	110	95	74

Analyzing collected system call logs, the malware only uses 136 system calls, the benign uses 127 system calls, most of the system calls of two episodes overlap, and there are 160 system calls appear in both sets. The rate of most appeared system calls on two datasets is shown Figure 3.

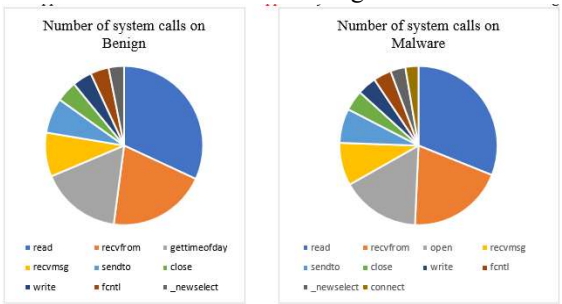

Figure 3: The rate of most appeared system calls

[2] https://www.tensorflow.org/

[3] https://colab.research.google.com/

Figure 3 indicates that the system calls that appear most on both the Mal_SystemCallLog and Beg_SystemCallLog datasets are relatively similar, especially two system calls "read" and "recvfrom" appeared most in the both datasets. So, it is not feasible to use each system call feature for classification in this case.

5.2. Evaluation metric

In our paper, several evaluation standards are used to evaluate the effectiveness of approach in terms of mainly three metrics included F1-Micro, F1-Weight, and accuracy.

The F1-score is the harmonic average of the recall and precision of one class.

$$\text{Precision} = \frac{TP}{TP + FP} \quad (12)$$

where True Positive (TP) indicates that the number of malware samples identified correctly; False Positive (FP) is the number of trusted programs is detected as malware.

Recall is fraction of system call sequence in ground truth that is correctly classified:

$$\text{Recall} = \frac{TP}{TP + FN} \quad (13)$$

where False Negative (FN) is the number of malware samples is taken as trusted programs.

F1-Macro: Average of the F1-scores of classes, characterizing classifier performance on small classes.

F1-Weight: Weighted average of the F1 scores of classes, with weight proportional to their support in the ground truth.

Accuracy can be described as:

$$\text{Accuracy} = \frac{TP + TN}{TP + NP + TN + FN} \quad (14)$$

In the above formula, True Negative (TN) is number of trusted applications identified correctly.

Performance of proposed model is investigated by length of syscall sequences, the number of hidden layers and the number of hidden units in each layer.

5.3. Effectiveness of system call sequence length

Both sentence and syscall sequence can be treated as sequences. However, sentence length is always shorter than syscall sequence length. According to C. Raymond [35], the system call log must be collected within a fixed time and the minimum length of the system call log is 1,500. If a system call sequence of program not enough long, it does not distinguish malicious behavior or normal behavior. In our experiments, the length of system call log less than 50 are generated by error samples such as lack of libraries, insufficient parameters to operate, errors initialize, etc.

In addition to that, there are many different words in natural language, but it has only 345 different system calls are available in MIPS architecture. In our syscall dataset, there are 160 syscalls that appear in both malware and benign sets while the number of different words is 10.000. Maximum length of sentence is about 100 in typical natural language dataset.

In our LSTM classifier, network is constructed with 4 hidden layers and 1000 units in each layer. Then, the classifier is used to detect under different maximum length of system call sequences, i.e., 50, 100, 150, 200, 250, 300, 350, 400, 450, 500. Results are shown in Table 3 and Figure 3.

Table 3: Results under different length of system call sequences

Length	50	100	150	200	300	400	500
Accuracy	97.83	96.47	**98.37**	89.67	88.59	85.87	85.59
F1-Macro	96.94	95.37	**97.81**	86.23	81.06	75.1	75.07
F1-Weight	97.79	96.53	**98.38**	89.79	87.22	83.56	83.31

Figure 3 and Table 3 indicate that when length is 150, the classifier can achieve high accuracy of 98.37% with F1-Macro of 97.81% and F1-Weight of 98.38 %. The network is unable to memorize more information and it is disturbed by additional noises when the sequence is very long, such as maximum length of sequences is 400 or 500. Figure 4 indicates that our classifier can discriminate malware from benign programs and achieve favorable accuracy.

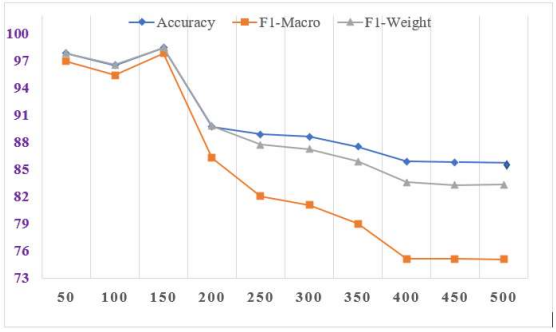

Figure 4: Accuracy, F1-Macro, F1-Weight under different syscall sequence length

5.4. Effectiveness of the number of hidden layers and the number of hidden units in each layer

Performance of LSTM classifier are analyzed under numbers of hidden layers in network from 1 to 6. Each layer of all networks has 1000 hidden units, and maximum length of system call sequence is 150. Results are shown in Table 4.

Table 4: Metrics under the number of hidden layers

Number of hidden layers	1	2	3	4	5	6
Accuracy	90.49	96.2	93.75	**98.37**	96.47	95.92
F1-Macro	88.44	94.73	91.81	**97.81**	95.2	94.74
F1-Weight	90.95	96.17	93.87	**98.38**	96.47	96.03

The table shows that the highest Accuracy of 98.37% with F1-Weight of 98.38% are achieved when there are 4 hidden layers. When the number of hidden layers is one, the neural network cannot catch enough useful information from the sequence. Whereas the number of hidden layers is greater than 4, network is overfitting. Thus, network with 4 layers is appropriate for our work.

Beside the number of hidden layers, relationship between the number of hidden units in each layer and detection performance in the LSTM classifier should be considered. The number of hidden layers in network is set as 4, maximum length of 150, and change the number of hidden units in each layer. Results are shown as Table 5.

Table 5: Metrics under the number of hidden units in each layer

Number of hidden units	600	800	1000	1200	1400
Accuracy	95.92	92.12	**98.37**	95.02	96.20
F1-Macro	94.73	90.20	**97.81**	94.03	95.03
F1-Weight	96.03	92.44	**98.38**	95.13	96.27

Table 5 shows that when the number of hidden units is 1000 in each layer our method has the highest accuracy of 98.37% and the highest F1-Weight of 98.38%. The reason is that when the number of hidden units is less than 1000, model is underfitting. When number is more than 1000, model is overfitting.

5.5. Comparison with machine learning approach using n-gram feature selection method

In order to make a comparison, three machine learning models, which use n-gram feature selection method, are typical detection methods using system call sequences. They are conducted with the same dataset as ours. Results are shown in Table 6.

From the Table 6, when the length of the system call sequence is 500, n-gram method [2] has the highest F1-Macro of 92.11% and F1-Weight of 97.09%, while LSTM classifier reach the highest F1-Macro of 97.81% and F1-Weight of 98.38% with length of system call sequence of 150. It can be obviously seen from table that our LSTM model is better than that of the machine learning models using n-gram method.

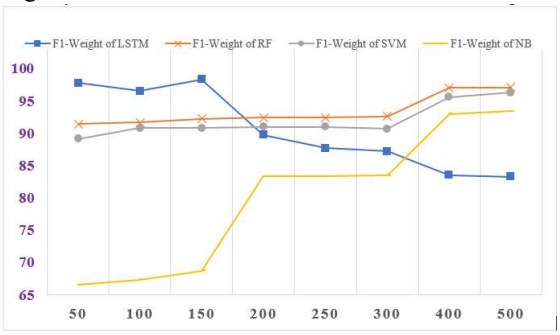

Figure 5: Comparison between our LSTM classifier, RF, SVM, and NB

Figure 5 indicates that LSTM classifier uses shortest system call sequence length of 150 with the highest F1-Weight of 98.38%. Therefore, the LSTM classifier model can be used to detect early and accurately malware in real analysis system.

6. CONCLUSIONS AND FUTURE WORKS

In summary, system call sequence of program has been employed to detect malware. In our classifier, there are two models is built based on LSTM language model to detect IoT malware on MIPS architecture with system call sequences. When a new sequence comes, two representative values are calculated from the two LSTM networks to classify the corresponding programs. The representative values demonstrate efficiency in classify programs based on system call sequence. Our experiments have been done to test the performance of our new classifier. The results show that our method can achieve the highest accuracy of 98.37% and the highest F1-Weight of 98.38%. Moreover, the comparison with the n-gram method and machine learning approach was conducted to show that our classifier is better than the approach base on n-gram feature selection method. In term of the length of system call sequence, our proposed method uses quite shorter system call sequence compare to other methods but still archive better result for IoT malware detection. Thus, the method can obviously be used to early detecting IoT malware in real systems.

In the future, we plan to evaluate our approach against larger and broader datasets, and other sequence analysis techniques can be extended to solve more complicated malware detection problems such as using the static features or combining dynamic features and static features. Deep learning methods combined with more other features could also be considered to detect exactly malware and early detection.

ACKNOWLEDGMENT

This research is fractionally funded by Ministry of Science and Technology of Vietnam, grant number ĐTĐLCN.46/20-C

REFERENCES

[1] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges", *Journal of Network and Computer Applications*, 153(January), 102526, 2020, https://doi.org/10.1016/j.jnca.2019.102526.

[2] Tran Nghi Phu, Hoang Dang Kien, Ngo Quoc Dung, Nguyen Dai Tho, "A Novel Framework to Classify Malware in MIPS Architecture-Based IoT Devices", *Hindawi Security and Communication Networks Volume 2019*, Article ID 4073940, 13 pages, 2019, https://doi.org/10.1155/2019/4073940.

[3] A. Kapoor and S. Dhavale, "Control flow graph based multiclass malware detection using Bi-normal separation", *Defence Science Journal*, vol. 66, no. 2, 2016, pp. 138–145, https://doi.org/ 10.14429/dsj.66.9701.

[4] Mohannad Alhanahnah, Qicheng Lin, Qiben Yan, Ning Zhang, and Zhenxiang Chen, "Efficient signature generation for classifying cross-architecture IoT malware", Proceedings of *the IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, Beijing, China, 2018, https://doi.org/10.1109/CNS.2018.8433203

[5] Ed Skoudis, Lenny Zeltser, "Malware: fighting malicious code", *Prentice Hall*, 2004, https://www.researchgate.net/publication/240105009_Malware_Fighting_Malicious_Code

[6] Graves A, "Supervised Sequence Labelling with Recurrent Neural Networks", *Studies in Computational Intelligence*, 2012, https://doi.org /10.1007/978-3-642-24797-2.

[7] Baysa, Donabelle & Low, Richard & Stamp, "MarkStructural entropy and metamorphic malware", *Journal of Computer Virology and Hacking Techniques*, 2013, https://doi.org/10.1007/s11416-013-0185-4.

[8] D. Carlin, A. Cowan, P. O'Kane, S. Sezer, "The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes", *IEEE Access 5*, 2017, pp. 17742–17752, https://doi.org/10.1109/ACCESS.2017.2749538.

[9] D. Bekerman, B. Shapira, L. Rokach and A. Bar, "Unknown malware detection using network traffic classification", *IEEE Conference on Communications and Network Security (CNS),* Florence, 2015, pp. 134-142, https://doi.org/10.1109/CNS.2015.7346821.

[10] Yuxin Ding, Xuebing Yuan, Ke Tang, Xiao Xiao, Yibin Zhang, "A fast malware detection algorithm based on objective-oriented association mining", *Computers & Security*, Volume 39, Part B, Pages 315-324, ISSN 0167-4048, 2013, https://doi.org/10.1016/j.cose.2013.08.008

[11] R. Canzanese, S. Mancoridis, and M. Kam, "System Call-based Detection of Malicious Processes", 2015, pp 119–124. https://doi.org/10.1109/QRS.2015.26.

[12] S. Hou, A. Saas, and L. Chen, "Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs", 2016, pp. 104–111. https://doi.org/10.1109/WIW.2016.15

[13] X. Xiao, S. Zhang, and F. Mercaldo, "Android malware detection based on system call sequences and LSTM", 2017, https://doi.org/10.1007/s11042-017-5104-0.

[14] Marko Dimjašević, Simone Atzeni, Ivo Ugrina, and Zvonimir Rakamaric, "Evaluation of Android Malware Detection Based on System Calls", In Proceedings of *the 2016 ACM on International Workshop on Security and Privacy Analytics (IWSPA '16).* Association for Computing Machinery, New York, NY, USA, 2016, pp.1–8, https://doi.org/10.1145/2875475.2875487

[15] Dimjaˇ, M., Atzeni, S., Ugrina, I., Rakamari, Z., & Dimjaˇ, M., "Evaluation of Android Malware Detection Based on System Calls", 2015, https://doi.org/10.1145/2875475.2875487.

[16] Gerardo Canfora, Eric Medvet, Francesco Mercaldo, and Corrado Aaron Visaggio, "Detecting Android malware using sequences of system calls". In Proceedings of *the 3rd International Workshop on Software Development Lifecycle for Mobile (DeMobile 2015).* Association for *Computing Machinery*, New York, NY, USA, 2015, pp. 13–20, https://doi.org/10.1145/2804345.2804349.

[17] Helenius, Marko, "A system to support the analysis of antivirus products' virus detection capabilities", Tampere University Press, 2002, https://www.researchgate.net/publication/35704262_A_system_to_support_the_analysis_of_antivirus_products%27_virus_detection_capabilities

[18] Nikolopoulos, Stavros & Polenakis, Iosif, "A graph-based model for malicious code detection exploiting dependencies of system-call groups", 2015, pp. 228-235, https://doi.org/10.1145/2812428.2812432.

[19] A. Costin, Z. Jonas, A. Francillon, and D. Balzarotti, "A large- scale analysis of the security of embedded firmwares", in Proceedings of *the 23rd USENIX Security Symposium*, 2014, pp. 95–110, https://www.usenix.org/conference/usenixsecurity14/techincal-sessions/presentation/costin.

[20] D. D. Chen, M. Egele, M. Woo, and D. Brumley, "Towards Automated Dynamic Analysis for Linux-Based Embedded Firmware", Carnegie Mellon University, Pittsburgh, PA, USA, 2015, https://doi.org/10.14722/ndss.2016.23415.

[21] S. Chaba, R. Kumar, R. Pant, and M. Dave, "Malware detection approach for android systems using system call logs", 2017, https://arxiv.org/abs/1709.08805.

[22] M. Sundermeyer, R. Schl, and H. Ney, "LSTM Neural Networks for Language Modeling", 2012, pp.194–197, https://www.researchgate.net/publication/266030628_LSTM_Neural_Networks_for_Language_Modelin.

[23] Pa, Yin & Suzuki, Shogo & Yoshioka, Katsunari & Matsumoto, Tsutomu & Kasama, Takahiro & Rossow, Christian, "IoTPOT: A novel honeypot for revealing current IoT threats", *J. Inf. Process*, vol. 24, no. 3, 2016, pp. 522-533, https://doi.org /10.2197/ipsjjip.24.522

[24] D. Uhrıcek, LiSa, (2020), "Multiplatform Linux Sandbox for Analyzing IoT Malware". [online] Available: http://excel.fit.vutbr.cz/submissions/2019/058/58.pdf.

[25] Le Hai Viet and Ngo Quoc Dung, "V-Sandbox for Dynamic Analysis IoT Botnet". in *IEEE Access*, vol. 8, pp. 145768-145786, 2020, https://doi.org/10.1109/ACCESS.2020.3014891.

[26] Tran Nghi Phu, Le Huy Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, and Nguyen Ngoc Binh, "CFDVex: A Novel Feature Extraction Method for Detecting Cross-Architecture IoT Malware. In Proceedings of the Tenth International Symposium on Information and Communication Technology (SoICT 2019)", *Association for Computing Machinery*, New York, NY, USA, 2019, pp. 248–254, https://doi.org/10.1145/3368926.3369702.

[27] S. D. Nikolopoulos, and I. Polenakis, "A graph-based model for malware detection and classification using system-call groups", *Journal of Computer Virology and Hacking Techniques*, 2016, https://doi.org/10.1007/s11416-016-0267-1

[28] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent pe-malware detection system based on association mining", *J. Comput*. Virol. 4 (4), 2008, pp. 323–334, https://doi.org/10.1007/s11416-_008-_0082-_4.

[29] Z. Fuyong, Z. Tiezhu, "Malware detection and classification based on n-grams attribute similarity", in: *2017 IEEE International Conference on Computational Science and*

Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), vol. 1, 2017, pp. 793–796, https://doi.org/10.1109/CSE-EUC.2017.157.

[30] D. Yuxin, Z. Siyi, "Malware detection based on deep learning algorithm", *Neural Comput*, Appl. 31 (2), 2019, pp. 461–472, https://doi.org/10.1007/s00521-_017-_ 3077-_6.

[31] I. Santos, F. Brezo, X. Ugarte-Pedrero, P.G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection", *Inf. Sci. 231, 64–82 data Mining for Information Security*, 2013, https://doi.org/10.1016/j.ins.2011.08.020.

[32] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification", CODASPY 16. In: Proceedings of *the Sixth ACM Conference on Data and Application Security and Privacy,* ACM, New York, NY, USA, 2016, pp. 183–194, https://doi.org/10. 1145/2857705.2857713.

[33] D. Gibert, C. Mateu, J. Planes, R. Vicens, "Classification of malware by using structural entropy on convolutional neural networks", In: *IAAI Conference on Artificial Intelligence*, 2018, pp. 7759–7764. https://www.aaai.org/ocs/index.php/AAAI/AAI18/paper/view/16133.

[34] M. Ghiasi, A. Sami, Z. Salehi, "Dynamic vsa: a framework for malware detection based on register contents". Eng. Appl. Artif. Intell. 44, 2015, pp. 111–122, https://doi.org/10.1016/j.engappai.2015.05.008

[35] C. Raymond, "Detection and classification of malicious processes using system call analysis". Drexel University, Philadelphia, PA, USA, 2015, https://www.researchgate.net/publication/336737030_Detection_and_Classification_of_Malicious_Processes_Using_System_Call_Analysis.

[36] G. Zhao, K. Xu, L. Xu, B. Wu, "Detecting apt malware infections based on malicious dns and traffic analysis", *IEEE Access 3*, 2015, pp. 1132–1142, https://doi.org/10.1109/ACCESS.2015.2458581

[37] Z. Salehi, A. Sami, M. Ghiasi, "Maar: robust features to detect malicious activity based on api calls, their arguments and return values", Eng. Appl. Artif. Intell. 59, 2017, pp. 93–102. http://www.sciencedirect.com/science/article/pii/S0952197616302512

[38] Internet of Things (IoT) active device connections installed base worldwide from 2015 to 2025: https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/(Accessed: 2021-03-06).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 6: Comparison between LSTM classifier and n-gram methods

Len	F1-Macro				F1-Weight			
	RF	SVM	NB	LSTM classifier	RF	SVM	NB	LSTM classifier
50	85.03	79.81	67.21	96.94	91.48	89.13	66.56	97.79
100	85.6	84.65	69.01	95.37	91.73	90.82	67.35	96.53
150	84.95	82.67	70.19	**97.81**	92.29	90.83	68.76	**98.38**
200	85.48	82.54	77.79	86.23	92.45	91.02	83.38	89.79
250	85.18	82.43	77.89	82.02	92.48	91.04	83.38	87.74
300	85.13	80.88	77.81	81.06	92.61	90.73	83.51	87.22
400	91.82	88.09	78.84	75.1	97.04	95.61	93	83.56
500	**92.11**	**90.64**	**79.49**	75.07	**97.09**	**96.34**	**93.46**	83.31