Transformer encoder-based Crypto-Ransomware Detection for Low-end Embedded Processors

Hyun-Ji Kim*, Se-Jin Lim*, Yea-Jun Kang*, Won-Woong Kim*, Hwa-Jeong Seo*
*IT Department, Hansung University, Seoul
{khj1594012, dlatpwls834, etus1211, dnjsdndeee, hwajeong84}@gmail.com

Abstract

Crypto-ransomware has a process to encrypt the victim's files, and crypto-ransomware requests the victim for money for a key to decrypt the encrypted file. In this paper, we present new approaches to prevent crypto-ransomware by detecting block cipher algorithms for Internet of Things (IoT) platforms. The generic software of the AVR package and the lightweight block cipher library (FELICS) written in C language was trained through the neural network, and then we evaluated the result. Unlike the previous technique, the proposed method does not extract sequence and frequency characteristics, but considers opcodes and opcode sequences as words and sentences, performs word embedding, and then inputs them to the neural network based on the encoder structure of the transformer model. Through this approach, the file size was reduced by 0.5 times while maintaining a similar level of classification performance compared to the previous method. The detection success rate for the proposed method was evaluated with the F-measured value, which is the harmonic mean of precision and recall. In addition to achieving 98% crypto-ransomware detection success rates, classification by benign firmware and lightweight cryptography algorithm, Substitution-Permutation-Network (SPN) structure, Addition-Rotation-eXclusive-or structure (ARX) and normal firmware classification are also possible.

Keywords: Deep learning, Cryptography, Ransomware, Internet of Things(IoT)

1. Introduction

In 2017, users of Microsoft Windows were infected by Wannacry ransomware virus, since the file sharing protocol has the the vulnerability[1]. It is the biggest attack in the history of ransomware, with more than 200,000 computers affected. There are two categories of ransomware, including cryptography type or locker type. Since the locker type locks the target machine, it can no longer be accessible by users[2]. However, the data can be copied to other devices and the data can be recovered because the data is not encrypted fully. On the other hand, files of devices are encrypted by the cryptography ransomware. Since cryptography algorithms used to encrypt the victim's files are designed to be secure in mathematical assumptions, it cannot be recovered without the valid secret key. The victim has to pay the ransom to the hacker to recover files. Then, the victim can recover original files using the secret key. Since most users are moving to digitization, the ransomware is a large threat to digital devices. In order to prevent ransomware attacks, many works devoted to detect the ransomware virus and recover damages. Four most common crypto ransomware programs are analyzed in [3]. All ransomware viruses that rely on the target system's available system tools are identified. The file can be recovered by shadow copies generated while the tool is running.

The ransomware detection is divided into the analysis of data traffic and the function call. In order to improve the performance, the machine learning is actively studied. In [4], the ransomware virus is identified by analyzing ransomware network behavior and packet selection. In [5], the light and deep networks to detect the ransomware virus were evaluated. The dominance of application programming interfaces (APIs) is analyzed to characterize and differentiate ransomware. In [6], a framework for multi-level big data mining is utilized. The ransomware is analyzed at different levels such as the function call, dynamic link library (DLL), and machine code level through supervised machine learning. Many researches focused on the cryptography function because of the nature of crypto ransomware.

In [7], the collected data through the dynamic binary analysis is utilized to characterize a specific aspect of cryptographic codes. In [8], an approach is presented to automatically identify parameters and block cipher algorithms in the binary code, and it is based on static. In [9], asymmetric key cryptographic (AKC) algorithms are targeted since the ransomware performs the public key algorithms to encrypt files. The encryption process performed by public-key cryptographic algorithms can be detected by monitoring integer multiplication instructions. However, the architecture of block cipher was paid little attention in previous works. Also, there is not many works to defend against ransomware in the Internet of Things (IoT) environment.

In this paper, we propose a new approach to prevent ransomware viruses by classifying the encryption process of block ciphers in low-end embedded processors. By analyzing the cryptographic algorithm opcode on the target embedded processor, it is classified into cryptographic ransomware and normal software. For this, an artificial neural network is used, and the block cipher algorithm of FELICS (Fair Evaluation of Lightweight Cryptographic Systems) and the binary code of the normal program of Alf and Vegard's RISC (AVR) processor package are learned and evaluated. The proposed method is based on the encoder structure of the transformer. The opcodes extracted from the binary file correspond to the words constituting the sentence. That is, the opcode sequence becomes one sentence in which words are listed, and the corresponding opcodes are expressed in a vector by word embedding, and then input into the neural network to learn information such as correlation, order, and pattern between opcodes. We can classify by inputting the binary file of the file encryption process or benign firmware to the trained model. The proposed method successfully classified the encryption process and detected the ransomware virus.

This paper is an extension of our previous work[10]. In previous work, after extracting the opcode sequence and opcode frequency from the binary file, training through two different neural networks. Then late fusion to classify each cryptographic algorithms, benign firmware, and crypto-ransomware is performed.

In Chapter 2 of this paper, related work is explained. Then, we propose and evaluate a new method to prevent crypto-ransomware in Section 3 and Section 4. Finally, we conclude this paper in Chapter 5.

2. Related Works

2.1 Ransomware on IoT Environments

Due to the rapid development of IoT, ransomware virus prevention and security enhancement are being established as basic components of IoT-based services [11]. For a safe IoT environment, many studies are being conducted as follows. In [12] presents a machine learning-based ransomware detection method. After monitoring the power consumption patterns of some processes, it detects malicious ransomware in benign applications. In [13], a deep learning-based method for detecting malicious code using an opcode sequence is presented, and the opcode is converted into a vector space and then learned. In [14] proposes a behavior-based approach. It extracts the Transmission Control Protocol/Internet Protocol (TCP/IP) header and puts it on the command and control (C&C) server blacklist to detect ransomware attacks. In [15], the sequence of instruction is transformed into an image, and then the multiple classes are separated using dimensionality reduction and statistical methods. However, these approaches have focused on advanced IoT platforms and low-end microcontrollers are used for IoT services to collect data from a distance. Therefore, in order to improve the security of IoT-based services, a ransomware detection mechanism for low-end microcontrollers is required. In this study, we propose a new mechanism to detect cryptographic ransomware in a resource-constrained, low-end IoT environment.

2.2 Ransomware Detection Methods based on Cryptographic Function Call

Crypto-ransomware uses encryption to encrypt the victim's files, so to detect it, you need to detect the encryption function. Table 1 compares the ransomware detection method based on the encryption function call. In [7], symmetric key and public key cryptography were detected through the features of encryption functions. This approach used a heuristic based on the target architecture. In [8] utilizes a data flow graph extracted from a binary file, and cryptographic function calls are identified using subgraph isoforms. In Reference [9], the multiplication instruction frequently called in the Rivest-Shamir-Adleman (RSA) algorithm was monitored to detect the public key encryption algorithm. Recently, the work[16] showed that deep learning algorithms can improve malware detection. However, this method does not target crypto-ransomware, and the high-end desktop is the target platform. In this paper, we present an crypto-ransomware detection mechanism targeting microcontrollers.

Table 1 Comparison of Ransomware Detection Techniques Based on Cryptographic Function Calls (SKC and AKC mean Symmetric Key Cryptography and Asymmetric Key Cryptography, respectively)

Category	[7]	[8]	[9]	This Work
Crypto graphy	SKC and AKC	SKC	AKC	SKC
Approach	Dynamic	Static	Dynamic	Static
Algorithm	Heuristics	Data graph flow	System monitor	Deep learning
Architecture	Desk top	Desk top	Desk top	Embedded Processor

2.3 Fair Evaluation of Lightweight Cryptographic Systems (FELICS)

In 2015, the University of Luxembourg published the Cryptographic Benchmarking Framework (FELICS) [17], and cryptographic engineers around the world submitted several block cipher implementations for embedded processors to FELICS. In this paper, the block cipher implementation of FELICS was used as data, and the target microcontroller used ATmega128, an 8-bit AVR widely used in low-end IoT environments. The microcontroller is an 8-bit single chip based on a modified Harvard architecture and uses registers and instructions in units of 8 bits. In addition, block ciphers targeted in this paper can be classified into two types as follows. There are Addition, Rotation, Bitwise eXclusive-or (ARX) and Substitution-Permutation-Network (SPN), the two architectures can be characterized with different operations and overall structures. In this paper, binary codes are classified according to the characteristics of the two architectures.

3. Proposed Method

In this paper, we propose an artificial neural network-based crypto-ransomware detection method for low-end IoT devices. In general, low-end IoT devices are used as leaf nodes to collect sensor data, etc. The base station (firmware server) manages the devices and updates the firmware of IoT devices regularly for better service. At this time, there is a risk of crpyto-ransomware being inserted into IoT devices, and the base station or the devices must detect the crypto-ransomware virus. The proposed method uses an artificial neural network to classify crypto-ransomware and benign firmware according to whether the encryption process is executed or not before the firmware is distributed to the device. We propose a method based on the encoder structure of the transformer, utilizing data extracted from binary code. Since ransomware encrypts the victim's files, the encryption process can be a characteristic of ransomware. The proposed method targets a lightweight block cipher algorithm to determine whether the encryption process is executed or not. If you compile the source code of the lightweight block cipher algorithm, we can get a binary file of the source code, and in the binary file, there are opcodes for various functions such as encryption, decryption, and key schedule. Since different cryptographic algorithms perform different operations, the core operation used by each algorithm can be characterized. Therefore, the opcodes of key functions that perform encryption are extracted from each algorithm and used as a feature to detect the encryption process. In addition, it enables neural network inference on low-end devices through the Tensorflow Lite model.

3.1 Transformers encoder-based crypto-ransomware detection

In this section, we propose a transformer encoder-based crypto-ransomware detection technique. Figure 1 is the overall system structure diagram. Extract the core function from the binary file of each encryption algorithm, and extract the opcode from the function. In this proposed method, string-type opcodes are used as words in sentences. That is, it does not convert to decimal, and after embedding each opcode as a vector, the vector is used as input data. Unlike previous research results, learning and reasoning are performed by configuring a single network, and it is classified into benign firmware and encryption algorithms, and the encryption algorithm is determined as crypto-ransomware.

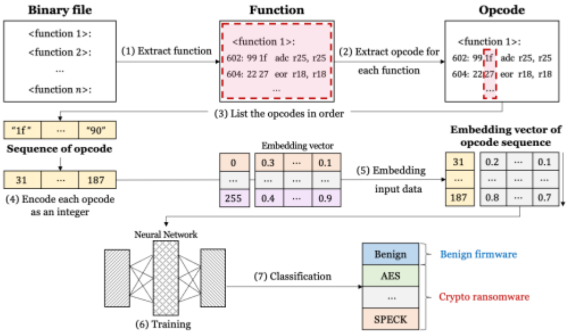

Figure 1 Diagram of transformer encoder-based crypto-ransomware detection system

3.2 Data generation and preprocessing

A binary file can be obtained by compiling the source code of the lightweight block cipher algorithm. Since the encryption process is a characteristic of ransomware, the function that performs encryption is extracted from the binary file, and the opcode is extracted for each function. Also, since the opcodes for each function become the characteristics of the function, as shown in Figure 2, each opcode is labeled with the name of the cryptographic algorithm including the function to which it belongs.

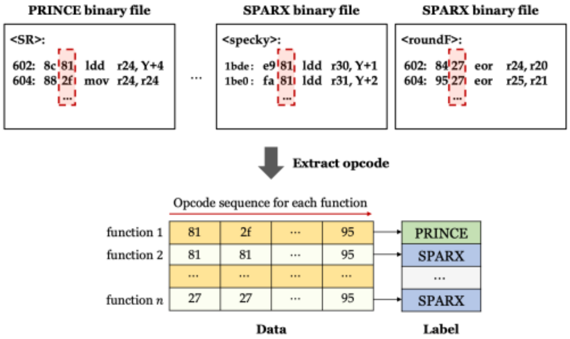

Figure 2 Extracting and labeling opcode sequences

After extracting the core function of each algorithm from the binary file, the opcode within the function is extracted. The extracted opcodes are listed in order and become time series data. In this proposed method, a transformer network is used to learn such time-series data. Therefore, we consider of each opcode as a word and the opcode sequence as a sentence. In order to input such data to the network, a preprocessing process through word embedding is required. Figure 3 shows the embedding of opcode for pre-processing.

This is the process of representing a single word, opcode, as a dimensional vector. Since there are a total of 256 opcodes, in case of one-hot encoding, one opcode is expressed as a 256-dimensional vector. The embedding dimension was set to 8 dimensions through experiments, and the number of words was set to 256, the number of opcodes. By reducing the embedding dimension from 256 dimensions to 8 dimensions, memory usage can be reduced, which is an advantage for inference in low-end IoT devices. Also, before the embedding process, the length of the array should be adjusted to 1,000 through zero padding. Since each opcode sequence array constructed has a different length, the length must be unified in order to be input to the neural network. For all sequence arrays, the minimum length is 14 and the maximum length is 13,859 (unless functions are used for all operations). Most of the source codes are written using functions, and they are only two in the total data. As a result of calculating the percentile, the case of length 878 occupies 90% of the total data. Therefore, the maximum length of the opcode sequence array is set to 1000. If the length is less than 1000, the remaining part is filled with zeros, and if the length is long, it is truncated to 1000.

This pre-processing process can be performed through various embedding techniques, and in this study, the embedding layer was used. Therefore, by attaching the corresponding layer to the front of the encoder structure of the transformer, it is used as an input layer of the neural network, and the embedding value is adjusted with the gradient during backpropagation in the learning process.

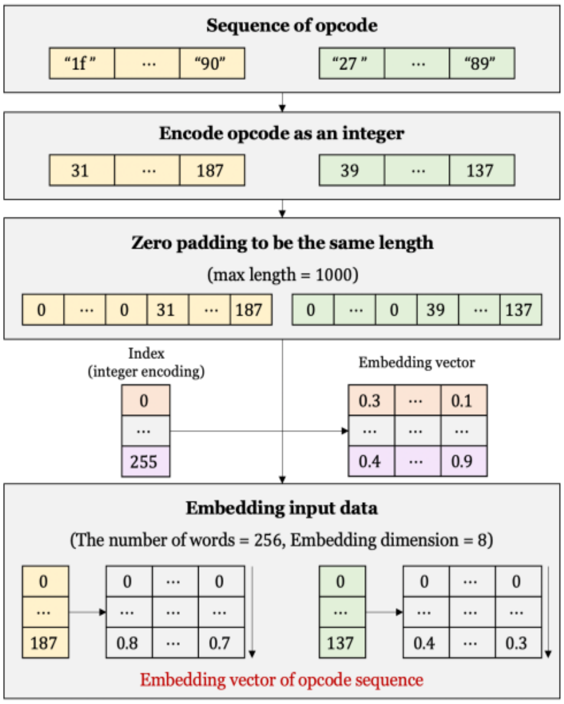

Figure 3 Opcode embedding process for input data pre-processing

3.3 Neural network architecture and training phase

Figure 4 is a schematic diagram of the neural network of the proposed method. Only the encoder is used among the transformer models, and a classifier for multi-class classification is added at the end of the encoder. By using the transformer model, the sequential processing of time series data learning is eliminated and input data is processed at once, so parallel processing is possible and efficiency is improved.

First, the embedding vector through the preprocessing process is input to the network. At this time, since it is time series data, 'Positional Encoding' is performed to consider order information. The process is encoded as a value between –1 and 1 through the sine(sin) and cosine(cos) functions, which can predict the distance between tokens, express long sentences, and predict position values. When the entire dimension is d dimension, a matrix containing position information is generated by applying sin to an even number and cos to an odd number to the position information value of each opcode. Thereafter, the matrix indicating the input embedded opcode and the element-by-element addition are performed. Through this, order information of the input opcodes is learned.

The value to which the location information is added is input to the encoder of the transformer model and then input to the classifier. That is, the proposed method uses only the encoder among the encoders and decoders of the transformer model. Through this, the correlation of all elements of the input opcode sequence is analyzed, and overall information and characteristics of the context are extracted. When it is input to the encoder, it goes through 'Multi-head self-attention' as shown in the figure. This part is the part that performs multiple attention, and the attention technique calculates how much each key affects based on the query, and then adjusts the weight required for learning by reflecting the result. At this time, since each opcode becomes a Query or Key, the association between opcodes and various features can be learned. Also, since each head is divided into different Value, Key, and Query, different results are obtained. That is, by using multiple heads, various relationships and characteristics for each opcode can be learned, and the number of heads was set to two through experiments. Also, the unit of the 'Dense' layer for feed-forward is set to 4. In addition, there are hyperparameters of the layer for normalization.

In the case of the proposed method, it is judged to be a more efficient and suitable method because the entire opcode sequence is input at once and the association between the entire context and elements is learned through the encoder.

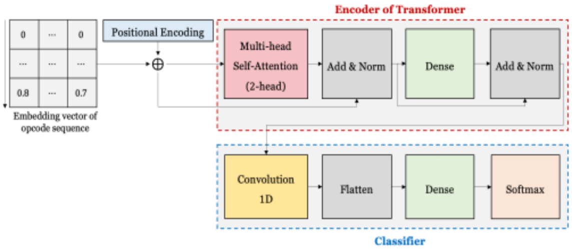

Figure 4 Diagram of transformer encoder-based neural network

After going through the encoder part of the transformer as above, input data is entered to the network for classification. The two parts are not separate networks, they input the encoder's output to the 'Convolution 1D' layer. The layer has fewer parameters because it shares weights through filters, and time-series data can be considered. After flattening it into a one-dimensional vector through the ‘Flatten’ layer, it is input to the output layer and classified into 12 classes. Table 2 is the hyperparameter of the proposed method, and the corresponding values were set through experiments.

Table 2 Hyperparameters for Transformer encoder-Based Networks

Hyperparameter	Description
The number of labels	12
Sequence max length	1,000
The number of words	256(0~255 opcodes)
Embedding dimension	8
The number of heads	2
The number of encoder	1
Epsilon of normalization	1e-6
Feed Forward unit	4
Conv1D filters	16
Conv1D kernel size	8
Conv1D kernel size	1
Batch size	32
Epochs	70
Loss	Sparse categorical crossentropy
Optimizer	Adam (lr = 0.0002)
Activation	ReLu (hidden), Softmax (output)

3.4 Detection of crypto-ransomware

Figure 5 shows the crypto-ransomware detection process in low-end IoT devices. After converting the trained network into a TFLite model that can be inferred from IoT devices, it is distributed. After writing the code to encrypt the file through the given encryption algorithm, and extracting 1,000 opcodes from the beginning of the binary file obtained by compiling, the opcodes are pre-processed through the proposed method. After that, if it is input to the learned neural network, it can be classified into normal firmware and each encryption algorithm.

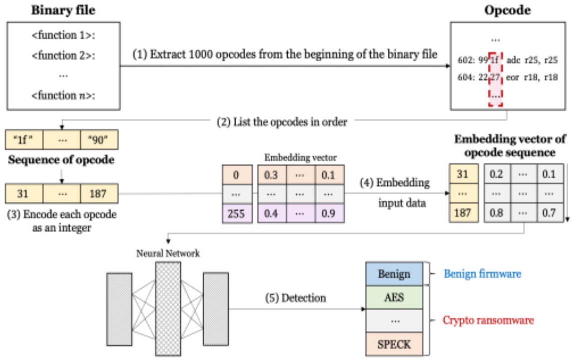

Figure 5 System diagram of transformer encoder-based crypto-ransomware detection for detection phase

4. Experiment and evaluations

In this experiment, Google Collaboratory, a cloud-based service, was used, and Intel Xeon CPU (13GB RAM), Nvidia GPU (12GB RAM) and Ubuntu 18.04.5 LTS are supported. Python 3.7.11 and Tensorflow 2.6.0 were used as programming environments. Also, ATmega128 was targeted as the target board, and GCC-AVR compiler was used.

4.1 Dataset

As mentioned in Chapter 3, when the crypto-ransomware is executed, unlike the benign firmware, it performs an encryption process, so the encryption operation is set as a characteristic of the ransomware. Table 3 is the dataset used in this experiment. It consists of a lightweight block cipher algorithm that corresponds to the substitution-permutation-network (SPN) and addition-rotation-exclusive-or (ARX) structures, which are symmetric key cryptographic algorithms, and general firmware in the IoT environment. Cryptographic algorithms belonging to SPN and ARX are cryptographic modules written in C language during the implementation of FELICS. And as the benign firmware, RFID (Radio-Frequency Identification), WiFi, xBee, Bluetooth, etc. are used. The value in parentheses is the number of data, and in the detection phase, the binary file of the lightweight block cipher belonging to the test set and the benign firmware are cut and used by 1,000 from the beginning.

Table 3 Details of dataset

Architecture	Examples
SPN (88)	RECTANGLE(5), PRIDE(25), PRINCE(33), PRESENT(10), and AES(15)
ARX (191)	SPECK(37), RC5(12), LEA(9), SIMON(19), HIGHT(58), and SPARX(56)
Benign (66)	XBee, GPS, WiFi, Bluetooth, and RFID

4.2 Transformers encoder-based crypto-ransomware detection

This experiment aimed to classify cryptographic algorithms and benign firmware and detect crypto-ransomware using a network composed of encoders and classifiers of transformers. Among the hyperparameters mentioned in Chapter 3, experiments were carried out according to the embedding dimension, the number of layers of the classifier, and the number of neurons in order to reduce the complexity of the model and make it lighter. In addition, there are factors such as the number of heads of multi-head self-attention and the number of units in the dense layer corresponding to the feed forward, and these factors were also set through hyperparameter tuning. In addition, after measuring the capacity of the tflite model for distribution to IoT devices, it was compared with previous work.

Also, in this experiment, the performance is measured using the F-measure, which is the harmonic average of precision and recall. In addition, since we use an unbalanced dataset in which the number of data is not the same for all classes, we evaluate using micro F-measure (considering the number of data in each class) for each experiment. and the corresponding result is the average value of 10 experiments. Validation F-measure and Test F-measure are the results of experiments through Validationset and Testset, respectively, and Detection F-measure is the result of testing after applying the pre-processing of the detection step rather than the pre-processing of training.

In addition, each evaluation is performed using 'Each algorithm vs. Benign firmware', 'SPN vs. ARX vs. Benign firmware', and 'Crypto-ransomware vs. Benign firmware' that is divided. Each experiment is an experiment to classify individual encryption algorithms and normal firmware, an experiment to classify an SPN structure encryption algorithm and ARX structure encryption algorithm and normal firmware, and an experiment to classify the encryption algorithm as crypto-ransomware.

In a resource-constrained environment such as a low-end device, a model with a small size and fast inference is used even if the performance is slightly lower. Recently, many studies have been conducted to reduce the size of the model while maintaining the model performance as much as possible. Therefore, the experiment was conducted to find an efficient network in consideration of the performance and model complexity, and the performance according to the embedding dimension, the number of layer neurons of the classifier, and the kernel size were compared. Except for the above three factors, the remaining hyperparameters were set through optimization, and finally, the hyperparameters in Table 3 were used. In addition, through the experimental results in Section 3, the F-measure for classifying 11 encryption algorithms and normal firmware was the lowest and the crypto-ransomware detection performance was the highest, so this experiment was conducted with 'Each algorithm vs. Measure validation F-measure for 'Benign firmware'. After performing the other two experiments with the values determined through the experiment, the results were analyzed. Table 4 and Table 5 show the number of F-measures and parameters according to the embedding dimension, the number of units of the convolutional layer, and the kernel size. Performance of 0.94 or more was achieved in all combinations, and the maximum difference was 0.03. There was no significant difference for each hyperparameter combination, and in the case of achieving the highest performance, the kernel size was 8, the number of units of the Conv1D layer was 16, and the embedding dimension was 8. By reducing the embedding dimension from 256 to 8 and minimizing the number of filters, the number of parameters was significantly reduced, and the number of parameters at this time was 202,432. This is a result suitable for targeting low-end IoT devices because it has the highest performance while having a small number of parameters compared to other cases.

Table 4 When the embedding dimension is 128, the validation F-measure and the number of parameters according to the number of units of the Conv1D layer and the kernel size (expressed as F-measure (number of parameters))

Embedding dimension = 128		The number of units of Conv1D		
		64	32	16
kernel size	8	0.95 (1,122,640)	0.95 (708,528)	0.96 (501,472)
	16	0.94 (1,182,032)	0.95 (738,224)	0.95 (516,320)

Table 5 When the embedding dimension is 8, the validation F-measure and the number of parameters according to the number of units of the Conv1D layer and the kernel size (expressed as F-measure (number of parameters))

Embedding dimension = 8		The number of units of Conv1D		
		64	32	16
kernel size	8	0.96 (777,520)	0.95 (394,128)	0.97 (202,432)
	16	0.95 (775,472)	0.94 (393,104)	0.95 (201,920)

Table 6 shows each algorithm vs. Benign firmware, SPN vs. ARX vs. Benign firmware, Crypto-ransomware vs. This is the result of performing an experiment on Benign firmware. As a result of analyzing the results of classification and detection, there was no misclassification of ARX and SPN structures, and there were cases of misclassifying SPECK and SPARX, and LEA and SPECK. In other words, SPN and ARX have different opcode patterns used in structure and have distinct characteristics. Also, in the case of crypto-ransomware detection, misclassification for SPECK and benign firmware occurred. As analyzed in previous work, the same cryptographic algorithms have very similar opcode patterns. Also, when belonging to the same structure, even different algorithms show similar opcode patterns, so we thought that such a result could be achieved.

Table 6 F-measure of Transformer Encoder-based Networks

Category	Training		Detection
	Validation F-Measure	Test F-Measure	F-measure
Each algorithm vs. Benign firmware	0.97	0.95	0.85
SPN vs. ARX vs. Benign firmware	1.00	0.98	0.98
Crypto-ransomware vs. Benign firmware	1.00	0.98	0.98

4.4 Comparison with previous work

Table 7 shows the file size and F-measure of the proposed method and the results of the previous method. Unlike the previous method, the proposed method in which the encoder structure of the transformer is applied by matching opcodes to words and opcodes appearing in a function to a sentence consisting of words does not include frequency information. But this approach can learn the relation and context of each opcode, and then a 0.01 higher F-measure was achieved. In addition, the file size of the convolutional fusion network to which pruning is applied is 1.96 times that of the transformer encoder-based network. Although the performance difference is not significant, the proposed method based on the transformer encoder structure is more effective in detecting cryptographic ransomware in a resource-constrained environment when the file size is compared.

Table 7 Comparison with previous work (File size and F-measure of detection phase)

Model		File Size	Detection F-measure
Convolution Fusion [10]	TFLite	1.74MB	0.97
	Pruned TFLite	1.64MB	0.97
Transformer encoder	TFLite	0.83MB	0.98

4.5 Comparison with other techniques

Table 8 is a table comparing the proposed method with other methods. For other studies, implementations of cryptographic algorithms such as OpenSSL and Cryptopp libraries were used. Since the proposed method is an 8-bit AVR environment, the source code implemented in C in FELICS was used.

Crypto-ransomware mainly uses AES and RSA algorithms. Therefore, these two algorithms must be detectable. Grobert et al. [7] have three approaches: chains, mnemonic-const, and verifier. Among them, the validator identification method has the best performance by checking the existence and parameters of the symmetric encryption process, and the validator method achieved a detection success rate of 0.946 in AES. However, the MD5 (Message-Digest) algorithm and RSA could not be detected, and both AES and RSA can be detected using the chains method. Caballero et al. [18] consider the key scheduling process, which is another factor that can identify the existence of the encryption process. The method can detect AES, Data Encryption Standard (DES), RC4, MD5 and RSA with a simple mechanism. The method proposed in this paper can handle more various encryption algorithms. Lightweight block cipher detection for low-end embedded processor environments was possible. Therefore, it can be used to detect cryptographic ransomware for IoT environments.

Table 8 Comparison with other method

	[7]	[17]	This Work
Algorithm	Heuristic	Field semantics inference	Deep learning
Implementation	OpenSSL, Cryptopp, Beecrypt	OpenSSL, Cryptopp, Beecrypt	FELICS
Description	AES, DES, RC4, and RSA	AES, DES, RC4, MD5, and RSA	See table 3

5. Conclusion

In this paper, we propose a new method to identify potential crypto-ransomware by classifying block cryptographic modules and normal firmware for embedded processors. For this, the lightweight cryptography implemented in C language among the implementations of FELICS was targeted, and the case where the encryption process was detected was classified as crypto-ransomware. A technique focused on reducing the size of the model in consideration of the resource-constrained environment while achieving a similar level of performance to the existing research results was proposed. After matching the opcode extracted from the binary file to the word constituting the sentence, it is converted into a vector through word embedding. It is trained after inputting the transformed data into the neural network based on the encoder structure of the transformer. By using this structure, the entire input data can be processed simultaneously instead of sequentially. In addition, sequences can be considered, and correlations and features between opcodes can be extracted and learned to classify successfully. In addition, through this proposed technique, a separate neural network considering frequency characteristics is no need to construct, and the size of the network was minimized by reducing the embedding dimension through experiments. This approach has a file size of 0.509 times that of the convolutional fusion network-based technique, and the F-measure achieved 0.98. It is 0.01 higher than previous work.

As future work, we will consider the key scheduling process, which is characteristic of another encryption process, and classification methods for other encryption modules, including public key encryption and hash functions. In addition, we plan to construct a more efficient neural network by utilizing various word embedding techniques.

References

[1] Mohurle, S.; Patil, M. A brief study of Wannacry threat: Ransomware attack 2017. International Journal of Advanced Research in Computer Science 2017, 8.

[2] Kharaz, A.; Arshad, S.; Mulliner, C.; Robertson, W.; Kirda, E. UNVEIL: A large-scale, automated approach to detecting ransomware. 25th USENIX Security Symposium (USENIX Security 16), 2016, pp. 757–772.

[3] Weckstén, M.; Frick, J.; Sjöström, A.; Järpe, E. A novel method for recovery from Crypto Ransomware infections. 2016 2nd IEEE International Conference on Computer and Communications (ICCC). IEEE, 2016, pp. 1354–1358.

[4] Tseng, A.; Chen, Y.; Kao, Y.; Lin, T. Deep learning for ransomware detection. IEICE Tech. Rep. 2016, 116, 87–92.

[5] Vinayakumar, R.; Soman, K.; Velan, K.S.; Ganorkar, S. Evaluating shallow and deep networks for ransomware detection and classification. 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE, 2017, pp. 259–265.

[6] Poudyal, S.; Dasgupta, D.; Akhtar, Z.; Gupta, K. A multi-level ransomware detection framework using natural language processing and machine learning. 14th International Conference on Malicious and Unwanted Software" MALCON, 2019.

[7] Gröbert, F.; Willems, C.; Holz, T. Automated identification of cryptographic primitives in binary programs. International Workshop on Recent Advances in Intrusion Detection. Springer, 2011, pp. 41–60.

[8] Lestringant, P.; Guihéry, F.; Fouque, P.A. Automated identification of cryptographic primitives in binary code with data flow graph isomorphism. Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, 2015, pp. 203–214.

[9] Kiraz, M.S.; Genç, Z.A.; Öztürk, E. Detecting large integer arithmetic for defense against crypto ransomware. Tech. Rep. 2017.

[10] Kim, H.; Park, J.; Kwon, H.; Jang, K.; Seo, H. Convolutional Neural Network-Based Cryptography Ransomware Detection for Low-End Embedded Processors. Mathematics 2021, 9, 705. https://doi.org/10.3390/math9070705

[11] Yaqoob, I.; Ahmed, E.; ur Rehman, M.H.; Ahmed, A.I.A.; Al-garadi, M.A.; Imran, M.; Guizani, M. The rise of ransomware and emerging security challenges in the Internet of Things. Computer Networks 2017, 129, 444–458.

[12] Azmoodeh, A.; Dehghantanha, A.; Conti, M.; Choo, K.K.R. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. Journal of Ambient Intelligence and Humanized Computing 2018, 9, 1141–1152.

[13] Azmoodeh, A.; Dehghantanha, A.; Choo, K.K.R. Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. IEEE Transactions on Sustainable Computing 2018, 4, 88–95

[14] Zahra, A.; Shah, M.A. IoT based ransomware growth rate evaluation and detection using command and 410 control blacklisting. 2017 23rd International Conference on Automation and Computing (ICAC). IEEE, 411 2017, pp. 1–6.

[15] Karimi, A.; Moattar, M.H. Android ransomware detection using reduced opcode sequence and image similarity. 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE). IEEE, 2017, pp. 229–234.

[16] Kumar, R.; Xiaosong, Z.; Khan, R.U.; Ahad, I.; Kumar, J. Malicious code detection based on image processing using deep learning. Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, 2018, pp. 81–85.

[17] Dinu, D.; Biryukov, A.; Großschädl, J.; Khovratovich, D.; Le Corre, Y.; Perrin, L. FELICS–fair evaluation of lightweight cryptographic systems. NIST Workshop on Lightweight Cryptography, 2015, Vol. 128.

[18] Caballero, J.; Poosankam, P.; Kreibich, C.; Song, D. Dispatcher: Enabling Active Botnet Infiltration Using Automatic Protocol Reverse-Engineering. Proceedings of the 16th ACM Conference on Computer and Communications Security; Association for Computing Machinery: New York, NY, USA, 2009; CCS '09, p. 621–634. doi:10.1145/1653662.1653737.