Received: 18 May 2020 | Revised: 5 November 2020 | Accepted: 25 November 2020

DOI: 10.4218/etrij.2020-0215

ORIGINAL ARTICLE

ETRI Journal WILEY

Evaluations of AI-based malicious PowerShell detection with feature optimizations

Jihyeon Song[1,2] | **Jungtae Kim**[2] | **Sunoh Choi**[3] | **Jonghyun Kim**[2] | **Ikkyun Kim**[2]

[1]ICT (Information Security Engineering), University of Science and Technology, Daejeon, Rep. of Korea

[2]Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea

[3]Department of Software Engineering, Jeonbuk National University, Jeonju, Rep. of Korea

Correspondence
Ikkyun Kim, Cyber Security Research Division, Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea.
Email: ikkim21@etri.re.kr

Funding information
This research was supported by the Institute for Information & Communications Technology Promotion (IITP) grant funded by the Korean government (MSIT) (no. 2019-0-00026, ICT infrastructure protection against intelligent malware threats).

Cyberattacks are often difficult to identify with traditional signature-based detection, because attackers continually find ways to bypass the detection methods. Therefore, researchers have introduced artificial intelligence (AI) technology for cybersecurity analysis to detect malicious PowerShell scripts. In this paper, we propose a feature optimization technique for AI-based approaches to enhance the accuracy of malicious PowerShell script detection. We statically analyze the PowerShell script and preprocess it with a method based on the tokens and abstract syntax tree (AST) for feature selection. Here, tokens and AST represent the vocabulary and structure of the PowerShell script, respectively. Performance evaluations with optimized features yield detection rates of 98% in both machine learning (ML) and deep learning (DL) experiments. Among them, the ML model with the 3-gram of selected five tokens and the DL model with experiments based on the AST 3-gram deliver the best performance.

KEYWORDS
Deep learning, feature optimization, fileless malware, machine learning, PowerShell script

1 | INTRODUCTION

Cyberattackers conduct criminal activities for various purposes against individual users, enterprises, and organizations worldwide. These cyberattacks are often difficult to identify with traditional signature-based detection because attackers continually find ways to bypass the detection methods [1,2].

In 2016, the amount of fileless malware, which is malware that does not exist in file systems, was determined to have rapidly increased [3]. Consequently, fileless attacks using a PowerShell script have been increasing. PowerShell is the scripting language and command-line shell provided by Microsoft [4] and is often used for system management and automation purposes because it provides powerful scripting capabilities. Attackers can also utilize the functional benefits of PowerShell scripts, for instance, performing malicious behavior and command operations using Windows Management Instrumentation or PowerShell, especially when it is installed on the victim's system [5].

Malware has been evolving over time, and unknown malware is difficult to detect as it may be combined with various types of existing malware or hide its operations from the system monitoring tools. Consequently, researchers have introduced artificial intelligence (AI) technology to perform intelligent detection of evolving malicious behavior [6,7] and have conducted studies to increase the performance of the model. Although in-depth research on PowerShell and the malware that exploits it has been scarce, recent studies [8,9]

regarding the analysis of malicious PowerShell scripts have been reported. However, these studies are limited in that they focused on the PowerShell obfuscation technique only.

The dataset used in this study contains the PowerShell script used by the Emotet malware that was distributed in december 2018. Emotet malware [10] was first identified in 2014 and still appears as a variant malware. Recently, it has been distributed in the form of a malicious document file attached to a phishing email that seems to convey information about COVID-19 infection status. The document file contains a PowerShell script for downloading the Emotet malware, and various techniques are used to hide the contents of these scripts.

According to IGLOO Security [11], attackers have developed obfuscation methods to minimize the source code exposure of their malicious PowerShell scripts and bypass antivirus products and other security solutions. Base64-encoded PowerShell scripts can be detected by behavior-based security solutions (such as endpoint detection and response). However, they are difficult to detect using existing pattern-based antivirus products. In addition, in the case of PowerShell, they can be more difficult to detect because multiple obfuscations are easy to implement.

Figure 1A shows a part of the malicious PowerShell script code containing the object linking and embedding (OLE) file used as the dataset in this paper. In Figure 1A, the capital letters are the Base64-encoded malicious part that the attacker intended to make difficult for the analyst to analyze. This string can be Base64 decoded to obtain the original code containing the shellcode that performs the malicious behavior. Figure 1B is a snippet of that code, and "$hI" is the shellcode that establishes a TCP connection to the attacker. Therefore, we first need to find and extract the PowerShell script included in the OLE file, and then decode the Base64-encoded

```
exec = "powershell -w 1 -C "sv S -;sv Dm ec;
sv uh ((gv S).value.toString()+(gv Dm).value.toString())
powershell (gv uh).value.toString()
'JABMAEUAIAA9ACAAJwAkAEYAYgAgAD0AIAAnACcAWwBEAGwAbABJAG0.
QBuAHQAUAB0AHIAIABWAGkAcgB0AHUAYQBsAEEAbABsAG8AYwAoAEkAb
0AGkAbwBuAFQAeQBwAGUALAAgAHUAaQBuAHQAIABmAGwAUAByAG8AdAB
GkAYwAgAGUAeAB0AGUAcgBuACAASQBuAHQAUAB0AHIAIABDAHIAZQBhA
AUwB0AGEAYwBrAFMAaQB6AGUALAAgAEkAbgB0AFAAdAByACAAbABwAFM
```

(A)

```
[Byte[]]$hI =
0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,0xe5,0x31,0xc0,
02,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,0x
,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0x01,0xd6,0x31,0xff,0xac
x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0
8,0x6e,0x65,0x74,0x00,0x68,0x77,0x69,0x6e,0x69,0x54,0x6
0x03,0x53,0x53,0x68,0xbb,0x01,0x00,0x00,0xe8,0xb0,0x00,
```

(B)

FIGURE 1 Malicious PowerShell script example: (A) PowerShell script containing the OLE file and (B) de-obfuscated shellcode

part in the script in order to detect the shellcode that performs the malicious behavior. This is a simple code sample that makes it possible to detect shellcode with a single Base64 decoding, but it can be difficult to detect malicious PowerShell scripts that are embedded within document files.

In this paper, we propose a feature optimization method to analyze PowerShell scripts statically to then determine the optimal feature combinations for AI-based malicious PowerShell detection. We evaluated the performance of our method by using our training dataset as input to three different machine learning (ML) and deep learning (DL) models and analyzing the performance of each model. The experimental results show that the random forest (RF) model with five token (5-token) 3-grams and the DL models with abstract syntax tree (AST) 3-grams have detection rates of approximately 98% and false-positive rates (FPRs) of $< 0.1\%$. In addition, the malicious script used in the Emotet malware described above is extracted, the Base64 is de-obfuscated, and the combination of features proposed in this paper is extracted. As a result of the experiment, it was confirmed that Emotet can be properly classified as malicious.

The remainder of this paper is organized as follows. Section 2 discusses previous research and related work. Section 3 introduces feature optimization techniques for malicious PowerShell script detection with feature extraction processes. Section 4 presents the structure of the pre-processing system and experimental AI models. Section 5 describes the feature extraction process in detail with the training datasets used for the evaluations. Then, the results of the performance evaluations of the ML and DL models are discussed in Section 6. Finally, Section 7 summarizes the proposed work with directions for future work.

2 | RELATED WORK

Prior to this study, we conducted a DL-based malicious PowerShell detection experiment with combinations of selected tokens [12]. Previously, 5-token types of the PowerShell script were selected to create a token combination for feature extraction. In our previous study, we utilized the 22 261 normal and 4150 malicious PowerShell scripts as a test dataset. Then, we evaluated the performance using 5-fold cross-validation with a convolutional neural network (CNN), long short-term memory (LSTM), and CNN-LSTM combined models. The experimental results of the three proposed models had an average detection rate of approximately 93% and an FPR of approximately 0.4%. However, we only conducted a validation experiment with a limited set of 4-tokens and did not conduct experiments with the other features for performance comparison. Therefore, our present study attempts to overcome the limitations of the previous study.

Hendler and others [13] used a natural language processing and character-level CNN-based detector with 60 098 normal and 6290 malicious PowerShell commands as datasets to detect malicious PowerShell commands. After analyzing the PowerShell commands with the deduplication process, they utilized a character set with a frequency of more than 1.4% of overall commands for the training feature information to exclude rare characters. In total, 62-length vector arrays were applied to the detector by including case-sensitive bit characters. They evaluated the performance using a variety of detectors, and as a result of 2-fold cross-validation, the 3-gram detector obtained a true-positive rate (TPR) of 0.98 when the FPR was at 1:1000. However, the limitation of the study was its narrow focus only on PowerShell commands without considering the entire script itself.

Rubin and others [14] proposed a contextual embedding method and DL to improve the overall performance. They attempted to overcome the lack of labeled data by learning the contextual embedding from the unlabeled data. With contextual embedding, words with similar meanings can be represented in a vector of embedding space; thus, malicious scripts can be detected with similarity matching. The authors performed training and evaluation by utilizing data collected from the Microsoft Antimalware Scan Interface (AMSI), which includes 111 593 normal and 5383 malicious instances of PowerShell code. They also included an unlabeled dataset of approximately 370 000 unlabeled PowerShell scripts and modules for the training data to simulate a realistic dataset. In this study, they conducted an experiment with a total of 12 models and performed 3-fold cross-validation. At an FPR level of 1:1000, the TPR of the CNN-W2V model was approximately 0.94. However, the contextual embedding was performed with no distinction between normal and malicious scripts, which may reduce the overall accuracy, instead of using a labeled script as training data.

Rusak and others [15] used the depth and node count information of the AST nodes to classify a malicious PowerShell script based on its family type information. They utilized 4079 malicious PowerShell scripts as a dataset. By recursively exploring the PowerShell script and extracting the depth and number of AST nodes, the hyperparameter of an RF classifier was optimized at the maximum node depth. Then, the PowerShell scripts were labeled according to each type of malware family by using a 3-fold cross-validation test with the RF classifier. In their experiment, they achieved 85% classification accuracy. In addition, the authors built an embedding matrix of the malicious PowerShell script according to its AST node types and performed experiments to show the relationship between similar embedding information. However, the classification of malicious types was limited to using the malicious PowerShell scripts only; consequently, the results were narrow in focus. In addition, although information about the depth and number of AST nodes facilitates the identification of the entire structure of the scripts, the distinction between a normal and malicious PowerShell script may not be clear.

TABLE 1 Token types: frequency and description

Token types	Count	Description
*Variable	1 757 455	Variable after the "$" character
Number	1 357 818	All numbers in the PowerShell scripts
String	770 950	All strings in the PowerShell scripts
* Member	762 072	Object properties and methods
* Command	577 446	Commands for the action to perform
* CommandParameter	474 777	Parameters used with Command
* Keyword	425 713	Condition, branch statement, etc.
* CommandArgument	367 884	Arguments used with Command

To overcome the limitations of previous studies, we analyzed the entire content and data structure of the PowerShell script, such as the AST node types (AST classes), which helps to consider various feature combinations from the detailed composition of the script content block and improve the detection accuracy.

3 | PROPOSED FEATURE OPTIMIZATION METHOD

This section describes the proposed method for PowerShell script analysis and the feature selection process of combining various features for performance enhancement.

3.1 | Pre-processing of PowerShell scripts

3.1.1 | Token-based keyword extraction

PowerShell scripts can be parsed by Microsoft-defined token units using PSParser Tokenize [16]. Microsoft classifies the tokens into 20 categories. Using the entire PowerShell script as the dataset, the frequency information of all related token types was extracted except for the tokens corresponding to operators, newlines, and parentheses. This dataset contains 22 261 normal scripts and 4214 malicious scripts, and the sources and types of script samples are detailed in Section 5.1. The pre-processing of the PowerShell script in this work mainly focused on the six types of tokens indicated by asterisks in Table 1. The Number and String tokens are

variables, and consequently, they are excluded to reduce the noise in the identification process to distinguish between normal and malicious scripts.

The token types for all types of behavior performed in PowerShell mostly consist of Command (Com), CommandParameter (Com_Param), and CommandArgument (Com_Arg), which correspond to execution commands with related variable and argument values, respectively. By analyzing these 3-token types, it is possible to determine the attacker's intention from the PowerShell script.

Keyword tokens include all tokens that are classified as keywords, including conditional or branch statements, which can be analyzed together with other tokens, such as Com tokens. For example, Keyword and Com tokens can be used together in the case of a command execution in PowerShell, especially when an attacker aims to perform the desired malicious behavior when a certain condition is met.

The Variable tokens contain all the variables created in the script. Variable names can be easily modified according to the user's needs. The number of variables can be very large when extracting Variable tokens because all of them are counted as different variables. However, this occurrence could be common when an attacker writes a script that references another script or when a variable name is used to clarify what a function is doing. In addition, when checking the condition of a specific variable in a conditional or branch statement, which frequently occurs in the analysis of a Keyword token, the syntax can be accurately assessed by checking a Variable token that appears later. Thus, it is logical to analyze the Keyword and Variable tokens together, regardless of their number.

According to Microsoft [17], objects play an important role in PowerShell; in particular, Member tokens represent the properties and methods of PowerShell objects. Although the features of this token type differ slightly from those of the other five types considered in this study, the Member token type is a crucial feature that represents the characteristics of PowerShell objects.

With the proposed six token types, the final issue to consider is that the PowerShell script is case insensitive when running commands. Certain malicious PowerShell scripts use a mixed case for script obfuscation. This characteristic makes it difficult for the user to read the contents, and the actual behavior is the same as when the script is represented in lowercase letters. Therefore, when analyzing the PowerShell scripts, the case-sensitivity issues of the extracted tokens can be neglected while processing the feature extraction and experiments.

3.1.2 | AST-based keyword extraction

The second method of parsing a PowerShell script is based on AST methods [18]. Although analysis using the Tokenize method can identify all the contents and lexical units of the script, analysis using the AST can identify the overall structure of the script by assessing its block units.

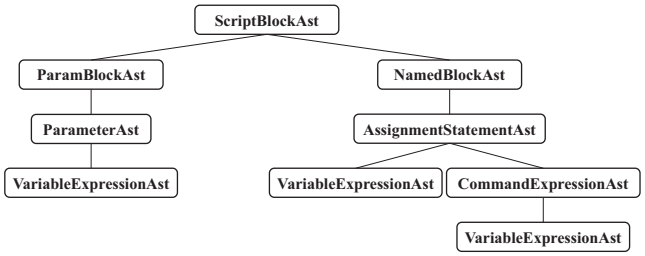

FIGURE 2 AST structure example

TABLE 2 Related groups of token types

Group	Token types
1	Command, CommandParameter, CommandArgument
2	Keyword, Variable
3	Member

Because normal (or benign) and malicious PowerShell scripts behave differently, their roles and overall structures differ. Therefore, we extracted the AST structure and selected it as a feature to distinguish between normal and malicious contents. Furthermore, the order of script construction can be analyzed with an n-grams approach to utilize the structural features more effectively. In this study, we selected the commonly used 3-gram approach to analyze three consecutive pieces of structural information together. As Microsoft has defined 108 AST classes [19], we analyzed all of the PowerShell scripts that were used as datasets and extracted all 108 classes accordingly. Figure 2 shows an example of an AST structure obtained by parsing a sample PowerShell script.

3.2 | Feature optimization

As explained in the previous section, the six proposed token types were used as the selected set of features in this study. We suggest three groups of related token types, as summarized in Table 2. For Group 1, the token types Com, Com_Param, and Com_Arg (Com, etc.) were selected, as they are related to each other while executing the commands. The Keyword and Variable tokens in Group 2 were selected, as they are often deployed together in conditional or branch statements of PowerShell. Lastly, the Member token is the only token type in Group 3, as it is a crucial component that represents the characteristics of an object, unlike other tokens.

Although the token types within the groups listed in Table 2 are interrelated, they may not be perfect combinations of token types for analyzing and detecting malicious PowerShell scripts. Groups 1 and 2 are not relevant in all

TABLE 3 Experimental results for token combinations

Token	Group 1	Group 1 and keyword	Group 1 and variable	5-tokens
Recall	90.0	90.3	84.0	90.6
FPR	4.3	5.0	4.3	4.0

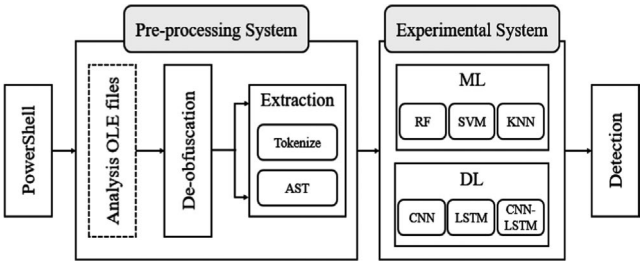

FIGURE 3 Structure of the detection system

situations, but can be used together to execute the desired commands under a certain condition. A combination such as this can be found especially in a malicious script with the aim of downloading malware from an attacker server (command) when the victim's PC is not a virtual machine (condition).

Consequently, five types of tokens exist in the combinations in Groups 1 and 2 that help to more accurately classify and detect malicious PowerShell scripts. The Members token in Group 3, as explained earlier, may not be significantly related to the other types of tokens. Therefore, we used a single Member token as a secondary feature instead of including it with the other token types. A simple experiment was conducted to determine the most appropriate combinations of the proposed token types. The results are provided in Table 3.

In this experiment, 1000 normal and malicious PowerShell script samples were randomly selected and used as a dataset to evaluate the effectiveness of the selected token combinations. Among them, 80% of the data were used for training, and the remaining 20% were used for testing purposes. The performance of each was measured with the CNN, LSTM, and CNN-LSTM combined models. Table 3 lists the averages of the experimental results based on the three models. The results show that the approach based on 5-tokens has the best performance overall, with a recall of 90.6% and an FPR of 4%.

4 | STRUCTURE OF THE DETECTION SYSTEM

Section 4 presents the overall structure of the proposed AI-based malicious PowerShell detection system used in this study. This system consists of data pre-processing and experimental system parts for malicious PowerShell detection and classification, as shown in Figure 3.

4.1 | Pre-processing system

The data pre-processing part of the detection system performs script analysis and generates training data. The PowerShell script is initially checked for the obfuscation methods used and then de-obfuscated to convert into plaintext script for static analysis. The de-obfuscated script is then further analyzed using the tokenize and AST methods to extract the different types of tokens and AST classes accordingly to generate the training data for the experimental AI models. Detailed descriptions of the pre-processing and procedures for the training data generation are provided in Sections 5.2 and 5.3, respectively.

4.2 | Experimental models

The experimental models employed in this study include three ML and three DL models. For the ML models, RF, support-vector machine (SVM), and K-nearest neighbor (K-NN) are used, and for the DL models, the CNN, LSTM, and CNN-LSTM models are used. The ML model was built using the Jupyter virtual environment in a Windows environment with the sklearn and Keras packages. The DL model was built using Keras, which is an open-source neural network library written in Python, on a Ubuntu server with a GPU (GeForceGTX 1080 Ti). For comparison with the results of the DL experiment, the length of the maximum sequence was set to 800 for both ML and DL. When the sequence length was insufficient, zero padding was added. The number of epochs and batch size were set to 1 for both ML and DL. The layer structures of the three DL models used in the experiments are depicted in Figure 4.

4.2.1 | RF

The RF model [20] is mainly used for classification and detection, and is an ensemble technique for learning multiple decision trees. The decision tree may have limitations for general use in certain fields because its overall performance tends to have a strong dependency on the given datasets. Therefore, RF was selected to overcome these problems. The RF may not be affected much by noise because the prediction of the decision tree is uncorrelated. Therefore, the generalization performance of RF is superior to that of the approach based on a decision tree.

4.2.2 | SVM

The SVM [21] model, also known as a support-vector network, is generally utilized for pattern recognition and data analysis for classification and regression. Based on the two types of datasets the SVM is provided with, the algorithm is optimized to determine the best matching type between two categories given

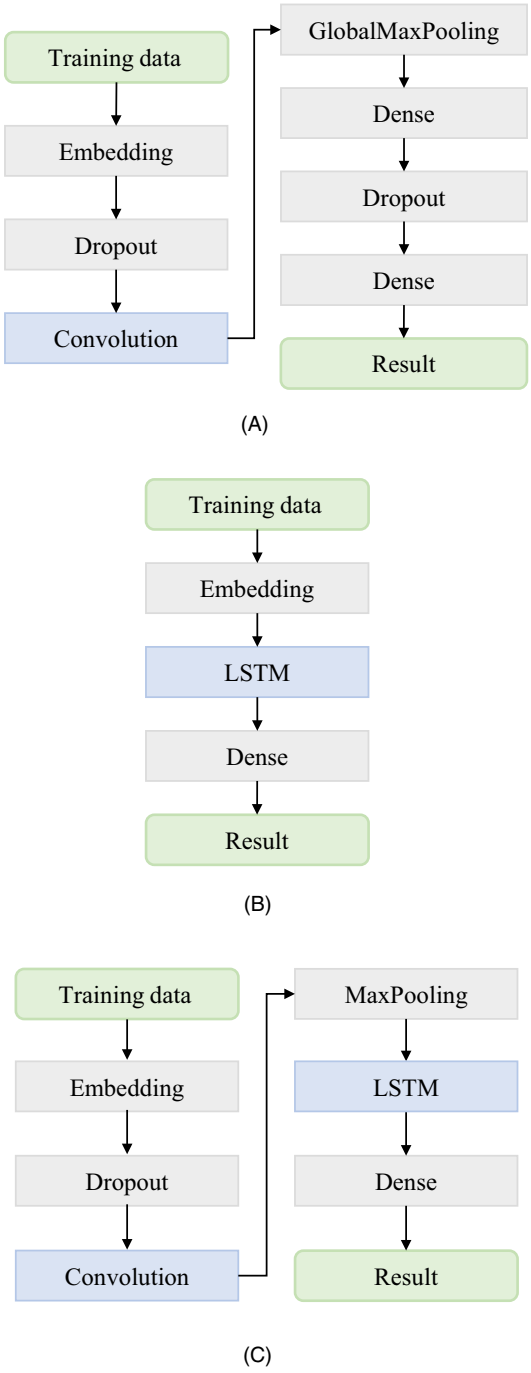

FIGURE 4 DL models: (A) CNN model, (B) LSTM model, and (C) CNN-LSTM model

unknown data. The SVM aims to maximize the margin, which is the distance between hyperplanes, for more accurate categorization of the different data types or separation of the datasets.

4.2.3 | K-NN

The K-NN algorithm [22] is a model designed for pattern recognition, especially for data classification and regression. The K-NN model with the previously labeled data helps to identify a similarity measure with the shortest distance from the k neighbor data when new data are available. The K-NN algorithm utilizes the Euclidean distance measurement for distance calculations. The disadvantage of the algorithm is the overall processing delay for data classification, especially when a large training dataset is used.

4.2.4 | CNN

The CNN model [23,24] is generally deployed to analyze and classify image, video, and text data, and is the most widely used algorithm for DL. CNN recognizes special patterns in the information consisting of input images or text, and extracts a meaningful feature set automatically. The fully connected neural networks are connected to all the neurons in adjacent layers. This neural network treats the input data as neurons of the same dimension and cannot use the spatial information of the data. Unlike a fully connected neural network, a CNN can utilize spatial information because it can maintain and process the shape information of the given data. In addition, the overall learning process requires less time than the recurrent neural network (RNN) model, which is described in the following section. The layered structure of the CNN model used in this study is shown in Figure 4A.

4.2.5 | LSTM

The RNN model [25] is specifically utilized to process sequential data types such as voice- and video-recorded format. The neural network does not support buffering of information obtained in the previous DL stage until the next stage. However, the RNN learns new data based on the information from the previous step, and this process is repeated for all new data. However, the RNN model experiences the vanishing gradient problem [26] when the previous step, in which the information was obtained, is far from the current step in which the information is used. The LSTM [27] model is designed to overcome these problems with a structural design that adds the cell state to the hidden state of the RNN, and consequently, it can maintain the intended information for a long time. The cell state of the LSTM can determine whether information is reflected in the data using an input, output, or forget gate. The layered structure of the LSTM model is shown in Figure 4B.

4.2.6 | CNN-LSTM

The CNN-LSTM model [28], as the name implies, combines the CNN and LSTM approaches. The combined model

utilizes the spatial information of the CNN and the temporal information of the LSTM together. Consequently, this approach has more flexibility for application in various DL fields than other models. In addition, the CNN-LSTM model sequentially integrates the local features extracted by the CNN into the LSTM. If only the CNN model was used to process the input data, it might not be able to identify the dependencies between the large datasets for a long period. Therefore, the problems associated with previous models can be solved by using the CNN and LSTM together. The layer configuration of the CNN-LSTM model can be found in Figure 4C.

5 | EXPERIMENTAL SETUP

5.1 | Datasets

In the experiment, a total of 26 475 PowerShell scripts were used, including 22 261 normal and 4214 malicious scripts. The PowerShell script samples contained approximately 3000 publicly available Base64-encoded scripts [29] and nearly 400 OLE files containing malicious PowerShell scripts provided by ESTsecurity [30]. The remainder of the dataset consists of a script that distinguishes between normal and malicious contents using VirusTotal [31] among the publicly available unlabeled PowerShell scripts [32]. The script labels were classified as malicious when a malicious PowerShell script was detected in 5% or more of the detection engines registered in VirusTotal.

5.2 | Pre-processing of datasets

The data used in the experiment can be broadly divided into Base64-encoded scripts or OLE files and regular scripts. Among them, the Base64-encoded scripts and OLE object files do not support static analysis as they are encoded; therefore, an additional pre-processing step is required before the proposed feature extraction process. We initially decoded the Base64-encoded PowerShell scripts and the analyzed OLE object files to find and extract the target PowerShell scripts and related features, as described in Section 3.1. Detailed information on the pre-processing is as follows.

5.2.1 | Analysis of the OLE files

The OLE allows a script to support embedding and linking to documents or objects. The provided OLE files contained malicious PowerShell scripts, and therefore, it was necessary to extract the embedded PowerShell scripts to utilize these files in the dataset. The Python-oletools package [33] is primarily used to analyze OLE and MS Office files for malware and debugging purposes. Several modules were provided as a software package for file analysis. For example, the MacroRaptor module enables the detection of malicious visual basic for applications (VBA) macros, and the olevba module provides a means of searching and extracting the VBA macro source code from the OLE and MS Office documents. In this study, we utilized the olevba module to find the malicious PowerShell scripts included in the OLE files to extract the embedded scripts.

5.2.2 | Simple de-obfuscation

An attacker can use a PowerShell script to perform any malicious behavior and even employ an obfuscation technique to avoid detection. In this study, because the features were extracted after static analysis without actually executing the script, the feature extraction would be difficult if the script was obfuscated. Therefore, a simple de-obfuscation process was performed before feature extraction. The PowerShell script can be de-obfuscated easily using Base64 decoding once or twice. In addition, a sophisticated obfuscation method utilized in the PowerShell scripts can also be de-obfuscated using publicly available de-obfuscation tools. The PowerShell options used to Base64 encode the scripts are "-enc," "-EncodedCommand," "-ec," and so on [29]. The pre-processing system searches for the PowerShell script encoded by the attacker using various options, after which the system decodes the script to obtain a plaintext script using the Base64 decoding module (b64decode).

5.3 | Feature extraction

After the de-obfuscation process, we can extract the feature information from the pre-processed plaintext PowerShell scripts. As explained in Section 3, we extracted the lexical unit token combinations and structural unit block of the script as feature information. At this time, the extracted feature strings were replaced by the unique corresponding IDs, and consequently, the same ID was assigned for each string within the script when the same string appeared in multiple scripts. In the case of the AST classes, 108 fixed classes can be extracted; thus, it was not necessary to convert them into lowercase. However, when using tokens as a feature, only the types of tokens to be extracted are specified and the total number is not fixed. As tokens (vocabulary) can have the same function but are case sensitive, all extracted tokens were converted into lowercase strings for deduplication purposes. The sequence information of the extracted tokens and AST classes could also be used as additional feature information by obtaining frequencies or reprocessing the sequence into n-gram sequences.

5.3.1 | Frequency

In malware analysis research, the frequency information of the malware opcodes is often extracted and used as training data [34]. In this study, because all six types of token and AST classes were used as the feature set, the frequency information about the proposed feature data, provided in Tables 1 and 2, could also be extracted and evaluated.

Because the total number of features extracted from the scripts was large, it was necessary to limit this number. Hence, we conducted a simple experiment by randomly assigning the top most frequently extracted token numbers to evaluate the overall performance of selection of the frequency information as a feature set. Among the top 100 to 5000 most frequently extracted feature sets, satisfactory performance was measured at the top frequency numbers of 500 and 1000. The 500 most frequently extracted tokens took less pre-processing time than the 1000 most frequently extracted tokens, but the performance was similar. Therefore, we extracted the top 500 frequently extracted tokens, and the corresponding results are listed in Table 4. In this experiment, the dataset we used was identical to that described in Section 5.1, and we measured the performance using the CNN model with 5-fold cross-validation tests using 5-token frequency without the Member token as learning data.

5.3.2 | 3-grams

As explained in Section 3.1.2, an analyst can identify the overall structure of the scripts by analyzing multiple AST structures together. Similarly, grouping-related token types can help researchers to determine the organization of the script content. In this study, a commonly used 3-gram sequence was utilized for the token and AST analysis; specifically, three consecutive pieces of structural information were analyzed together. If the extracted token and AST sequences were shorter than a 3-gram, zero padding was added to construct a complete 3-gram sequence for building the learning data.

5.4 | Learning the data configuration

In this study, the proposed feature optimization techniques with the six tokens and AST were utilized with the sequence, frequency, and 3-gram sequences to produce the learning data. Although several different types of features were used, the main objective was to arrange the extracted features in sequence for the learning data. The structures of the training data files applied in the ML and DL models are shown in Figure 5, which depicts part of a training data file consisting of a 5-token sequence extracted from the same normal PowerShell script.

TABLE 4 Experimental results for 5-token frequency

Frequency	100	200	500	1000	5000
Recall	62.2	69.2	93.2	92.3	0.7
FPR	4.1	3.3	8.3	7.3	0.02

The first column of the ML training data file in Figure 5A is labeled to distinguish benign (b) from malicious (m). The second column contains the corresponding sequence number of the extracted feature data. Therefore, Figure 5A can be interpreted as a list of feature sequences extracted from the normal PowerShell scripts.

On the other hand, the first column of the DL training data file in Figure 5B represents the file names of the PowerShell scripts, followed by the number of feature data extracted in the second column. The third column is a label (0 or 1) that can be used to distinguish a script as benign (0) or malicious (1), and, finally, the sequence of extracted feature data is represented in the remaining columns. For instance, Figure 5B shows a set of feature sequence data extracted from five normal PowerShell scripts.

6 | PERFORMANCE EVALUATION

This section presents the performance evaluation results obtained by applying the training data files to the ML and DL models. A total of 10 features were used in this study. Among them, nine of the features are mainly related to simple sequence information of token and AST, as listed in Table 5. We also experimented with the case-sensitive issue for the 5-token that were extracted to prove that it is meaningful to convert the token to lowercase when it is extracted.

In all six models, 5-fold cross-validation tests were performed, and the results of the detection and FPRs were rounded to two decimal places. In Table 5 and 6 , the top results of the ML and DL experiments are marked with an asterisk.

Benign1	1	2	3	1	1	4	5
Benign2	26	26	12	27	28	29	26
Benign3	1	1	29	30	29	12	31
Benign4	1	3	45	1			
Benign5	12	12	12	46	47	48	49

(A)

Normal1	28	0	1	2	3	1	1
Normal2	19	0	26	26	12	27	28
Normal3	7	0	1	1	29	30	29
Normal4	4	0	1	3	45	1	
Normal5	35	0	12	12	12	46	47

(B)

FIGURE 5 Data configuration: (A) ML data and (B) DL data

TABLE 5 Experimental results

	RF		SVM		K-NN		CNN		LSTM		CNN-LSTM	
Feature	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**	**Recall**	**FPR**
AST	88.5	0.1	84.8	0.05	87.4	1.0	89.0	0.60	88.1	1.3	85.8	1.20
AST 3-gram	89.5	0.1	86.0	0.05	87.7	0.6	* 98.5	* 0.08	* 98.0	* 0.01	* 98.4	* 0.06
AST frequency	94.2	0.2	86.5	0.60	91.1	0.5	79.5	0.60	77.3	1.3	79.0	1.50
5-token	* 95.6	* 0.3	86.3	0.10	89.7	0.6	93.8	0.30	* 94.9	* 0.5	91.1	0.40
5-token 3-gram	* 98.9	* 0.1	80.5	0.20	* 94.5	* 0.2	75.1	2.80	78.2	3.9	75.5	3.60
5-token frequency	* 96.1	* 2.2	70.0	0.20	85.1	1.2	93.2	8.30	34.6	1.4	18.2	0.40
Member	92.2	0.5	86.6	0.30	88.2	0.7	92.0	0.50	* 94.8	* 0.5	90.9	0.50
Member 3-gram	* 96.6	* 0.6	88.3	0.30	91.6	0.4	92.4	0.40	91.1	0.4	91.8	0.50
Member frequency	93.6	0.7	84.4	1.50	91.4	0.8	44.0	0.80	7.90	0	22.1	3.20

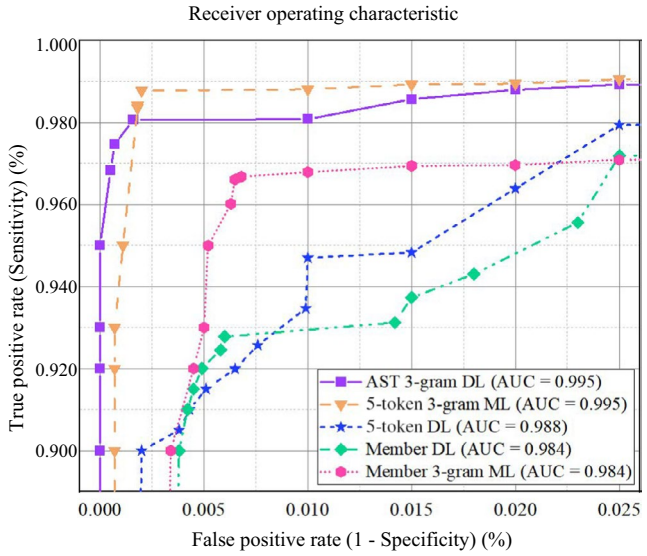

FIGURE 6 ROC curve

In the ML case, the best feature combination was the 5-token 3-gram case with the RF model, which shows a significantly higher detection rate and lower FPR than the others. Second, the 5-token frequency and 3-gram of Member tokens showed a detection rate of approximately 96% with a higher FPR. Third, the 5-token sequence also shows a detection rate of 95.6% and a FPR of 0.3%. Overall, the overall performance of the 5-token feature combinations in the ML models is satisfactory. Although the best results were obtained with the 5-token sequence cases in the ML, the AST 3-gram case had the highest detection rate of 98.5% for the DL. In particular, when the AST 3-gram was used, the detection rate was greater than 98% and the FPR was less than 0.1% in all of the DL models. The next highest detection rate was obtained using the 5-token and Member token sequences in the DL case. Direct comparison was difficult because of the different experimental models and features that were used. The results of the receiver operating characteristic (ROC) curve experiments [35] are summarized in Figure 6.

Overall, the detection rates in the ML experiments were higher than those in the DL experiment, but the FPR appears to be significantly lower in the DL case. As the trade-off between sensitivity and specificity among different experiments can be represented by the area under the ROC curve, which is equivalent to a measure of accuracy, the AST 3-gram case with the DL and 5-token 3-gram case with the ML models achieved the highest detection accuracy.

When AI technology is used for malware detection, a high detection rate is the main objective, but a low FPR is also very important. Therefore, the DL with AST 3-gram as a feature can be seen as the most effective feature optimization method for malicious PowerShell script detection.

As mentioned in Section 5.3, token extraction requires the extracted characters to be converted to lowercase. When writing scripts, there may be cases in which users intentionally mix both uppercase and lowercase characters. However, as indicated in Table 6, the case-sensitivity issues of PowerShell scripts can be neglected at the functional system level. To prove this proposal, we prepared a case-sensitive 5-token sequence and a sequence in which all tokens were converted to lowercase to conduct the experiment. In the same way as the previous experiment, we performed 5-fold cross-validation on six models and compared the results. We argue that the performance is superior for lowercase 5-token than when the token is case-sensitive. Therefore, we propose converting all extracted feature information into lowercase characters to reduce the number of duplicated tokens and to optimize the features for detecting malicious scripts based on similar malicious behavior.

Table 7 compares the studies surveyed in Section 2. Direct comparison of the experimental results is difficult because the datasets, detection models, extracted features, and experimental methods used in each study are different, but they can be indirectly compared based on the detection performance presented in the study. In the study of Rusak

and colleagues, a malicious PowerShell script family classification experiment was conducted, and 85% classification performance was obtained by 3-fold cross-validation, but the subject of the experiment is different and thus cannot be compared. According to Table 7, the experimental results with the proposed feature optimization methods show a similar detection rate, but 10 times lower FPRs compared with related studies on malicious PowerShell script detection. Experimental results show that the features and optimization methods used are appropriate for detecting malicious PowerShell scripts.

7 | CONCLUSION AND FUTURE WORK

The number of cybercrimes is constantly increasing, and these acts are often difficult to identify via traditional signature-based detection. As cyberattackers continue to find ways to bypass detection methods, the amount of research on this topic has been increasing in recent years. However, recent studies merely focused on PowerShell obfuscation techniques and ways to bypass them. In addition, studies on the use of AI to select appropriate features for learning and to increase the detection rate are scarce.

Therefore, in this study, we evaluated feature optimization and related performance for malicious PowerShell detection using various AI technologies. We utilized combinations of the extracted token and sequences of the AST as learning features for analyzing the PowerShell scripts statically. There are various types of tokens used in the PowerShell script, but in this study, we extracted token types that appear frequently and even tokens that can act as noise for optimal feature combinations. Using all of these features, we conducted a comparative experiment. The combination of features proposed in this paper has the best detection performance among them. According to the experimental results, the feature combinations proposed in this study achieved a detection rate of more than 98% with both the ML and DL models. Among them, the 5-token 3-gram and AST 3-gram cases showed the best results in the ML and DL models, respectively. PowerShell can be downloaded and used in macOS, Linux, and Windows. In this paper, we proposed an optimized combination of features for detecting malicious PowerShell scripts. We extracted script content and structure information through static analysis without running scripts to organize feature combinations. Therefore, regardless of the environment in which the script runs, if the script is collected, statically analyzed, and the combination of the features we propose is extracted, the detection system can use it as a dataset.

PowerShell scripts are primarily used as tools to download additional malware or execute commands to steal the victim's personal information to the C&C server. Therefore, it can be very rare that a script contains user personal information. In the operation of the detection system, there are two prerequisites in order to become a situation in which concerns related to personal information may arise. First, the personal information must be included inside the PowerShell script, and second, the information contained in the script must correspond to the five types of tokens used as features in this paper. If the user using the detection system is concerned about personal information that may be contained in the PowerShell script, then only the AST feature information can be used. In the case of AST, it does not extract the contents of the script, but only extracts the entire structure information of the script with a fixed number. Therefore, there is no problem related to privacy protection, and the detection performance is good.

Several approaches could be followed to expand upon this research in the future. First, the feature combinations

TABLE 6 Results of 5-token case-sensitive experiments

		5-token lowercase	5-token case-sensitive
RF	Recall	* 95.6	93.40
	FPR	* 0.3	0.20
SVM	Recall	86.3	74.80
	FPR	0.1	0.06
K-NN	Recall	89.7	88.50
	FPR	0.6	0.90
CNN	Recall	93.8	91.50
	FPR	0.3	0.40
LSTM	Recall	* 94.9	90.90
	FPR	* 0.5	0.60
CNN-LSTM	Recall	91.1	90.60
	FPR	0.4	0.50

TABLE 7 Comparison with related work. AUC: Area under the ROC curve

Comparison	ML	DL	[13]	[14]
Data set				
Nor	22 261	22 261	60 098	111 593
Mal	4214	4214	6290	5383
Unlabel	—	—	—	368K
Features	5-Token 3-gram	AST 3-gram	Character frequency	Token embedding
Algorithm	RF	LSTM	3-gram	CNN-w2v
N-fold	5	5	2	3
Recall	0.989	0.980	0.980	0.944
FPR	1:1000	1:10 000	1:1000	1:1000
AUC	0.995	0.995	0.990	0.994

could be optimized further by measuring the importance of features or utilizing an attention mechanism [36]. Second, as other researchers are interested in PowerShell script obfuscation technology, an advanced pre-processing technique could also be applied to analyze and extract the features of scripts with more complicated obfuscations than the Base64 decoding technique. Third, the proposed PowerShell detection system can be divided into two separate parts: the pre-processing and experimental systems. Combining them into a single automated system would help to achieve effective and efficient processing capabilities in terms of the overall system performance. Finally, the detection model could be hardened to be robust against adversarial examples. In this paper, we focus on proposing a feature optimization method for detecting malicious PowerShell scripts. However, recently, research is actively being conducted to create adversarial samples that make them look like malware or non-malware using algorithms such as generative adversarial networks [37]. Therefore, it is not currently considered, but in the future, research to implement a robust detection model for evasive PowerShell scripts can be conducted by generating adversarial samples for PowerShell scripts and retraining them.

ORCID


Jihyeon Song https://orcid.org/0000-0002-3318-506X

REFERENCES

1. A. Osipov, Trickbot trojan leveraging a new windows 10 uac bypass, Jan. 2020, available at https://blog.morphisec.com/trickbot-uses-a-new-windows-10-uac-bypass.
2. A. Katrenko, Malware sandbox evasion: Techniques, principles & solutions, Mar. 2020, available at https://www.apriorit.com/dev-blog/545-sandbox-evading-malware.
3. Symantec, The increased use of powershell in attacks, 2016, available at https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en.pdf.
4. Microsoft, What is powershell?, May 2020, available at https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.
5. A. Mellen, Fileless malware 101: Understanding non-malware attacks, Sept. 2019, available at https://www.cybereason.com/blog/fileless-malware.
6. M. Kim, *Supervised learning-based ddos attacks detection: Tuning hyperparameters*, ETRI J. **41** (2019), no. 5, 560–573.
7. I. Ko, D. Chambers, and E. Barrett, *Unsupervised learning with hierarchical feature selection for DDos mitigation within the isp domain*, ETRI J. **41** (2019), no. 5, 574–584.
8. D. Bohannon and L. Holmes, Revoke-obfuscation: Powershell obfuscation detection using science, 2017, available at https://www.blackhat.com/docs/us-17/thursday/us-17-Bohannon-Revoke-Obfuscation-PowerShell-Obfuscation-Detection-And%20Evasion-Using-Science.pdf.
9. D. Bohannon, Invoke-obfuscation v1.8.2, 2018, available at https://github.com/danielbohannon/Invoke-Obfuscation.
10. Trendmicro, Emotet uses coronavirus scare in latest campaign, targets japan, 2020, available at https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/emotet-uses-coronavirus-scare-in-latest-campaign-targets-japan.
11. IGLOO security, Monthly security report, 2020, available at http://www.igloosec.co.kr/pdf/igloosec_security_report_202002_en.pdf.
12. J. H. Song et al., *Implementation of a static powershell analysis based on the cnn-lstm model with token optimizations*, in Proc. World Conf. Inform. Secur. Appl. (Jeju, Rep. of Korea), Aug. 2019, pp. 99–107.
13. D. Hendler, S. Kels, and A. Rubin, *Detecting malicious powershell commands using deep nerual networks*, in Proc. Asia Conf. Comput. Commun. Secur. (Incheon, Rep. of Korea), June 2018, pp. 187–197.
14. A. Rubin, S. Kels, and D. Hendler, *Amsi-based detection of malicious powershell code using contextual embeddings*, available at arXiv preprint CoRR, 2019 arXiv: 1905.09538v2.
15. G. Rusak, A. Al-Dujaili, and U. M. O'Reilly, *Ast-based deep learning for detecting malicious powershell*, in Proc. Conf. Comput. Commun. Secur. (Toronto, Canada), Oct. 2018, pp. 2276–2278.
16. Microsoft, Pstokentype enum, 2019, available at https://docs.microsoft.com/en-us/dotnet/api/system.management.automation.pstokentype?view=pscore-6.2.0.
17. S. Wheeler, Viewing object structure (get-member), 2017, available at https://docs.microsoft.com/en-us/powershell/scripting/samples/viewing-object-structure--get-member-?view=powershell-7.
18. P. Singh, Powershell: Tokenization and abstract syntax tree, 2017, available at https://geekeefy.wordpress.com/2017/06/07/powershell-tokenization-and-abstract-syntax-tree.
19. Microsoft, System.management.automation.language.namespace, 2019, available at https://docs.microsoft.com/en-us/dotnet/api/system.management.automation.language?view=pscore-6.2.0.
20. T. Yiu, Understanding random forest, 2019, available at https://towardsdatascience.com/understanding-random-forest-58381e0602d2.
21. R. Gandhi, Support vector machine–introduction to machine learning algorithms, 2018, available at https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.
22. O. Harrison, Machine learning basics with the k-nearest neighbors algorithm, 2018, available at https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.
23. T. Y. Kim, Text input binary classification model recipe, 2017, available at https://tykimos.github.io/2017/08/17/Text_Input_Binary_Classification_Model_Recipe.
24. Y. Kim, *Convolutional neural networks for sentence classification*, in Proc. Conf. Empir. Methods Nat. Language Process. (Doha, Qatar), Oct. 2014, pp. 1746–1751.
25. A. Amidi and S. Amidi. Recurrent neural networks cheatsheet, 2019, available at https://stanford.edu/shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.
26. C. F. Wang, The vanishing gradient problem, Jan. 2019, available at https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.
27. C. Olah, Understanding lstm networks, 2015, available at https://colah.github.io/posts/2015-08-Understanding-LSTMs.
28. J. Wang et al., *Dimensional sentiment analysis using a regional cnn-lstm model*, in Proc. 54th Annu. Meet. Assoc. Comput. Linguist. (Berlin, Germany), Aug. 2016, pp. 225–230.

29. J. White, Pulling back the curtains on encodedcommand powershell attacks, 2017, available at https://unit42.paloaltonetworks.com/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks.
30. ESTsecurity, available at https://www.estsecurity.com.
31. Virustotal, available at https://www.virustotal.com.
32. Powershell corpus, available at https://aka.ms/PowerShellCorpus.
33. P. Legadec, oletools-python tools to analyze ole and ms office files, 2018, available at https://www.decalage.info/python/oletools.
34. I. Santos et al., *Idea: Opcode-sequence-based malware detection*, in Proc. Int. Symp. Secur. Eng. Softw. Syst. (Pisa, Italy), Feb. 2010, pp. 35–43.
35. S. Narkhede, Understanding auc-roc curve, 2018, available at https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.
36. A. Vaswani et al., *Attention is all you need*, in Proc. Adv. Neural Inform. Process. Syst. (Long Beach, CA, USA), 2017, 5998–6008.
37. L. Tong et al., *Improving robustness of ML classifiers against realizable evasion attacks using conserved features*, in Proc. 28th USENIX Secur. Symp. (USENIX Security 19), (Santa Clara, CA, USA), Aug. 2019, pp. 285–302.

AUTHOR BIOGRAPHIES

Jihyeon Song is a PhD student at the University of Science and Technology (UST), Daejeon, Rep. of Korea. She received her BS degree in information security engineering from Soonchunhyang University, Asan, Rep. of Korea, in 2018, and her MS degree in Information Security Engineering from UST in 2020. Her research interests include cybersecurity, malware analysis, and AI-based security.

Jungtae Kim received his BE degree in information technology from Charles Sturt University, Australia, in 2001, and his MS degree in network computing from Monash University, Australia, in 2004. He joined the Electronics and Telecommunications Research Institute (ETRI) in 2004. He is currently working in the field of network security analysis technologies as a member of the engineering staff of the Network Security Research Laboratory. His current research interests include malware analysis, advanced cyberattacks, and behavior-based dynamic security analysis technology.

Sunoh Choi Sunoh Choi has been an assistant professor in the Department of Computer Engineering at Honam University since 2019. He received his BS and MS degrees from Korea University and his PhD degree from Purdue University, respectively. He held employment with the ETRI from 2014 to 2019. His research interests include data security and AI-based security.

Jonghyun Kim received his MS and PhD degrees in computer science from the University of Oklahoma, USA, in 2000 and 2005, respectively. He was a researcher with Samsung Electronics from 1995 to 1997, and a system consultant with Samsung SDS in 2000. He is currently a principal researcher with the ETRI, Daejeon, Rep. of Korea, where he is also the project leader of the Intelligence Security Group. He is involved in standardization activities as an associate rapporteur of Q.4 (cybersecurity) with ITU SG17. His research interests include information security, cybersecurity, network management, AI-based Security Information and Event Management, and AI-based malware detection.

Ikkyun Kim is an assistant vice president of the Cyber Security Research Division at the ETRI, Daejeon, Rep. of Korea. He joined the ETRI in 1996. He received his MSCE and PhD degrees in computer engineering from Kyungpook National University, Daegu, Rep. of Korea, in 1996 and 2009, respectively. His research interests include network security, network forensics, cloud security, and hardware security.