electronics

MDPI

Article

Malicious PowerShell Detection Using Attention against Adversarial Attacks

Sunoh Choi

Department of Computer Engineering, Honam University, Gwangju 62399, Korea; suno@honam.ac.kr

Received: 14 September 2020; Accepted: 30 October 2020; Published: 2 November 2020

Abstract: Currently, hundreds of thousands of new malicious files are created daily. Existing pattern-based antivirus solutions face difficulties in detecting such files. In addition, malicious PowerShell files are currently being used for fileless attacks. To prevent these problems, artificial intelligence-based detection methods have been suggested. However, methods that use a generative adversarial network (GAN) to avoid AI-based detection have been proposed recently. Attacks that use such methods are called adversarial attacks. In this study, we propose an attention-based filtering method to prevent adversarial attacks. Using the attention-based filtering method, we can obtain restored PowerShell data from fake PowerShell data generated by GAN. First, we show that the detection rate of the fake PowerShell data generated by GAN in an existing malware detector is 0%. Subsequently, we show that the detection rate of the restored PowerShell data generated by attention-based filtering is 96.5%.

Keywords: Malicious PowerShell detection; adversarial attack; GAN

1. Introduction

Every day, hundreds of thousands of new malicious files are generated, and there are currently about 1 billion malicious files in circulation [1]. To detect these malicious files, malware analysts are required to provide new malicious file patterns to existing pattern-based antivirus solutions. Thus, most existing pattern-based antivirus solutions have difficulties in detecting new malicious files [2]. To solve these problems, artificial intelligence (AI)-based malicious file detection methods have been proposed [3–11].

Recently, malicious PowerShell files have been observed [12,13]. Table 1 compares AI-based detection methods. PowerShell, a scripting language used to manage Windows systems, has powerful functions; however, it is being used in malicious endeavors. An example of a malicious PowerShell is as follows: The dataset used in this study contains PowerShell scripts that were used in the Emotet malware that was distributed in December 2018. Emotet malware [14] 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 the status of coronavirus infection. The document file contains a PowerShell script for downloading Emotet malware, and various techniques such as obfuscation are used to hide the contents of these scripts.

AI-based malware detection involves two steps. The first step is to extract the feature data from malicious files. We can perform static analysis [3,4] or dynamic analysis [5–7] to extract feature data. In malicious PowerShell data, we perform static analysis using PSParser [15]. The second step is to train a deep learning model using training data and test the deep learning model using test data. We can use the convolutional neural network (CNN)-based model, the long short-term memory (LSTM)-based model, and the CNN–LSTM combined model for AI-based malware detection [9–11].

Table 1. Comparison of AI-based Detection.

Reference	File Type	Feature Extraction	Malware Detection	Year
[3]	PE	Static	DNN	2015
[4]	PE	Static	CNN	2016
[5]	PE	Dynamic	DNN	2013
[6]	PE	Dynamic	RNN	2015
[7]	PE	Dynamic	DNN	2016
[8]	PE	Dynamic	LCS	2015
[9]	PE	Static	SC-LSTM	2018
[10]	PE	Static	Attention	2020
[11]	PE	Static	TLSH + LSTM	2020
[12]	PowerShell	Static	CNN + LSTM	2019
[13]	PowerShell	Static	CNN	2018

However, methods to avoid these deep learning-based malware detections have been proposed nowadays [16–18]. Fake data similar to normal data are generated from malicious data using GAN. The GAN has a generator and a discriminator. The generator generates fake data similar to normal data from malicious data, and the discriminator is trained to distinguish fake data from normal data. By repeating this process, an attacker generates fake data similar to normal data from malicious data. Therefore, the fake data is determined as normal in the existing AI-based detector. This is called an adversarial attack.

In this study, we propose an attention-based filtering method to prevent adversarial attacks using GAN, as shown in Figure 1. Attention [19] is a variant of the LSTM model, which computes the weights of each token in the input data based on the output data. The attention-based filtering method is as follows: First, we compute the weights of each token in the training data based on the output data using attention and generate a malicious token list containing tokens with top k weights in each input data. Second, when fake data generated by GAN are provided, we generate restored PowerShell data using the attention-based filtering method. The fake data were similar to the normal data. However, the restored data become similar to malicious data if they are originally malicious because the tokens in the restored data are in the malicious token list. Hence, we prevent adversarial attacks using the attention-based filtering method.

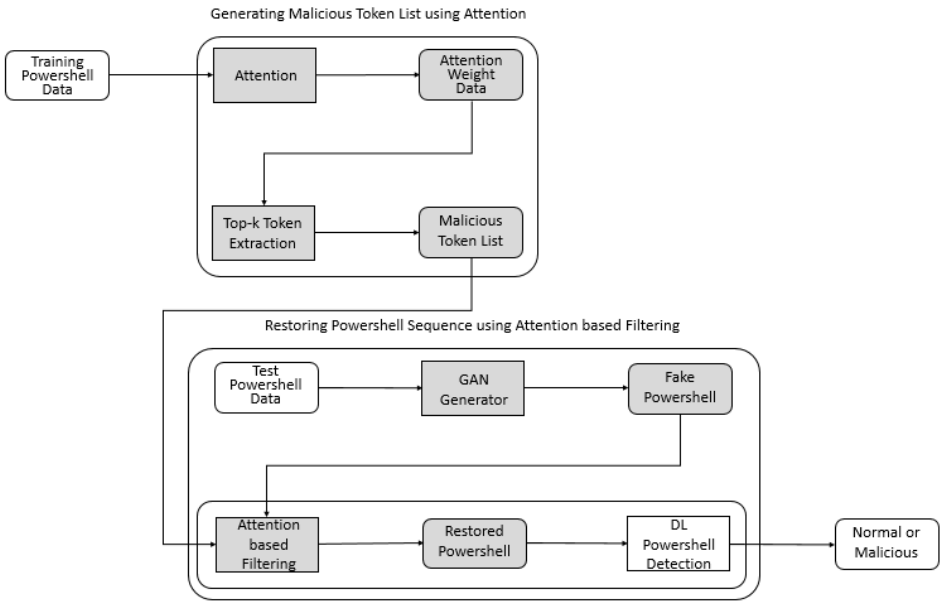

Figure 1. Malicious PowerShell Detection System Using Attention against Adversarial Attacks.

This study makes the following contributions: First, we show that the detection rate of malicious PowerShell files was 93.5% in deep learning-based malicious PowerShell detection. Second, we generated fake PowerShell data using a GAN generator and showed that its detection rate was reduced to 0%. Third, we generated restored PowerShell data using the attention-based filtering method and showed that its detection rate increased to 96.5%. Thus, we verify that we prevent adversarial attacks using the attention-based filtering method.

The remainder of this paper is organized as follows. In Section 2, we introduce related work. In Section 3, we present a malicious PowerShell detection deep learning model and provide adversarial attacks using GAN for malicious PowerShell. In Section 4, we introduce the attention mechanism and propose an attention-based filtering method to prevent adversarial attacks. In Section 5, we present the experimental results. Finally, in Section 6, we conclude with a discussion.

2. Related Work

Malicious file analysis was performed using static and dynamic analyses. A static analysis analyzes strings, import tables, byte n-grams, and opcodes [20]. However, the analysis is difficult if files are obfuscated or packed [21]. Dynamic analysis is used to analyze files by running them. Recently, AI-based malware detection has been widely used. In Saxe et al. [3] and Gibert [4], feature data were extracted using static analysis, and a deep learning model was used to determine whether the files were malicious. In Dahl et al. [5], Pascanu et al. [6], Huang et al. [7], and Ki et al. [8], feature data were extracted using dynamic analysis. In Dahl et al. [5], a deep neural network-based deep learning model was used. In Pascanu et al. [6], a recurrent neural network was used. In Huang et al. [7], the authors proposed a deep learning model that simultaneously performs detection and classification. In Ki et al. [8], API system calls were extracted, and the longest common subsequence (LCS) was used.

In addition, deep learning-based malicious PowerShell detection methods have been proposed. In Song et al. [12], five tokens of the PowerShell script were selected to create a token combination for feature extraction. We evaluated the performance using the CNN, LSTM, and CNN–LSTM combined models. Hendler et al. [13] used natural language processing and a character-level CNN-based detector with malicious PowerShell commands to detect malicious PowerShell commands. They focus only on the PowerShell commands without considering the entire script.

The GAN was proposed by Goodfellow. In Goodfellow et al. [16], adding some small perturbations to the original data makes a discriminator unable to classify them correctly. Grosse et al. [17] show that fixed-dimensional feature-based malware detection is vulnerable under adversarial attacks. In Hu et al. [18], to generate adversarial examples from API sequences, they consider adding other APIs to the original sequences. In this study, we use the same method to generate fake PowerShell data and verify that malicious PowerShell data are also not detected under adversarial attacks using GAN. In addition, we propose an attention-based filtering method to prevent adversarial attacks.

Methods for preventing adversarial attacks have been previously proposed [22–24]. In Goodfellow et al. [22], adversarial training was used to augment the training data with adversarial examples. However, if the attackers use other attack models, adversarial training does not work well. In Papernot et al. [23], defensive distillation was used to train the classifier using distillation. However, it does not prevent black box attacks. In Samangouei et al. [24], the Defense-GAN generator was used to obtain restored sequences.

3. Adversarial Attacks on Deep Learning-Based Malicious PowerShell Detection

In this section, we introduce a deep learning-based malicious PowerShell detection method and adversarial attacks using GAN.

3.1. Deep Learning-Based Malicious PowerShell Detection

Deep learning-based malicious PowerShell detection involves two steps, as presented in Figure 2. The first step is to extract feature data from PowerShell files, and the second step is to train a deep learning model using training data and test it using test data to detect malicious PowerShell files.

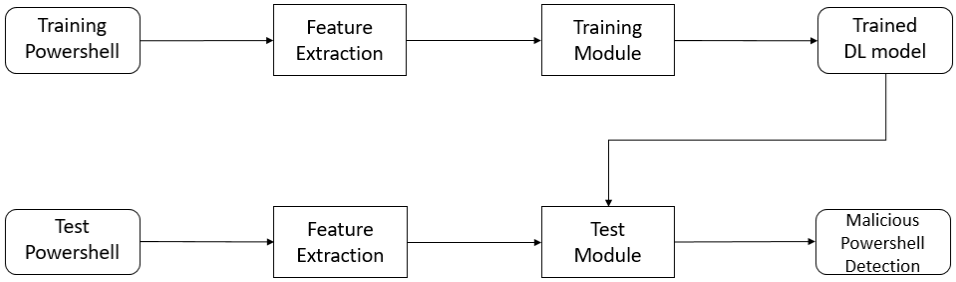

Figure 2. Deep Learning-based Malicious PowerShell Detection System.

The PowerShell feature data are extracted using the Tokenize method in the PSParser class [15]. PowerShells have 20 tokens, as follows:

{Attribute, Command, CommandArgument, CommandParameter, Comment, GroupEnd, GroupStart, Keyword, LineContinuation, LoopLabel, Member, NewLine, Number, Operator, Position, StatementSeparator, String, Type, Unknown, Variable}

Among the 20 token types, we use 6 token types as feature data [12].

{Command, CommandArgument, CommandParameter, Keyword, Member, Variable}

We can extract the PowerShell sequence data from a PowerShell file using PSParser as follows:

$$(x_1, x_2, \ldots, x_n)$$

Next, we train a CNN-based deep learning model [25] to detect malicious PowerShells using the PowerShell sequence data extracted from the training data. Thereafter, we tested the CNN-based deep learning model using test data. Thus, we determine whether the PowerShell file is malicious.

3.2. Adversarial Attack

In this section, we introduce a method to attack a deep learning-based malicious PowerShell detection system. As shown in Figure 3, we generate fake PowerShell data from malicious PowerShell data using GAN.

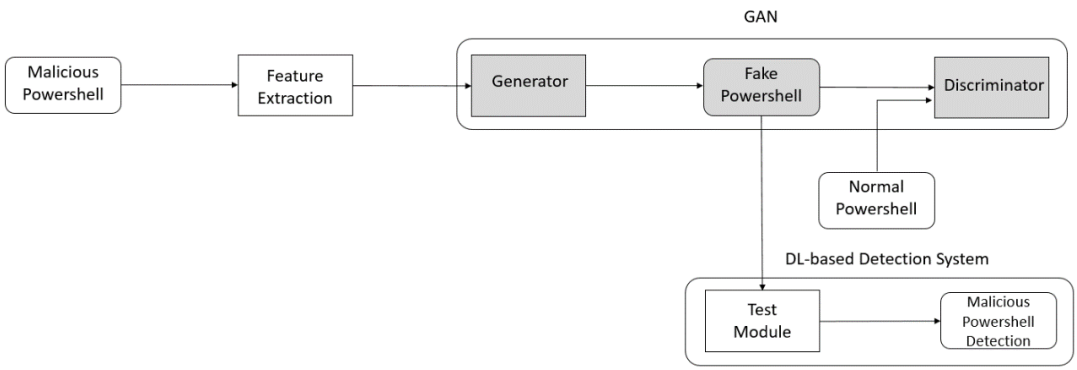

Figure 3. Adversarial Attack about Deep Learning-based PowerShell Detection.

The GAN has a generator and a discriminator [16]. The GAN generator generates fake PowerShell data, and the GAN discriminator is trained with the fake PowerShell data and normal PowerShell data. In addition, the GAN generator is trained with the result of training the GAN discriminator. By repeating this process, the generator generates fake PowerShell data similar to the normal PowerShell data from the malicious PowerShell data. Finally, the deep learning-based detection system determines that the fake PowerShell data generated from malicious PowerShells using GAN is not malicious.

The GAN is used in various domains. In art, a picture similar to trained pictures, such as Van Gogh, was generated [26]. In music, a song similar to trained songs, such as Beethoven, was generated [27]. In malware detection, a fake malware similar to normal files was generated [18].

Fake PowerShell data are generated as follows. Suppose that a malicious PowerShell data sequence is given.

$$(x_1, x_2, \ldots, x_n)$$

Using GAN, we generate a fake PowerShell sequence similar to a normal PowerShell sequence as follows.

$$\left(x'_1, x'_2, \ldots, x'_n\right)$$

However, we should not replace the original PowerShell token with another token because we should ensure that the original PowerShell tokens are contained in the fake PowerShell sequence data and malicious behaviors occur [18]. Therefore, we generate fake PowerShell sequence data as follows, ensuring that the original tokens are contained.

$$(g_{1,1}, \ldots, g_{1,L}, x_1, g_{2,1}, \ldots, g_{2,L}, x_2, \ldots, g_{n,1}, \ldots, g_{n,L}, x_n)$$

New tokens $g_{i,j}$ are added to the PowerShell sequence. The repeated training of the GAN model generates a fake PowerShell sequence that is determined not to be malicious by the deep learning-based detection system.

Note that L is the length of the new tokens, $g_{i,j}$ and is added to the PowerShell sequence. L can be random. However, for simplicity, we make L constant in Section 5.2.

4. Malicious PowerShell Detection Using Attention against Adversarial Attacks

In this section, we introduce an attention mechanism [19] and propose a malicious PowerShell detection method that uses attention against adversarial attacks.

4.1. Attention

Attention is a variant of the LSTM model [28]. We trained the sequence data using the LSTM model. In addition, using the LSTM model, we translate a language and make a chatbot [29]. Attention is a deep learning model used to find a part of the input that has a greater impact on the output. Using attention, we compute the weight $a_{t,i}$ of each token x_i of input based on the output y_t, as shown in Figure 4.

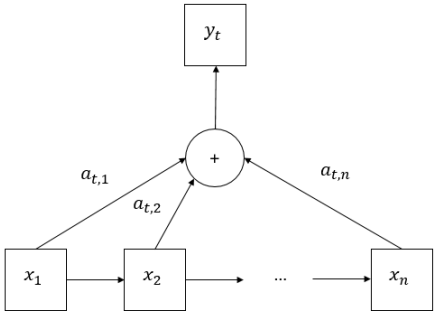

Figure 4. Attention.

Note that in the LSTM model, the output can be as follows.

$$y_1, y_2, \ldots, y_t$$

However, in malware detection systems, the output is only whether or not the input is malicious. Therefore, in malware detection systems, t is set to 1.

When the weight of a token is large, the token is important. Attention is used in text summarization [19]. For example, consider a sentence as follows:

> Russian defense minister Ivanov, called Sunday for the creation of a joint front to combat global terrorism

It can be summarized using attention as follows.

> Russia called for a joint front for terrorism

The RNN model is used in neural machine translation (NMT) [29]. For example, a German sentence is translated into an English sentence. NMT encodes a source sentence to a vector and decodes an output sentence based on the vector. Attention allows the decoder to refer to each part of the input sentence based on the output sentence. In Figure 4, x is a source sentence and y is an output sentence. An output word y_t depends on the combination of the weights $a_{t,i}$ of input words x_i. $a_{t,i}$ is a weight that shows how large each input word impacts the output word. For example, when $a_{3,2}$ is large, the third word in the output sentence refers to the second word in the input sentence.

4.2. Malicious PowerShell Detection Using Attention

In this section, we propose a malicious PowerShell detection method that uses attention against adversarial attacks. It has two steps. The first step is to generate a malicious token list using attention from the PowerShell training data, as shown in Figure 5. The second step is to generate the restored PowerShell data from the fake PowerShell data using the malicious token list.

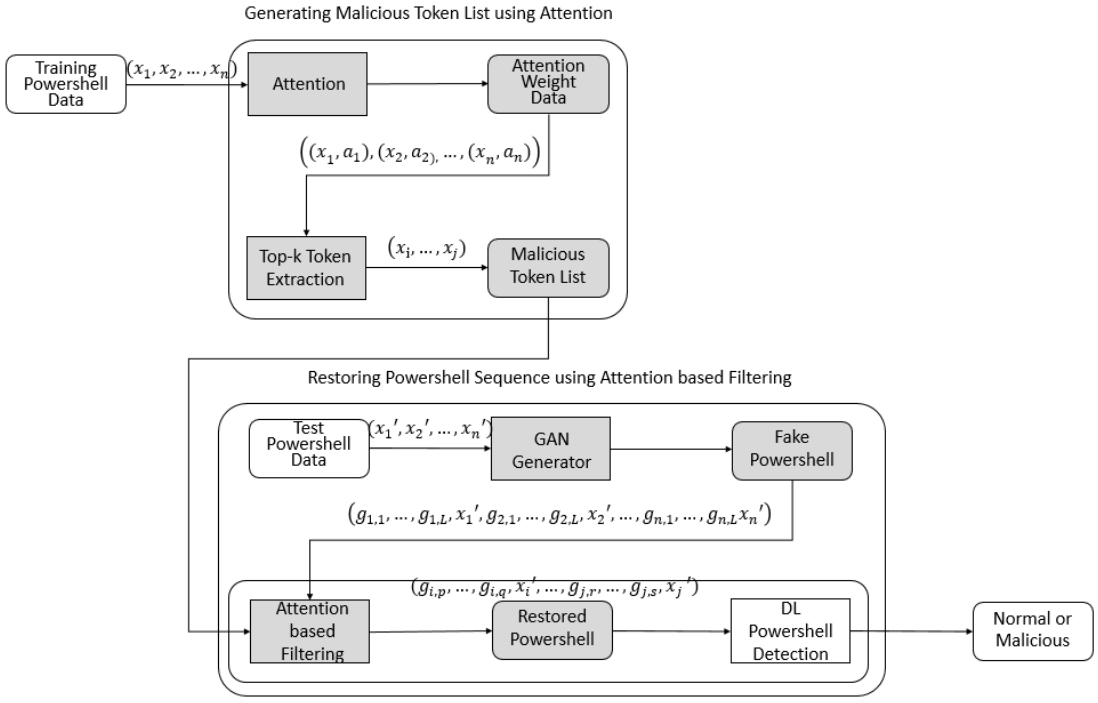

Figure 5. Malicious Token List Generation and Attention-based Filtering.

In the first step, we first suppose that the following PowerShell data sequence is given.

$$(x_1, x_2, \ldots, x_n)$$

Second, we compute the weights a_i of each token x_i in the PowerShell sequence data using attention.

Third, we find the k-th largest weight a_j in each PowerShell sequence and add tokens whose weight is larger than a_j to a malicious token list if the PowerShell sequence is malicious. In contrast, if the PowerShell sequence is normal, we add tokens whose weight is larger than a_j to a normal token list. Thus, we generate two token lists as follows:

{Normal_token_list, Malicious_token_list}

The intersection of the two token lists is a common token list. Using the two token lists, we generate three token lists as follows:

{Normal_only_token_list, Malicious_only_token_list, Common_token_list}

In the second step, we perform attention-based filtering using the malicious_only_token_list. We suppose that a fake PowerShell sequence generated by an adversarial attack is given as follows:

$$\left(g_{1,1}, \ldots, g_{1,L}, x'_1, g_{2,1}, \ldots, g_{2,L}, x'_2, \ldots, g_{n,1}, \ldots, g_{n,L}, x'_n\right)$$

First, we generate a restored PowerShell sequence containing tokens that exist in the malicious_only_token_list from the fake PowerShell sequence as follows:

$$\left(g_{i,p}, \ldots, g_{i,q}, x'_i, \ldots, g_{j,r}, \ldots, g_{j,s}, x'_j\right)$$

Second, we determine whether the restored PowerShell sequence is malicious using the existing deep learning-based malicious PowerShell detection system.

The advantages of the attention-based filtering method are analyzed as follows. We define the difference, $diff_{fake}$, between the fake PowerShell sequence generated by GAN and the original malicious PowerShell sequence.

$$diff_{fake} = \sum_{a=1}^{n/(L+1)} \left|x'_a - x_a\right| + \sum_{b=1}^{n/(L+1)} \sum_{c=1}^{L} \left|g_{b,c} - x_{b,c}\right|$$

On the other hand, the difference, $diff_{restored}$, between the original malicious PowerShell sequence and the restored PowerShell sequence generated by the attention-based filtering method is as follows.

$$diff_{restored} = \sum_{a=i}^{j} \left|x'_a - x_a\right| + \sum_{b=u}^{v} \sum_{c=p}^{q} \left|g_{b,c} - x_{b,c}\right|$$

Because $1 \le i \le j \le n/(\mathrm{L}+1)$ and $1 \le u \le v \le n/(\mathrm{L}+1)$, and $1 \le p \le q \le L$, the following condition is always satisfied.

$$diff_{restored} \le diff_{fake}$$

Therefore, we conclude that the attention-based filtering method reduces the difference between the fake PowerShell sequence generated by GAN and the original malicious PowerShell sequence. This means that even if the fake PowerShell sequence is determined as normal in the existing malware detector, the restored PowerShell sequence generated by attention-based filtering is determined as malicious.

5. Experimental Results

In this section, we present the experimental results. First, in Section 5.1, we introduce the experiment environment. Second, in Section 5.2, we present the performance metric. Third, in Section 5.3,

we describe the experimental result of an adversarial attack. Finally, in Section 5.4, we show the experimental results of malicious PowerShell detection using attention-based filtering against adversarial attacks.

5.1. Setup

We used 1000 normal PowerShell data files and 1000 malicious PowerShell data files provided by the Information Security Research Division of Electronics and Telecommunications Research Institute (ETRI) [30]. We generated PowerShell sequence data by extracting six types of PowerShell tokens from each PowerShell file, as shown in Figure 6. We set the length of the PowerShell sequence to 800. We used 5-fold cross validation [31]. Thus, 80% of the data were used for training, and 20% of the data were used for testing. We used 800 normal PowerShell data and 800 malicious PowerShell data for training, and 200 normal PowerShell data files and 200 malicious PowerShell data files for testing. We performed the experiments five times by changing the training data and test data.

```
1 2 2 2 3 2 4 5 6 7 1 2 3 8 9 10 11 3 12 8 8 8 13 14 15
70 71 72 73 33 74 75 75 76 0 0 0 0 0 0 0 0 0 0 0 0 0 0 C
77 78 79 80 81 82 80 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 C
35 83 84 85 86 87 85 87 86 88 89 90 90 90 0 0 0 0 0 0 0
91 92 91 92 93 94 95 96 93 94 95 96 97 96 98 97 96 98 0
99 100 101 102 103 9 18 103 99 100 103 101 103 102 104 1
115 116 117 36 118 98 119 98 120 121 122 123 0 0 0 0 0 C
108 48 109 124 116 48 125 126 127 0 0 0 0 0 0 0 0 0 0 0
108 48 109 128 48 129 48 130 130 131 108 48 109 128 48 1
140 141 142 143 144 145 146 147 148 77 147 33 149 148 14
```

Figure 6. PowerShell Sequence Data.

We performed two experiments. In the first experiment, we performed adversarial attacks against deep learning-based malicious PowerShell detection by generating fake PowerShell data using GAN introduced in Section 3.2. In the second experiment, we detected the restored PowerShell sequence data using attention-based filtering from the fake PowerShell data proposed in Section 4.2.

These two experiments were performed on a Windows 10 system. We implemented the GAN and attention-based filtering method using Keras [32]. The detailed experimental conditions are listed in Table 2.

Table 2. Experimental Environment.

Name	Specification
OS	Windows 10 Pro
CPU	Intel i7 2.2 GHz
RAM	16 GB
GPU	GeForce RTX 2060
Cuda	8.0

5.2. Performance Metric

In this section, performance evaluation indicators are described before presenting the experimental results. The indicators used in this study are accuracy, precision, recall (detection rate), F1 score, and false positive rate (FPR). The confusion matrix used to calculate these values is presented in Table 3.

True positive (TP) indicates that a file has been correctly evaluated by the system as malicious, and true negative (TN) indicates that the system has correctly determined that a benign file is normal. Furthermore, false positive (FP) indicates that a normal file has been incorrectly assessed by the system

as malicious, and false negative (FN) indicates that the system incorrectly identified a malicious file as normal. Each indicator is calculated as follows:

$$\text{Accuracy} = (TP + TN)/(TP + FP + FN + TN)$$

$$\text{Precision} = TP/(TP + FP)$$

$$\text{Recall (Detection Rate)} = TP/(TP + FN)$$

$$\text{F1 score} = 2 \times \text{Precision} \times \text{Recall}/(\text{Precision} + \text{Recall})$$

$$\text{FPR} = FP/(FP + TN)$$

Table 3. Confusion Matrix.

-	Malware	Normal File
Predicted Malware	TP	FP
Predicted Normal File	FN	TN

5.3. Adversarial Attack

First, we trained a malicious PowerShell detection deep learning model with 800 normal PowerShell data and 800 malicious PowerShell data. Second, we generated a fake PowerShell data sequence using the GAN generator. As mentioned in Section 3.2, we varied the value of L from 0 to 4. When L was set to 0, the fake PowerShell sequence was the same as the original PowerShell sequence.

As shown in Figure 7, when L was set to 0, the detection rate of 200 malicious PowerShell data was 93.5%. When L was set to 1, the detection rate was 93%. However, when L was set to 2, the detection rate was 53.5%, and when L was 3, it was 36%. Finally, when L was 4, it decreased to 0%. Through this experiment, we verified that an adversarial attack on malicious PowerShell data is possible using GAN.

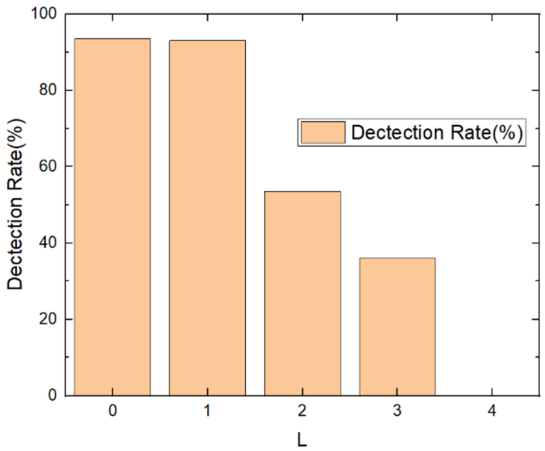

Figure 7. Adversarial Attack Results.

Subsequently, we measured the fake PowerShell data generation time using the GAN as shown in Figure 8. We varied the number of training PowerShell data from 400 to 1600 by increments of 400. Epoch was set to 100. Fake PowerShell generation time includes the time to detect 200 test PowerShell data in each epoch. When the number of training data was 400, the fake PowerShell generation time was 438 s, and when the number was 1600, the generation time was 720 s. We think that the fake PowerShell data generation time is reasonable.

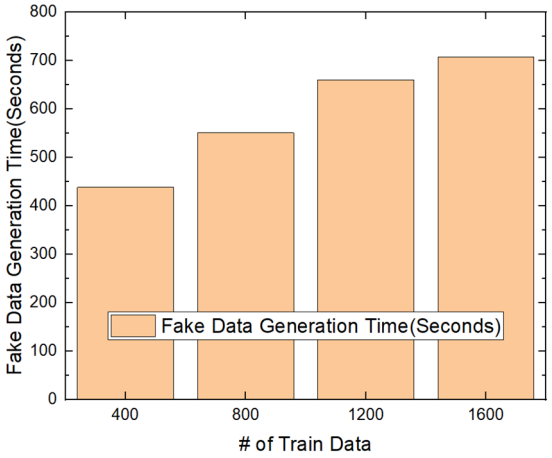

Figure 8. Fake PowerShell Data Generation Time.

5.4. Malicious PowerShell Detection Using Attention against Adversarial Attack

In the second experiment, we generated restored PowerShell sequence data using attention-based filtering from 200 fake PowerShell data that were generated by GAN, and we measured the detection rate of the restored PowerShell sequence data in the existing malicious PowerShell detection system, as indicated in Figure 9. As stated in Section 4.2, we varied the value of k from 1 to 5. Note that we computed the weights of each token in the PowerShell sequence data using attention. Then, we found the k-th largest weight in each PowerShell sequence and added tokens whose weights were larger than the k-th largest weight to a malicious token list.

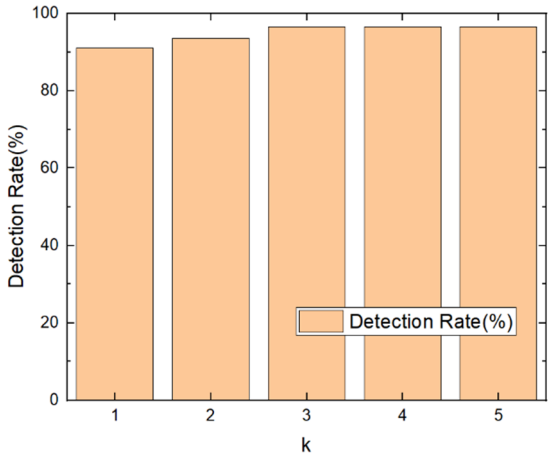

Figure 9. Attention Filtering Results.

When L was 4, the detection rate of the fake PowerShell data was 0%. However, when k was set to 1, the detection rate of the restored PowerShell data was 91%, and when k was set to 2, the detection rate was 93%. When k was 3, 4, or 5, the detection rate increased to 96.5%. This was higher than the original detection rate of 93.5%. We show that the attention-based filtering method improves the detection rate of the existing malicious PowerShell detection system and prevents adversarial attacks.

Next, we measured the attention-based filtering time, as shown in Figure 10. When there were 50 fake PowerShell data, the attention-based filtering time was 131 ms, and when there were 200 fake PowerShell data, the filtering time was 452 ms. It is approximate 2.5 ms per fake PowerShell data on average. We think that the attention-based filtering time against adversarial attacks is reasonable [33].

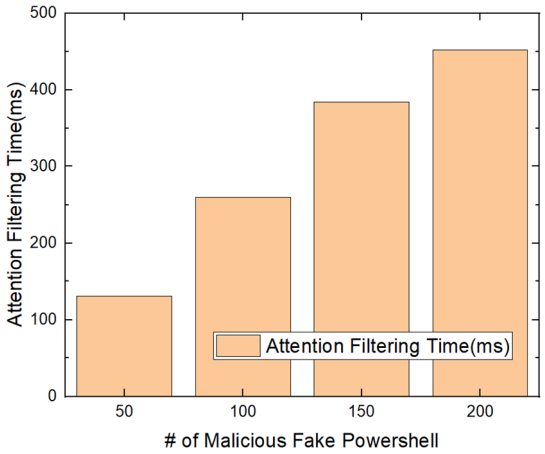

Figure 10. Attention Filtering Time.

Next, we measured the false positive rate (FPR) as shown in Figure 11. We used 200 normal PowerShell data and performed attention-based filtering and measured the FPR in the existing deep learning-based detection model. When k was 1 or 2, the FPR was 1.5%. However, when k was 3, it increased to 32%. When k was 4, it decreased to 7.5%, and when k was 5, it was 3.5%. When k increases, the length of the malicious token list increases. We discover that FPR depends on the length of the malicious_only_token_list.

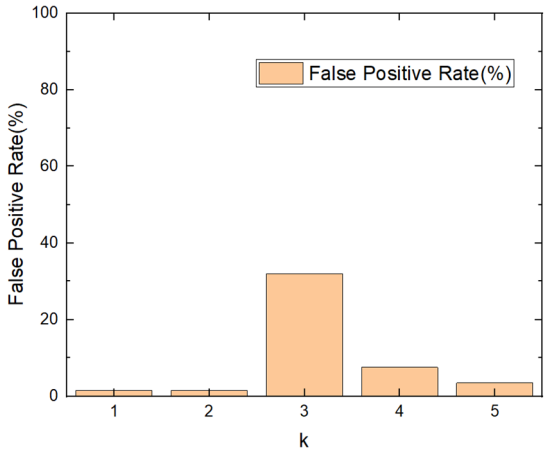

Figure 11. False Positive Rate.

Table 4 shows the size of the malicious only token list according to k. As k increases, the size of the malicious only token list increases. This means that when the size of the list was 93, the attention-based filtering method extracted tokens among the 93 malicious tokens only from the fake PowerShell sequence. Generally, if the malicious only token list is longer, the FPR decreases. However, in some cases (e.g., k = 3), some restored normal PowerShell sequences in the attention-based filtering can be determined as malicious. Only a few tokens are not enough for the restored normal PowerShell sequence to be determined as normal.

Table 4. Size of malicious only token list.

	Top-1	**Top-2**	**Top-3**	**Top-4**	**Top-5**
Size of Malicious Only Token List	93	165	183	213	233

Finally, we compared the attention-based filtering method with adversarial training [22]. Adversarial training trains the fake PowerShell sequence data generated by the GAN. When we trained 800 normal PowerShell data and 800 malicious PowerShell data, we additionally added fake PowerShell data to the training data from 200 to 800. As shown in Figure 12, the detection rate of attention-based filtering was slightly lower than that of adversarial training. However, as shown in Figure 13, the false positive rate of attention-based filtering was significantly lower than that of adversarial training. Therefore, we think that attention-based filtering is better than adversarial training.

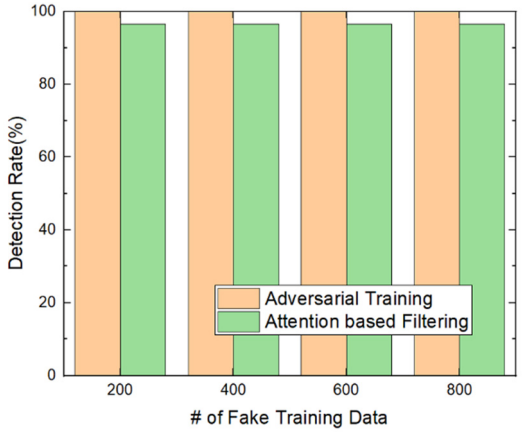

Figure 12. Detection Rate Comparison.

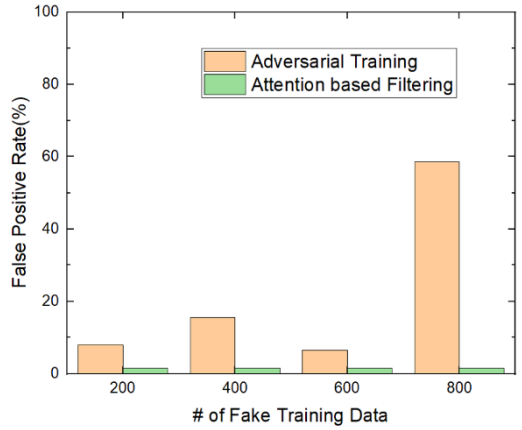

Figure 13. False Positive Rate Comparison.

6. Discussion

In this study, we generated a fake PowerShell data sequence using GAN and showed that its detection rate decreased to 0% when using an existing detection method. Then, we first generated a malicious only token list using attention. Second, we generated a restored PowerShell data sequence using attention-based filtering and verified that its detection rate increased to 96.5%. We showed that adversarial attacks are prevented using attention-based filtering.

In contrast, research has been conducted to generate adversarial attacks against intrusion detection systems [34]. We think that attention-based filtering is also useful to prevent adversarial attacks against intrusion detection. In future work, we will research a method to prevent adversarial attacks against intrusion detection systems.

Funding: This work was supported by a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No.2019R1G1A11100261) and was supported by the HPC support project by MSIT and NIPA.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. AV-TEST. Available online: https://www.av-test.org (accessed on 9 September 2020).
2. Gavrilut, D.; Cimpoesu, M.; Anton, D.; Ciortuz, L. Malware Detection Using Machine Learning. In Proceedings of the International Multiconference on Computer Science and Information Technology, Mragowo, Poland, 12–14 October 2009.
3. Saxe, J.; Berlin, K. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. In Proceedings of the International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015.
4. Gibert, D. Convolutional Neural Networks for Malware Classification. Master's Thesis, Universitat de Barcelona, Barcelona, Spain, 2016.
5. Dahl, G.E.; Stokes, J.W.; Deng, L.; Yu, D. Large-Scale Malware Classification Using Random Projections and Neural Networks. In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, BC, Canada, 26–31 May 2013.
6. Pascanu, R.; Stokes, J.W.; Sanossian, H.; Marinescu, M.; Thomas, A. Malware Classification With Recurrent Networks. In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, QLD, Australia, 19–24 April 2015.
7. Huang, W.; Stokes, J.W. MtNet: A Multi-Task Neural Networks for Dynamic Malware Classification. In Proceedings of the International Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), San Sebastian, Spain, 7–8 July 2016.
8. Ki, Y.; Kim, E.; Kim, H.K. A Novel approach to detect malware based on API call sequence analysis. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 659101. [CrossRef]
9. Bae, J.; Lee, C.; Choi, S.; Kim, J. Malware Detection model with skip-connected LSTM RNN. *J. Korean Inst. Inf. Sci. Eng.* **2018**, *45*, 1233–1239. [CrossRef]
10. Choi, S.; Bae, J.; Lee, C.; Kim, Y.; Kim, J. Attention-based automated feature extraction for malware analysis. *Sensors* **2020**, *20*, 2893. [CrossRef] [PubMed]
11. Choi, S. Combined kNN Classification and hierarchical similarity hash for fast malware detection. *Appl. Sci.* **2020**, *10*, 5173. [CrossRef]
12. Song, J.; Kim, J.; Choi, S.; Kim, J.; Kim, I. Implementation of a Static Powershell Analysis Based on the Cnn-Lstm Model With Token Optimizations. In Proceedings of the WISA Workshop, Jeju, Korea, 21–24 August 2019.
13. Hendler, D.; Kels, S.; Rubin, A. Detecting Malicious Powershell Commands Using Deep Neural Networks. In Proceedings of the ACM ASIACCS, Incheon, Korea, 4–8 June 2018.
14. Trendmicro. Emotet Uses Coronavirus Scare in Latest Campaign, Targets Japan. Available online: http://trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-thrests (accessed on 9 September 2020).
15. Tokenizing PowerShell Scripts. Available online: http://powershell.one/powershell-internals/parsing-and-tokenization/simple-tokenizer (accessed on 8 September 2020).
16. Goodfellow, I.J.; Abadie, J.P.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative Adversarial Nets. In Proceedings of the 28th Annual Conference on Neural Information Processing Systems NIPS, Montreal, QC, Canada, 8–13 December 2014.
17. Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; McDaniel, P. Adversarial Examples for Malware Detection. In Proceedings of the 22nd European Symposium on Research in Computer Security ESORICS, Oslo, Norway, 11–13 September 2017.
18. Hu, W.; Tan, Y. Black-box attacks against RNN Based Malware Detection Algorithms. In Proceedings of the Workshops of the Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018.
19. Rush, A.M.; Harvard, S.E.A.S.; Chopra, S.; Weston, J. A neural Attention Model for Sentence Summarization. In Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP, Lisbon, Portugal, 17–21 September 2015.
20. Kendall, K.; McMillan, C. *Practical Malware Analysis*; BlackHat: Las Vegas, NV, USA, 2007.
21. Moser, A.; Kruegel, C.; Kirda, E. Limits of Static Analysis for Malware Detection. In Proceedings of the 23rd IEEE International Conference on Computer Security and Applications, Miami Beach, FL, USA, 10–14 December 2007; pp. 421–430.

22. Goodfellow, I.J.; Shlens, J.; Szegedy, C. Explaining and Harnessing Adversarial Examples. In Proceedings of the 3rd International Conference on Learning Representations ICLR, San Diego, CA, USA, 7–9 May 2015.
23. Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; Swami, A. Distillation As a Defense to Adversarial Perturbations Against Deep Neural Networks. In Proceedings of the IEEE Symposium on Security and Privacy Workshop, San Jose, CA, USA, 23–25 May 2016.
24. Samangouei, P.; Kabkab, M.; Chellappa, R. DEFENSE-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. In Proceedings of the 6th International Conference on Learning Representations ICLR, Vancouver, BC, Canada, 30 April–3 May 2018.
25. Krizhevsky, A.; Sutskever, I.; Hinton, G. ImageNet Classification With Deep Convolutional Neural Networks. In Proceedings of the International Conference on Neural Information Processing Systems, Lake Tahoe, CA, USA, 3–6 December 2012.
26. Jones, K. GANGogh: Creating Art with GANS. Available online: http://towardsdatascience.com/gangogh-creating-art-with-gans-8d087d8f74a1 (accessed on 9 September 2020).
27. Engel, J. GANSynth: Making Music with GANS. Available online: http://magenta.tensorflow.org/gansynth (accessed on 1 November 2020).
28. Understanding LSTM Networks. Available online: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed on 27 July 2020).
29. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In Proceedings of the 6th International Conference on Learning Representations, San Diego, CA, USA, 7–9 May 2015.
30. Information Security Research Division, Electronics and Telecommunications Research Institute (ETRI). Available online: http://etri.re.kr (accessed on 9 September 2020).
31. N-fold Cross Validation. Available online: https://en.wikipedia.org/wiki/Cross-validation_(statistics) (accessed on 9 September 2020).
32. Keras. Available online: http://keras.io (accessed on 9 September 2020).
33. Antivirus Performance Comparisons. Available online: http://sharedit.co.kr/posts/424 (accessed on 5 October 2020).
34. Lin, Z.; Xue, Z.; Shi, Y. IDSGAN: Generative adversarial networks for attack generation against intrusion detection. *arXiv* **2018**, arXiv:1809.02077.