Evasion attempt for the malicious PowerShell detector considering feature weights

Kou Sugiura[1] and Mamoru Mimura[1][0000−0003−4323−9911]

National Defense Academy of Japan, Yokosuka, Japan
[em62004,mim]@nda.ac.jp

Abstract. In recent years, dependence on digital technology has increased, raising the risk of cyber attacks. Particularly, PowerShell, known for its high convenience, has been exploited by many attackers. Prior research has proposed methods using natural language processing and machine learning to detect malicious PowerShell. On the other hand, there have been reports of the potential for evasion attacks against detectors using machine learning or neural networks. Nevertheless, the assessment of evasion attacks against malicious PowerShell detectors remains inadequate. Particularly, there have been no reported evaluations of evasion attacks targeted at detectors utilizing neural networks. In this study, we examined the feasibility of evasion attacks on models designed for detecting malicious PowerShell using neural networks. We utilized words with high attention weights extracted using the Attention mechanism for benign features. We conducted evasion attempts on models employing Deep Neural Networks (DNN), Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and a combined model of Attention and LSTM. The results showed a decrease in recall rates for all detectors, confirming the possibility of evasion attacks. Furthermore, we observed that the method of inserting words with high attention weights as benign features is more effective than other insertion methods.

Keywords: PowerShell, Evasion attack, neural, network malware

1 Introduction

In contemporary society, the dependence on digital technology is increasing. Consequently, it is crucial to implement measures against cyber attacks. One of the recent trends in cyber attacks involves targeting widely used software present within the attack surface [10]. In particular, PowerShell, standard in Windows, boasts powerful capabilities for configuring Windows OS settings, automating tasks, and more [22][11]. However, the high functionality of PowerShell has been abused by attackers [18][25][14]. While PowerShell is designed with access restrictions imposed by administrators to mitigate vulnerabilities, these restrictions can be easily bypassed by using arguments that ignore execution policies [27]. Additionally, PowerShell commands can be easily generated, encrypted, and obfuscated. For these reasons, PowerShell is increasingly utilized in cyber attacks,

appearing in various scenarios such as ransomware, script jacking, and fileless attacks, as reported in numerous instances [18].

Previous research on the detection of malicious PowerShell has proposed methods combining natural language processing techniques with machine learning [10][29]. While approaches using machine learning models show promise in detecting unknown samples, there is a recognized threat of attacks aimed at evading detection, such as adversarial samples. Various attacks have demonstrated effectiveness against machine learning-based image recognition systems [24][5][7]. However, if such methods are directly applied to evasion attacks, there is a high risk of disrupting the infection functionality of malware. Therefore, it is necessary to attempt evasion attacks while maintaining the original behavior of the malware.

For PowerShell, it has been reported that evasion attacks are possible against models using natural language processing and machine learning, by incorporating features of benign PowerShell into malicious PowerShell [21]. To the best of our knowledge, there has been no evaluation of evasion attacks on machine learning models for malicious PowerShell detection, and particularly, the evaluation of evasion attacks against detection models using neural networks has not been conducted.

In this study, we attempted an evasion attack for a malicious PowerShell detection model that combines LSTM and attention mechanism by inserting words characterizing benign PowerShell script, extracted using attention weights, into malicious scripts. During this process, we combined the insertion with functions that do not impact the behavior, ensuring that it does not affect the original malicious script. Subsequently, we confirmed whether the detection rate decreased. Additionally, evasion attacks were evaluated against detection models using other neural networks. Furthermore, a comparison of the detection rates was conducted by inserting frequently occurring words in benign PowerShell scripts. The main contributions of this paper are as follows:

1. It was confirmed that evasion attacks are possible on the malicious PowerShell detection model combining LSTM and Attention mechanism, resulting in a decrease of approximately 0.23 in recall rate.
2. Evasion attacks were conducted on detection models using other neural networks, confirming a decrease in recall rates.
3. Comparison of the recall rates between inserting words that only appear in benign scripts and inserting words extracted using attention weight revealed a more significant decrease in recall rates when inserting words that contribute frequently to characterizing benign PowerShell.

The structure of this paper is shown below. Section 2 introduces related research and section 3 introduces related techniques used in this study. Section 4 describes the evaluation method, and Section 5 describes the evaluation experiment and its results. Section 6 produces some considerations, and finally, we conclude this paper.

2 Related work

2.1 Deobfuscation malicious PowerShell scripts

In malicious PowerShell, obfuscation is often used to make it harder to increase the difficulty of detection and removal of traces left by attackers. Obfuscation is a technique that involves compressing, partial encrypting, and inserting meaningless code into a program, enabling syntax modifications without affecting the program's functionality. Obfuscated PowerShell scripts make detection challenging because several methods have been suggested for deobfuscation.

Ugarte et al. propose PowerDrive [30]. This is an open-source, multi-stage deobfuscation tool for PowerShell attacks, which includes static and dynamic analysis. PowerDrive measures PowerShell code and gradually reveals obfuscation steps to analysts, thereby enabling a progressive deobfuscate process. Liu et al. proposed a deobfuscation technique known as PSDEM [17]. This deobfuscation technique involves a two-layered process to retrieve the original PowerShell script. Li et al. proposed a lightweight deobfuscation technique that focuses on the Abstract Syntax Tree (AST) subtrees [16]. This method performs obfuscation detection and emulation-based restoration at the subtree level.

In this study, deobfuscation is conducted as a preprocessing step. Regular expression matching is employed for deobfuscation, where obfuscated portions are extracted using regular expressions. The extracted portions undergo processing such as converting to lowercase, replacing Base64 encoding, and handling line breaks with newline characters.

2.2 Detection of malicious PowerShell

We present research on the detection of malicious PowerShell. Hendler et al. proposed a method for detecting malicious PowerShell using deep neural networks [10]. This approach yielded a high detection rate, combining natural language processing techniques with character-level convolutional neural networks for dynamic analysis. Fang et al. proposed a method for detecting malicious PowerShell based on multiple features [6]. This method involves extracting semantic features using FastText, followed by extracting features from abstract syntax trees, including sentences, tokens, and nodes. These features are then combined for classification. Alahmadi et al. proposed a detection method using Malicious PowerShell Script Autodetect (MPSAutodetect) [1]. This method utilizes a stacked denoising autoencoder to extract features, which are then classified using XGBoost. Tajiri et al. proposed a method for detecting malicious PowerShell scripts using a word-based language model based on static analysis [29]. This involves generating a word-based language model and using machine learning to classify unknown PowerShell scripts. In the study by Mezawa et al., the Attention mechanism is used to extract which words in the source code characterize benign and malicious PowerShell, and these are ranked by contribution frequency [20].

Thus, while machine learning or deep learning-based detection models achieve high detection rates, their evaluation under attempted evasion attacks has not been thoroughly explored. Therefore, in this study, we assume a scenario in which attackers recognize the detection method and attempt evasion attacks.

2.3 Evading Machine Learning Models

Methods to evade detection have been proposed against detection methods utilizing machine learning models. Maiorca et al. classified known vulnerabilities in PDF malware detectors and identified potential defense mechanisms to mitigate the impact of new threats compromising the detectors [19]. Chen et al. proposed a method to evade malware detection using CNN for PE file analysis [4]. In their research, they demonstrated evasion by adding source code to PE files, and they also presented means to counteract attacks through adversarial learning and pre-detection of adversarial samples. Grosse et al. showed that it is possible to evade detection in Android malware by adding characteristics of benign applications [9]. Kolosnjaji et al. introduced a gradient-based white-box attack against MalConv, generating adversarial malware binaries (AMB). To make the generated adversarial malware binaries behave as closely as possible to the original malware, they appended bytes predicted to be benign to the end of the original malware file [15]. Hu and Tan proposed a new algorithm for generating adversarial examples to attack an RNN-based malware detection system [12]. Huang et al. demonstrated that it is possible to evade detection for an LSTM (Long Short-Term Memory) based DDoS detector by inserting or replacing certain packets of the original input sample [13]. Yamamoto et al. proposed the possibility of evading machine learning model-based detection for VBA malware by incorporating characteristics of benign VBA macros while maintaining the malware's functionality. In the research by Mezawa et al., it was reported that evading attacks are feasible for machine learning models designed for detecting malicious PowerShell scripts by incorporating features of benign PowerShell scripts into malicious ones [21].

As demonstrated, the potential for evasion attacks on detectors utilizing machine learning and neural networks is evident. However, no reported evaluation of evasion attacks against malicious PowerShell detectors, except for existing research [21]. Specifically, there are no reports on the evaluation of evasion attacks against detectors using neural networks. This study aims to evaluate evasion attacks on malicious PowerShell detection models using neural networks by incorporating features of benign samples into malicious samples, similar to evasion attacks observed in other malware scenarios.

3 Related technique

In this section, we describe the machine learning models employed in this study and the attacks against these machine learning models.

3.1 Long Short Term Model(LSTM)

Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN). The prototype of LSTM was proposed by Hochreiter et al. [8]. It addresses the issue of vanishing and exploding gradients commonly encountered by RNNs. The LSTM model is illustrated in Figure 1. The LSTM model operates according to

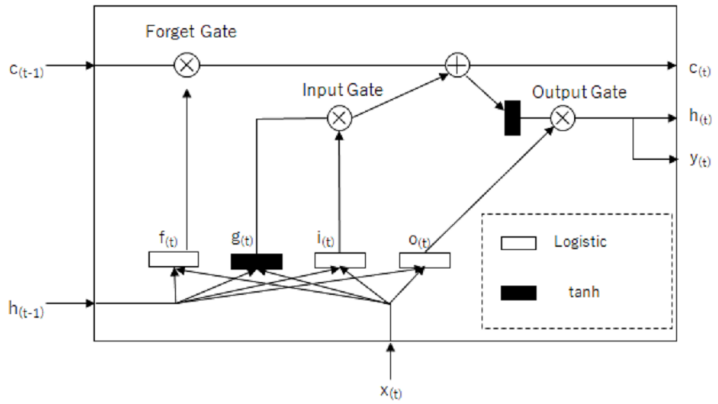

Fig. 1. LSTM cell

the following equation.

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{xi}x_{(t)} + W_{hi}h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{xf}x_{(t)} + W_{hf}h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{xo}x_{(t)} + W_{ho}h_{(t-1)} + b_o) \\
g_{(t)} &= tanh(W_{xg}x_{(t)} + W_{hg}h_{(t-1)} + b_g) \\
c_{(t)} &= f_{(t)}c_{(t-1)} + i_{(t)}g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)}tanh(c_{(t)})
\end{aligned}
$$

W_xi, W_xf, W_xo, W_xg represent the weight matrices between the input $x_{(t)}$ and four fully connected layers. W_hi, W_hf, W_ho, W_hg represent the weight matrices between the preceding short-term state $h_{(t-1)}$ and the four fully connected layers. b_i, b_f, b_o, b_g denote the bias terms for the four fully connected layers.

$g_{(t)}$ is constructed based on the current input $x_{(t)}$ and the preceding short-term state $h_{(t-1)}$ The forget gate is controlled by $f_{(t)}$, determining which part of the long-term memory to erase. The input gate is controlled by $i_{(t)}$ deciding which part of $g_{(t)}$ to store in memory. The output gate, controlled by $o_{(t)}$ determines which part of the long-term state to read and output. The long-term

state $c_{(t)}$ is created by discarding some memories from the preceding long-term state $c_{(t-1)}$ via the forget gate and adding information that passed through the input gate. The output $y_{(t)}$ is formed by passing the long-term state $c_{(t)}$ through the tanh function and output gate, while the short-term state $h_{(t)}$ is similarly generated.

3.2 Attention

The Attention mechanism is a type of deep learning that guides the prediction model on where to focus within the input data. By visualizing the weights assigned to input data tokens, known as attention weights, Attention reveals which parts of the input data it focused on during the prediction. In this study, Self-Attention is employed. Self-Attention creates three vectors from the input data: Query, Key, and Value. The relationship score, which expresses the correlation of input data, is obtained by taking the dot product of Query and Key. The attention weights are calculated using this score and the softmax function. The output is acquired by multiplying these attention weights with the Value. The self-attention is illustrated in Figure 2.

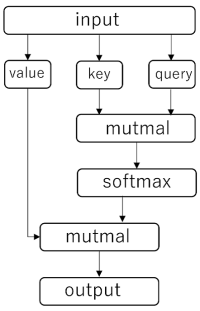

Fig. 2. Overview Diagram of Self-Attention

If we denote Query as Q, Key as K, and Value as V, the expression is represented as follows.

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

3.3 Attack against the machine learning model

There are various methods of attacking machine learning models. In this section, we introduce attacks against machine learning. According to research [26], attacks on machine learning are broadly classified as shown in the figure 3. In

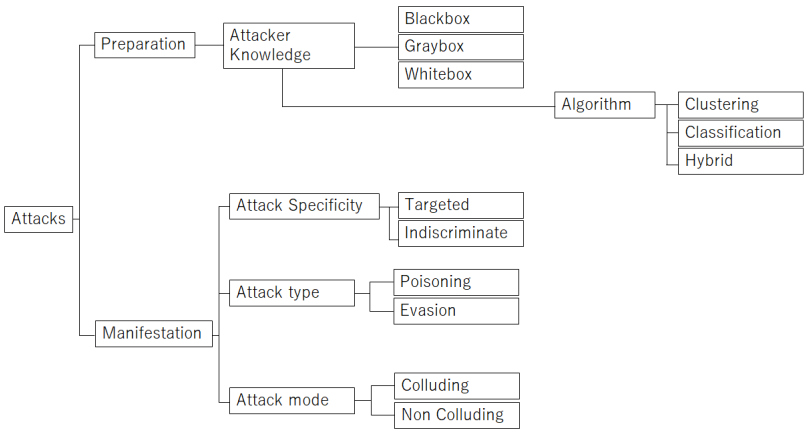

Fig. 3. taxonomy of attacks against machine learning

the preparation phase, the attacker gathers information to implement an attack plan and identify their resources. "Attacker Knowledge" refers to the knowledge that an attacker possesses about the target machine learning model. There are three categories: black box, gray box, and white box. In a black-box scenario, the attacker doesn't have Ground Truth and the learning algorithm. In a gray-box scenario, the attacker has partial internal information, and in a white-box scenario, the attacker has complete internal information. "Algorithm" is classified as (i) clustering, (ii) classification, or (iii) hybrid styles, Based on the machine learning algorithm used.

In the Manifestation phase, the attacker launches the attack against the machine learning system. "Attack Specificity" refers to the range of data points that attackers target. "Targeted" means the focus of the attack is on specific samples, while "indiscriminate" implies that attackers target samples from very general classes without specific selection. "Attack Type" refers to how much impact a machine learning system is affected by the attack. Poisoning refers to contaminating the training data, introducing errors into the model after training, and decreasing the accuracy of the model. poisoning attacks have been reported to be possible against various classification models, including SVM (Support Vector Machine) and neural networks [23][2]. An evasion attack refers to intentionally providing specific inputs to a target model during the testing phase, to cause the model to make incorrect inferences or predictions. This attack technique has been reported to apply to various classifiers, including malware classifiers, spam email filters, image recognition models, and others [3][28][7]. "Attack Mode" indicates the possibility of attackers collaborating to conduct an attack.

4 Evaluation method

In this section, we describe the experimental method for verifying evasion attacks against a malicious PowerShell detection model.

4.1 Condition

In this study, the conditions are illustrated in Figure 4. the attacker has prior knowledge of the specific machine learning model being used by the target. Additionally, it is assumed that the attacker has access to the training data. The attacker aims to create malicious PowerShell that can evade detection by applying evasion techniques against the detection model for malicious PowerShell. The attacker creates adversarial samples by adding benign features to malware, aiming to bring the adversarial sample closer to the feature space of benign files. If we denote the original source code as x_{origin} , the adversarial sample as x_{adv} , the benign features-indicating word as ϵ , the function to convert to tokens as Tokenizer and the new feature vector as V_{adv}, the creation of an adversarial sample can be expressed using the following formula

$$x_{\text{adv}} = x_{\text{origin}} + \varepsilon$$

$$V_{\text{adv}} = Tokenize(x_{\text{origin}} + \varepsilon)$$

Given these conditions, the attacks to be validated assume gray-box knowledge, the algorithm used is Classification, Attack Specificity is targeted, attack type is Evasion, and the Attack Mode is Colluding.

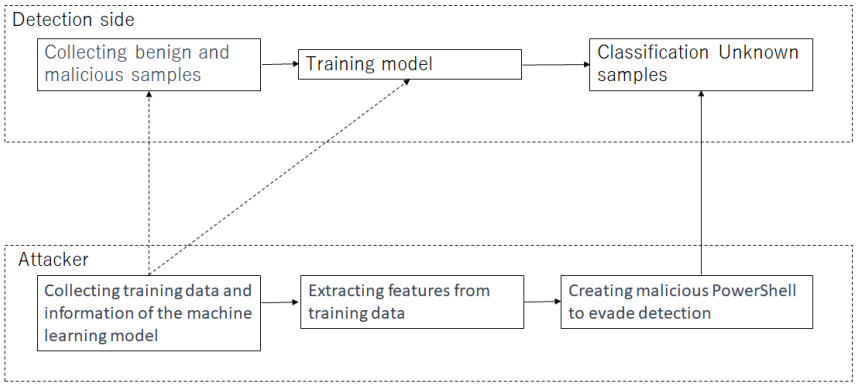

Fig. 4. Verification Procedure

4.2 Outline

The verification process is illustrated in Figure 5 and is broadly divided into three stages: the training process, detection evasion process, and testing process. The dataset consists of training and testing data, each containing samples of both benign and malicious PowerShell. Training data is used to train the machine learning model that acts as the classifier while testing data serves as the basis for generating disguised data. Initially, preprocessing is performed on the training data, including deobfuscation, data cleansing, and tokenization. Subsequently, the training process involves training the machine learning model, which acts as the classifier. Following the training process, the detection evasion process generates disguised data by adding benign PowerShell features to the malicious PowerShell in the testing data. After preprocessing the disguised data, the trained model is employed in the testing process to classify the disguised data, and the recall of the classification is verified. The details of each step are explained below.

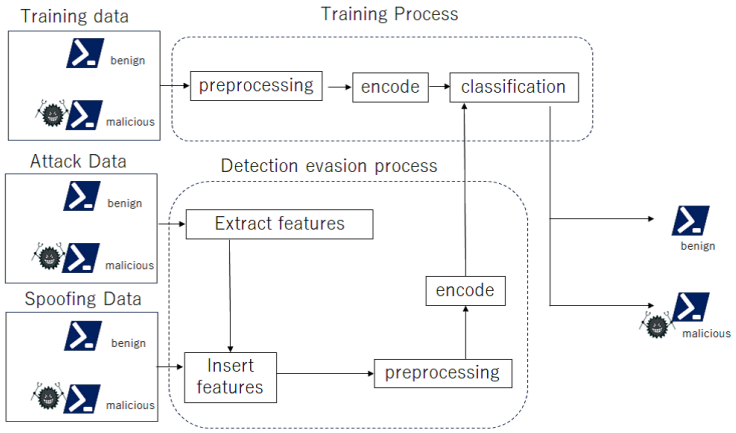

Fig. 5. Outline of the evasion method

4.3 Preprocess

In the preprocessing stage, each data undergoes deobfuscation, data cleansing, and tokenization, with each process using regular expressions. For deobfuscation, regular expressions are employed to extract obfuscated parts. The extracted parts are then processed by converting to lowercase, replacing Base64 encoding, and handling line breaks at terminal characters. In data cleansing, various elements such as comments, URLs, IP addresses, and multibyte characters are replaced, making them usable as features for subsequent processes. Tokenization

involves splitting strings using symbols that serve as boundaries for commands and variables in PowerShell, such as spaces, PowerShell terminal characters, brackets, operators, and other symbols (+.”’ ,:).

4.4 Training Process

In the training process, the machine learning model, acting as the classifier, is trained. Initially, the preprocessed training data is tokenized using a tokenizer to split the source code into words and assign a unique ID to each word. Subsequently, the training data is input into the machine learning model for training. Once trained, the machine learning model becomes capable of classifying samples. After classifying the training data, attention weights of words within each sample are compared. The top 50 words with the highest values are then extracted.

4.5 Detection Evasion Process

In the detection evasion process, evasion techniques are applied to the testing data. Algorithm 1 outlines the steps. One word, y , is randomly selected from the words that characterize benign PowerShell scripts with the top 50 high attention weights. it is then inserted into the malicious PowerShell script, x , within the testing data. To maintain PowerShell functionality, another word g is randomly selected from the arguments G of functions that do not affect the overall operation, such as case-insensitive conversion and character count. This process aims to generate disguised data. Subsequently, preprocessing is applied to the generated disguised data. The tokenizer is then used to split the source code into words and assign a unique ID to each word. An example of the insertion process is illustrated in Figure 6.

Algorithm 1 Generating spoofing Data

Require: $Y = \{Y_n\}$, k, $G = \{g_n\}$
  while Number of adds $< k$ **do**
    $y =$ random Y
    $g =$ random G
    $x^* = x\ + y + g$
  end while
  return x^*

5 Experiment

5.1 Dataset

In this study, we use the same dataset revealed in previous research [20], The dataset consists of 589 PowerShell scripts collected from HybridAnalysis, 355

```
C:¥Windows¥System32¥WindowsPowerShell¥v1.0¥p
owershell.EXE -nop -ep bypass -e
IEX (New-Object
Net.WebClient).downloadstring('http://v.bdd
p.net/wm?hdp')
```

```
C:¥Windows¥System32¥WindowsPowerShell¥v1.0¥p
owershell.EXE -nop -ep bypass -e
IEX (New-Object
Net.WebClient).downloadstring('http://v.bdd
p.net/wm?hdp')
"pipeline".ToUpper()
"name".Length
```

Fig. 6. Example of the insert process

PowerShell scripts collected from AnyRun, and 5000 benign PowerShell scripts obtained from GitHub. The collection period spans from January 2019 to March 2020, and all samples are publicly available on these platforms. For samples that were identified as threats by two or more of the five security vendors (Kaspersky, McAfee, Microsoft, Symantec, TrendMicro), we labeled them as malicious. Samples that were not identified as threats by any of the vendors were labeled as benign. Samples that did not fall into either category were excluded. To simulate real-world malicious PowerShell detection, as we cannot use unknown samples for AI training, we split the collected samples in chronological order. Samples obtained from HybridAnalysis and AnyRun were divided based on timestamps into those before June 2019 and those after July 2019. Samples obtained from GitHub, without timestamps, were randomly divided into two sets. One set was used as training data, and the other as testing data.

Table 1. Detail of dataset

AnyRun, HybridAnalysis			github
dataset type	malicious	benign	benign
training data, attack data	309	232	4901
spoofing data	171	92	

5.2 Setting

Table 2 shows the setting used for the experiments, and Table 4 presents the libraries used in implementing the machine learning model.

Table 2. Experimental setting

CPU	Core i7-9700K 3.60GHz
Memory	64GB
OS	Windows10 Home
language	Python3.8.9

Table 3. Libraries used in experiment

scikit-learn	1.0.2
Keras	2.11.0
keras-self-attention	0.51.0
tensorflow-estimator	2.11.0

5.3 Experimental model

The experimental model used in this study is presented in Table 5. Applying the parameters that demonstrated the highest performance in previous studies [20], the input data length was set to 256, the batch size to 8, the number of epochs to 16, and the dropout rate to 0.5.

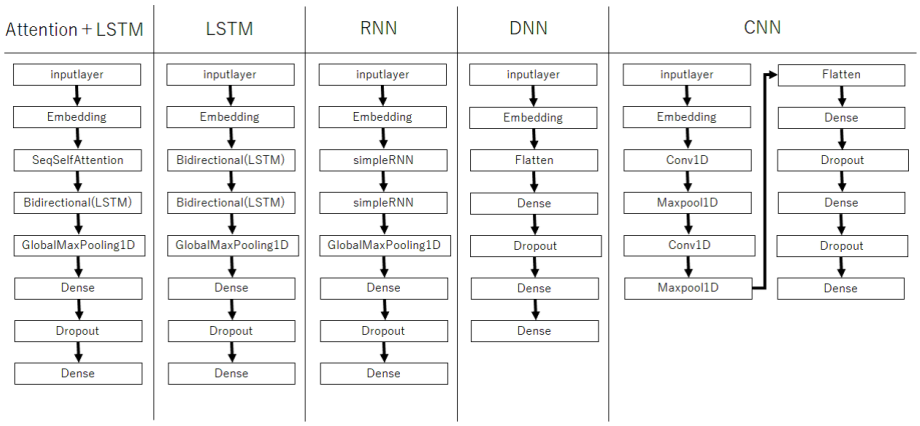

Fig. 7. The model used in the experiment

5.4 Experiment contents

In the validation experiments, we evaluate evasion attacks on the malicious PowerShell detection model that combines LSTM and Attention mechanisms, using the same model as in previous research [20]. Within the evaluation method, we

vary the number of insertions into malicious PowerShell from 0 to 100. Subsequently, the spoofing data is classified with the trained model, and the recall of malicious PowerShell is measured. We compare the recall rates when randomly inserting words characterizing benign PowerShell scripts extracted using attention weights from the training data into malicious PowerShell with the recall rates when inserting randomly selected words that are frequent only in benign PowerShell. Additionally, for comparison with models using other neural networks, including DNN, CNN, RNN, and LSTM, we conduct evaluation experiments and compare the changes in recall rates. Given the anticipated variability in experimental results, we conducted 10 trials for each measurement and reported the average value.

5.5 Result

In this section, we confirm the effect of the evaluation method. Figure 8 illustrates the change in recall rates for the model combining LSTM and attention mechanism. Figure 9,10,11,12 illustrates the change in recall rates for the model using LSTM,RNN,DNN and CNN. The vertical axis of each graph represents the recall rate of malicious PowerShell classified by each model, while the horizontal axis represents the number of insertion processes in the verification method. In this study, we focus on the recall rate of malicious PowerShells because we want to evaluate the degree of influence of evasion attacks on the detection of malicious PowerShells. In the model combining LSTM and attention mechanism, the maximum decrease was approximately 0.19 when inserting words that appear only in benign scripts and approximately 0.23 when inserting words extracted using attention weights. In all models using other neural networks, inserting words extracted using attention weights resulted in a greater decrease in recall compared to inserting words that appear only in benign scripts.

6 Discussion

6.1 Potential evasion attacks on the model combining LSTM and Attention

An attempt was made to conduct evasion attacks on the model combining LSTM and Attention by inserting words characterizing benign PowerShell into malicious PowerShell. Consequently, the recall decreased by approximately 0.23, indicating the effectiveness of the proposed attack method. Moreover, when compared to inserting words that only appear in benign cases, inserting words characterizing benign PowerShell extracted using attention weights resulted in a maximum recall difference of approximately 0.04. This suggests that the proposed method is capable of performing more effective evasion attacks.

After inserting words approximately 90 times, the recall stopped changing. This is likely because the model reads a fixed length from the beginning, and as the number of insertions increases, the portion being read becomes smaller.

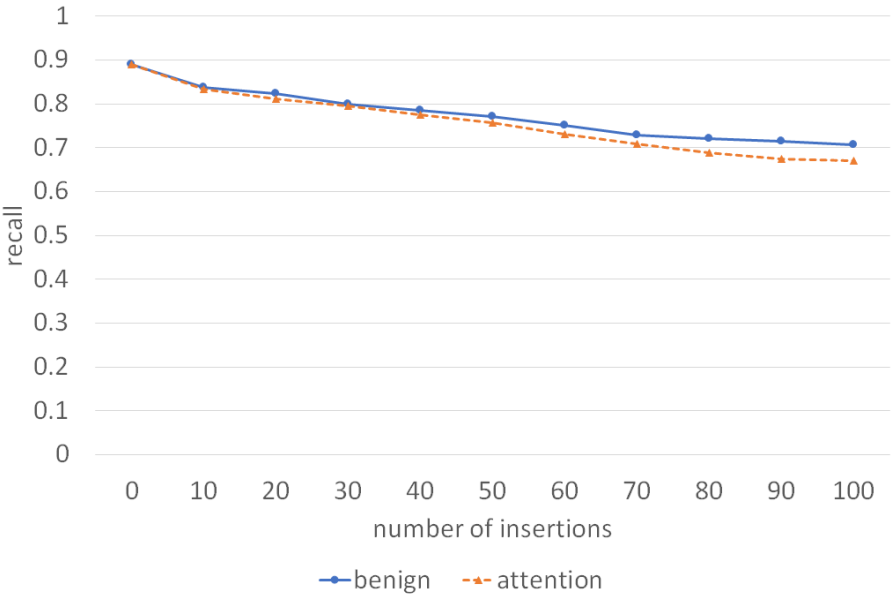

Fig. 8. The recall rate of the models using attention+LSTM.

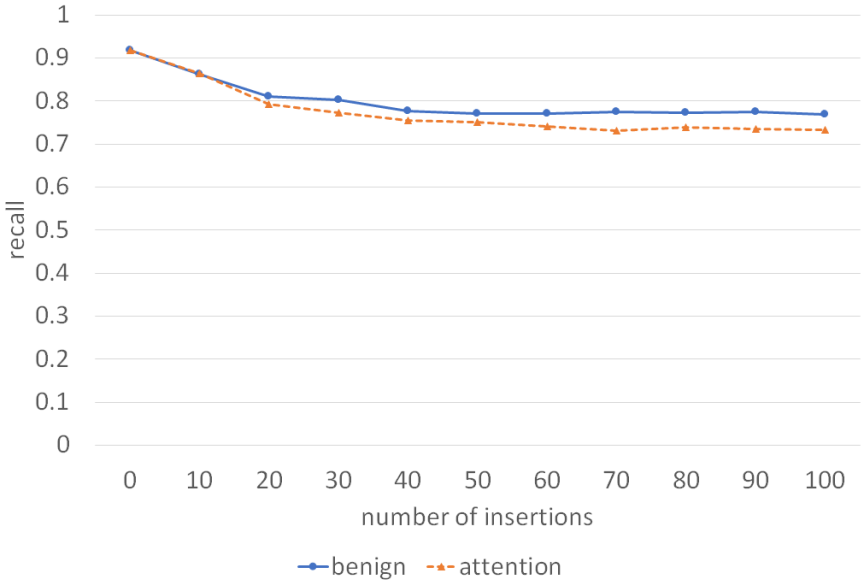

Fig. 9. The recall rate of the models using LSTM.

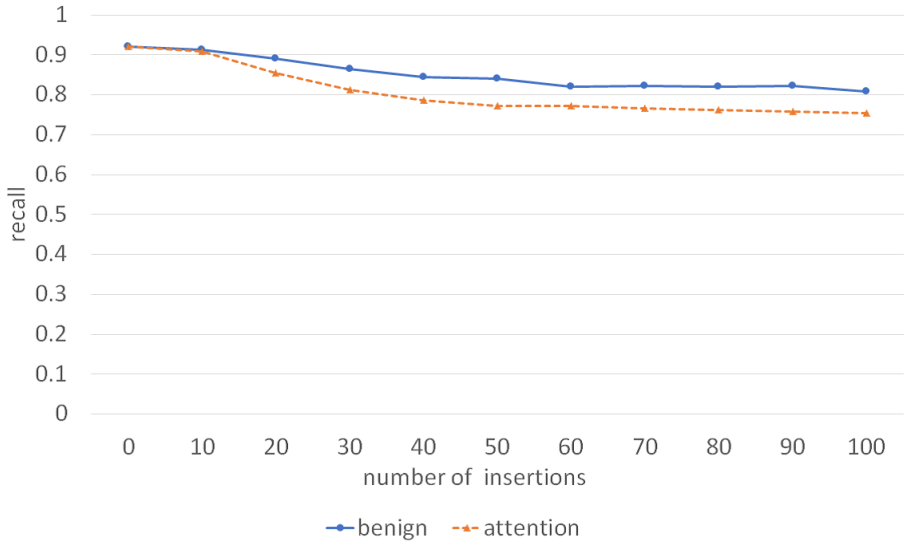

Fig. 10. The recall rate of the models using RNN.

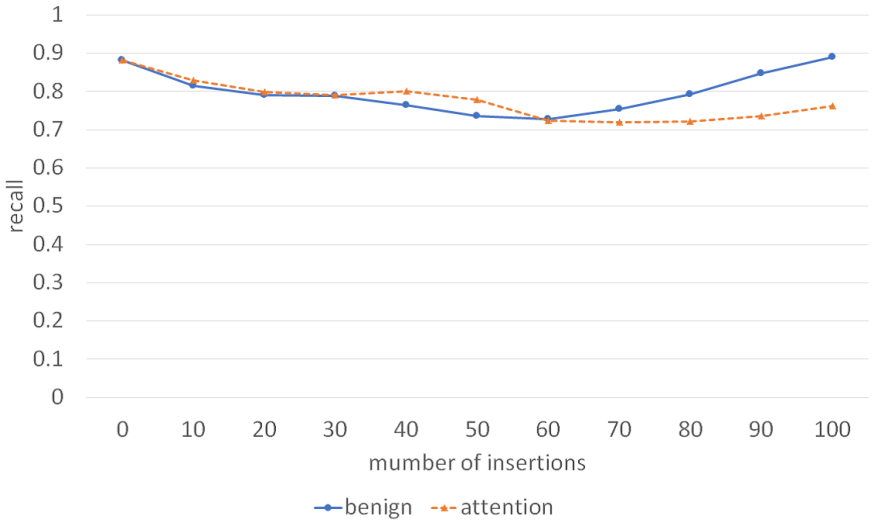

Fig. 11. The recall rate of the models using DNN.

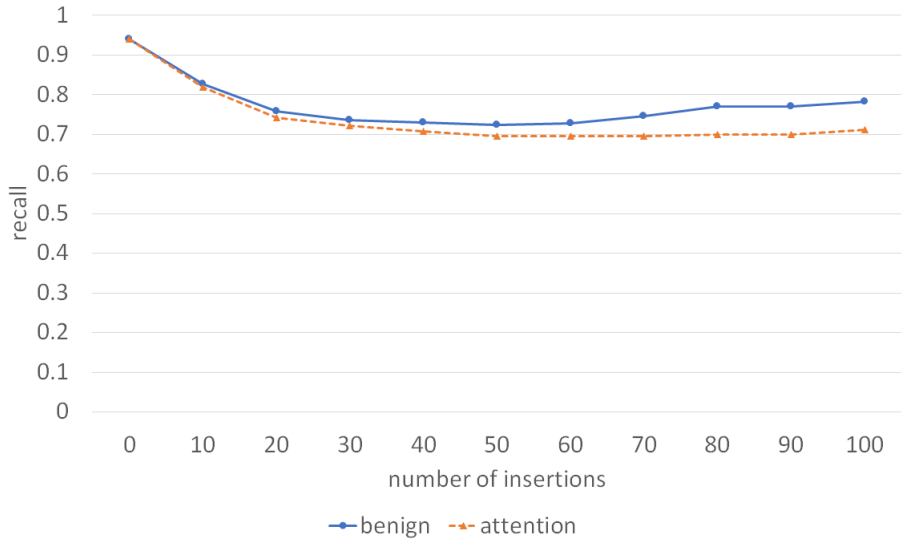

Fig. 12. The recall rate of the models using CNN.

In this experiment, the parameters with the highest performance in previous studies were used, but it is considered that the longer the fixed length the model reads, the lower the recall.

6.2 Potential evasion attacks on other neural network models

Attempts were made to conduct evasion attacks on models using other neural networks by inserting words characterizing benign PowerShell into malicious PowerShell. Consequently, recall decreased in all models. This decline is attributed to the fact that models utilizing neural networks classify based on somewhat common words. Thus, it is considered that the proposed attack method is effective against other neural network models. Moreover, in most models, inserting words characterizing benign PowerShell extracted using attention weights resulted in lower recall compared to inserting words that only appear in benign cases. The proposed method is considered capable of performing more effective evasion attacks.

6.3 Maintaining Malware Behavior

In this experiment, the objective was to preserve the original behavior of the malware by inserting words characterizing benign features and functions that do not affect the operation at the end of the source code. However, it has not been verified whether the newly created samples using sandboxes and similar tools exhibit the same behavior as the original samples. If there is no variation in the

insertion position or inserted words during the insertion process, it is conceivable that the insertion process may be identified using simple detection methods or become a new characteristic of malicious PowerShell. In fact, in models using Deep Neural Networks (DNN), recall increases to some extent as the number of insertions increases. This is because the insertion process is recognized as a new characteristic of malicious PowerShell and is classified as malicious. Therefore, it is necessary to perform insertion processes in the source code or increase the types of functions and words to make the insertion less detectable. In doing so, it is important to consider the structure of the source code to ensure that the insertion process does not affect the original malware behavior.

6.4 Feasibility of the Attack

In this experiment, the condition was that the attacker had access to the training data and the used model on the detection side. In a real-world scenario, an attacker would need to conduct attacks such as estimating training data or extracting the model to obtain information about the training data and the used model before attempting evasion attacks. Therefore, the feasibility of conducting attacks under realistic conditions is considered low. Additionally, commercial detectors capture various features for detection, so it is difficult to implement the proposed method in this study, which focuses solely on evading detection through machine learning, in a real-world environment. However, understanding evasion attacks is significant for taking proactive measures against attackers.

6.5 Research ethics

In this study, all PowerShell samples used are publicly available. Additionally, the libraries used for implementing the machine learning model, such as gensim and scikit-learn, are also freely accessible. Therefore, reproducing a similar environment to this study is straightforward, and the reproducibility of this research is deemed to be high.

6.6 Limitations

In this study, experiments were conducted using several samples equivalent to [20]. However, it can be challenging to claim that the number of malicious PowerShell samples is necessarily sufficient. Increasing the number of samples for experimentation could potentially enhance the accuracy of the validation.

7 Conclusion

In this study, we demonstrated that it is possible to evade neural network-based machine learning models for malicious PowerShell scripts by adding words extracted using attention weight. We also showed that the evasion attack is more effective when inserting words extracted using attention weight compared to

words that only frequently appear in benign scripts, resulting in a lower detection rate.

One future challenge is to confirm the functionality of the modified samples by comparing their behavior with the original samples. While theoretically, it is believed that maintaining the functionality of malicious PowerShell is possible even after modification, practical confirmation of the actual behavior has not been conducted.

Acknowledgments This work was supported by JSPS KAKENHI Grant.

References

1. Alahmadi, A., Alkhraan, N., Binsaeedan, W.: Mpsautodetect: A malicious powershell script detection model based on stacked denoising auto-encoder. Comput. Secur. 116, 102658 (2022), https://doi.org/10.1016/j.cose.2022.102658
2. Biggio, B., Nelson, B., Laskov, P.: Poisoning attacks against support vector machines. In: Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012. icml.cc / Omnipress (2012), http://icml.cc/2012/papers/880.pdf
3. Biggio, B., Roli, F.: Wild patterns: Ten years after the rise of adversarial machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 2154–2156. ACM (2018), https://doi.org/10.1145/3243734.3264418
4. Chen, B., Ren, Z., Yu, C., Hussain, I., Liu, J.: Adversarial examples for cnn-based malware detectors. IEEE Access 7, 54360–54371 (2019), https://doi.org/10.1109/ACCESS.2019.2913439
5. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 1625–1634. Computer Vision Foundation / IEEE Computer Society (2018), http://openaccess.thecvf.com/content_cvpr_2018/html/Eykholt_Robust_Physical-World_Attacks_CVPR_2018_paper.html
6. Fang, Y., Zhou, X., Huang, C.: Effective method for detecting malicious powershell scripts based on hybrid features. Neurocomputing 448, 30–39 (2021), https://doi.org/10.1016/j.neucom.2021.03.117
7. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015), http://arxiv.org/abs/1412.6572
8. Graves, A.: Supervised Sequence Labelling with Recurrent Neural Networks, Studies in Computational Intelligence, vol. 385. Springer (2012), https://doi.org/10.1007/978-3-642-24797-2
9. Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.D.: Adversarial examples for malware detection. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part

II. Lecture Notes in Computer Science, vol. 10493, pp. 62–79. Springer (2017), https://doi.org/10.1007/978-3-319-66399-9_4
10. Hendler, D., Kels, S., Rubin, A.: Detecting malicious powershell commands using deep neural networks. In: Kim, J., Ahn, G., Kim, S., Kim, Y., López, J., Kim, T. (eds.) Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018. pp. 187–197. ACM (2018), https://doi.org/10.1145/3196494.3196511
11. Holmes, L.: Windows PowerShell Cookbook: The Complete Guide to Scripting Microsoft's Command Shell. O'Reilly Media (2012)
12. Hu, W., Tan, Y.: Black-box attacks against RNN based malware detection algorithms. In: The Workshops of the The Thirty-Second AAAI Conference on Artificial Intelligence, New Orleans, Louisiana, USA, February 2-7, 2018. AAAI Technical Report, vol. WS-18, pp. 245–251. AAAI Press (2018), https://aaai.org/ocs/index.php/WS/AAAIW18/paper/view/16594
13. Huang, W., Peng, X., Shi, Z., Ma, Y.: Adversarial attack against lstm-based ddos intrusion detection system. In: 32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020. pp. 686–693. IEEE (2020), https://doi.org/10.1109/ICTAI50040.2020.00110
14. Kazanciyan, R., Hastings, M.: Investigating powershell attacks. Black Hat p. 25 (2014)
15. Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., Roli, F.: Adversarial malware binaries: Evading deep learning for malware detection in executables. In: 26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018. pp. 533–537. IEEE (2018), https://doi.org/10.23919/EUSIPCO.2018.8553214
16. Li, Z., Chen, Q.A., Xiong, C., Chen, Y., Zhu, T., Yang, H.: Effective and lightweight deobfuscation and semantic-aware attack detection for powershell scripts. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019. pp. 1831–1847. ACM (2019), https://doi.org/10.1145/3319535.3363187
17. Liu, C., Xia, B., Yu, M., Liu, Y.: PSDEM: A feasible de-obfuscation method for malicious powershell detection. In: 2018 IEEE Symposium on Computers and Communications, ISCC 2018, Natal, Brazil, June 25-28, 2018. pp. 825–831. IEEE (2018), https://doi.org/10.1109/ISCC.2018.8538691
18. Liu, S., Peng, G., Zeng, H., Fu, J.: A survey on the evolution of fileless attacks and detection techniques. Comput. Secur. 137, 103653 (2024), https://doi.org/10.1016/j.cose.2023.103653
19. Maiorca, D., Biggio, B., Giacinto, G.: Towards adversarial malware detection: Lessons learned from pdf-based attacks. ACM Comput. Surv. 52(4), 78:1–78:36 (2019), https://doi.org/10.1145/3332184
20. Mezawa, Y., Mimura, M.: An attention mechanism for visualizing word weights in source code of powershell samples: Experimental results and analysis. In: Barolli, L. (ed.) Advances on Broad-Band Wireless Computing, Communication and Applications - Proceedings of the 17th International Conference on Broad-Band Wireless Computing, Communication and Applications (BWCCA-2022), Tirana, Albania, 27-29 October 2022. Lecture Notes in Networks and Systems, vol. 570, pp. 114–124. Springer (2022), https://doi.org/10.1007/978-3-031-20029-8_11
21. Mezawa, Y., Mimura, M.: Evaluating the possibility of evasion attacks to machine learning-based models for malicious powershell detection. In: Su, C., Gritzalis,

D., Piuri, V. (eds.) Information Security Practice and Experience - 17th International Conference, ISPEC 2022, Taipei, Taiwan, November 23-25, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13620, pp. 252–267. Springer (2022), https://doi.org/10.1007/978-3-031-21280-2_14
22. Microsoft: Powershell, https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.4
23. Muñoz-González, L., Biggio, B., Demontis, A., Paudice, A., Wongrassamee, V., Lupu, E.C., Roli, F.: Towards poisoning of deep learning algorithms with back-gradient optimization. In: Thuraisingham, B., Biggio, B., Freeman, D.M., Miller, B., Sinha, A. (eds.) Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017. pp. 27–38. ACM (2017), https://doi.org/10.1145/3128572.3140451
24. Paya, A., Arroni, S., García-Díaz, V., Gómez, A.: Apollon: A robust defense system against adversarial machine learning attacks in intrusion detection systems. Comput. Secur. 136, 103546 (2024), https://doi.org/10.1016/j.cose.2023.103546
25. Pereira, A.J.: Tracking, detecting, and thwarting powershell-based malware and attacks. Trend Micro (2020), https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/tracking-detecting-and-thwarting-powershell-based-malware-and-attacks
26. Pitropakis, N., Panaousis, E., Giannetsos, T., Anastasiadis, E., Loukas, G.: A taxonomy and survey of attacks against machine learning. Comput. Sci. Rev. 34 (2019), https://doi.org/10.1016/j.cosrev.2019.100199
27. Symantec: The increased use of Powershell in attacks. https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/increased-use-of-powershell-in-attacks-16-en. (2016)
28. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014), http://arxiv.org/abs/1312.6199
29. Tajiri, Y., Mimura, M.: Detection of malicious powershell using word-level language models. In: Aoki, K., Kanaoka, A. (eds.) Advances in Information and Computer Security - 15th International Workshop on Security, IWSEC 2020, Fukui, Japan, September 2-4, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12231, pp. 39–56. Springer (2020), https://doi.org/10.1007/978-3-030-58208-1_3
30. Ugarte, D., Maiorca, D., Cara, F., Giacinto, G.: Powerdrive: Accurate de-obfuscation and analysis of powershell malware. In: Perdisci, R., Maurice, C., Giacinto, G., Almgren, M. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment - 16th International Conference, DIMVA 2019, Gothenburg, Sweden, June 19-20, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11543, pp. 240–259. Springer (2019), https://doi.org/10.1007/978-3-030-22038-9_12