PowerDetector: Malicious PowerShell Script Family Classification Based on Multi-Modal Semantic Fusion and Deep Learning

Xiuzhang Yang[1,2,3], Guojun Peng[1,2,*], Dongni Zhang[1,2], Yuhang Gao[1,2], Chenguang Li[1,2]

[1] School of Cyber Science and Engineering, Wuhan University, Wuhan 430072, China
[2] Key Laboratory of Aerospace Information Security and Trusted Computing, Ministry of Education, Wuhan 430072, China
[3] School of Information, Guizhou University of Finance and Economics, Guiyang 550025, China
[*] The corresponding author, email: guojpeng@whu.edu.cn

Cite as: X. Yang, G. Peng, *et al*., "Powerdetector: Malicious powershell script family classification based on multi-modal semantic fusion and deep learning," *China Communications*, vol. 20, no. 11, pp. 202-224, 2023. **DOI**: 10.23919/JCC.fa.2022-0509.202311

Abstract: PowerShell has been widely deployed in fileless malware and advanced persistent threat (APT) attacks due to its high stealthiness and live-off-the-land technique. However, existing works mainly focus on deobfuscation and malicious detection, lacking the malicious PowerShell families classification and behavior analysis. Moreover, the state-of-the-art methods fail to capture fine-grained features and semantic relationships, resulting in low robustness and accuracy. To this end, we propose PowerDetector, a novel malicious PowerShell script detector based on multi-modal semantic fusion and deep learning. Specifically, we design four feature extraction methods to extract key features from character, token, abstract syntax tree (AST), and semantic knowledge graph. Then, we intelligently design four embeddings (i.e., Char2Vec, Token2Vec, AST2Vec, and Rela2Vec) and construct a multi-modal fusion algorithm to concatenate feature vectors from different views. Finally, we propose a combined model based on transformer and CNN-BiLSTM to implement PowerShell family detection. Our experiments with five types of PowerShell attacks show that PowerDetector can accurately detect various obfuscated and stealth PowerShell scripts, with a 0.9402 precision, a 0.9358 recall, and a 0.9374 F_1-score. Furthermore, through single-modal and multi-modal comparison experiments, we demonstrate that PowerDetector's multi-modal embedding and deep learning model can achieve better accuracy and even identify more unknown attacks.

Keywords: deep learning; malicious family detection; multi-modal semantic fusion; PowerShell

Received: Jul. 24, 2022
Revised: Nov. 25, 2022
Editor: Min Lei

I. INTRODUCTION

PowerShell is an interactive scripting language and cross-platform task automation solution based on the Microsoft .NET framework, which has been pre-installed into the operating system since Windows 7 [1]. Moreover, PowerShell can quickly call Windows API and assist in managing the operating system, hence is used for task automation and configuration management [2]. Unfortunately, since PowerShell can execute directly in memory to carry out living-off-the-land and fileless attacks [3–5], and easily bypass existing firewalls or intrusion detection systems (IDS) through obfuscation [6], hackers often leverage PowerShell to conduct advanced persistent threat (APT) attacks [7, 8]. Thus, it is crucial to detect and trace APT malware [9, 10]. According to McAfee's latest report,

the number of new PowerShell malware increased by 689% in the first quarter of 2020 compared to the previous quarter [11]. Specifically, recent incidents (e.g., Stuxnet [12], APT34 using PowDesk to steal information [13], the explosion of the Donoff family's TrojanDownloader in 2020 [11], and APT29's fileless PowerShell backdoor, namely PoshSpy [14]) have demonstrated that sophisticated and stealthy APT attacks can have a catastrophic impact on the economy, society, and human lives. Therefore, it is urgent to build an effective PowerShell detector and prevent such threats.

To combat PowerShell attacks, current works have proposed to deobfuscate and identify malicious PowerShell scripts. In terms of deobfuscation, PSDEM [15] and PowerDrive [16] are multi-stage deobfuscation methods to extract PowerShell scripts. Revoke-Obfuscation [17] and PowerDecode [18] are two open-source frameworks for the deobfuscation and analysis of PowerShell scripts. Moreover, Li et al. [3] proposed a new subtree-based deobfuscation method, which can eliminate the obfuscation of PowerShell scripts well through abstract syntax trees and simulated recovery. In terms of malicious detection, Unit42 cyber research introduced the obfuscation technology of hidden data within PowerShell scripts through static analysis and built a scoring system to assess the risk of scripts [19]. Microsoft leveraged antimalware scan interface (AMSI) [20] for detecting malicious PowerShell codes, which can provide defense for Windows systems. However, the above methods rely heavily on expert knowledge and need to construct a large number of rules manually, which results in poor portability and low accuracy.

With the rapid development of machine learning, malicious PowerShell detection methods based on machine learning have been widely proposed to prevent various attacks, which can solve the shortcomings of the above methods. For example, Hendler et al. [21] implemented a PowerShell detector based on natural language processing (NLP) and convolutional neural networks (CNN). Subsequently, they used the information provided by AMSI to study malicious PowerShell codes and constructed a detector based on contextual embedding [2]. In addition, Fang et al. [22] designed a hybrid feature-based detection method for malicious PowerShell scripts and analyzed the characters, functions, tokens, and node features of malicious samples and normal samples. Rusak et al. [23] combined deep learning and abstract syntax trees (AST) to detect the malicious PowerShell family.

However, there are no existing works to construct multi-modal algorithms or semantic relationship analysis to extract key features of PowerShell scripts, which makes it impossible to achieve fine-grained features. More specifically, current studies mainly focus on deobfuscation and malicious detection (i.e., binary classification). However, in real-world attacks, we need to further determine malicious family categories and critical characteristics of raw codes to trace APT attacks and analyze attack organization portraits. Obviously, multi-classification studies for PowerShell are crucial and more complex than previous binary classification. Traditional deep learning-based detectors ignore feature extraction and embedding fusion, which will lead to low accuracy and suffer from evasion attacks that can easily bypass firewalls or IDSs, especially when facing PowerShell attacks [5]. Therefore, it is essential to build an effective multi-class detector to identify family labels of malicious PowerShell scripts. When the PowerShell family is successfully identified, it can help security analysts determine attack types and malicious functions, predict attack behavior, and reason attack intent, thereby preventing PowerShell attacks immediately [24].

Moreover, there are many challenges in designing multi-modal embedding for feature extraction, constructing an effective neural network model, and extracting semantic features using a knowledge graph. Firstly, extracting features of various modalities for PowerShell scripts is complex, and existing methods cannot accurately extract PowerShell features of fusion characters, tokens, trees, and graphs. Secondly, since there is a huge semantic gap between the low-level malicious PowerShell code and the high-level malicious behavior analysis, it seriously affects the accuracy of malware detection or APT attack tracing. Then, it is challenging to construct effective embeddings to complete the vector representation and embedding fusion of multiple modalities [25]. Finally, it is difficult to construct a deep learning model suitable for the multi-modal embedding of the PowerShell script. In a word, we hope to build a classification model that can meet the characteristics of PowerShell codes, and then accurately extract semantic features and dependencies of different modalities or dimen-

sions, which can efficiently identify PowerShell malicious families or behaviors.

To address the above challenges, we design and implement PowerDetector, a novel malicious PowerShell script detector based on multi-modal semantic fusion and deep learning. PowerDetector can effectively extract the fine-grained features of different modalities of PowerShell scripts and accurately identify malicious PowerShell families. Specifically, we propose four feature extraction methods based on the characteristics of PowerShell. These methods extract features from the character level, the token level, the abstract syntax tree's structure level, and the semantic knowledge graph level, thereby capturing different levels and fine-grained features of malicious PowerShell scripts. To overcome the challenge that existing methods cannot achieve fine-grained features, we intelligently design four embeddings (i.e., Char2Vec, Token2Vec, Rela2Vec, and AST2Vec) and construct a multi-modal fusion algorithm to combine the feature vectors of different views. Finally, we build a combined model based on multi-head self-attention, transformer, and CNN-BiLSTM to perform the multi-classification task (i.e., five families identification). In particular, we combine the ATT&CK framework [26] and real-world attacks to construct a PowerShell dataset with five malicious families, a total of 3000 PowerShell samples. Next, we implement comparative experiments and analyze the performance of various models and modalities, thereby verifying that PowerDetector has better accuracy and robustness. Our contributions are summarized as follows:

- To our best knowledge, we are the first to design and implement PowerDetector, a novel malicious PowerShell script detector based on multi-modal semantic fusion and transfer learning. PowerDetector can extract fine-grained features of PowerShell scripts based on four feature extraction algorithms and four embedding representation methods (i.e., Char2Vec, Token2Vec, Rela2Vec, and AST2Vec). The detector can extract different level features of PowerShell scripts and detect family labels, including characters, tokens, functions, tree structures, and semantic relationship graphs. Therefore, the feature extraction and multi-modal fusion methods proposed in this paper are better than state-of-the-art ones.
- We construct a PowerShell detector based on multi-head self-attention, transformer, and CNN-BiLSTM for the first time. The proposed model can closely associate the multi-modal embedding and capture multi-level semantic relationships, thus enhancing the model's generalization ability and identifying more PowerShell attacks with less prior knowledge. In particular, the work in this paper belongs to a multi-classification task, which is more in-depth than the previous binary classification and can effectively identify the malicious behavior of PowerShell attacks. Therefore, PowerDetector can motivate future research efforts in APT attack detection and attack intent reasoning.
- To ensure family classification, we systematically survey the ATT&CK framework and malicious functions of typical PowerShell attacks. Then, we construct a PowerShell malicious family dataset based on real-world attacks, which is divided into five categories: Bypass, Downloader, Injector, Payload, and TaskExecution. Furthermore, we evaluate the performance of PowerDetector. The experimental results show that PowerDetector has better accuracy and robustness, and can even identify unknown attacks, which indicates that the proposed model can be effectively applied to detect obfuscated, stealthy, and fileless attacks.

Note that the multi-modal fusion proposed in this paper is based on the PowerShell script structure and modality, which is different from traditional feature fusion. The previous method only fuses different features (e.g., combining API sequence features extracted by dynamic analysis and string features extracted by static analysis), which ignores the modality between features and the semantic information between the extracted features. In other words, PowerDetector extracts features from the four modalities of character, token sequence, tree structure, and graph structure, and then constructs four embeddings (i.e., Char2Vec, Token2Vec, Rela2Vec, and AST2Vec). Our method regards the four structures of PowerShell (i.e., point, line, tree, and graph) as modalities or dimensions. Also, the semantic information between the features affected by each modality is different, which can more effectively find the semantic relationship between different malicious families. Therefore, we define this method as the multi-modal semantic fusion, which can

combine subsequent deep learning models to identify PowerShell malicious families.

The rest of this paper is organized as follows. In Section II, we summarize the background and related works. Section III presents the design details of PowerDetector. Section IV evaluates the performances of PowerDetector. Section V concludes this paper.

II. BACKGROUND AND RELATED WORK

In this section, we give a brief background of PowerShell malware and threats. Then, we discuss existing defenses against such attacks in detail and provide an overview.

2.1 PowerShell Malware and Threats

Advanced persistent threat (APT) attack is a novel network attack with well-organized attackers, long-term campaigns, and stealthy and evasive techniques (e.g., the zero-day attack), which has caused serious security threats and financial losses worldwide [7, 8, 27]. Moreover, script attacks are common forms of APT attacks, such as PowerShell [1], JavaScript [28], and VBScript [29]. Recently, Symantec's technical report on the malicious abuse of PowerShell [30] showed a sharp increase in the number of malicious PowerShell samples from penetration tools. Obviously, there is an urgent need to develop effective ways to detect malicious PowerShell scripts.

Another reason for PowerShell being widely used is that it can build fileless attacks using command-line shells and scripting languages. Also, open-source penetration frameworks (e.g., PowerSploit [31], Nishang [32], Empire [33], and CobaltStrike [34]) are very easy to generate PowerShell attack samples. In general, PowerShell scripts are used for real-world attacks during installation, payload delivery, persistence, and execution phases [2, 3, 22]. For example, malicious PowerShell code can be downloaded from a remote server, or the target host can be controlled by executing a PowerShell payload. In particular, such threats often use living-off-the-land and obfuscating techniques to bypass family detection [3, 4], which have created significant security risks worldwide. All in all, it is critical to consider how to build a model to identify PowerShell malicious families.

2.2 Related Survey

As shown in Table 1, we summarize related works of PowerShell with different aspects such as techniques, focus, deobfuscation, abstract syntax trees (AST), knowledge graph (KG), multi-modal, and transformer.

Deobfuscation. Deobfuscation mainly leverages regular expressions [16–18], abstract syntax trees [3, 36], monitor processes [3, 15, 36], and cmdlet overriding [16, 18] to extract the source code of PowerShell. Moreover, these methods implement multi-stage deobfuscation with dynamic and static analysis, which can extract the raw PowerShell code to help security personnel analyze the sample. Unfortunately, these methods ignore detecting PowerShell malicious behaviors (or families) and fail to use systematic feature extraction, multi-modal fusion, and transfer learning, resulting in limited application scenarios.

Malicious Detection. Malicious detection is a binary classification task that identifies whether PowerShell scripts are malicious or benign. Recently, deep learning has been widely used in malware analysis [37, 5, 24], including PowerShell attack detection. PowerShell malicious detection methods commonly utilize machine learning (e.g., random forests [23, 35]) and deep learning (e.g., convolutional neural networks [2, 21] and autoencoders [38]). Next, deep learning methods combining abstract syntax trees [22], AMSI [2], and contextual embeddings [2] are proposed, which can recognize some malicious PowerShell scripts. However, these methods lack semantic relationship analysis of PowerShell features and have not yet utilized multi-modal fusion to extract key features at various levels. In fact, PowerShell scripts have multiple levels of features (e.g., characters, functions, tree structures, and semantic relationships). In addition, the existing methods rely excessively on expert knowledge to realize a large amount of data annotation and fail to adopt transformers to capture long-distance dependent features [39]. Also, how to implement few-shot learning and malware detection in specific environments (e.g., fileless attacks or malicious code in the cloud platforms) will be the focus of future research [3, 40, 41]. More specifically, the goal of existing methods is only to identify whether PowerShell is malicious or not. However, real-world attacks need to determine the attack families or behaviors. Therefore, it is crucial to construct an effective PowerShell

Table 1. *Related work comparison. In this table, ✓ stands for fully cover, ◯ stands for partial cover, × means cannot cover.*

Related work	Techniques	Focus	Deobfuscation	AST	KG	Multi-modal	Transformer
Li et al. [3]	subtree-based deobfuscation OOA mining algorithm	deobfuscation	✓	✓	×	×	×
PSDEM [15]	two-layer deobfuscation monitor process by dynamic analysis	deobfuscation	✓	×	×	×	×
PowerDrive [16]	multi-stage deobfuscator static analysis by regex dynamic analysis by cmdlet	deobfuscation	✓	×	×	×	×
PowerDecode [18]	syntax check and remove base64 encoding deobfuscate by cmdlet overriding deobfuscate by regex	deobfuscation	✓	×	×	×	×
Hendler et al. [21]	character-level CNN 4-layer CNN	binary classification	×	×	×	×	×
Fang et al. [22]	hybrid features FastText random forest	binary classification	✓	✓	×	◯	×
Hendler et al. [2]	AMSI-based detection contextual embeddings Token-Char-FastText	binary classification	×	×	×	×	×
FC-PSDS [35]	features combination random forest and DNN	binary classification	✓	✓	×	×	×
Ruscak et al. [23]	abstract syntax tree random forest	multi-classification task	×	✓	×	×	×
Our method PowerDetector	multi-modal embedding Transformer-CNN-BiLSTM multi-layer deobfuscation algorithm	malicious family detection multi-classification task	✓	✓	✓	✓	✓

detector to identify different functional families.

To this end, we first propose PowerDetector to identify the malicious families of PowerShell scripts. As illustrated in Table 1, PowerDetector can extract semantic relationships through knowledge graph, extract fine-grained features through multi-modal fusion, and integrate transfer learning to improve the robustness and accuracy of the model. Therefore, our detector can provide assistance for subsequent APT attack tracing and malicious behavior reasoning.

III. SYSTEM DESIGN

In this section, we first present the architecture of PowerDetector and describe the data preprocessing. Then, we present four feature extraction methods and a multi-modal fusion algorithm. Finally, we present the design and implementation of this model in detail.

3.1 Overview

To combat mentioned obfuscation attacks and combine the advantages of multi-level features, we first design and implement PowerDetector, a malicious PowerShell script detector based on multi-modal feature fusion and deep learning. The overall architecture of our PowerShell detector is depicted in Figure 1. PowerDetector mainly consists of six phases: data preprocessing, feature extraction, embedding transformation, multi-modal fusion, model construction, and classification.

Data preprocessing. Since PowerShell scripts are usually obfuscated and heterogeneous in real-time attacks, the PowerDetector first performs data preprocessing to get raw input data (Step ① in Figure 1). PowerDetector obtains high-quality data through deobfuscation and data cleaning. Then, PowerShell scripts are processed with the text, token, and abstract syntax trees to extract different types of features. Finally, the preprocessed data is split into training and testing datasets.

Feature extraction. We design four feature extraction methods to capture various levels and fine-grained features of PowerShell scripts. Firstly, character statistics-based feature extraction can calculate the character frequency. Secondly, PowerDetector mines token features through the feature extraction method based on statistical analysis. Then, the abstract syntax tree-based feature extraction method is used to extract semantic representations of PowerShell structures, which preserves the relationship of contextual features and removes extraneous parameter interference. Finally, the semantic relationship features of different malicious families are discovered through a knowledge graph-based feature extraction method (details in Section 3.3).

Embedding. To transform the extracted features into

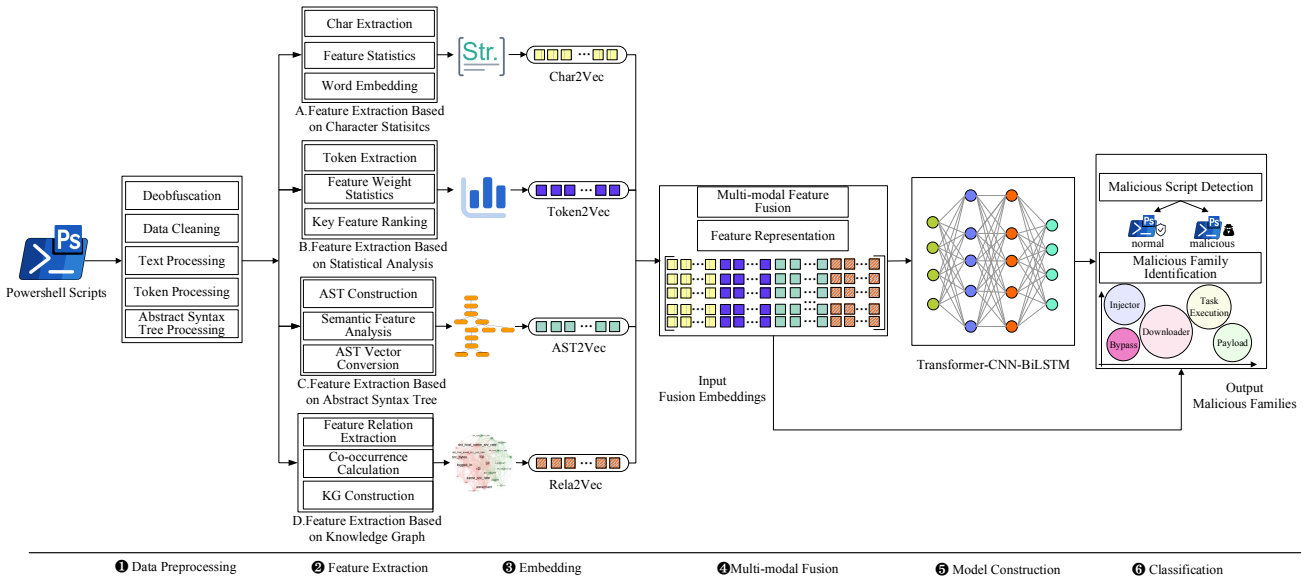

Figure 1. *The architecture of PowerDetector.*

vectors for subsequent neural network model learning, PowerDetector maps word features of four levels in turn through Word2vec. Next, it splines them into corresponding sentence vectors according to PowerShell scripts. To this end, four embedding representation methods are generated, which are Char2Vec, Token2Vec, AST2Vec, and Rela2Vec, respectively (Step ③ in Figure 1).

Multi-modal fusion. Since multiple modalities can provide more semantic information than a single one by describing the same PowerShell script, PowerDetector realizes joint representation learning by fusing features from multiple levels. At this stage, PowerDetector extracts richer and more features by fusing four-dimensional embeddings (i.e., character, token text, abstract syntax tree structure, and semantic knowledge graph), which provides better embedding representations for subsequent PowerShell malicious family classification (details in Section 3.4).

Model construction. This paper transforms the PowerShell malicious family classification problem into the sequence classification problem, and captures the semantic features of multi-modal fusion by building a combined model (Step ⑤ in Figure 1). Our model consists of a multi-head self-attention, transformer, and CNN-BiLSTM. In addition, the transformer and attention can effectively highlight the global co-occurrence information, the dependencies of abstract syntax trees and knowledge graph features, thereby improving the accuracy and robustness of detection.

Classification. The classification component of PowerDetector is mainly to identify malicious families or attack types (i.e., multi-class classification). Note that our task focuses on malicious functional family detection. We divided five types of families by combining the ATT&CK framework and PowerShell popular attack techniques. The detailed data description will be introduced in the following experiment section.

3.2 Data Preprocessing

In real-world network attacks, there are a lot of obfuscation PowerShell codes, including character transformation (i.e., replacing keywords through string segmentation, case obfuscation, and character splicing), encoding (i.e., converting English letters into corresponding ASCII codes or hexadecimal codes), encryption (i.e., processing the code using encryption algorithms like AES), and compression (i.e., compressing the original code using the Compress-Archive library functions). Moreover, the identification of key commands is crucial for PowerShell family classification. For example, the "System.Net.Webclient" object is an important label of the downloader. The obfuscation technology greatly reduces the syntax structure of the original code, making the classification model unable to recognize the original features and have low accu-

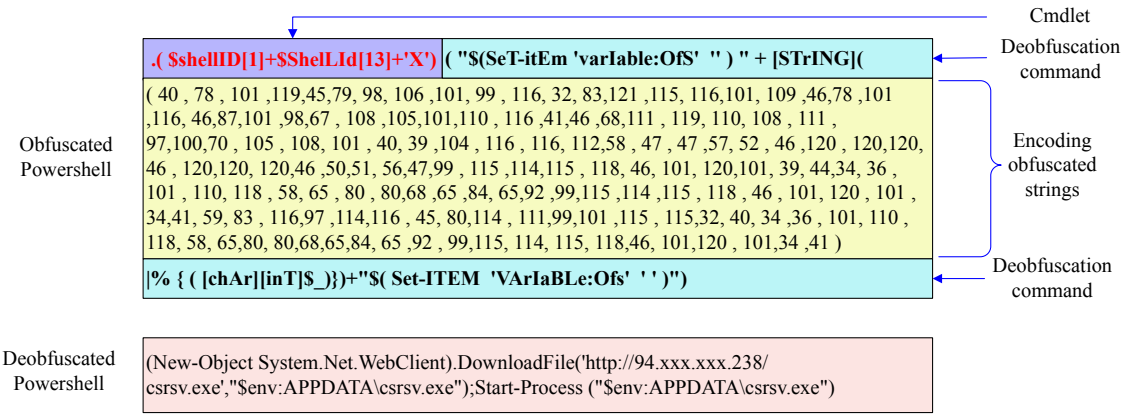

Figure 2. *An example of obfuscation and deobfuscation.*

racy. Therefore, to obtain a high-quality PowerShell script dataset, PowerDetector performs detailed data preprocessing. This part mainly includes deobfuscation, data cleaning, text processing, token processing, and abstract syntax tree processing.

First of all, this paper designs a multi-layer deobfuscation algorithm, which includes abstract syntax trees, regular expressions, and emulation. Among them, abstract syntax tree-based deobfuscation focuses on the structural changes of code. Regular expression-based deobfuscation is used to deal with string confusion. Emulation-based deobfuscation can recover the encoded and compressed code and PowerShell script code with the deobfuscation commands. By doing so, we can effectively extract the commands, functions, objects, variables, parameters, and keywords of the obfuscated script. Figure 2 shows a deobfuscation example. The upper part of Figure 2 shows the obfuscated PowerShell processed by character encoding, including a cmdlet command (i.e., IEX), encoded obfuscated strings, and deobfuscation commands. The results of the lower part show that our deobfuscation algorithm can effectively recover the raw code.

To enhance the performance of PowerDetector, we remove redundant PowerShell scripts and standardize the dataset. In addition, we design three different PowerShell script processing methods, including text processing, token processing, and AST processing, which will be applied in Section 3.3.

Finally, we perform the one-hot encoding to convert the label of the malicious PowerShell family to a unique category, which assigns the current category bit to 1 and other low bits to 0. For example, the Downloader label in Table 2 will be converted to the form [0,1,0,0,0]. Moreover, we divide the whole dataset into training and testing datasets.

3.3 Feature Extraction

Since attackers adopt stealthy or obfuscated techniques to bypass intrusion detection systems in real scenarios, we propose four feature extraction methods to deal with malicious PowerShell scripts and extract fine-grained features across different modalities. Furthermore, they generate corresponding embeddings, namely character embedding, token Embedding, AST embedding, and relationship embedding. In short, PowerDetector is dedicatedly designed to deal with malicious PowerShell scripts with obfuscation and diversity, especially for fileless attacks.

3.3.1 Character Statisitcs-Based Feature Extraction

In general, new features or words often appear in the testing set rather than the training set. However, it is difficult to define a complete dictionary and identify malicious features without prior knowledge. Therefore, we apply the skip-gram model to extract character-level features, which can prevent the out-of-vocabulary (OOV) problem that word-level features would suffer [42]. In addition, this method can reduce the influence of redundant features on the model.

Next, we combine the frequency and context of different PowerShell script characters to generate the character-level embedding. Then, we map each PowerShell script into a character-level sentence vector. This Char2Vec operation can further improve the robustness of PowerDetector.

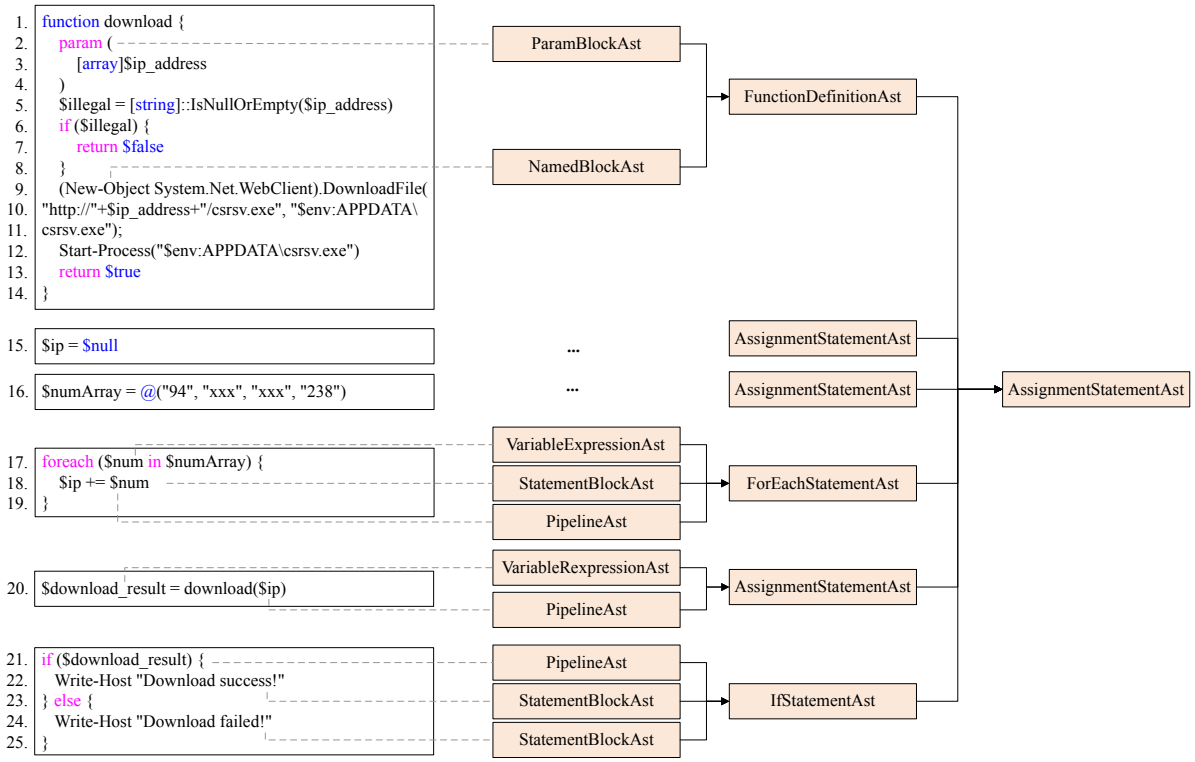

Figure 3. *An example of converting PowerShell code into AST statement nodes*

3.3.2 Statistical Analysis-Based Feature Extraction

To solve the problem that character-level vectors cannot capture syntactic or semantic relations, we design a token-level feature extraction method based on statistical analysis, which can represent the PowerShell corpus by word-level vectors. This method's steps are as follows: (1) Extract token features of each PowerShell script, including commands, variables, functions, keywords, parameters, and strings. (2) Calculate the weights of token features using statistical analysis methods. (3) Select critical features through a sorting algorithm and convert each PowerShell script into a corresponding token sequence vector, as shown in Equation (1).

$$V^{(token)} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nm} \end{bmatrix}. \quad (1)$$

In Equation (1), $V^{(token)}$ represents all sequence vectors eventually generated by Token2Vec. Each line represents a sequence vector of a PowerShell script, and v_{ij} represents a word or token vector generated by word2vec. Taking the PowerShell script of Figure 2 as an example, our method recognizes "New-Obejcet" and "Start-Process" as Commands (i.e., Cmdlets), and recognizes "DownloadFile" as the Member (i.e., a method or function). Similarly, it recognizes "System.Net.WebClient" as the CommandArgument, and recognizes the URL and path (e.g., "$env:APPDATA\csrsv.exe") as string parameters. In short, token features play an important role in malicious PowerShell detection or classification tasks.

3.3.3 Abstract Syntax Tree-Based Feature Extraction

PowerShell uses the abstract syntax tree to sense the semantic expression, which typically represents the logical structure of the script in the form of a multiway tree. Therefore, AST can be used to analyze PowerShell functions by extracting the context structure features of codes. To this end, we propose an abstract syntax tree-based feature extraction method to mine structure-level features, which can better detect malicious families or specific functions (e.g., Downloader, Injector). In particular, this method can represent the PowerShell code of the same malicious family

in a logical structure, thus reducing the impact of confusion or structural transformation.

Figure 3 shows an example of converting PowerShell code into an AST structure. The left part of the figure is the raw code, showing the download and execution process of the remote malware. The AST statement nodes are on the right, extracted by PowerDetector. For example, our method can identify the "download" function as FunctionDefinitionAst, and the foreach loop as ForEachStatementAst. In this way, even if the code structure changes, the AST-based feature extraction method can discover the deeper logic of the code with similar functions.

Algorithm 1 shows the calculation process of AST-based feature extraction and AST2Vec, which includes three steps: (1) Extract the abstract syntax tree nodes of the PowerShell script and use the post-order traversal method to generate the corresponding set S. (2) Establish a set F, and add all non-repeated AST features into F. (3) Use Word2Vec to calculate each AST feature's word vector, convert each PowerShell script into a corresponding sequence vector according to AST nodes and finally generate the vector $V^{(ast)}$.

Algorithm 1. *Feature extraction based on abstract syntax tree.*

Input: $X = \{x_1, x_2, ..., x_n\}$, where x_i is the i^{th} PowerShell script.

Output: $V^{(ast)} = \{v_1, v_2, ..., v_n\}$, where v_i is the i^{th} sequence vector generated by AST-based feature extraction method (i.e., AST2Vec).

$V^{(ast)} \leftarrow \emptyset$, $S \leftarrow \emptyset$, $F \leftarrow \emptyset$, $W \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  $t_i = ExtractAstSequences(x_i)$
  $s_i = PostorderTraversal(t_i)$
  $S.append(s_i)$
end for
$F = BuildFeatureSetFromAst(S)$
// Count all distinct features of AST sequences.
for each $f_k \in F$ **do**
  $w_k = CalculateWordVectors(f_k)$
  $W.append(w_k)$
end for
// Calculate word vectors for each AST node.
for each $s_i \in S$ **do**
  $v_i = GenerateAstEmbedding(s_i)$
  $V^{(ast)}.append(v_i)$
end for
return $V^{(ast)}$

3.3.4 Knowledge Graph-Based Feature Extraction

Most existing feature extraction methods have only focused on key features or weight distribution, while ignoring the fact that the same family has strong semantic relationships. In addition, the above methods require a large amount of prior knowledge, which makes it challenging to mine the correlation among features and fail to identify unknown malicious features. However, knowledge graphs can associate similar or isomorphic features through semantic relationship analysis, thereby enhancing the robustness of the detector and identifying new PowerShell attacks. This paper proposes a feature extraction method based on the knowledge graph, which constructs a knowledge graph through semantic feature analysis and co-occurrence calculation and generates relationship embedding.

$$TokenPairs[j][k] = \begin{cases} v_{jk} + 1 & \exists < f_j, f_k > \\ v_{jk} + 0 & other \end{cases}. \quad (2)$$

Algorithm 2 shows the process of knowledge graph-based feature extraction and Rela2Vec generation. Firstly, the key token features of each PowerShell script are extracted and stored in collection S. Secondly, the co-occurrence relationship of all features is calculated in pairs, and the generated matrix $TokenPairs$ is obtained by Equation (2). Among them, if feature f_j and f_k co-occur, the corresponding relation matrix $TokenPairs[j][k]$ is added by one, otherwise it remains unchanged.

Thirdly, the entity set E is constructed, and the triple (e.g., $\langle f_j, v_{jk}, f_k \rangle$) is used to construct the relation set R. Then, the feature pairs with strong semantic relations in the training set are calculated. For example, $\langle New - Object, DownloadFile \rangle$ and $\langle New - Object, DownloadString \rangle$ often appear in the Downloader family, and $\langle IEX, UseSellExecute \rangle$ and $\langle IEX, Start - job \rangle$ often appear in the Injector family. Finally, the word vector of each relation feature is calculated in turn, and the sentence vector is spliced by Rela2Vec to form a $V^{(kg)}$ embedding.

PowerDetector successfully extracts four levels of

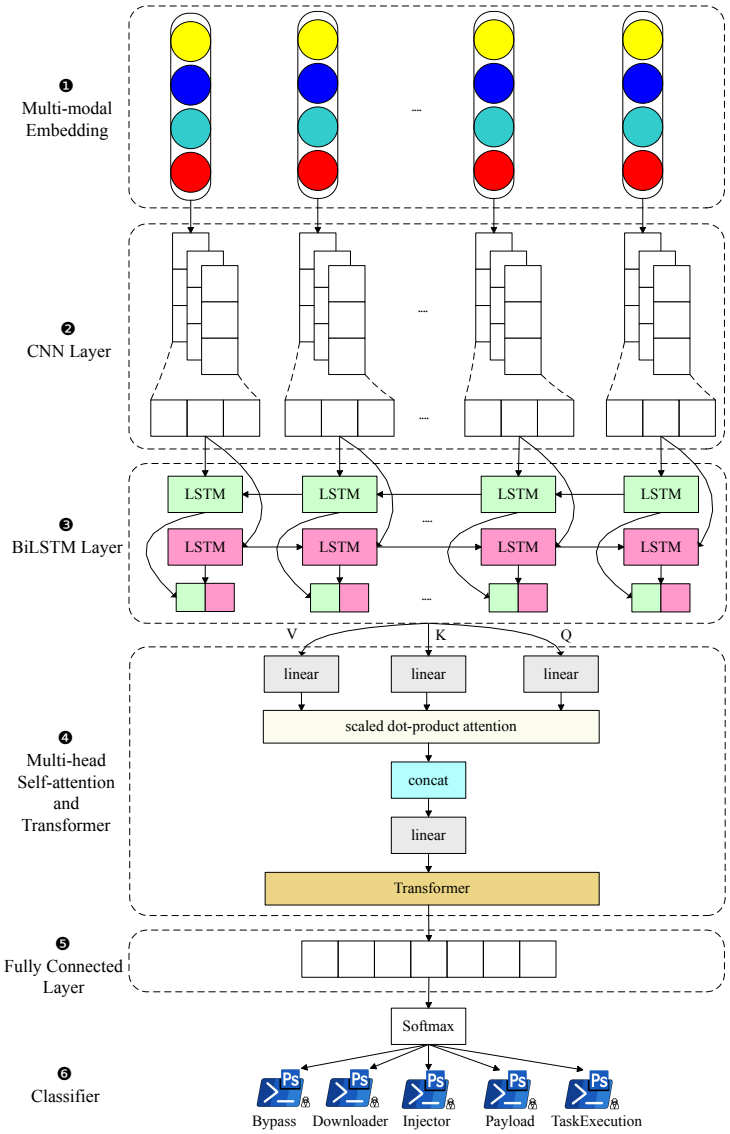

Figure 4. *The overall architecture of the combined model.*

features of PowerShell scripts, namely character-level, token-level, ast-level, and relation-level features.

3.4 Multi-Modal Fusion

In this stage, we design and implement a novel multi-modal feature fusion method, which is different from the traditional feature fusion methods as described in the introduction. This method can fuse the feature vectors extracted from different modalities, and then feed the multi-modal embeddings into a classification model.

The whole process of Multi-modal fusion is shown in Algorithm 3. Firstly, the inputs are four embeddings generated by the previous feature extractions and embedding methods (i.e., Char2Vec, Token2Vec, AST2Vec, and Rela2Vec). Next, different vectors are aligned and spliced. Finally, a high-level joint representation $V^{(multi)}$ is generated as the output. Obviously, our detector can always capture higher-level features through multi-modal fusion, including character-level features, token-level features, AST structure-level features, and semantic relation-level features. Under this fusion scheme, PowerDetector can learn the advantages of different feature extraction methods and extract fine-grained features, thereby better detecting malicious PowerShell family from multiple dimensions, which are not realized by existing work. Through the above processing, we can

Algorithm 2. *Feature extraction based on knowledge graph.*

Input: $X = \{x_1, x_2, ..., x_n\}$, where x_i is the i^{th} PowerShell script.
Output: $V^{(kg)} = \{v_1, v_2, ..., v_n\}$, where v_i is the i^{th} sequence vector generated by KG-based feature extraction method (i.e., Rela2Vec).

$V^{(kg)} \leftarrow \emptyset$, $S \leftarrow \emptyset$, $F \leftarrow \emptyset$, $TokenPairs \leftarrow \emptyset$, $E \leftarrow \emptyset$, $R \leftarrow \emptyset$, $W \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  $s_i = ExtractTokenSequences(x_i)$
  $S.append(s_i)$
  for $j \leftarrow 1$ **to** $len(s_i)$ **do**
    for $k \leftarrow j + 1$ **to** $len(s_i)$ **do**
      $F.append(f_j, f_k)$
      if f_j and f_k are co-occurring **then**
        $TokenPairs[j][k] += 1$
      end if
    end for
  end for
end for
// Calculate co-occurring feature pairs.
$E = CalculateEntity(F)$
$R = CalculateRelationship(F, TokenPairs)$
$M = GetTopFeaturePairs(E, R)$
$S^{'} = SelectSemanticFeatures(S, M)$
// Filter key semantic relation features.
for each $f_i \in F$ **do**
  $w_i = CalculateWordVectors(f_i)$
  $W.append(w_i)$
end for
for each $s_i \in S^{'}$ **do**
  $v_i = GenerateKGEmbedding(s_i)$
  $V^{(kg)}.append(v_i)$
end for
return $V^{(kg)}$

improve the robustness and performance of the model, which significantly contributes to the following classification tasks.

3.5 Model Construction

We construct the first effective and combined model based on multi-head self-attention, transformer, and CNN-BiLSTM. The overall architecture of the model is shown in Figure 4, which mainly includes six parts: (1) multi-modal embedding; (2) CNN layer; (3) BiLSTM layer; (4) multi-head self-attention and transformer; (5) full connection layer; (6) classifier.

Algorithm 3. *The process of multimodal feature fusion.*

Input: $V^{(char)}$, $V^{(token)}$, $V^{(ast)}$, $V^{(kg)}$
Output: $V^{(multi)}$

$V^{(multi)} \leftarrow \emptyset$
for $i \leftarrow 1$ **to** n **do**
  $Alignment(v_i^{(char)}, v_i^{(token)}, v_i^{(ast)}, v_i^{(kg)})$
  $v_i^{(multi)} = Concat(v_i^{(char)}, v_i^{(token)}, v_i^{(ast)}, v_i^{(kg)})$
  $V^{(multi)}.append(v_i^{(multi)})$
end for
return $V^{(multi)}$

Multi-modal embedding. We process and map the PowerShell script corpus into multi-modal vectors via the previous method. The multi-modal embedding is composed of Char2Vec, Token2Vec, AST2Vec, and Rela2Vec, which integrates the features of four layers and serves as the input for the subsequent neural network model.

CNN layer. Convolutional neural network (CNN) consists of convolutional layers and pooling layers [43]. Since convolution kernels can capture features between multiple consecutive words, convolution layers are used to extract local key features of sequences in PowerDetector. We construct a 3-layer CNN model, and the calculation equation of convolution operation is as follows:

$$c_i^t = f(v_i^{(multi)} \times w_t + b_t), \tag{3}$$

where $v_i^{(multi)}$ represents the fusion feature vector as the input, and $v_i^{(multi)} \in R^{n \times D}$, n represents the number of features, and D represents the dimension of word vectors. w_t is a convolution kernel, and b_t is the corresponding bias. Also, f represents an activation function (i.e., ReLU), and the final generated feature is denoted as c_i^t. We then perform the pooling layer processing and select the maximum pooling method to reduce the computation of the model by sampling the vector, which can preserve the key features of PowerShell. The final output vector is denoted as s.

BiLSTM layer. Long short-term memory (LSTM) is a variant of recurrent neural network (RNN), which protects and controls cell state through gate structure (i.e., forget gate, input gate, and output gate) [44]. Note

that LSTM can retain essential information, forget secondary information, and solve the challenge that traditional neural networks lose context features [45]. The BiLSTM includes a forward LSTM and a backward LSTM (step ③ in Figure 4), which can extract sequence features from two directions. For example, "DownLoadFile" and "New-Object" often appear in the Downloader family, and there is a certain distance between them. The corresponding output vectors of the forward LSTM layer and backward LSTM layer are $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$, as shown in Equation (4).

$$\begin{cases} \overrightarrow{h_t} = f(w_1 \cdot s_t + w_2 \cdot \overrightarrow{h_{t-1}}) \\ \overleftarrow{h_t} = f(w_3 \cdot s_t + w_5 \cdot \overleftarrow{h_{t-1}}), \\ y_t = g(w_4 \cdot \overrightarrow{h_t} + w_6 \cdot \overleftarrow{h_t}) \end{cases} \tag{4}$$

where s_t is the input vector at time t, w_1 to w_6 are weight parameters corresponding to the BiLSTM structure. Besides, f and g are the activation functions (i.e., sigmoid and tanh), and y_t is the output vector of the BiLSTM model, which integrates feature information in two directions.

In addition, the BiLSTM model can efficiently learn the semantic features of the abstract syntax tree and knowledge graph structure among the four modalities. Due to the tree structure's position order and the graph structure's semantic relationship among the above features, they can be abstractly represented as features similar to time series, which can be better learned by the LSTM model [46]. Also, the LSTM model can retain essential information from AST-level and KG-level features and forget useless information through forgetting gates, output gates, input gates, and forgetting units. Subsequent experiments will further verify the effectiveness of the BiLSTM model. In summary, PowerDetector utilizes the BiLSTM model to capture contextual dependencies and avoid the gradient explosion and vanishing.

Multi-head self-attention and transformer. The attention mechanism can highlight the key information of the sequence. The transformer can calculate the correlation among features and improve the processing ability of the large-scale data [47, 48]. To better capture the local features of the input sequence and filter essential information, we design a multi-head self-attention and transformer model in PowerDetector. Our model can capture multiple layers of semantic information and long-distance dependencies, thereby improving the model's recognition effect. In particular, the transformer model is very suitable for multi-modal embedding, and the distance of features in the sequence can be ignored due to the location relation. The proposed model can better extract the global dependency information and apply it to different scenarios, even identifying unknown types of attacks. As shown below, self-attention utilizes scaled dot-product attention to calculate the output of the query (**Q**), key (**K**), and value (**V**) of an input vector sequence.

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}. \tag{5}$$

In Equation (5), **Q** is a query matrix, **K** is a key matrix, and **V** is a value matrix, all of which are calculated by the dot product between the state matrix and the corresponding weight matrix. The dimension of the matrix is represented by d_k. The softmax function can normalize the dot product of **Q** and **K**, and then multiply it with **V** to obtain the sum of the multi-head attention weights. Next, the multi-head attention mechanism is used to repeat the operation n times and calculate the attention score to obtain the contextual features in multiple subspaces. Finally, the results are spliced into the output of multi-head self-attention, as shown in Equation (6).

$$MHA(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = Concat(head_1, \cdots, head_n)\mathbf{W}^o, \tag{6}$$

where $\mathbf{W}_o$ is the output weight matrix, connecting n parallel heads (i.e., $head_1$ to $head_n$), and the final result is $MHA(\mathbf{Q}, \mathbf{K}, \mathbf{V})$. Each $head_j$ can be represented as follows:

$$head_j = Attention(\mathbf{Q}\mathbf{W}_j^Q, \mathbf{K}\mathbf{W}_j^K, \mathbf{V}\mathbf{W}_j^V), \tag{7}$$

where $\mathbf{W}_j^Q$, $\mathbf{W}_j^K$, and $\mathbf{W}_j^V$ are different weights, and linear transformation processing is performed on three matrixes of **Q**, **K**, and **V**. The final output is passed to a transformer model. Overall, PowerDetector effectively utilizes multi-head self-attention and transformers to capture rich semantic knowledge of features and

improve the model's adaptability to different scenarios. In addition, the first four stages of Figure 4 implement the weight calculation of feature vectors, thereby better identifying PowerShell attacks.

Full connected layer and classifier. The fully connected (FC) layer is often used as a classifier in neural network models, which maps learned distributed feature representations to the sample label space. PowerDetector uses the softmax function to build a classifier and identify five families of PowerShell malicious scripts (i.e., Bypass, Downloader, Injector, Payload, and TaskExecution). In short, we successfully construct a hybrid model based on multi-head self-attention, transformer, and CNN-BiLSTM to identify PowerShell attacks through the six steps in Figure 4.

IV. EVALUATION

This paper focuses on detecting malicious PowerShell families based on multi-modal fusion and deep learning, so the performance of different modalities and models will be compared in detail. To this end, we have designed and implemented an extensive evaluation framework to compare PowerDetector and the baseline approaches. In the remainder of this section, we first introduce the dataset and experimental setup. Then, we analyze the differences between PowerDetector and state-of-the-art models to highlight the effectiveness and robustness of the proposed detector. Finally, we compare the performance of these methods, and discuss and prove that multi-modal fusion can accurately extract fine-grained features.

4.1 Dataset

Since PowerShell lacks a bench-marked dataset, this paper combines multiple data sources to construct the dataset shown in Table 2. The data source mainly consists of six parts: (a) the open-source Powershell dataset from unit42 security research [49]; (b) malicious PowerShell code collected by Github; (c) malicious PowerShell scripts generated by the open API of HyBrid analysis service [50]; (d) PowerShell samples extracted from the AnyRun online sandbox [51]; (e) PowerShell samples constructed using Metasploit, PowerSploit, and CobaltStrike penetration tools [34]; (f) PowerShell samples used by APT organizations (e.g., APT29, Lazarus, and OceanLotus) [14].

Generally, PowerShell scripts are always utilized in the installation, payload delivery, persistence, and execution phases of real-time attacks [16, 19, 21]. To ensure the fairness and effectiveness of the experiment, we analyze the techniques and tactics commonly used by PowerShell in the MITRE ATT&CK framework [52] and APT attacks [13, 14]. Then, we divide PowerShell into five malicious families according to its functions, including Bypass, Downloader, Injector, Payload, and TaskExecution. In addition, we use VirusTotal, ThreatBook, Windows Defender, and HuoRong security software to scan PowerShell scripts in turn, and collect feedback on malware family results. Next, we calculate the maximum possibility that each script belongs to the malicious family and label it as the corresponding family combined with the function. Also, the generated dataset is shown in Table 2.

Table 2. *Malicious PowerShell script dataset.*

No	Label	Total	Train set	Test set
0	Bypass	200	124	76
1	Downloader	800	452	348
2	Injector	1380	799	581
3	Payload	270	148	122
4	TaskExecution	350	200	150

In Table 2, there are 1723 training samples and 1277 testing samples, which are randomly grouped and cover the five functional malware families.

Bypass. Bypass is an attack that can bypass the firewalls or intrusion detection systems through obfuscation (e.g., character segmentation) or encoding (e.g., ASCII code). Bypass is usually in the stage of payload delivery. Since it is often combined with other functions, the whole dataset contains 200 samples.

Downloader. Downloader is an attack that downloads malicious payloads or malware from remote servers, including 800 samples.

Injector. Injector is an attack that utilizes the design flaws of hosts or websites to achieve illegal credentials or permission access. The sub-categories of our dataset include ShellCode Inject, Trojan/Injector, and Unicorn, with a total of 1380 samples.

Payload. Payload is generated by penetration tools, such as PowerSploit, Nishang, Empire, and CobaltStrike, which can implement privilege escalation, information collection, and lateral movement. This kind

of our dataset consists of 270 samples.

TaskExecution. TaskExecution is designed to execute malicious tasks or start jobs, such as keylogging, executing COM tasks, persistence, and command control, including 350 samples in the dataset.

Note that the test set contains 751 unknown attack samples whose sub-categories do not appear in the train set and cover five families (e.g., Backdoor/Meterpreter.as and Trojan/Injector.pd). We compare the robustness of PowerDetector and the existing models through unknown attack detection in the following Section 4.6. Through the above processing, we successfully constructe a PowerShell dataset with typical malicious families. Next, we mainly focus on the experimental evaluation of PowerDetector.

4.2 Evaluation Metrics

To evaluate the performance of the proposed PowerDetector, we adopt precision, recall, F_1-score, and accuracy to conduct comparative experiments, which are also standard evaluation metrics in classification tasks. The calculation process of the i^{th} malicious family is as follows.

$$Precision_i = \frac{TP_i}{TP_i + FP_i}. \tag{8}$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i}. \tag{9}$$

$$F_1 - score_i = \frac{2 \times Precision_i \times Recall_i}{Precision_i + Recall_i}. \tag{10}$$

$$Accuracy_i = \frac{TP_i + TN_i}{TP_i + TN_i + FP_i + FN_i}. \tag{11}$$

Since our study is a multi-classification task and the distribution of sample categories is uneven, we use the weighted average of the precision, recall, F_1-score, and accuracy for fairness. Moreover, the experimental results are the average of ten classification results, thereby reducing the noise influence of a possible abnormal result. The final four evaluation indicators are shown in equations (12) to (15), where weight w_i is the proportion of the number of the i^{th} label samples to the total number of samples.

$$Precision = \sum_{i=1}^{N} Precision_i \times w_i. \tag{12}$$

$$Recall = \sum_{i=1}^{N} Recall_i \times w_i. \tag{13}$$

$$F_1 - score = \sum_{i=1}^{N} F_1 - score_i \times w_i. \tag{14}$$

$$Accuracy = \sum_{i=1}^{N} Accuracy_i \times w_i. \tag{15}$$

4.3 Experimental Setup

In this section, we implement various models by Python (version 3.7), TensorFlow, Keras, and Sklearn. All experiments were conducted on a dedicated computer with an Intel Core i7-8700K CPU, one GTX 1080Ti GPU, 64 GB memory, and 500 GB SSD. To measure the improvements achieved by PowerDetector, we establish two baselines:

Machine learning-based models. We use machine learning algorithms to construct models and identify malicious PowerShell scripts, including logistic regression (LR), random forest (RF), support vector machine (SVM), and K-nearest neighbor (KNN).

Deep learning-based models. We choose TensorFlow and Keras to construct deep learning models and compare them with PowerDetector. The existing models include CNN, TextCNN, BiLSTM, BiGRU, and attention-based CNN-BiLSTM.

For a fair comparison, we perform the same activation function (i.e., ReLU) and normalization method on PowerDetector and the baseline models. Besides, the number of convolutional neurons in the CNN model is 128. The number of hidden dimensions in the forward and backward directions of BiLSTM and BiGRU models is 128. The Adam optimizer is utilized to solve the minimum loss, and the initial learning rate of each model is 0.0005. Moreover, a dropout mechanism (with a parameter of 0.3) is introduced to prevent over-fitting.

In particular, we have implemented different models (including single-modal and multi-modal) to highlight

Table 3. *Performance comparison of various PowerShell detection models.*

Feature type	Model	Precision	Recall	F_1-score	Detection Time
Token-level	RF	0.9088	0.8676	0.8723	1.66
	CNN	0.9241	0.8974	0.9002	10.32
	BiLSTM	0.9099	0.9069	0.9076	25.98
	BiGRU	0.9164	0.9021	0.9041	17.58
	Transformer	0.9234	0.9107	0.9123	**79.57**
	CNN-BiLSTM+ATT	**0.9298**	**0.9123**	**0.9142**	36.13
Character-level	RF	0.9017	0.8567	0.8610	2.61
	CNN	0.9107	0.8808	0.8826	17.49
	BiLSTM	0.9134	0.9036	0.9037	42.71
	BiGRU	0.9180	0.8966	0.8989	29.43
	Transformer	0.9077	0.9029	0.9053	**85.37**
	CNN-BiLSTM+ATT	**0.9189**	**0.9076**	**0.9081**	60.76
AST-level	RF	0.9042	0.8786	0.8807	1.94
	CNN	0.9144	0.8998	0.9019	11.66
	BiLSTM	0.9145	0.9107	0.9126	34.01
	BiGRU	0.9178	0.9045	0.9092	28.53
	Transformer	0.9141	0.9092	0.9116	**59.28**
	CNN-BiLSTM+ATT	**0.9272**	**0.9139**	**0.9144**	48.84
KG-level	RF	0.9088	0.8676	0.8723	1.03
	CNN	0.9294	0.8998	0.9025	10.25
	BiLSTM	0.9284	0.9025	0.9054	41.03
	BiGRU	0.9256	0.9013	0.9046	26.34
	Transformer	0.9127	0.9115	0.9121	**85.81**
	CNN-BiLSTM+ATT	**0.9313**	**0.9123**	**0.9139**	70.60
Related Work	Fang et al. [22]	0.9277	0.8843	0.9055	4.02
	Ruscak et al. [23]	0.9167	0.8798	0.8979	2.88
	FC-PSDS [35]	0.9288	0.9076	0.9181	22.33
PowerDetector (Multi-modal+Transformer-CNN-BiLSTM)		**0.9402**	**0.9358**	**0.9374**	**106.45**

the advantages of multi-modal fusion and deep learning. The detailed experiments will be introduced in Section 4.4 to 4.5. Then, we have performed semantic feature relationship analysis using Algorithm 2 and the gephi tool to reflect the effectiveness of the multi-modal embedding. Obviously, we have built a robust evaluation framework and performed detailed experiments to prove that PowerDetector can better detect malicious PowerShell families and obtain the best F_1-score and accuracy.

4.4 Evaluation Results

The experimental comparison results of various models are shown in Table 3. To demonstrate the performance of PowerDetector, we compare traditional machine learning models (e.g., SVM) and existing deep learning models (e.g., CNN, BiLSTM, BiGRU, Transformer, and CNN-BiLSTM+ATT). Meanwhile, we evaluate the results of four modalities (i.e., token, character, AST, and KG) and multi-modal scenarios in detail. In particular, most malicious PowerShell detection methods are binary classification tasks (i.e., judging whether PowerShell scripts are malicious or benign), and their source codes are not open source. Therefore, to better evaluate the performance of Pow-

erDetector, this paper reconstructs three models based on the description and methods of related works and conducts a comparison experiment of PowerShell malicious family classification on the dataset shown in Table 2. The three methods include Fang et al. [22], Rusak et al. [23], and FC-PSDS [35]. Also, to effectively measure the time cost and algorithm complexity of the model, this paper adds an important indicator for evaluating malicious script detection. Column 6 of Table 3 gives the detection times for different models in seconds.

As shown, our PowerDetector can achieve the best performances on precision, recall, and F_1-score, which are 0.9402, 0.9358, and 0.9374, respectively. Compared with the attention-based CNN-BiLSTM model, which performs well in traditional methods, the F_1-score of PowerDetector is improved to a certain extent, including a 2.32% increase in token level, a 2.93% increase in character level, a 2.30% increase in AST level, and a 2.35% increase in KG level. Also, the F_1-score of our method increased by 3.19%, 3.39%, and 1.93% compared with that of Fang [22], Ruscak [23], and FC-PSDS [35], respectively. Moreover, the detection time of PowerDetector for the test set is 106.45 seconds, and it can effectively detect more malicious PowerShell scripts of different families. The Random Forest algorithm (i.e., machine learning model) has the lowest detection time, and the single models with relatively high detection time are BiLSTM and Transformer, which have a more complex neural network structure. In short, our model's entire detection time efficiency is in the upper-middle range. Compared with the improvement of the F_1-score, the time cost is within an acceptable range, only slightly higher than other deep learning models, and does not show an exponential increase. Also, all deep learning models occupy nearly the same memory space.

Based on the results, we prove that PowerDetector can effectively identify malicious PowerShell functional families, and the multi-modal fusion algorithm can capture fine-grained features with four dimensions. In addition, our transformer-based CNN-BiLSTM model can improve the classifier's performance and effectively capture long-distance dependencies and semantic relationships. To our best knowledge, the proposed model in this paper can achieve the best F_1-score for identifying malicious families of PowerShell scripts.

Table 4. *The identified results of five families by PowerDetector.*

Malicious family	Precision	Recall	F_1-score
Bypass	0.6064	0.7500	0.6706
Downloader	0.9587	0.9339	0.9461
Injector	0.9931	0.9966	0.9948
Payload	0.8833	0.8689	0.8760
TaskExecution	0.9078	0.8533	0.8797
Weighted avg	**0.9402**	**0.9358**	**0.9374**

The classification results of each malicious PowerShell family based on PowerDetector are shown in Table 4. The performance of Injector and Downloader labels is better, and their F_1-score are 0.9948 and 0.9461, respectively. Unfortunately, the Bypass identification results are poor because the number of real-world attack samples is too small (only 200) to capture detailed feature information and associations, and the sub-categories are complex and changeable. In contrast, the other four categories have a larger number of scripts, resulting in richer semantic information and co-occurrence of various dimensional features. So far, PowerDetector can correctly identify more PowerShell scripts, and the final average weight F_1-score of the five families is 0.9374. In the future, we will further expand the Bypass sample to improve the detection performance of this family.

In particular, the PowerShell detection lacks a bench-marked dataset with the functional family. Moreover, there are few related studies on the multi-classification task, and some binary classification codes have not been open-sourced. Therefore, comparison models of PowerShell detection in this paper are realized by combining existing deep learning algorithms and ideas to reconstruct the code. All experiments are implemented in the same dataset and environment for fairness.

4.5 Compare Multi-Modal and Single-Modal

To prove the effectiveness of PowerDetector's four feature extraction algorithms and multi-modal fusion method, a detailed evaluation of multi-modal and single-modal is carried out in this paper. The analysis results are shown in Table 5.

As shown, four machine learning algorithms (i.e.,

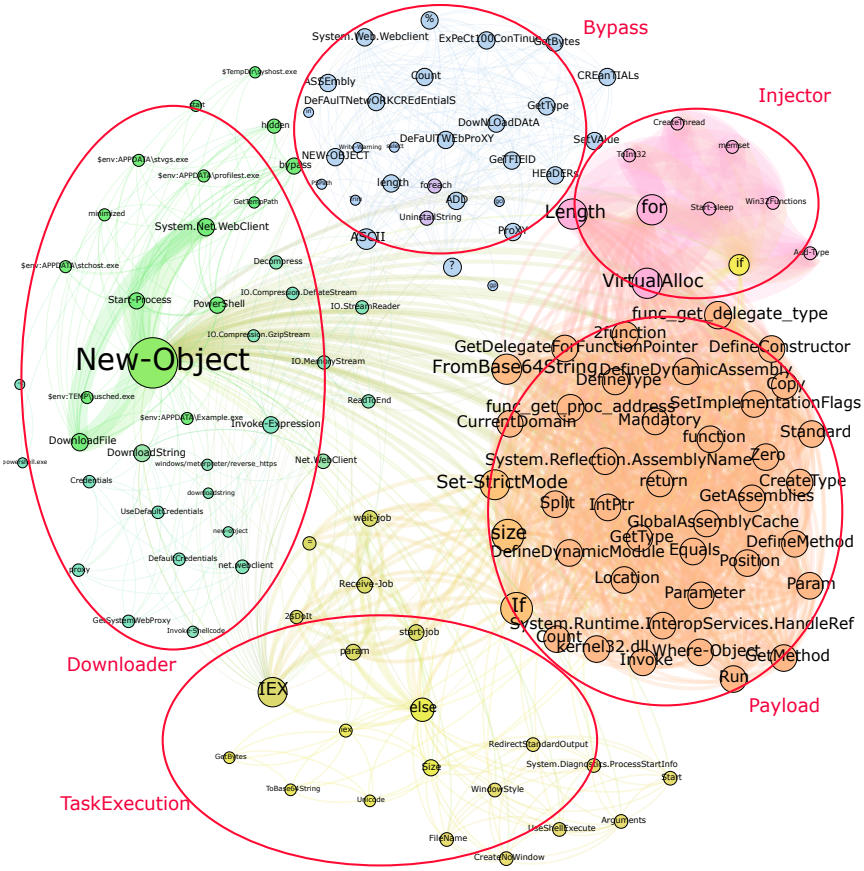

Figure 5. *The semantic token feature relationships of five malicious families are extracted by knowledge graph.*

Table 5. *Detailed performance comparison of single-modal and multi-modal.*

Model	Single-modal								Multi-modal	
	Token-level		Character-level		AST-level		KG-level			
	F_1	Acc	F_1	Acc	F_1	Acc	F_1	Acc	F_1	Acc
LR	0.8727	0.8629	0.8496	0.8528	0.8661	0.8700	0.8646	0.8559	0.8895	0.8857
RF	0.8723	**0.8676**	**0.8610**	**0.8567**	**0.8807**	**0.8786**	0.8723	0.8676	**0.9017**	**0.8943**
SVM	0.8764	0.8661	0.8527	0.8519	0.8755	0.8786	**0.8771**	0.8676	0.8934	0.8912
KNN	0.8706	0.8669	0.8554	0.8536	0.8644	0.8637	0.8741	**0.8715**	0.8804	0.8771
CNN	0.9002	0.8974	0.8826	0.8808	0.9019	0.8998	0.9025	0.8998	0.9153	0.9115
TextCNN	0.9049	0.9013	0.9012	0.8966	0.9083	0.9076	0.9036	0.9005	0.9186	0.9178
BiLSTM	0.9076	0.9069	0.9037	0.9036	0.9126	0.9107	0.9054	0.9025	0.9226	0.9209
BiGRU	0.9041	0.9021	0.8989	0.8966	0.9092	0.9045	0.9046	0.9013	0.9205	0.9201
Transformer	0.9123	0.9107	0.9053	0.9029	0.9116	0.9092	0.9121	0.9115	0.9224	0.9178
CNN-BiLSTM+ATT	**0.9142**	**0.9123**	**0.9081**	**0.9076**	**0.9144**	**0.9139**	**0.9139**	**0.9123**	**0.9262**	**0.9209**
Transformer-CNN-BiLSTM	**0.9236**	**0.9225**	**0.9170**	**0.9169**	**0.9248**	**0.9233**	**0.9204**	**0.9201**	**0.9374**	**0.9358**

LR, RF, SVM, and KNN), six deep learning algorithms (i.e., CNN, TextCNN, BiLSTM, BiGRU,

Transformer, and CNN-BiLSTM+ATT), and the proposed model (i.e., Transformer-CNN-BiLSTM) can achieve the best F_1-score and accuracy in multi-modal scenarios, which are better than the single-modal results. Among machine learning algorithms, the best multi-modal model is RF, whose F_1-score and accuracy are 0.9017 and 0.8943. Among them, the F_1-score of RF model at the multi-modal level is 2.87% higher than the average of the four machine learning algorithms at the token level. Similarly, the F_1-score of RF model is 4.70%, 3.00%, and 2.97% higher than the average of the character level, AST level, and KG level. In addition, among existing deep learning algorithms, the best multi-modal model is CNN-BiLSTM+ATT, whose F_1-score and accuracy are 0.9262 and 0.9209. Compared with six deep learning algorithms at token level, character level, AST level, and KG level, the F_1-score values of CNN-BILSTM+ATT are improved by 1.90%, 2.62%, 2.77%, and 1.92%, respectively. More specifically, the combined model based on multi-head self-attention, transformer, and CNN-BiLSTM in this paper has also been compared with different modalities. As a result, our detector can obtain a 0.9374 F_1-score and a 0.9358 accuracy in multi-modal scenarios, which are 1.60% and 1.51% higher than the average results of the four single modalities.

All in all, the multi-modal fusion method with four feature extraction algorithms and four embeddings (i.e., Char2Vec, Token2Vec, AST2Vec, and Rela2Vec) can improve the effect of malicious PowerShell family detection. Also, PowerDetector can obtain the best result, which means that multi-level semantic feature extraction and multi-modal feature fusion are feasible. Moreover, we conduct pairwise and three-three combinations from the features of the four modalities, respectively. However, the F_1-score and accuracy are slightly improved, and the results are lower than PowerDetector. Therefore, Table 5 mainly compares the experimental results of single-modal and four modalities fusion.

In particular, through multi-modal and single-modal comparative analysis, we find that the PowerDetector's components have different detection effects on the semantic features of different modalities. Among them, the BiLSTM model can better identify the semantic features of the abstract syntax tree and the knowledge graph, which have a unique position sequence or semantic structure (i.e., the structure of the tree or graph); the transformer can effectively recognize the token sequence and the knowledge graph semantic features; the CNN model can extract key features and speed up the detection time of our model. As a result, PowerDetector fits very well with the four modal features and four embeddings proposed in this paper, enabling our model to achieve the best F_1-score and accuracy.

We have designed and implemented four feature extraction algorithms to extract character features, token (or function) features, abstract syntax tree structure features, and semantic relationship features of PowerShell scripts. The previous experiments have proved that PowerDetector can achieve the best results. Next, to make sure that PowerDetector can extract semantic relationships of various features, we further analyze the knowledge graph-based feature extraction method. We use algorithm 1 to extract the structure relationship of the abstract syntax tree and use algorithm 2 to extract the semantic relationship of the knowledge graph. The corresponding co-occurrence knowledge graph of semantic features is shown in Figure 5. Among them, nodes represent key token features (e.g., commands, functions, objects, variables, and keywords) of PowerShell scripts, and edges represent the co-occurrence relationship between two features. Besides, the larger node means the feature plays a more important role, and the thicker edge means the semantic relationship is closer between the two features.

In addition, we find that the entire knowledge graph spreads from the center to the surrounding areas through further analysis. These nodes are effectively clustered into five clusters, corresponding to the five functional categories of Bypass, Downloader, Injector, Payload, and TaskExecution. In Figure 5, the left part of the green node group is the Downloader family, the bottom yellow node group is the TaskExecution family, the top blue node group is the Bypass family, the upper right pink node group is the Injector family, and the lower right red node group is the Payload family. PowerDetector leverages these nodes with rich semantic associations to improve the detection effect. Moreover, we extracted critical features with close relationships in each family and formed the results shown in Table 6. For example, "New-Object", "DownloadFile", "DownloadString", and "System.Net.WebClient" often appear to-

Table 6. *Key semantic features extracted by the knowledge graph.*

Malicious family	Typical relationship feature
Bypass	IEX, ASCII, UninstallString, DownloadString, Assembly, bypass, hidden, minimized, DownloadFile, New-Object, ProXY, length
Downloader	New-Object, IEX, DownloadString, DownLoadFile, Start-Process, UseDefaultCredentials, ReadToEnd, System.Net.WebClient, bypass, Invoke-Expression, IO.MemoryStream, Decompress
Injector	VirtualAlloc, Add-Type, Start-sleep, CreateThread, Length, memset, Win32Functions, ToInt32, GetBytes, ToBase64String, IEX, kernel32.dll
Payload	FromBase64String, New-Object, Set-StrictMode, func_get_proc_address, Location, UseShellExecute, size, System.Reflection.AssemblyName, VirtualAlloc, GetType, start-job, Receive-Job
TaskExecution	IEX, start-job, Receive-job, WindowsStyle, FileName, CreateNoWindow, UseShellExecute, wait-job, System.Diagnostics.ProcessStartInfo, ADD, ProXY, Get-Process

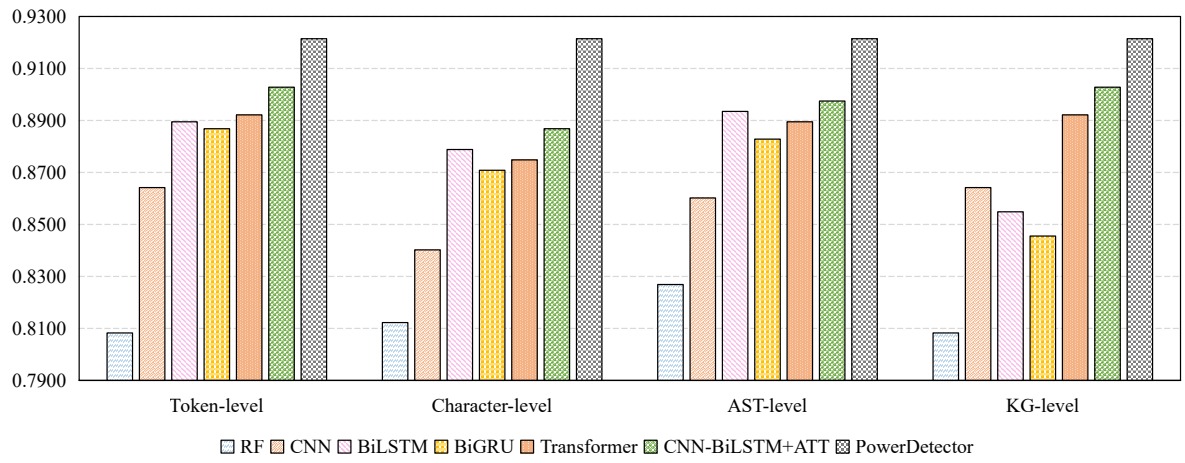

Figure 6. *The accuracy of different models to identify unknown attack of PowerShell.*

gether in the Downloader family, which can carry out the attack that downloads remote malicious codes. Also, "VirtualAlloc", "Start-sleep", "CreateThread", "Win32Functions", and "memset" often appear together in the Injector family. All in all, through the analysis of feature semantic relations, we demonstrate that the proposed method can effectively extract fine-grained features and achieve feature-family mapping. Thus, our feature extraction algorithms and the multi-modal fusion method outperform state-of-the-art methods, and the multi-modal embedding plays an important role in malicious PowerShell detection.

4.6 Performance of PowerDetector

Finally, we evaluate the performance of PowerDetector. On the one hand, we evaluate the robustness and scalability of different models by comparing the accuracy of identifying unknown attacks in the test set. On the other hand, we compare the learning ability of various models to detect malicious families through the loss and accuracy curve of the train set.

As mentioned in Section 4.1, the test set included 751 unknown attack samples whose subcategories do not appear in the train set. Among them, Bypass includes the "VirTool/PS.Obfuscator" and "Remove AV" sub-categories; Downloader includes "Downloader IEXDS", "Downloader Proxy", and "Downloader Kraken" sub-categories; Injector

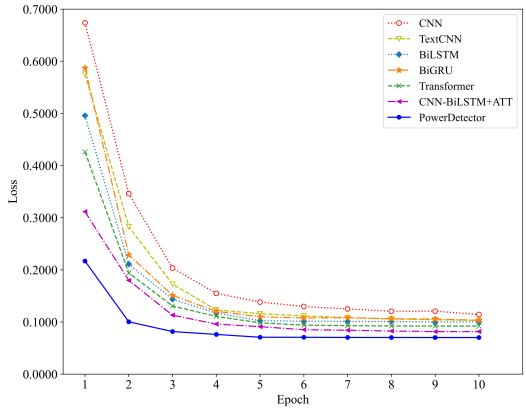

Figure 7. *The loss curve of different models.*

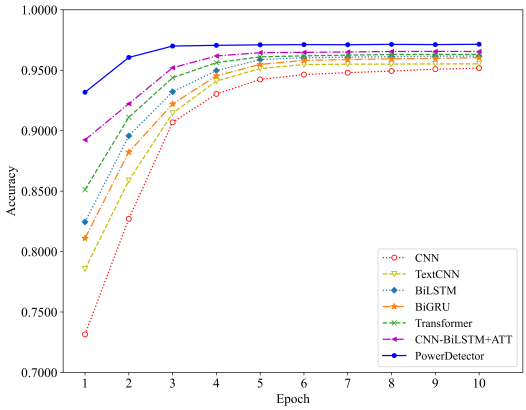

Figure 8. *The accuracy curve of different models.*

includes "Trojan/Injector.pd", "PowerShell/Ploty.F", and "Unicorn"; Payload includes "Powerfun Reverse/Bind", "Backdoor/Meterpreter.as", and "Veil Stream" sub-categories; TaskExecution includes "VB Task", "BITSTransfer", "Scheduled Task COM", and "DynAmite Launcher". Next, we compare the experimental results of PowerDetector and various models in different modalities in detail, as shown in Figure 6. Specifically, PowerDetector is a Transformer-CNN-BiLSTM model built in multi-modal scenarios. The final accuracy of PowerDetector is 0.9214, which is better than RF, CNN, BiLSTM, BiGRU, Transformer, and CNN-BilSTM+ATT models. More specifically, PowerDetector's accuracy is 10.75% higher than the average value of the four single modalities of RF. Similarly, other deep learning models also calculate the average accuracy of the four single modalities. The accuracy of our detector is 6.42%, 4.23%, 4.99%, 3.43%, and 2.40% higher than CNN, BiLSTM, BiGRU, Transformer, and CNN-BiLSTM+ATT. In short, the above results effectively prove that PowerDetector can better detect unknown attacks and is robust.

Table 7. *The experimental results of PowerShell deobfuscation.*

Model	Deobfuscated
AST Layer	66.97%
Emulation Layer	49.6%
RE Layer	48.03%
PSDEM [15]	52.5%
PowerDetector	**83.23%**

In order to systematically reflect the performance of PowerDetector, we use the train set to evaluate the learning ability of the proposed model and obtain the error change curve as shown in Figure 7 and the accuracy change curve as shown in Figure 8. In Figure 7, the horizontal axis is the training period, and the vertical axis is the overall error of model training. As shown, the error of PowerDetector converges faster with the training period, and the curve quickly flattens in the second epoch, achieving the best and fastest convergence effect. This experiment demonstrates the fact that the proposed model is feasible to apply the malicious PowerShell family detection. Figure 8 shows that PowerDetector's training process is more stable and achieves higher accuracy in fewer batches.

Finally, to demonstrate that PowerDetector has a good deobfuscation effect, we compared the multi-layer deobfuscation algorithm proposed in this paper with the existing work. The deobfuscation results are shown in Table 7. The PowerDetector has a deobfuscation success rate of 83.23%, which outperforms existing single-level deobfuscation algorithms. Compared with the deobfuscation algorithm based on the abstract syntax tree, simulated execution, and regular expression, the success rate of our model is increased by 16.26%, 33.63%, and 35.20%. In addition, the success rate of our method is 30.73% higher than that of PSDEM [15]. This experiment proves the effectiveness of the multi-layer deobfuscation algorithm designed in this paper, which includes abstract syntax

trees, regular expressions, and simulated execution. Through multi-layer superposition, it can better eliminate the obfuscation technology of PowerShell malicious code. In addition, since this paper focuses on the malicious PowerShell scripts families or behavior detection, only a brief description of the deobfuscation is given in the last part of the evaluation.

In summary, the experimental results of this part show that the designed model in this paper based on multi-head self-attention, transformer, and CNN-BILSTM can effectively capture semantic features and long-distance dependencies, so as to accurately identify PowerShell categories.

V. CONCLUSION

In this paper, we designed and implemented PowerDetector, a novel malicious PowerShell script detector based on multi-modal semantic fusion and deep learning. To overcome the challenge that existing models fail to achieve fine-grained features and semantic relationships, we proposed four feature extraction methods and four embeddings (i.e., Char2Vec, Token2Vec, AST2Vec, and Rela2Vec), which extract critical features from character level, token (or function) level, abstract syntax tree level, and semantic knowledge graph level. Furthermore, to enhance the robustness and accuracy of the model, we designed the first multi-modal fusion algorithm and a combined deep learning model to detect different families of PowerShell attacks. In particular, our classifier consists of multi-head self-attention, transformer, and CNN-BILSTM. Moreover, we applied PowerDetector to identify different malicious families and evaluated them using a dataset created by ourselves which contains 3000 malicious PowerShell scripts, covering five families of real-world attacks (i.e., Bypass, Downloader, Injector, Payload, and TaskExecution). Our evaluation shows that PowerDetector has the best performance in malicious PowerShell families detection compared with the state-of-the-art models, with a 0.9402 precision, a 0.9358 recall, and a 0.9374 F_1-score. Specifically, experimental results demonstrate that PowerDetector's multi-modal embedding and deep learning model can accurately detect various obfuscated and stealth PowerShell scripts and even identify more unknown attacks. In short, we conclude that PowerDetector can effectively identify malicious PowerShell functional families with better robustness and accuracy, which is helpful for subsequent APT attack tracing and malicious behavior reasoning.

ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No.62172308, No.U1626107, No.61972297, No.62172144, and No.62062019).

References

[1] Microsoft, "What is powershell? - powershell — microsoft docs." https://docs.microsoft.com/en-us/powershell/scripting/overview. 2022.

[2] D. Hendler, S. Kels, *et al.*, "Amsi-based detection of malicious powershell code using contextual embeddings," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, pp. 679–693. ACM, 2020.

[3] Z. Li, Q. Chen, *et al.*, "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 (CCS)*, pp. 1831–1847. ACM, 2019.

[4] T. Ongun, J. W. Stokes, *et al.*, "Living-off-the-land command detection using active learning," in *Proceedings of the 24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pp. 442–455. ACM, 2021.

[5] F. Barr-Smith, X. Ugarte-Pedrero, *et al.*, "Survivalism: Systematic analysis of windows malware living-off-the-land," in *Proceedings of the 2021 IEEE Symposium on Security and Privacy (SP)*, pp. 1557–1574, 2021.

[6] J. White, "Practical behavioral profiling of powershell scripts through static analysis (part2)." https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-powershell-scripts-through-static-analysis-part-2. 2017.

[7] X. Han, T. Pasquier, *et al.*, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[8] Q. Wang, W. U. Hassan, *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis," in *Proceedings of the 27th Annual Network and Distributed System Security Symposium (NDSS)*, 2020.

[9] S. Li, Q. Zhang, *et al.*, "Attribution classification method of apt malware in iot using machine learning techniques," *Security and Communication Networks*, vol. 2021, pp. 1–12, 2021.

[10] W. Chen, X. Helu, *et al.*, "Advanced persistent threat organization identification based on software gene of malware," *Transactions on Emerging Telecommunications Technologies*, vol. 31, no. 12, p. e3884, 2020.

[11] McAfee, "Mcafee labs covid-19 threats report." https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-july-2020.pdf. 2020.

[12] A. Nourian and S. Madnick, "A systems theoretic approach to the security threats in cyber physical systems applied to stuxnet," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 2–13, 2015.
[13] Clearsky, "Powdesk: Powershell script for landesk management agent hosts." https://www.clearskysec.com/powdesk/ 2022.
[14] M. Dunwoody, "Dissecting one of apt29's fileless wmi and powershell backdoors (poshspy)." https://www.mandiant.com/resources/dissecting-one-ofap. 2017.
[15] C. Liu, B. Xia, *et al.*, "Psdem: A feasible de-obfuscation method for malicious powershell detection," in *Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 825–831, 2018.
[16] D. Ugarte, D. Maiorca, *et al.*, "Powerdrive: Accurate de-obfuscation and analysis of powershell malware," in *Proceedings of the 16th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pp. 240–259. Springer, 2019.
[17] D. Bohannon, "Revoke-obfuscation." https://github.com/danielbohannon/Revoke-Obfuscation. 2020.
[18] G. M. Malandrone, V. Giovanni, *et al.*, "Powerdecode: A powershell script decoder dedicated to malware analysis," in *Proceedings of the Italian Conference on Cybersecurity (ITASEC)*, vol. 2940, pp. 219–232. CEUR-WS.org, 2021.
[19] J. White, "Practical behavioral profiling of powershell scripts through static analysis (part1)." https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-powershell-scripts-through-static-analysis-part-1. 2017.
[20] Microsoft, "Antimalware scan interface (amsi) - win32 apps." https://docs.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal. 2019.
[21] D. Hendler, S. Kels, *et al.*, "Detecting malicious powershell commands using deep neural networks," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (AsiaCCS)*, pp. 187–197. ACM, 2018.
[22] Y. Fang, X. Zhou, *et al.*, "Effective method for detecting malicious powershell scripts based on hybrid features," *Neurocomputing*, vol. 448, pp. 30–39, 2021.
[23] G. Rusak, A. Al-Dujaili, *et al.*, "Ast-based deep learning for detecting malicious powershell," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 2276–2278. ACM, 2018.
[24] E. Downing, Y. Mirsky, *et al.*, "Deepreflect: Discovering malicious functionality through binary reconstruction," in *Proceedings of the 30th USENIX Security Symposium*, pp. 3469–3486, 2021.
[25] W. Lian, G. Nie, *et al.*, "Cryptomining malware detection based on edge computing-oriented multi-modal features deep learning," *China Communications*, vol. 19, no. 2, pp. 174–185, 2022.
[26] J. Straub, "Modeling attack, defense and threat trees and the cyber kill chain, att&ck and stride frameworks as blackboard architecture networks," in *Proceedings of the 2020 IEEE International Conference on Smart Cloud (SmartCloud)*, pp. 148–153, 2020.
[27] C. Xiong, T. Zhu, *et al.*, "Conan: A practical real-time apt detection system with high accuracy and efficiency," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 551–565, 2020.
[28] A. Fass, M. Backes, *et al.*, "Hidenoseek: Camouflaging malicious javascript in benign asts," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 1899–1913. ACM, 2019.
[29] J. W. Stokes, R. Agrawal, *et al.*, "Detection of malicious vbscript using static and dynamic analysis with recurrent deep learning," in *Proceedings of the 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2887–2891, 2020.
[30] C. Wueest, "Powershell threats grow further and operate in plain sight." https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/powershell-threats-grow-further-and-operate-plain-sight. 2018.
[31] W. HarmJ0y, "Powersploit - a powershell post-exploitation framework." https://github.com/PowerShellMafia/PowerSploit. 2020.
[32] Samratashok, "Nishang - offensive powershell for red team, penetration testing and offensive security." https://github.com/samratashok/nishang. 2021.
[33] C. Ross, "Empire is a powershell and python post-exploitation agent." https://github.com/EmpireProject/Empire. 2019.
[34] HelpSystems, "Cobalt strike — adversary simulation and red team operations." https://www.cobaltstrike.com. 2022.
[35] Y. Liu, B. Liu, *et al.*, "Powershell malware detection method based on features combination," *Journal of Cyber Security*, vol. 6, no. 1, pp. 40–53, 2021.
[36] C. Xiong, Z. Li, *et al.*, "Generic, efficient, and effective de-obfuscation and semantic-aware attack detection for powershell scripts," *Frontiers of Information Technology & Electronic Engineering*, vol. 23, no. 3, pp. 361–381, 2022.
[37] X. Zhang, Y. Zhang, *et al.*, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pp. 757–770. ACM, 2020.
[38] A. Alahmadi, N. Alkhraan, *et al.*, "Mpsautodetect: A malicious powershell script detection model based on stacked denoising auto-encoder," *Computers & Security*, vol. 116, p. 102658, 2022.
[39] Y. Luo, L. Cheng, *et al.*, "Deepnoise: Learning sensor and process noise to detect data integrity attacks in cps," *China Communications*, vol. 18, no. 9, pp. 192–209, 2021.
[40] Y. Chai, L. Du, *et al.*, "Dynamic prototype network based on sample adaptation for few-shot malware detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 5, pp. 4754–4766, 2022.
[41] S. Li, Y. Li, *et al.*, "Malicious mining code detection based on ensemble learning in cloud computing environment," *Simulation Modelling Practice and Theory*, vol. 113, p. 102391, 2021.
[42] N. Prasad, S. Saha, *et al.*, "A multimodal classification of noisy hate speech using character level embedding and attention," in *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2021.
[43] L. Yann, L. Bottou, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[45] N. Zhang, Y. Yan, *et al.*, "A novel user behavior prediction model based on automatic annotated behavior recognition in smart home systems," *China Communications*, vol. 19, no. 9, pp. 116–132, 2022.
[46] M. Ring, D. Schlör, *et al.*, "Malware detection on windows audit logs using lstms," *Computers & Security*, vol. 109, p. 102389, 2021.
[47] A. Waswani, N. Shazeer, *et al.*, "Attention is all you need," in *Proceedings of the 2017 Neural Information Processing Systems (NIPS)*, pp. 5998–6008, 2017.
[48] F. Yang, X. Quan, *et al.*, "Multi-document transformer for personality detection," in *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, vol. 35, no. 16, pp. 14 221–14 229, 2021.
[49] J. White, "Pulling back the curtains on encodedcommand powershell attacks." https://unit42.paloaltonetworks.com/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks. 2022.
[50] F. Sandbox, "Free automated malware analysis service." https://www.hybrid-analysis.com. 2022.
[51] "Any.run - interactive online malware sandbox." https://any.run. 2022.
[52] Mitre, "Mitre att&ck framework." https://attack.mitre.org. 2022.

Biographies

Xiuzhang Yang received the B.S. and M.S. degrees in software engineering from Beijing Institute of Technology (BIT) in 2014 and 2016. He is currently pursuing a Ph.D. degree with the School of Cyber Science and Engineering, Wuhan University. His research interests include information system security, malicious code analysis, and machine learning.

Guojun Peng is currently a professor at the School of Cyber Science and Engineering, Wuhan University. He received the Ph.D. degree from Wuhan University in 2008. His research interests are system security and trust computing. He is a member of CCF and CSAC.

Dongni Zhang received the B.S. degree in computer science and technology from Central South University (CSU) in 2017, and the M.S. degree from University College Londen (UCL) in 2018. She is currently pursuing a Ph.D. degree with the School of Cyber Science and Engineering, Wuhan University. Her current research is information system security.

Yuhang Gao received the B.S. degree in cyberspace security from Wuhan University in 2019. He is currently working towards the M.S. degree in School of Cyber Science and Engineering, Wuhan University. His current research is malicious code analysis.

Chenguang Li received the B.S. degree in cyberspace security from Wuhan University in 2021. He is currently working towards the M.S. degree in School of Cyber Science and Engineering, Wuhan University. His current research is malicious code analysis.