Received 23 November 2022, accepted 22 December 2022, date of publication 26 December 2022, date of current version 2 January 2023.

Digital Object Identifier 10.1109/ACCESS.2022.3232505

RESEARCH ARTICLE

PowerDP: De-Obfuscating and Profiling Malicious PowerShell Commands With Multi-Label Classifiers

MENG-HAN TSAI[1,2,3], (Member, IEEE), CHIA-CHING LIN[1], ZHENG-GANG HE[1], WEI-CHIEH YANG[1,2,3], AND CHIN-LAUNG LEI[1]
[1]Graduate Institute of Electrical Engineering, National Taiwan University, Taipei 106319, Taiwan
[2]Center for Cybersecurity Service, Institute for Information Industry, Taipei 106214, Taiwan
[3]National Center for Cyber Security Technology, Taipei 106043, Taiwan

Corresponding author: Meng-Han Tsai (d02921015@ntu.edu.tw)

This work was supported in part by the Ministry of Science and Technology, Taiwan, under Grant 110-2221-E-002-073-MY2.

ABSTRACT In recent years, PowerShell has become the common tool that helps attackers launch targeted attacks using living-off-the-land tactics and fileless attack techniques. Unfortunately, malware-derived PowerShell Commands (PSCmds) have typically been obfuscated to hide the malicious intent from detection and analysis. Also, malicious PSCmds' expansive use of multiple obfuscation strategies and encryption methods makes them difficult to be revealed. Despite the advances in malicious PSCmds detection incorporating new approaches such as machine learning and deep learning, there is still no consensus on the solution to de-obfuscating malicious PSCmds and profiling their behavior. To address this challenge, we propose a hybrid framework that combines deep learning and program analysis for automatic *PowerShell De-obfuscation and behavioral Profiling* (PowerDP) through multi-label classification in a static manner. First, we use character distribution features to forecast obfuscation types of malicious PSCmds. Second, we developed an extensive de-obfuscator utilizing static regular expression replacement to recover the original content of obfuscated PSCmds based on the predicted obfuscation types. Finally, we profile the behavior of PSCmds by features extracted from the abstract syntax tree of PSCmds after de-obfuscation. Our results show that PowerDP achieves a promising 99.82% accuracy and 0.18% hamming loss in obfuscation multi-label classification using deep learning. Furthermore, the successful recovery rate of the de-obfuscator against 15 obfuscation types is 98.11% on average with semantic similarity comparison, and the accuracy of the behavior multi-label classification for identifying 5 behaviors in malicious PSCmds averages 98.53%. The evaluation indicates that PowerDP is able to classify and profile complicated PSCmds.

INDEX TERMS PowerShell, de-obfuscation, machine learning, deep learning, abstract syntax trees, multi-label classification, behavioral profiling.

I. INTRODUCTION

With the rise of hacktivism [1] and cybercriminals' growing aspirations for higher profits [2], targeted attacks using malicious emails have gradually expanded over the past decade. According to the knowledgable MITRE ATT&CK Matrix for Enterprise,[1] fileless malware spreading via emails is one of the mainstream types of stealth attacks [3], evading detection by most security solutions and thwarting forensic analysis efforts. Even though security monitoring and defense effectiveness increase, emails and decoy Office documents are still the best combinations for malware delivery media. Because people remain susceptible to manipulation, human psychological weaknesses result in the main vulnerabilities that can be exploited through social engineering, e.g., spear-phishing attacks. In this scenario, attackers typically attached a well-customized malicious document containing

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Masucci.

[1]https://attack.mitre.org/matrices/enterprise/

PowerShell Commands (PSCmds) to a forged email. Additionally, impersonation is often used in sender names or contact information to lure targets into opening malicious files.

Living-off-the-Land (LotL) tactics and fileless attack techniques are the emerging trends in targeted attacks [4], [5]. Attackers use legitimate and preinstalled system tools or run scripts such as PSCmds directly in memory to perform lateral movement within a corporate network. By leaving fewer artifacts and traces on the target system, attackers can minimize the risk of being detected by security defenses and increase the success of attacks. Considering that PowerShell is widely used to administrate and manage Windows operating systems, PowerShell has become one of the primary attack tools used in fileless attacks.

PowerShell is a powerful scripting language, interactive command-line interface, and scripting platform built on the Microsoft .NET framework. With a wide range of libraries and useful cmdlets, PowerShell is gaining popularity among software developers and IT operators due to its versatility and reliability. Unfortunately, cybercriminals and malware authors are also rapidly taking advantage of PowerShell for malicious usage [6]. As with other scripting languages used by malware, PSCmds' expansive use of multiple obfuscation strategies and encryption methods makes them difficult to be revealed. As a result, attackers use the flexibility of PowerShell to hide malicious intent from the context of code, compromising analysis by security analysts.

Table 1 shows three commands from the native Windows command line, the Powershell cmdlet, and the dotNet code. PowerShell could interpret all three commands to produce the same output even though they are three different invocation methods. In addition, PowerShell supports numerous encoding and encryption mechanisms, such as Base64, Hex, XOR, and SecureString. Using PowerShell's natural properties, attackers can combine multiple obfuscation strategies to prevent malicious PSCmds from being quickly and automatically broken by a sandbox or analyst. Also, malware authors typically produce novel generations of malware with polymorphic and metamorphic properties [7] through advances in areas such as automated obfuscation engines [8], code generation tools, code protection methods [9], and packers to circumvent detection. With the rapid developments of obfuscation techniques and metamorphic engines, malicious software and script populations have increased dramatically [10], [11]. Therefore, detection evasion and obfuscation techniques represent a non-neglected challenge that needs to be mitigated in today's IT environment.

While security vendors and researchers have proposed many works and solutions from various aspects to combat such threats, most mainly focus on dynamic execution or static parsing to detect PSCmds. The monitoring schemes using the .NET framework [12], [13] and script block logging via Antimalware Scan Interface (AMSI)[2] require in-depth

[2]https://learn.microsoft.com/en-us/windows/win32/amsi/antimalware-scan-interface-portal

TABLE 1. Different invocation methods for the same purpose in PowerShell.

#	Command	Description
1	PWD	Native command line
2	Get-Location	PowerShell cmdlet
3	[System.IO.Directory]:: GetCurrentDirectory()	dotNet code

knowledge of the system and implementation within the dynamic mechanisms. The static mechanisms [14], [15], [16], [17] manually picking with the regular expression (regex) to match the string-level or token-level signatures have been shown to facilitate the analysis and de-obfuscation of malicious PSCmds. Although security analysts benefit from dynamic and static solutions for analyzing malicious PSCmds, the propagation speed of metamorphic malware containing variants of various obfuscations is still faster than real-world detection and analysis processes. As a result, the arms race between malware authors and security analyzers continued, and a more effective solution is required to combat highly obfuscated and encrypted threats.

On the other hand, recently emerging techniques such as Machine Learning (ML) and Deep Learning (DL) have shown that they could offer researchers alternative solutions [18], [19], [20], [21] for developing cutting-edge methods to combat cybersecurity challenges. In general, several studies achieve better performance than traditional signature scanning and execution monitoring mechanisms, including detection with vector representation features from Abstract Syntax Tree (AST) [22], [23], [24], [25], [26], Natural Language Processing (NLP) [27], [28], [29], [30], and Graph Neural Network (GNN) [31] inference to differentiate between malicious and benign scripts. However, to the best of our knowledge, previous studies by ML and DL cannot be considered conclusive as they mainly focus on binary classification that discriminates malicious PSCmds from benign ones, and often fail to reveal semantics or malicious intent behind the obfuscated PSCmds. Therefore, the problem remains unsolved, and analysts still have to put much effort into finding out the behavior of PSCmds and the targets of attackers through dynamic or static analysis for forensic purposes.

To address this challenge, we propose a hybrid framework that combines deep learning and program analysis for automatic *PowerShell De-obfuscation and behavioral Profiling* (PowerDP) through multi-label classification in a static manner. We first collect malicious PSCmds from an internal sandbox and external open datasets to generate the ground truth, including obfuscation and behavioral labels. Then we adopt an obfuscation multi-label classifier to forecast the obfuscation types of PSCmds. As per the prediction, we utilize the static regex replacement to accurately de-obfuscate and recover the original content of obfuscated PSCmds. Afterward, taking advantage of AST, a tree-like representation of the abstract syntactic structure of code, we process each de-obfuscated PSCmd of arbitrary length into a real-valued vector representation for behavioral profiling with multi-label classification. PowerDP not only efficiently automates labor-intensive tasks and relieves analysts of mental effort but also provides a coarse-to-fine grained report from the results of the multi-label classification, which can be easily combined with previous works on detecting malicious PSCmds. Security analysts can benefit from the proposed automation mechanism to discover malicious intent behind the malware-derived PSCmds and reduce the burden of malware and forensics analysis.

The significant contributions of this study can be summarized as follows:

- We present a novel research task to automatically identify the obfuscation types of malicious PSCmds through character distribution features and obfuscation multi-label classification.
- We empirically extend previous works on static regex to develop a de-obfuscator to recover obfuscated PSCmds to the original content with precise regex replacement in a recommended order.
- To address behavioral profiling for forensic analysis, we use the real-valued vector representations from the AST of PSCmds as input to a behavior multi-label classifier to predict potential behavioral functionalities.

The remainder of this paper is organized as follows. In Section II, we provide background information, the various obfuscation types of PSCmds used by attackers to thwart the analysis efforts, and the challenges of de-obfuscating and behavioral profiling for PSCmds. Section III gives an overview of our framework and entails our approach to describe how we statically de-obfuscate and profile PSCmds. We present the ground truth dataset creation and correction process in Section IV, including how we collect and preprocess PSCmds to detect obfuscation types and profile behavior of PSCmds. Section V illustrates the evaluation results, followed by a brief discussion of the proposed method limitations in Section VI. Related work is summarized in Section VII. Finally, we conclude this work and propose a direction for future improvement in Section VIII.

II. BACKGROUND AND MOTIVATION

Because of PowerShell's versatility and reliability, PowerShell is abused as one of the most popular scripting languages used by cybercriminals. Among all attacks, PSCmds are commonly used to download and execute malicious payloads. Listing 1 shows a typical example of malicious PSCmds often stored in a well-customized malicious document. In this section, we introduce various obfuscation types and discuss the challenges of de-obfuscation and behavioral profiling to uncover malicious intent behind the PSCmds.

A. *PowerShell OBFUSCATION*

Obfuscation is a technique to evade detection and thwart malware analysis, bringing an invisible cloak for malware and malicious scripts. Through obfuscation, attackers can effectively launch an underneath intrusion without awareness. There are various methods of obfuscating PSCmds,

```
PowerShell -ExecutionPolicy Bypass
    -NoProfile -WindowStyle Hidden -Command
    (New-Object
    System.Net.WebClient).DownloadFile(
    "http://malicious.website/malware.exe",
    "$env:APPDATA\bad.exe"); Start-Process
    ("$env:APPDATA\bad.exe")
```

LISTING 1. **Maclicious PSCmds for malware download and execution.**

many of which were implemented in the framework "Invoke-Obfuscation" created by Daniel Bohannon [32] in 2016 and other tools such as PowerShell Empire[3] and PowerSploit.[4] Three categories of obfuscation methods are typically employed by malicious PSCmds:

1) **Compression.** This strategy applies compression to compress the malicious PSCmds and then utilizes Base64 to encode the result.
2) **Encoding schemes.** Encoding is the most common method used in PowerShell obfuscation. We briefly review the seven encoding schemes typical in the real world.
    - Base64: Encodes entire PSCmds to a Base64 encoded string.
    - ASCII: Represents each printable character in PSCmds with an ASCII code value.
    - Hexadecimal: Converts each printable character in PSCmds to a hexadecimal value.
    - Octal: Converts each printable character in PSCmds to an octal value.
    - Binary: Generates encoded payload using binary representation for entire PSCmds.
    - Binary Exclusive OR: Takes PSCmds and a specified key as inputs and produces the encoded result with a bitwise XOR operation.
    - SecureString: Generates an AES-encrypted SecureString object from PSCmds using a specified key.
3) **String manipulation.** Since PowerShell provides many string-level and token-level manipulation methods, attackers are always trying to modify the easily recognizable patterns of functions, parameters, and variables in PSCmds for malicious purposes. Below are four common methods for manipulating strings.
    - Concatenation: Splits the PSCmds into multiple parts and combines all divided content with "+" operators or a join operator for recovery.
    - Reordering: Divides the PSCmds into several parts and reassembles the string with the specified indexes via the "-f" operator to format and rearrange a string expression.
    - Reversing: Makes the PSCmds appear in reverse order and recreates the string expression with some helper functions before executing the original command.

[3]https://github.com/EmpireProject/Empire
[4]https://github.com/PowerShellMafia/PowerSploit

- Tick: Inserts backticks "`", known as grave accents, into a string. Backticks are escape characters that are not interpreted when executing PSCmds.

Apart from the methods mentioned above, three helper functions are often adopted or combined with other obfuscations to increase the complexity.

1) **Splitting.** This helper function splits an entire obfuscated string into a token array to remove extraneous symbols or special characters not present in the original PSCmds.
2) **Replacement.** Instead of using the split function, replacing is another way to eliminate nonsensical characters in the expression string for the same purpose as splitting.
3) **Randomcase.** In this case, each alphabet in the PSCmds is converted with a randomly mixed case to break reading comprehension and thwart malware analysis.

The complete list of obfuscation methods discussed in this paper can be found in Table 2. Examples of obfuscated PSCmds derived by applying the above 12 obfuscation methods and three helper functions to the PSCmds in Listing 1 are given in the public GitHub repository.[5] As can be seen from the examples, the original contents and intent of PSCmds are hidden by the obfuscation. Furthermore, the textual representation of PSCmds has also been changed in different obfuscations. Thus, it is difficult to understand what is being executed by the PSCmds without de-obfuscation, affecting forensic analysis and impact evaluation. However, the number of possible combinations of different obfuscation strategies can be large and complex. Therefore, we need a procedure or recommended order to handle all combinations of obfuscation methods for recovery.

B. PowerShell BEHAVIORAL PROFILING

Behavioral profiling of malicious PSCmds provides security analysts and researchers with a quick view of malicious behavior and potential activities after executing the code. Additionally, the process helps reduce labor-intensive tasks like analyzing thousands of files in a limited time. Some authors [15], [17] have provided practical approaches for static analysis of PSCmds for PowerShell behavioral profiling. For example, after identifying behaviors via keywords, they use many custom regex to remove various types of obfuscation and score the malicious code accordingly. However, although previous work has proven useful for behavioral profiling, the process still needs to be performed iteratively to unravel new content and look for specific keywords that may be obfuscated or hidden from context.

At the same time, recent advances in ML algorithms and DL architectures have improved significantly and are booming, particularly in the areas of computer vision, video and speech recognition, and NLP. Moreover, studies like Code2vec [33] have shown excellent performance and effectiveness in terms of code search and comprehension. By leveraging the neural network with a fixed-sized vector input representing arbitrary-sized snippets of code, researchers can achieve algorithm prediction [34], semantic code labeling [35], [36], and multi-label classification [37] for code functionalities.

Several state-of-the-art studies [10], [11], [38], [39], [40], [41] using ML/DL and genetic algorithm achieve performance gains in detecting and analyzing malicious executables and mobile apps. Additionally, they have shown that automated behavioral profiling or semantic labeling for malware analysis can benefit security analysts and reduce manual work. Therefore, it is essential to study how the ML/DL techniques, which in previous work mainly focused on the binary classification of malicious PSCmds, can be extended to include multi-label classification in behavioral profiling to understand the malicious intent behind the PSCmds.

[5]https://github.com/mhtsai1010/pscmd-obfuscation-examples

TABLE 2. The complete list and recommended order for de-obfuscation.

Priority	Obfuscation method	Category
1	Tick	String manipulation
2	Concatenation	String manipulation
3	Reversing	String manipulation
4	Reordering	String manipulation
5	Replacement	Helper function
6	Splitting	Helper function
7	ASCII encoding	Encoding scheme
8	Binary encoding	Encoding scheme
9	Hexadecimal encoding	Encoding scheme
10	Octal encoding	Encoding scheme
11	Binary XOR encoding	Encoding scheme
12	Compression	Compression
13	SecureString encoding	Encoding scheme
14	Base64 encoding	Encoding scheme
15	Randomcase	Helper function

III. THE PROPOSED PowerDP FRAMEWORK

To counter the complexity and threat of malicious PSCmds, we proposed PowerDP, a hybrid framework for automatic PowerShell de-obfuscation and behavioral profiling through multi-label classification in a static manner. Fig. 1 illustrates the overview of the PowerDP framework that generates a coarse-to-fine grained multi-label classification report of malicious PSCmds. In the following subsections, we explain the technical details of each component in PowerDP.

A. SYSTEM OVERVIEW

As shown in Fig. 1, the proposed PowerDP framework consists of two phases, each of which involves a multi-label classification task, as described below.

1) PHASE 1: DE-OBFUSCATING PowerShell COMMANDS

In the first phase, we apply static techniques rather than dynamic mechanisms to de-obfuscate the PSCmds for efficiency. We extract textual features and statistical characteristics of character distribution from PSCmds to represent the malicious PowerShell samples. Given the representation

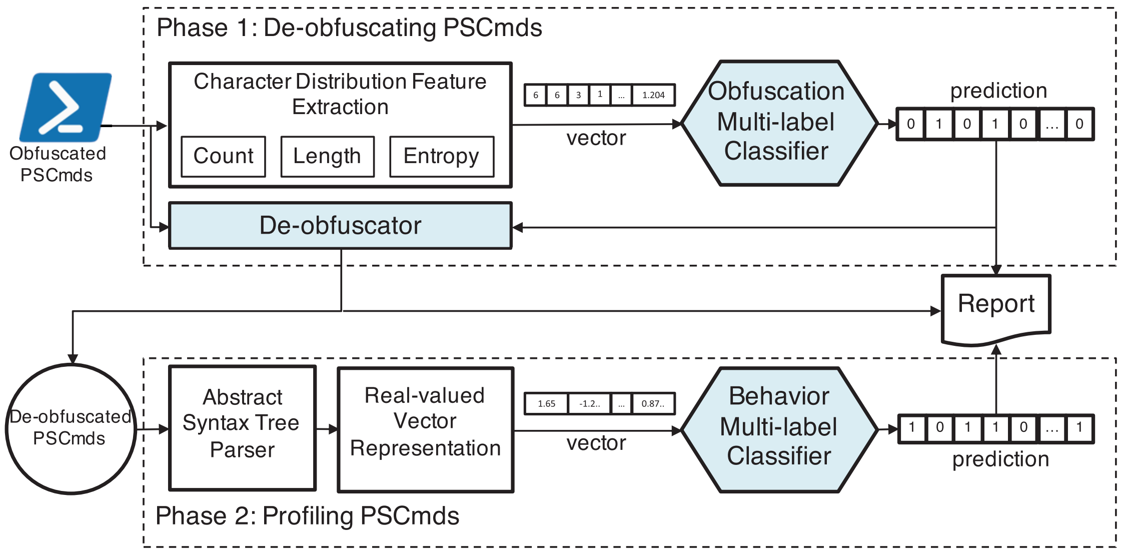
Phase 1: De-obfuscating PSCmds
Obfuscated PSCmds
Character Distribution Feature Extraction
Count
Length
Entropy
6 6 3 1 ... 1.204
vector
Obfuscation Multi-label Classifier
prediction
0 1 0 1 0 ... 0
De-obfuscator
Report
De-obfuscated PSCmds
Abstract Syntax Tree Parser
Real-valued Vector Representation
1.65 -1.2.. ... 0.87..
vector
Behavior Multi-label Classifier
1 0 1 1 0 ... 1
prediction
Phase 2: Profiling PSCmds

FIGURE 1. The overview of the proposed PowerDP framework.

vector of character distribution, we then train a obfuscation multi-label classifier to forecast the obfuscation types of the PSCmds. After that, we empirically extend previous studies [14], [15], [16], [17] on static regex to develop a PowerShell de-obfuscator in python language, which utilizes precise regex replacement against 15 obfuscation types mentioned in Section II to recover the obfuscated PSCmds into the original content based on the prediction from the classifier.

2) PHASE 2: PROFILING PowerShell COMMANDS

After de-obfuscating the PSCmds, the original malicious PSCmds are revealed, allowing us to profile their behavior from context to uncover the intent behind the code. Instead of sequentially retrieving behavioral patterns like keywords from the de-obfuscated context of PSCmds, we use deep learning with neural networks to classify the potential behavior on a high abstraction level of AST. Specifically, we develop an AST parser to encapsulate the underlying meanings of discrete symbols in PSCmds in an ensemble vector, such as AST node types and the parent-child relationships among the AST nodes to represent the malicious PowerShell samples, as will be detailed in Section III-C1. Finally, we train a behavior multi-label classifier with the real-valued vector representations from the AST of PSCmds to predict five behaviors of the PSCmds. We discuss how we choose the five predefined behaviors in Section IV.

3) MULTI-LABEL CLASSIFICATION

Multi-label classification [37] is the problem of finding a model that maps inputs X to binary vectors $Y = \{y_1, y_2, \ldots, y_n\}$, where $y_i \in \{0, 1\}^{M_f}$ representing the label vector for X, n is the number of samples, and M_f is the class dimension; that is, it assigns each element (label) in Y a value of 0 or 1, where $y_i = 1$ indicates the presence of the label. At present, several methods are well documented in the literature to address the problem of multi-label classification. And these methods can be divided into two categories as proposed in [37]: **Problem Transformation Methods** and **Algorithm Adaptation Methods**. Problem transformation methods aim to decompose the problem into a set of binary problems. On the other hand, algorithm adaptation methods extend specific learning algorithms to deal with multi-label data directly. We share our design of the two multi-label classifiers aiming to achieve obfuscation and behavior multi-label classification, respectively.

B. DE-OBFUSCATING PowerShell COMMANDS

In this phase, we focus on de-obfuscating malicious PSCmds based on the obfuscation types predicted by a multi-label classifier, with each design component detailed below.

1) FEATURES BASED ON THE CHARACTER DISTRIBUTION

Since different obfuscation methods change the same PSCmds through their properties to different appearances in the text, we try to keep the most meaningful and representative abstraction of the textual code in the PSCmds. Unlike other studies, we do not preprocess the PSCmds to eliminate or normalize redundant tokens or characters. For obfuscation multi-label classification, we collect 97 character-based features from the minimum unit, which is a printable character in PSCmds.

a: CHARACTER STATISTICS

We count every printable character whose ASCII decimal value is 32 to 126 in PSCmds. As a result, we have the quantity of existence of each character in PSCmds and represent it as 95 features.

b: CHARACTER LENGTH

From our observation, the length of obfuscated PSCmds in encoding schemes is usually much longer than that with obfuscation in string manipulation and compression. Therefore, we count the length of the code as one feature for PSCmds.

c: INFORMATION ENTROPY

Entropy is a measure to represent the statistical characteristics of different characters in one source with a probability distribution. We can use it to analyze the distribution of different characters. The entropy can be calculated as follows:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_{10} p(x_i) \begin{cases} X & = \{x_1, x_2, \ldots, x_n\} \\ W & = \sum_{i=1}^{n} x_i \\ p(x_i) & = \frac{x_i}{W} \end{cases} \tag{1}$$

In Eq. (1), x_i and W refer to the count of each distinct character and the count of all characters in the PSCmds, and $p(x_i)$ represents the frequency of the i^{th} distinct character.

2) OBFUSCATION MULTI-LABEL CLASSIFIER

For obfuscation multi-label classification to forecast obfuscation types, we compare the performance of adopting Linear Support Vector Machine (SVM) and Random Forest (RF) from Problem Transformation methods, K-Nearest Neighbors (KNN) and Decision Tree (DT) from Algorithm Adaptation methods, and the Deep Neural Network (DNN). All models were built using the python language on the Google Colaboratory[6] with the Scikit-learn [42] and Keras[7] libraries. The parameters of the ML algorithms are determined using the grid search strategy and listed in Table 3, and the architecture of the DNN is presented in Table 4. In the end, we employ the classifier with the best performance after evaluation.

3) THE DE-OBFUSCATOR

Based on the predicted obfuscation types, we then build a PowerShell de-obfuscator as follows. We empirically extend previous work [14], [15], [16], [17] on static regex replacement to develop the de-obfuscator to recover obfuscated PSCmds to the original content. After receiving the prediction results from the classifier, the de-obfuscator searches for specified signatures and patterns in PSCmds for substitution and recovery. To increase the accuracy of our designed de-obfuscator, we provide in Table 2 a recommended order for the de-obfuscation process for PSCmds with multiple obfuscation types. The order of de-obfuscation arises from the practical problems we faced when investigating different types of obfuscation from the in-the-wild PowerShell samples and the repeatable procedures such as iterative processing and testing in previous work.

Due to the complexity and variety, string manipulation methods must be treated as the top priority for de-obfuscation.

[6]https://colab.research.google.com/

[7]https://keras.io/

TABLE 3. The parameters for different machine learning algorithms.

Algorithm	Parameter
SVM	`SVC, kernel="linear", C=1, random_state=42`
RF	`n_estimators=300, max_depth=19, random_state=42`
KNN	`n_neighbors=3, p=1, manhattan_distance`
DT	`max_depth=15, random_state=42`

TABLE 4. The architecture of the DNN for obfuscation multi-label classification (Learning rate: 0.0001; Batch size: 256; Optimizer: Adam).

#	Layer type	# of neuron	Activation
1	Dense	512	ReLU
2	Dense	256	ReLU
3	Dense	128	ReLU
4	Dense	64	ReLU
5	Dense	32	ReLU
6	Dense	15	Sigmoid

Attackers use a combination of operators, string representation, and string-based helper functions such as replacement and splitting to embed the malicious command in the expression string, rendering the string nearly unreadable. Therefore, the first consideration should be to recover the obfuscation results of string manipulation into the de-obfuscated content using the de-obfuscator. Meanwhile, the encoding schemes differ from command tokenization or partial substitution commonly encountered in string manipulation. All methods in encoding schemes change the entire PSCmds with a specified representation of their corresponding range of coding values. Thus, the encoding schemes are the secondary candidates for de-obfuscation processing. We can detect the range of the coding values in the obfuscated PSCmds and use the corresponding transformation to recover the original expression string.

Aside from the obfuscation methods discussed previously, Base64 encoding is the simplest and most common anti-detection method among all obfuscation schemes and appears to be present in compression, SecureString encoding, and Base64 encoding itself. Because compression and SecureString encoding require additional operations to de-obfuscate the PSCmds, such as unzip function and decryption with keys, we designed the de-obfuscator to recover the PSCmds that contain a Base64 encoded string in the third stage to avoid the misjudgment. Finally, the de-obfuscator transfers all alphabets in the PSCmds to lowercase when Randomcase is recognized in the end, in case some uppercase characters are required in prior de-obfuscation processes.

C. PROFILING PowerShell COMMANDS

In this phase, we aim at behavior profiling for malicious PSCmds, which have been de-obfuscated in the previous phase.

1) FEATURES BASED ON THE ABSTRACT SYNTAX TREE

Knowing that AST is widely used in programming language processing systems to represent the abstract syntactic

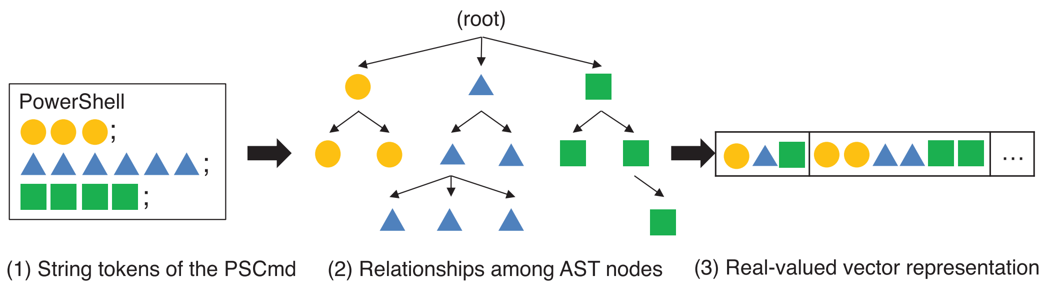
(root)
PowerShell
(1) String tokens of the PSCmd
(2) Relationships among AST nodes
(3) Real-valued vector representation

FIGURE 2. **The concept of assembling real-valued vector representations from string tokens via the relationships among AST nodes.**

structure of programs, we develop an AST parser using PowerShell commands "ParseFile" and "FindAll" to determine the AST of the de-obfuscated PSCmds. We saved the complete AST information as a text file in a depth-first-search order, including parent-child relationships and all AST node types. After parsing the AST structure from each PSCmds, we extract five statistical features, including the number of node types, the number of nodes, the number of leaves, the maximum degree of the node, and the height of the tree. Additionally, there are 45 distinct AST node types in our dataset, each of which is associated with a feature vector, as detailed below.

We have noticed that vector embedding representations are suitable for general ML/DL tasks such as clustering, recommendation, and classification. Therefore, we employed [43] to learn the real-valued vector representations of the 45 different AST node types. Furthermore, besides the vector representation for each node type in the AST, we try to preserve the parent-child relationships among AST nodes in the vector. Therefore, a node's vector representations in the AST should not only be learned from different node types but also be "coded" by its children's representations, which means that a node's vector representations could be treated as a summary of all functionalities of its child nodes. To give an example, for each subtree with parent node P and n direct child nodes $C_1, \ldots, C_n$ in an AST, we present them as $vec(P), vec(C_1), \ldots, vec(C_n)$. The primary objective is that

$$vec(P) \approx tanh\left(\sum_{i=1}^{n} l_i W_i \cdot vec(C_i) + \boldsymbol{b}\right) \tag{2}$$

where $l_i = (\#\, leaves\ of\ C_i)/(\#\, leaves\ of\ P)$ is the coefficient of the weight and $\boldsymbol{b} \in \mathbf{R}^{N_f}$ is a bias vector. Besides, $W_i \in \mathbf{R}^{N_f \times N_f}$ is the weights matrix for child node C_i, which can be defined as follows:

$$W_i = \frac{n-i}{n-1} W_l + \frac{i-1}{n-1} W_r \tag{3}$$

where $W_l, W_r \in \mathbf{R}^{N_f \times N_f}$ are trainable weights matrices. The objective is to make the Euclidean distance between the two vector representations on the left and right sides in Eq. (2) as small as possible. Moreover, to overcome the problem that different nodes may have different numbers of children, [44] proposed the idea of a "continuous binary tree," where W_l and W_r are model parameters, and W_i is a linear combination of W_l and W_r according to the position of node i. Formally, P has n ($n \geq 2$) children, if $n = 1$, $W_i = \frac{1}{2} W_l + \frac{1}{2} W_r$. In our experimental setting, we empirically set N_f to 22, which indicates the length of the vector of each AST node.

TABLE 5. **The architecture of the DNN for behavior multi-label classification (Learning rate: 0.00001; Batch size: 8; Optimizer: Adam).**

#	Layer type	# of neuron	Activation
1	Dense	512	ReLU
2	Dense	128	ReLU
3	Dense	32	ReLU
4	Dense	5	Sigmoid

After optimizing all parameters, we could generate the real-valued vector representations of all AST nodes using the methodology in [43]. For leaf nodes in the AST, they are just the vector representations learned by the algorithm. Meanwhile, for each non-leaf node P, there are two representation vectors. One is the left side of Eq. (2) learned from the algorithm, and the other is the right side of Eq. (2) coded by the child nodes. Hence we denote the combined vector of node P as $\boldsymbol{P}$. We have

$$\boldsymbol{P} = \frac{1}{2} \times vec(P) + \frac{1}{2} \times tanh\left(\sum_{i=1}^{n} l_i W_i \cdot vec(C_i) + \boldsymbol{b}\right). \tag{4}$$

Since each non-leaf node in the AST can be represented as a real-valued vector using Eq. (4), we adopted a breadth-first search to create the tree vector. Fig. 2 shows our idea of abstracting all information in PSCmds and encapsulating it from string tokens to a real-valued vector via the AST node relationships. Because the number of nodes in each tree is different, the length of each tree vector is not the same. However, for the ML/DL model, the input length of all data should be identical. To solve this problem, we zero-padded the end of each tree vector and fixed the most extended length as L. In our experiments, we will discuss how the length L of the vector representations affects performance.

2) BEHAVIOR MULTI-LABEL CLASSIFIER

After extracting features from AST of PSCmds, we then build a multi-label classifier for behavioral profiling. Specifically,

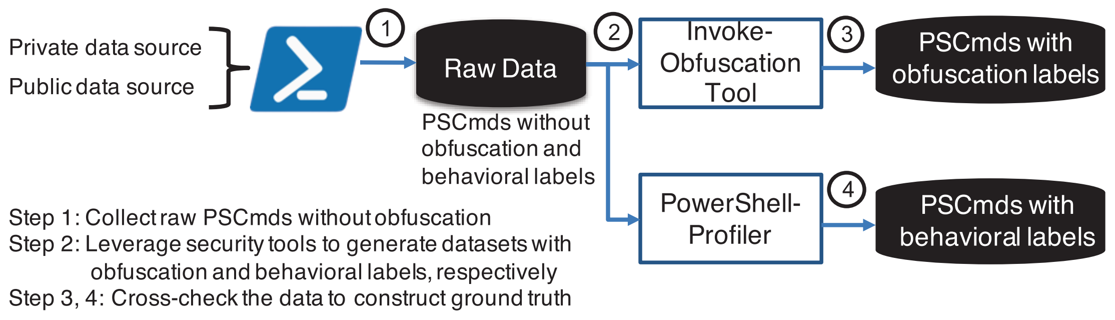
Private data source
Public data source
1
Raw Data
PSCmds without obfuscation and behavioral labels
2
Invoke-Obfuscation Tool
3
PSCmds with obfuscation labels
PowerShell-Profiler
4
PSCmds with behavioral labels
Step 1: Collect raw PSCmds without obfuscation
Step 2: Leverage security tools to generate datasets with obfuscation and behavioral labels, respectively
Step 3, 4: Cross-check the data to construct ground truth

FIGURE 3. The flow diagram of the dataset creation.

we implement a different DNN architecture, with detailed information described in Table 5. We trained the classifier with different experimental settings and selected the one with the best performance to verify the effectiveness of behavior multi-label classification.

IV. THE DATASET

There are many publicly available PowerShell corpora on the internet, such as Microsoft PowerShell gallery, GitHub repositories, and blogs. However, most of them are non-malicious, non-obfuscated, and unlabeled. In order to evaluate our system, we need a labeled dataset that can provide the obfuscation types of PSCmds and the malicious behavioral labels. Therefore, we first collect malware-derived PSCmds without obfuscation as our primary dataset. We then utilize security tools to generate the desired obfuscated PSCmds with obfuscation labels and annotate the non-obfuscated PSCmds with their behavioral labels. Fig. 3 shows the flow diagram of our dataset creation. We explain the process in detail as follows.

A. RAW DATA COLLECTION

We obtained malicious PSCmds from internal sandbox analysis reports as our initial dataset. For enrichment, we also gathered PSCmds from the open dataset, including Palo Alto's research data[8] and various online sandboxes such as Any.run,[9] FileScan.IO,[10] Hatching Triage,[11] and VirusShare,[12] a malware repository for research. Besides Palo Alto's data, we used the keyword "PowerShell" with case insensitive search to discover desired malicious PSCmds from each dataset. We examined all detected scripts between 2019 and 2021 according to their submitted time. Table 6 shows the number of malicious PSCmds we obtained from each dataset. All received scripts and commands are identified as malicious by dynamic analysis of sandbox or static detection by at least three anti-virus programs. After manually removing duplicated and obfuscated PSCmds from the previous collection stage, we ended up with 3,057 non-obfuscated PSCmds executed by malware.

[8]https://github.com/pan-unit42/iocs/tree/master/psencmds
[9]https://any.run/malware-reports/
[10]https://www.filescan.io/reports/
[11]https://tria.ge/reports/public/
[12]https://virusshare.com/

TABLE 6. The collection of non-obfuscated malicious PSCmds.

Dataset	# of samples	Period
Internal sandbox reports	495	2019 to 2021
Palo Alto's research data	1,644	before 2017
Any.run	85	2021
FileScan.IO	175	2021
Hatching Triage	156	2020 to 2021
VirusShare	502	2019 to 2021

B. GROUND TRUTH CREATION

After collecting the raw samples of PSCmds, we generate two datasets for obfuscation multi-label classification and behavior multi-label classification, respectively. First, to achieve the obfuscation multi-label classification, we manually generate obfuscated PSCmds based on our raw data using the Invoke-Obfuscation tool [32], a PowerShell v2.0+ compatible PowerShell command and script obfuscator, to build the ground truth of the obfuscated dataset. Then, we apply 12 obfuscation methods in three categories and three helper functionalities introduced in Section II for the 3,057 non-obfuscated PSCmds. After this step, we have 39,741 PSCmds samples, which comprise 3,057 non-obfuscated samples and 36,684 obfuscated samples with labels.

Second, to perform behavioral profiling with multi-label classification, we need to annotate the predefined behavioral labels for each PSCmds. However, it is challenging and infeasible to annotate all behaviors for PSCmds. Therefore, we leverage the PowerShellProfiler,[13] a pattern-based PowerShell analyzer developed by Palo Alto's researchers [17], to manually identify negative, neutral, and benign behaviors in 37 predefined behavioral labels. After analyzing the 3,057 non-obfuscated PSCmds, we have 23 behavioral labels in our dataset. To avoid the problem of imbalanced data, we only keep behavioral labels with at least 1,000 samples and remove ambiguous and non-behavioral labels. As a result, we use the top 5 behavioral labels for our behavioral profiling assessment, as shown in Table 7.

C. DATA CROSS-CHECK

To confirm that the two datasets generated by the security tools are appropriate for our evaluation, two security professionals helped review and cross-check the datasets.

[13]https://github.com/pan-unit42/public_tools/tree/master/powershell profiler

TABLE 7. Top 5 behavioral labels with at least 1,000 samples for non-obfuscated PSCmds.

#	Behavioral labels	# of samples
1	Sleeps	1,413
2	Known Malware	1,409
3	Code Injection	1,237
4	Byte Usage	1,228
5	Downloader	1,226

The number of labels for PSCmds in the obfuscation dataset is between 0 and 4, i.e., the maximum obfuscation methods applied to PSCmds are four in our dataset, and the methods for non-obfuscated PSCmds are zero. Meanwhile, the maximum and the minimum number of behavioral labels for non-obfuscated PSCmds range from 0 to 5. After manually correcting some mislabeling for obfuscation and behavioral labels, we formed the final ground truth datasets for the multi-label classification.

V. EVALUATION

In this section, we present the evaluation results and show the performance and effectiveness of our proposed work. Specifically, we conduct extensive experiments to answer the following research questions:

- **RQ1:** What is the performance of using character distribution features to detect and differentiate multiple obfuscation types?
- **RQ2:** What is the successful recovery rate of the de-obfuscator in PowerDP against various obfuscation methods?
- **RQ3:** How effective is PowerDP using the real-valued vector representations to profile multiple malicious behaviors of PSCmds on a high abstraction level of AST?

A. EVALUATION METHODOLOGY

In this work, we employ the following standard metrics and methods to evaluate the performance and correctness of PowerDP.

1) PERFORMANCE METRICS

Various evaluation measures have been developed for ML/DL solutions. In addition to the standard Accuracy, Precision, Recall, and F1 scores, we also adopt Hamming loss, a popular metric for evaluating multi-label classification problems.

- Hamming loss (HL): It reports how many times on average, the relevance of a sample to a class label is mispredicted. The smaller the Hamming loss value, the better the classification performance. It can define as Eq. (5), where Y_i^j is the prediction giving the predicted labels for i^{th} sample and j^{th} class, and Z_i^j is the target giving the actual labels for the i^{th} sample and the j^{th} class.

$$HL = \frac{1}{k}\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k}(Y_i^j \oplus Z_i^j) \tag{5}$$

TABLE 8. The Hamming loss, Accuracy score, Precision score, Recall score, and F1 score comparison with different classification approaches.

Algorithm	HL(%)	AS(%)	PS(%)	RS(%)	F1(%)
SVM	0.94	87.26	95.76	93.65	94.56
RF	**0.18**	97.56	**99.69**	98.98	99.33
KNN	0.69	93.16	96.96	95.88	96.29
DT	0.43	95.91	98.22	98.13	98.17
DNN	**0.18**	**99.82**	99.44	**99.53**	**99.48**

- Accuracy score (AS): It is also called exact match ratio or subset accuracy. As shown in Eq. (6), it is the most strict metric, indicating the proportion of samples where all labels are correctly classified, where I is the indicator function.

$$AS = \frac{1}{n}\sum_{i=1}^{n} I(Y_i = Z_i) \tag{6}$$

- Precision score (PS): A standard metric as shown in Eq. (7), where the precision score is the proportion of predicted correct labels to the number of predicted labels, averaged across all samples.

$$PS = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Y_i|} \tag{7}$$

- Recall score (RS): A standard metric as shown in Eq. (8), where the recall score is the proportion of predicted correct labels to the total number of actual labels, averaged across all samples.

$$RS = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap Z_i|}{|Z_i|} \tag{8}$$

- F1 score (F1): It seeks a balance between precision and recall and is interpreted as the harmonic mean of the precision and recall scores. Eq. (9) shows the formula for the computation of the F1 score.

$$F1 = \frac{2 * PS * RS}{PS + RS} \tag{9}$$

2) SIMILARITY COMPARISON

Ideally, the PSCmds de-obfuscation process can recover the same contents as the original ones. However, it is not easy to achieve such perfection in practice. Therefore, we need an indicator to evaluate the overall recovery effect of our work. We apply clone detection methods using abstract syntax suffix trees [45] to compare the similarity between the de-obfuscated PSCmds and their original content. Previous research has divided the clones into three types:

- **Type 1** clones are exact copies without modifications, except for possible differences in whitespace and comments.
- **Type 2** clones are syntactically identical copies with parameter changes such as function names, types, and variables.
- **Type 3** clones are syntactically dissimilar copies with modifications of statements but are semantically similar.

TABLE 9. The Hamming loss of different classification approaches against different obfuscation methods.

HL(%) Obf. / Algo.	spl.	rep.	rnd.	cmp.	ascii	b64.	bin.	bxor.	hex.	oct.	secstr.	concat.	rod.	rev.	tick
SVM	0.05	**2.5**	1.5	0.15	**3.32**	0.09	0.16	**3.46**	0.35	0.15	0.14	0.11	0.23	1.77	0.05
RF	0.0	**0.72**	**0.89**	0.0	0.25	0.0	0.0	**0.31**	0.08	0.0	0.01	0.13	0.05	0.2	0.03
KNN	0.09	**2.55**	1.11	0.08	1.47	0.04	0.05	**1.92**	0.08	0.31	0.01	0.28	0.1	**1.57**	0.75
DT	0.24	**1.07**	**1.65**	0.11	**0.84**	0.03	0.11	0.77	0.36	0.11	0.04	0.33	0.3	0.35	0.11
DNN	0.01	**0.52**	**0.81**	0.0	0.5	0.0	0.0	**0.54**	0.08	0.05	0.0	0.08	0.0	0.16	0.01

Since we leverage the methods in [45] to compare similarities by converting the AST of PSCmds into a semantic string sequence using every unique letter in 'A-Za-t' as an identifier for each AST node type, we can treat Type 1 and Type 2 as the same type of clones. Therefore, for the exact clones detection (Type1 & Type 2), we check the *Longest Common Subsequence* (LCS) [46] metric of the de-obfuscated PSCmds and their original content. If the LCS metric is identical to the semantic string sequence of the original PSCmds, we claim that they are full-cloned. The LCS metric is defined as follows:

$$LCS(X_i, Y_j) = \begin{cases} 0, \text{ if } & i = 0 \text{ or } j = 0 \\ LCS & (X_{i-1}, Y_{j-1}) \wedge x_i, \\ \text{if} & i, j > 0 \text{ and } x_i = y_j \\ max & \{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\}, \\ \text{if} & i, j > 0 \text{ and } x_i \neq y_j \end{cases} \tag{10}$$

where $X = \{x_i, i = 1, 2, \ldots, m\}$ is the semantic string sequence of de-obfuscated PSCmds, and $Y = \{y_j, j = 1, 2, \ldots, n\}$ is the semantic string sequence of the original PScmds.

For the semantic clones detection (Type 3), we calculate the *EditRatio* from the Levenshtein distance [47] as a comparable metric between the de-obfuscated PSCmds and the original content. The Levenshtein distance between two strings is the minimum number of single-character operations required to change one into the other, including insertions (cost of 1), deletions (cost of 1), and substitutions (cost of 2). For example, the following equation calculates the *EditRatio* between two semantic string sequences of PSCmds:

$$EditRatio(X, Y) = 1 - \frac{LevenshteinDistance(X, Y)}{|X| + |Y|} \tag{11}$$

The *EditRatio* close to 1 suggests the excessive semantic similarity between two PSCmds, which means the effective recovery results, whereas the value closed to 0 indicates the opposite. We set our threshold to 0.7 and take only the samples whose *EditRatio* exceeds the threshold, claiming that they are copy-and-modified.

B. EVALUATION RESULTS

1) PERFORMANCE OF MODELS ON OBFUSCATION MULTI-LABEL CLASSIFICATION

In our first experiment, we used all 36,684 obfuscated samples with 15 labels from 3 different obfuscation categories and 3,057 non-obfuscated PSCmds with all zero labels in raw data. After mixing and shuffling all 39,741 samples, we followed the 80/20 rule to separate our dataset as training and testing sets. A total of 97 character-based features were used in the experiment. To find the best performance by the automatic multi-label classification results of the obfuscation classifier, we performed 5-fold cross-validation tests in all models. As shown in Table 8, the best result in each evaluation metric is marked in bold.

The results showed that the minimum value of the hamming loss came from the RF and DNN models, which showed excellent multi-label classification performance compared to others. As for the comparison between the RF and DNN models, even though the precision of RF is superior to DNN, we still chose DNN model as our desired classifier in PowerDP. This is because the DNN model has a higher recall score than RF, which means DNN has fewer misprediction results. Thus, it is reasonable to state that DNN performs more reliably in obfuscation multi-label classification. In PowerDP, the prediction of the obfuscation multi-label classification is passed to the extensive PowerShell de-obfuscator. Moreover, the de-obfuscator attempts to recover the obfuscation to the original content by using different methods based on the predicted labels. Therefore, fewer misprediction lead to a more successful recovery.

2) ACCURACY OF PREDICTION AGAINST DIFFERENT OBFUSCATION TYPES

Table 9 presents the hamming loss of 5 ML/DL models against different obfuscation labels. Hamming loss evaluates how many times a label is misclassified. The greater the value of hamming loss means worse performance. We marked the top 3 misclassified labels for each model in bold. As shown in Table 8, the obfuscation using "Replacement," "Binary Exclusive OR," and "Randomcase" methods are the three most misclassified types.

After reviewing the distribution of characters in the PSCmds obfuscated by the "Binary Exclusive OR" methods, we found that they are very similar to other PSCmds with different obfuscation. Therefore, they may be poorly classified due to not too many differences in character distribution. In addition, as the helper functions, "Replacement" and "Randomcace" methods are often combined with other obfuscation strategies. However, from the statistical information, they do not show many unique characteristics compared with different obfuscation types. Instead, it is easier to detect them if we use traditional regex patterns

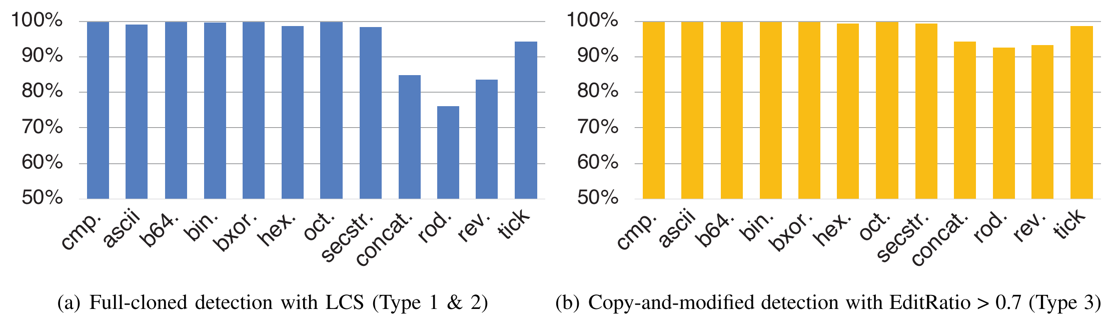

FIGURE 4. The successful recovery ratio of the de-obfuscator against different obfuscation methods.

TABLE 10. The Hamming loss, Accuracy score, Precision score, Recall score, and F1 score comparison with different number of nodes to construct the AST vector.

# of nodes	Coverage of samples	Vector size	HL(%)	AS(%)	PS(%)	RS(%)	F1(%)
0	0.0%	5	10.62	89.38	89.72	96.47	92.97
$\leq$ 10	9.39%	225	4.15	95.85	95.15	97.93	96.52
$\leq$ 20	30.59%	445	**1.47**	**98.53**	**97.94**	98.31	**98.12**
$\leq$ 50	50.15%	1105	1.67	98.33	97.64	**98.47**	98.05
$\leq$ 100	59.99%	2205	1.67	98.33	97.42	98.39	97.90
$\leq$ 1392	100%	30629	1.96	98.04	97.03	97.85	97.44

to discover the keywords for "Replacement" or to detect specific commands in mixed cases for "Randomcase."

Regarding the four obfuscation methods in the string manipulation category, "Reversing" is the worst one in classification. It is obvious to have such results, compared to other methods in the same category, the "Reversing" method does not provide too much insertion or modification to the original PSCmds except for reversing the whole expression string. On the other hand, DNN performed the classification well on average among all models. Five obfuscation types can be 100 percent predicted with the DNN model in our experiment, including "Compression," "Base64 Encoding", "Binary Encoding," "SecureString," and "Reordering." From the point of human vision, all of them have unique textual representations in obfuscation.

3) CORRECTNESS OF THE EXTENSIVE PowerShell DE-OBFUSCATION

When there are multiple obfuscations, our empirically designed de-obfuscator endeavors to recover obfuscated PSCmds to their original content according to the de-obfuscation order in Table 2. We evaluate the recovery quality by comparing the similarity between the de-obfuscated PSCmds and their original content. In this experiment, we use all 36,684 obfuscated samples generated by the 12 obfuscation methods in the "Invoke-Obfuscation" tool with their ground-truth obfuscation labels mentioned above.

As shown in Fig. 4, the diagram on the left side presents the successful recovery ratio of the de-obfuscator, which uses full-cloned detection for similarity comparison; the diagram on the right side shows the same ratio but uses copy-and-modified detection instead. Among all, the recovery ratios of eight obfuscation methods in the categories of "Compress" and "Encoding" are higher than 98.5% with full-cloned detection, which means only 1.5% of obfuscated samples using the "Compress" and "Encoding" strategies can not be recovered ideally by the de-obfuscator. This is because "Compress" and "Encoding" schemes encapsulate the whole expression string of PSCmds in their encoding representatives. Once the de-obfuscator discovers the core encapsulation, it can transform the encapsulation to the original content according to the encoding schemes.

Concerning the four obfuscation methods in the "String manipulation," all recovery ratios are higher than 76.09% with full-cloned detection, which decreases significantly from the other two categories. Since "String manipulation" changes the whole expression string with multiple insertions, deletions, and substitutions, the de-obfuscated contents are not strictly the same as the original ones. Therefore, we use copy-and-modified detection to compare the semantic similarity between the de-obfuscated and original samples. As shown in Fig. 4(b), the recovery ratio increased remarkably from 76.09% to 92.54% for de-obfuscated PSCmds in the "String manipulation" category with the copy-and-modified detection (*EditRatio* > 0.7). In the end, the average recovery ratio of the de-obfuscator is 94.54% with full-clone detection, and it increases to 98.11% on average with copy-and-modified detection.

4) EFFECTIVENESS ON BEHAVIOR MULTI-LABEL CLASSIFICATION

To validate the effectiveness of behavior multi-label classification, we select five behavioral labels, including sleep, known malware, code injection, byte usage, and downloader, to annotate the behaviors in 3,057 non-obfuscated PSCmds. We converted PSCmds to their AST structures and then built the real-valued vector representations from the vectors of

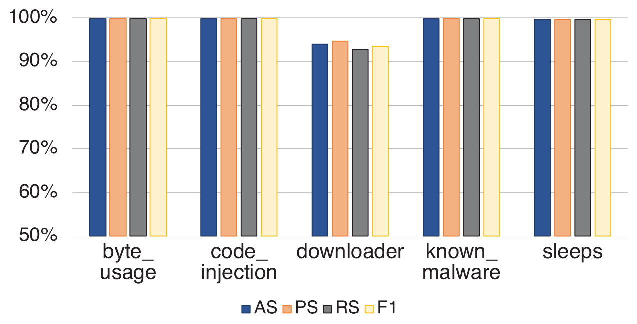

FIGURE 5. The performance of behavioral profiling with real-valued vector representations in 20 AST nodes.

AST node types as the features. First, since the size of each real-valued vector varies based on the number of nodes in the AST of PSCmds before padding, we evaluate the impact of the real-valued vector size on the performance in behavior multi-label classification. In our experimental setting, we set the vector size of each AST node type to 22 and used five statistical features mentioned in Section III as default features.

In our dataset, more than 40% of samples have more than 100 AST nodes, and the maximum number of nodes is 1,392. Therefore, we compared the number of nodes in {0, 10, 20, 50, 100, 1392} to form the real-valued vector representations and examine the impact of real-valued vector size on the effectiveness. Table 10 shows the overall comparative results. From the experiment, we found that the basic statistical features of the AST were enough to profile behaviors of the PSCmds with an accuracy score of 89.38%. If we assemble sub-vectors of 20 AST nodes as the real-valued vector feature in breadth-first-search order, we can escalate the accuracy score to 98.53% and decrease the hamming loss from 10.62% to 1.47%. The notable improvement shows that the ensemble order and the size of the real-valued vector are both significant factors for effectiveness. Furthermore, we also noted that the performance decreased marginally when we used more sub-vectors of AST nodes to compose the real-valued vector, which illustrated negative impacts on zero-padding in the vectors.

Fig. 5 reports the effective results of behavior multi-label classification with the experimental settings in 20 AST nodes and the DNN model. Except for the "downloader" label, the DNN model can predict all other four behaviors among all samples with an accuracy score of over 99.5%. After examing the content and the AST structure of the misclassified samples with the "downloader" label, we discovered that the downloading commands are placed in the tail of these PSCmds. In addition, some commands generated too many nodes after transforming the PSCmds into the AST. Therefore, with the settings of assembling sub-vectors of 20 AST nodes to the real-valued vector, the feature vector may lack critical information of "downloader" behavior in the vector representations. From the results, it is clear that the length L of the real-valued vector representations is a tradeoff factor for the effectiveness of behavioral classification.

VI. DISCUSSION AND LIMITATION

PSCmds with obfuscation are complex and provide flexibility for attackers to hide malicious intent from the context of code. Although many authors have conducted studies on PowerShell detection and de-obfuscation, this problem is still insufficiently explored. This paper introduces many obfuscation methods and attempts to use a hybrid methodology combineing deep learning and program analysis to de-obfuscate PSCmds and then profile their behaviors with multi-label classification. With the native character-based features from program analysis and the ensemble real-valued vector representations from AST abstraction, we highlight the opportunities for improvement on automatic de-obfuscation and behavioral profiling tasks and bridge the gap between malware analysis and machine learning. However, we found that some obfuscation does not present a different textual representation and unique distribution of characters. Besides, it is also difficult to exactly recover some obfuscation methods with string manipulation to their original content. Thus, we transform the recovery problem into the semantic similarity comparison for proof-of-concept, which may pose a risk in some cases.

Due to the lack of labeled data with ground truth, we created two datasets that rely on open-sourced security tools developed by other security researchers. Even though the tools provide various options and methods to generate our desired data, some other obfuscation methods and behavioral labels have not been comprehensively discussed in our work. One of our limitations is that the ground truth is a closed source and does not fully cover all obfuscations and behaviors of malicious PSCmds. Second, we empirically extend other previous works on static regex replacement to design a PowerShell de-obfuscator in python language that employs regex for a quick and effective de-obfuscation process. However, we do not handle multi-layer obfuscation and other more complex techniques, such as mixed obfuscation with native windows commands, PowerShell cmdlets, and dotNet APIs.

VII. RELATED WORK

This section provides an overview of the literature on malicious PowerShell detection and de-obfuscation over the last few years. Most ML/DL works focused on binary classification that discriminates malicious PSCmds from benign ones. Meanwhile, other researchers proposed innovative and practicable methods using the subtree in the AST or classical regex for de-obfuscation.

A. DETECTION OF MALICIOUS PowerShell

Dynamic mechanisms using system monitoring [12], [13] or script block logging via AMSI to detect malicious PowerShell have already been suggested as straightforward ideas in the previous work. Several security vendors also proposed their practical solutions. For example, Jeff White [16] from Palo Alto provided a set of observations of malicious PowerShell behavior with in-depth static analysis. With the advent of emerging ML/DL, Bohannon et al. from

FireEye [18] leveraged machine learning methods like linear regression and gradient descent algorithms to detect malicious PowerShell. Additionally, many DL approaches combine the notions of AST, NLP, or Graph, which perform excellently in differentiating between malicious and benign scripts.

AST is commonly used in programming language processing systems to represent the abstract syntactic structure of programs. Rusak et al. [22] proposed malicious PowerShell detection methods by combining AST and DL. They model PowerShell code using AST, and then utilize the structural information of AST to create the ensemble vector representations of each PowerShell. Using the novel idea with DL, they successfully identify common intrinsic patterns of PowerShell codes and distinguish malware families. Song et al. [25] try to optimize feature selection with a mixture of token-based and AST-based keyword extraction. By combining the vocabulary and structure of PowerShell code, the performance in detecting malicious PowerShell is increased.

Since DL methods for NLP have had great success recently, Hendler et al. [27], [28] have shown that DL-based detectors that use character-level information and contextual embeddings of words from commands contribute most to malicious PowerShell detection. In the first work, they extract character-level information as inputs to a convolutional neural network (CNN) [48] from the perspective of DL. Their evaluation findings indicate that an ensemble detector that combines an NLP-based classifier with a CNN-based classifier performs best. In the later work, they project semantically similar words to approximate vectors in the embedding space with contextual word embeddings. Combining character-level and token-level details for DL, they attained impressive performance with a low false positive rate of less than 0.1%. Meanwhile, Fang et al. [24] and Mimura et al. [29] proposed conceptually identical working structures using traditional NLP-based methods such as FastText [49] and Doc2vec [50]. By extracting hybrid features from PowerShell, their models demonstrate the capability to classify complex PSCmds and make improvements over other works.

Finally, Ongun et al. [30] adopt active learning and cmd2vec feature generation to expand the detection range to LotL threats, which encompass attacks with more system preinstallation tools and commands. Sunoh Choi [31] provides another perspective in turning detection into a graph inference problem using GNN and an adjacency matrix generation method via Jaccard similarity to detect malicious PowerShell. Without a doubt, ML/DL detection methods benefit in many ways. However, the full potential of ML/DL is not realized in malicious PowerShell analysis, particularly in de-obfuscation and behavioral profiling. In this paper, we propose a coarse-to-fine grained framework PowerDP that can be easily combined with previous detection-based methods and further infers the intents of malicious PSCmds through behavior multi-label classification.

B. DE-OBFUSCATION OF MALICIOUS *PowerShell*

Despite the advances of ML/DL in malicious PowerShell detection, there is still no consensus on the perfect solution to de-obfuscate malicious PowerShell. Liu et al. [14] proposed a framework called PSDEM that applies a two-layer de-obfuscation technique to recover the original content of obfuscated malicious PowerShell codes in Word documents. From the results, PSDEM can statically extract the commands and present the original script to users with no false positives compared to other security tools and anti-virus programs. In related works, Jeff White [17] has developed a well-designed tool called PowerShellProfiler, which not only can handle de-obfuscation for PowerShell code but also statically create behavioral profiles using numerous predefined regex patterns. Ugarte et al. [15] designed the PowerDrive, an open-source static and dynamic multi-stage de-obfuscator for PowerShell attacks. PowerDrive leverages regex for recursive de-obfuscation through multi-layer recovery. In addition, a taxonomy of behavioral models and a comprehensive list of malicious domains during the analysis are reported.

Seemingly, the regex technique is still a dominant solution to provide the analyst with effectiveness against various obfuscation methods. However, related solutions may easily fall into a cat-and-mouse race with attackers and lack of efficiency in recursively unraveling new contents of specific patterns or keywords that may be obfuscated or hidden from context. On the other hand, AST provides researchers with another solution to gain unprotected insight into the structure of obfuscated PSCmds. Li et al. [23] proposed the first semantic-aware PowerShell attack detection system that performs obfuscation detection and emulation-based recovery. The system de-obfuscates the obfuscation at the level of subtrees in the AST of PowerShell scripts. Based on the evaluation results with the dataset, the system outperforms both Windows Defender[14] and VirusTotal.[15] However, the system does not support behavioral profiling and the number of handled obfuscation methods is limited. Ultimately, our approach is inspired by the work of previous eminent researchers. How to balance the efficient de-obfuscation process with ML/DL and preserve the complementary semantic context for program analysis are vital issues.

VIII. CONCLUSION AND FUTURE WORK

This paper introduced PowerDP, a hybrid framework combining deep learning and program analysis for automatic PowerShell de-obfuscation and behavioral profiling. We leveraged multi-label classification to forecast 15 obfuscation types to improve the accuracy of an extensive de-obfuscator and successfully profiled five behaviors of malicious PowerShell commands. Our method achieves superior performance in classifying different obfuscation types using character distribution features and detecting malicious behaviors with real-valued vector representations from the abstract syntax

[14]https://www.microsoft.com/en-us/windows/comprehensive-security
[15]https://www.virustotal.com/

tree. To the best of our knowledge, PowerDP is the first work to employ the multi-label classification that infers the obfuscation types and behavioral labels of malicious PowerShell commands before and after the de-obfuscation, respectively. In the future, we will expand our work against more obfuscation types and malicious behaviors. We will also look at multi-layer obfuscation and improve classification performance and accuracy.

REFERENCES

[1] J. J. George and D. E. Leidner, "From clicktivism to hacktivism: Understanding digital activism," *Inf. Org.*, vol. 29, no. 3, Sep. 2019, Art. no. 100249.

[2] S. Afroz, V. Garg, D. McCoy, and R. Greenstadt, "Honor among thieves: A common's analysis of cybercrime economies," in *Proc. 8th IEEE APWG eCrime Res. Summit*, Sep. 2013, pp. 1–11.

[3] I. Ghafir, V. Prenosil, M. Hammoudeh, F. J. Aparicio-Navarro, K. Rabie, and A. Jabban, "Disguised executable files in spear-phishing emails: Detecting the point of entry in advanced persistent threat," in *Proc. 2nd Int. Conf. Future Netw. Distrib. Syst. (ICFNDS)*, 2018, pp. 1–5.

[4] Sudhakar and S. Kumar, "An emerging threat fileless malware: A survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, Dec. 2020.

[5] F. Barr-Smith, X. Ugarte-Pedrero, M. Graziano, R. Spolaor, and I. Martinovic, "Survivalism: Systematic analysis of Windows malware living-off-the-land," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2021, pp. 1557–1574.

[6] S. M. Pontiroli and F. R. Martinez, "The Tao of .NET and powershell malware analysis," in *Proc. Virus Bull. Conf.*, 2015, pp. 1–26.

[7] A. G. Kakisim, M. Nar, and I. Sogukpinar, "Metamorphic malware identification using engine-specific patterns based on co-opcode graphs," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103443.

[8] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. 5th Int. Conf. Broadband, Wireless Comput., Commun. Appl. (BWCCA)*, Nov. 2010, pp. 297–300.

[9] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, and K. De Bosschere, "Software protection through dynamic code mutation," in *Proc. 7th Int. Workshop. Inf. Secur. Appl. (WISA)*. Berlin, Germany: Springer, 2006, pp. 194–206.

[10] D. Javaheri, P. Lalbakhsh, and M. Hosseinzadeh, "A novel method for detecting future generations of targeted and metamorphic malware based on genetic algorithm," *IEEE Access*, vol. 9, pp. 69951–69970, 2021.

[11] J.-Y. Kim and S.-B. Cho, "Obfuscated malware detection using deep generative model based on global/local features," *Comput. Secur.*, vol. 112, Jan. 2022, Art. no. 102501.

[12] A. Rousseau, "Hijacking .NET to defend PowerShell," 2017, *arXiv:1709.07508*. Accessed: Nov. 23, 2022.

[13] M. Manna, A. Case, A. Ali-Gombe, and G. G. Richard, "Memory analysis of .NET and .Net core applications," *Forensic Sci. Int., Digit. Invest.*, vol. 42, Jul. 2022, Art. no. 301404.

[14] C. Liu, B. Xia, M. Yu, and Y. Liu, "PSDEM: A feasible de-obfuscation method for malicious PowerShell detection," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 825–831.

[15] D. Ugarte, D. Maiorca, F. Cara, and G. Giacinto, "Powerdrive: Accurate de-obfuscation and analysis of PowerShell malware," in *Proc. 16th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment (DIMVA)*. Cham, Switzerland: Springer, 2019, pp. 240–259.

[16] J. White. (2017). Pulling back the curtains on encodedcommand PowerShell attacks. Palo Alto Networks. Accessed: Nov. 23, 2022. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-pulling-back-the-curtains-on-encodedcommand-powershell-attacks/

[17] Palo Alto Networks. (2019). *Practical Behavioral Profiling of Powershell Scripts Through Static Analysis (Part 1—Part 3)*. Accessed: Nov. 23, 2022. [Online]. Available: https://unit42.paloaltonetworks.com/practical-behavioral-profiling-of-owershell-scripts-through-static-analysis-part-1/

[18] D. Bohannon and L. Holmes, "Revoke-obfuscation: PowerShell obfuscation detection using science," in *Proc. Black Hat USA*. Milpitas, CA, USA: FireEye, 2017, pp. 1–20.

[19] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 3, pp. 773–788, Mar. 2019.

[20] Y. Shen and G. Stringhini, "ATTACK2VEC: Leveraging temporal word embeddings to understand the evolution of cyberattacks," in *Proc. 28th USENIX Secur. Symp.* Berkeley, CA, USA: USENIX Association, 2019, pp. 905–921.

[21] S. Ndichu, S. Kim, and S. Ozawa, "Deobfuscation, unpacking, and decoding of obfuscated malicious Javascript for machine learning models detection performance improvement," *CAAI Trans. Intell. Technol.*, vol. 5, no. 3, pp. 184–192, Sep. 2020.

[22] G. Rusak, A. Al-Dujaili, and U.-M. O'Reilly, "AST-based deep learning for detecting malicious PowerShell," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2018, pp. 2276–2278.

[23] Z. Li, Q. A. Chen, C. Xiong, Y. Chen, T. Zhu, and H. Yang, "Effective and light-weight deobfuscation and semantic-aware attack detection for PowerShell scripts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Nov. 2019, pp. 1831–1847.

[24] Y. Fang, X. Zhou, and C. Huang, "Effective method for detecting malicious PowerShell scripts based on hybrid features," *Neurocomputing*, vol. 448, pp. 30–39, Aug. 2021.

[25] J. Song, J. Kim, S. Choi, J. Kim, and I. Kim, "Evaluations of AI-based malicious PowerShell detection with feature optimizations," *ETRI J.*, vol. 43, no. 3, pp. 549–560, Jun. 2021.

[26] H. Chai, L. Ying, H. Duan, and D. Zha, "Invoke-deobfuscation: AST-based and semantics-preserving deobfuscation for PowerShell scripts," in *Proc. 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2022, pp. 295–306.

[27] D. Hendler, S. Kels, and A. Rubin, "Detecting malicious PowerShell commands using deep neural networks," in *Proc. Asia Conf. Comput. Commun. Secur. (AsiaCCS)*, May 2018, pp. 187–197.

[28] D. Hendler, S. Kels, and A. Rubin, "AMSI-based detection of malicious PowerShell code using contextual embeddings," in *Proc. 15th ACM Asia Conf. Comput. Commun. Secur. (AsiaCCS)*, Oct. 2020, pp. 679–693.

[29] M. Mimura and Y. Tajiri, "Static detection of malicious PowerShell based on word embeddings," *Internet Things*, vol. 15, Sep. 2021, Art. no. 100404.

[30] T. Ongun, J. W. Stokes, J. B. Or, K. Tian, F. Tajaddodianfar, J. Neil, C. Seifert, A. Oprea, and J. C. Platt, "Living-off-the-land command detection using active learning," in *Proc. 24th Int. Symp. Res. Attacks, Intrusions Defenses (RAID)*, Oct. 2021, pp. 442–455.

[31] S. Choi, "Malicious powershell detection using graph convolution network," *Appl. Sci.*, vol. 11, no. 14, p. 6429, Jul. 2021.

[32] D. Bohannon. (2016). Invoke-obfuscation: PowerShell obFUsk8tion Techniques & How to (Try to) D e'Tec'T'th'+'em'. DerbyCon. Accessed: Nov. 23, 2022. [Online]. Available: https://www.danielbohannon.com/blog-1/2016/9/25/invoke-obfuscation-public-release

[33] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: Learning distributed representations of code," in *Proc. ACM Program. Lang. (PACMPL)*, vol. 3, 2019, pp. 1–29.

[34] W. Wang, Y. Zhang, Y. Sui, Y. Wan, Z. Zhao, J. Wu, P. S. Yu, and G. Xu, "Reinforcement-learning-guided source code summarization using hierarchical attention," *IEEE Trans. Softw. Eng.*, vol. 48, no. 1, pp. 102–119, Jan. 2022.

[35] B. Gelman, B. Hoyle, J. Moore, J. Saxe, and D. Slater, "A language-agnostic model for semantic source code labeling," in *Proc. 1st Int. Workshop Mach. Learn. Softw. Eng. Symbiosis (MASES)*, Sep. 2018, pp. 36–44.

[36] R. Wang, H. Zhang, G. Lu, L. Lyu, and C. Lyu, "Fret: Functional reinforced transformer with BERT for code summarization," *IEEE Access*, vol. 8, pp. 135591–135604, 2020.

[37] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int. J. Data Warehousing Mining*, vol. 3, no. 3, pp. 1–13, 2007.

[38] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, Y. Wang, and Y. Xiang, "A3CM: Automatic capability annotation for Android malware," *IEEE Access*, vol. 7, pp. 147156–147168, 2019.

[39] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android malware category classification using semi-supervised deep learning," in *Proc. IEEE Int. Conf Dependable, Autonomic Secure Comput., Int. Conf Pervasive Intell. Comput., Int. Conf Cloud Big Data Comput., Int. Conf Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522.

[40] F. N. Ducau, E. M. Rudd, T. M. Heppner, A. Long, and K. Berlin, "Automatic malware description via attribute tagging and similarity embedding," 2019, *arXiv:1905.06262*. Accessed: Nov. 23, 2022.

[41] M. R. Smith, N. T. Johnson, J. B. Ingram, A. J. Carbajal, B. I. Haus, E. Domschot, R. Ramyaa, C. C. Lamb, S. J. Verzi, and W. P. Kegelmeyer, ''Mind the gap: On bridging the semantic gap between machine learning and malware analysis,'' in *Proc. 13th ACM Work. Artif. Intell. Secur. (AISec)*, 2020, pp. 49–60.

[42] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay, ''Scikit-learn: Machine learning in Python,'' *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[43] H. Peng, L. Mou, G. Li, Y. Liu, L. Zhang, and Z. Jin, ''Building program vector representations for deep learning,'' in *Proc. 8th Int. Conf. Knowl. Sci., Eng. Manage. (KSEM)*, vol. 9403. Cham, Switzerland: Springer, 2015, pp. 547–553.

[44] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, ''Convolutional neural networks over tree structures for programming language processing,'' in *Proc. 13th AAAI Conf. Artif. Intell.* Palo Alto, CA, USA: AAAI, 2016, pp. 1287–1293.

[45] R. Koschke, R. Falke, and P. Frenzel, ''Clone detection using abstract syntax suffix trees,'' in *Proc. 13th Work. Conf. Reverse Eng. (WCRE)*, 2006, pp. 253–262.

[46] L. Bergroth, H. Hakonen, and T. Raita, ''A survey of longest common subsequence algorithms,'' in *Proc. 7th Int. Symp. String Process. Inf. Retr. (SPIRE)*, 2000, pp. 39–48.

[47] L. Yujian and L. Bo, ''A normalized Levenshtein distance metric,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1091–1095, Jun. 2007.

[48] X. Zhang, J. Zhao, and Y. LeCun, ''Character-level convolutional networks for text classification,'' in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, vol. 28. Red Hook, NY, USA: Curran Associates, 2015, pp. 649–657.

[49] S. Wu and U. Manber, ''Fast text searching: Allowing errors,'' *Commun. ACM*, vol. 35, no. 10, pp. 83–91, Oct. 1992.

[50] Q. Le and T. Mikolov, ''Distributed representations of sentences and documents,'' in *Proc. 31st Int. Conf. Mach. Learn. (ICML)*, 2014, vol. 32, no. 2, pp. 1188–1196.

MENG-HAN TSAI (Member, IEEE) received the B.S. degree in computer science and information engineering from National Chung Cheng University, Chiayi, Taiwan, in 2008, and the M.S. degree in computer and communication engineering from National Cheng Kung University, Tainan, Taiwan, in 2010. He is currently pursuing the Ph.D. degree in electrical engineering with National Taiwan University, Taipei, Taiwan. He joined at the Institute for Information Industry (III), Taipei, in 2010, where he is currently a Cybersecurity Engineer. His current research interests include network security, malware analysis, and intrusion detection.

CHIA-CHING LIN received the B.S. degree in electrical engineering and the M.S. degree in communication engineering from National Taiwan University, Taiwan, in 2007 and 2009, respectively, where he is currently pursuing the Ph.D. degree with the Department of Electrical Engineering. From 2011 to 2014, he was a Software Engineer at Media Tek Inc., Hsinchu, Taiwan. From 2016 to 2019, he was a Research Assistant at the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His current research interests include computer vision, deep learning, and image processing.

ZHENG-GANG HE received the B.S. degree in computer science from National Chengchi University, Taipei, Taiwan, in 2020, and the M.S. degree in electrical engineering from National Taiwan University, Taipei, in 2022. His current research interests include network security and deep learning.

WEI-CHIEH YANG received the B.S. and M.S. degrees in computer science and engineering from Tunghai University, Taiwan, in 2003 and 2005, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan. From 2005 to 2022, he was a Cybersecurity Engineer at the Institute for Information Industry (III), Taipei. His current research interests include network security and user entity behavior analytics (UEBA).

CHIN-LAUNG LEI received the B.S. degree in electrical engineering from National Taiwan University, Taipei, in 1980, and the Ph.D. degree in computer science from The University of Texas at Austin, in 1986. From 1986 to 1988, he was an Assistant Professor at the Computer and Information Science Department, Ohio State University, Columbus, OH, USA. In 1988, he joined as a Faculty Member with the Department of Electrical Engineering, National Taiwan University, where he is currently a Professor. He has published more than 250 technical papers in scientific journals and conference proceedings. His current research interests include network security, cloud computing, the Internet of Things, and big data analytics. He was a co-recipient of the First IEEE LICS Test-of-Time Award.