Received 14 August 2023, accepted 31 August 2023, date of publication 4 September 2023, date of current version 13 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3311889

RESEARCH ARTICLE

User Behavior Detection Using Multi-Modal Signatures of Encrypted Network Traffic

JEE-TAE PARK[1], **CHANG-YUI SHIN**[2], **UI-JUN BAEK**[1], **AND MYUNG-SUP KIM**[1]
[1]Department of Computer and Information Science, Korea University, Seoul 30019, South Korea
[2]C4ISR System Development Quality Team, Defense Agency for Technology and Quality, Daejeon 34327, South Korea

Corresponding author: Myung-Sup Kim (tmskim@korea.ac.kr)

This work was supported in part by the Technology Innovation Program Grant funded by the Ministry of Trade, Industry and Energy (MOTIE), South Korea, and the Korea Evaluation Institute of Industrial Technology (KEIT) (Development of SaaS SW Management Platform Based on 5Channel Discovery Technology for IT Cost Saving) under Grant 20008902; and in part by the Regional Innovation Strategy (RIS) through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (MOE) under Grant 2021RIS-004.

ABSTRACT With the development of the network environment and the emergence of new applications, network traffic has become increasingly complex. This paper focuses on user behavior detection based on encrypted traffic analysis. User behavior information plays a critical role in network management and security, leading to extensive research in this domain. This paper introduces two main contributions. Firstly, we present a categorization method for application types and a behavior definition approach for user behavior detection research. This enables consistent behavior definition for each application type, facilitating objective performance comparison with other studies in the field. Secondly, a behavior detection method based on multi-modal signatures is introduced. The multi-modal signatures represent the multiple signatures extracted from encrypted traffic, including header, SNI, and PSD signatures, which are subsequently defined as a rule. To validate the effectiveness of our proposed method, we conducted 4 experiments on 5 SaaS applications. As a result of the experiments, the proposed method achieves an F-measure of 94∼99% and can detect other types of application behaviors with high performance. As this study conducts user behavior detection research based on encrypted traffic analysis, the proposed method can be applied to other research areas that utilize encrypted traffic.

INDEX TERMS User behavior detection, encrypted traffic classification, multi-modal signature, signature-based traffic analysis.

I. INTRODUCTION

A. RESEARCH BACKGROUND

Various applications are occurring due to the development of network environments and technology. Network traffic classification research is important in network service, cost management, and security. Network traffic classification varies depending on the level at which it is classified, and most studies have been conducted at the application level. The classification of application traffic has been extensively studied, focusing on services, applications, and processes [1], [2], [3]. Traffic classification methods can generally be categorized into traditional, signature-based, and learning-based analysis methods [5].

Traditional traffic classification methods include port-based and payload-based analysis methods [4], [5], [6], [7]. The port-based method was performed for applications using fixed port numbers and is not used now because most applications use dynamic ports. The payload-based method utilizes the payload content of the packet and derives high classification accuracy [30], [31], [32]. However, it cannot be applied because the payload contents are encrypted by the application of encrypted traffic [16]. Recently, packets in the TLS (Transport Layer Security) handshake process that is not encrypted before encrypted communication are being used [15]. During the TLS Handshake process, research on utilizing SNI (Server Name Indication) information is conducted

The associate editor coordinating the review of this manuscript and approving it for publication was Hosam El-Ocla.

since the web address information of the destination server is shown in the SNI field in the Client Hello packet.

The signature-based method analyzes common characteristics of traffic generated from target applications and defines them as signatures [5], [6], [7], [8]. The signature-based analysis methods can be defined in various ways according to traffic characteristics, including header, payload, statistics, and behavior signatures [9], [10]. This method has derived high detection accuracy in the past. However, as traffic patterns become more complex and encrypted, the performance of the method of applying a single signature degraded. In addition, there is a problem in that the process of generating a signature is very complicated and consumes time and effort [10], [11], [12], [13].

The learning-based method uses a machine learning algorithm to extract network traffic characteristics and classify applications [13], [14], [15], [16]. As machine learning algorithms gradually diversify and develop, the performance of learning-based analysis methods has dramatically improved, especially showing high detection performance even for encrypted packets. However, learning-based analysis methods consume a lot of time and effort to find appropriate features and models and require a large amount of labeled data [16], [17], [18].

Recently, several types of research have been conducted that expand application traffic classification, such as fault detection, website fingerprinting, user behaviors detection, and operating systems identification [5]. In this paper, we focus on user behavior detection.

Information on user behavior plays a crucial role in network security and management. In terms of network security, malicious actors can exploit user behavior information to analyze the vulnerability of a target network. They can also misuse authorized user behavior patterns for malicious activities or unauthorized access to sensitive information [33], [34], [35]. Regarding network management, user behavior information aids in efficient network optimization, resource management, and user control [37]. As a result, various studies have been conducted in the field of network user behavior detection [19], [20], [21], [22], [23], [24], [25], [26]. The definition of user behavior varies depending on the application's characteristics and the research objectives. For example, in [20], the authors defined 11 behaviors specific to KakaoTalk, a messenger application type, including actions such as joining or leaving a chat room, blocking a user, and synchronizing friend lists. In [21], the authors focused on Instagram, a social media and SNS (Social Network Service) application type, and defined 9 behaviors, such as entering, logging in, posting, and browsing.

In this paper, we conduct research targeting SaaS (Software as a Service) applications. SaaS is a cloud-based application that provides software in a cloud environment over the Internet. SaaS has many advantages, such as easy accessibility, rapid deployment, and secure security. Enterprises using SaaS services are gradually increasing, and representative examples include Microsoft Office 365 and Google Apps. SaaS has different characteristics from installed applications. It is provided as a service in the form of a subscription, and the cost varies depending on the using service, the number of users, and the user period. The cost can be reduced if the user subscribes to an appropriate license. However, excessive spending may occur if the number of people allowed by the license subscribed is greater than the number of actual users or if more services and functions are included in the license than necessary services. In addition, it is also required to verify that the person using the license subscribed within the company is an appropriate user. Therefore, we define 4 behaviors (i.e., Application Start, Login, Logout, and Application End) for SaaS application usage information.

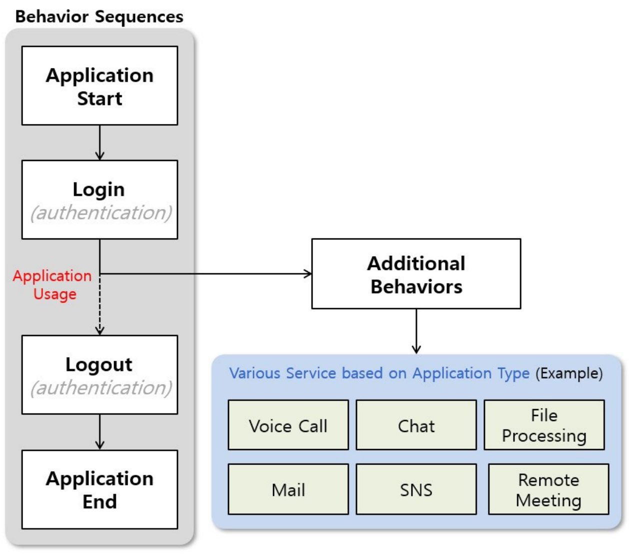


FIGURE 1. An example of behavior sequence in SaaS application.

Since SaaS is provided on the internet and requires a user authentication process to use the application. The 4 behaviors have a series of behavior sequences considering these processes, which we define as a behavior sequence. The behavior sequence is shown in Fig. 1. For example, a login is performed after the application start, and a logout is performed after the application start and login.

These 4 behaviors, representing the behavior sequence, can be applied to various SaaS applications, while additional behaviors can be defined specifically for each application. For example, in Office 365, additional behaviors such as file uploads or saving can be defined. Additional behaviors can be categorized into different types of services and application types based on the intended usage of the application. However, in this paper, we did not explore additional behaviors extensively, as they are less relevant to the primary focus on usage information.

B. PROBLEM STATEMENTS

In this section, we remark on the problems of user behavior detection research, including our previous study.

First, it is hard to make objective performance comparisons in user behavior detection research because each research defines the behaviors differently. In addition, since there is no public dataset in this research field, most of the existing research uses the private dataset of their target application.

Second, most studies use a learning-based traffic analysis method such as machine and deep learning [14], [15]. Learning-based methods are widely used in various research fields and show high performance [30], [31], [32]. However, a lot of time and resources are consumed depending on the model structure and amount of data. In addition, large amounts of labeled data are required for high-performance results. However, in real networks, most traffic datasets consist of unlabeled or semi-labeled data, and labeled data is limited.

Third, signature-based methods have been widely used in traffic analysis [4], [5], [6], [7], [8], [9]. However, existing signature-based methods use only one signature based on analyzed traffic characteristics. A single signature that considers a single characteristic of traffic degrades detection performance for encrypted traffic.

Forth, we proposed a rule-based behavior detection method in previous research [28]. However, there are several problems in the previous study. In [28], there is a high dependency on SNI because the rules are generated using only header and SNI information. Therefore, it is difficult to apply when there is no common SNI information on the behavior of the target application. In addition, detection performance is significantly degraded when the traffic pattern of a target application changes due to an application update.

C. CONTRIBUTION

The main contribution of this paper can be summarized as follows:

- Although there are many researches on user behavior detection, existing researches have respectively defined user behavior for their purposes. However, it is necessary to define user behavior consistently for objective verification and comparison. In this paper, we categorize the application types and present a guideline for behavior definition according to the application type.
- In this paper, we performed traffic analysis for SaaS applications, and we present a behavior sequence considering the subscription-type SaaS application characteristics. We verified that behavior sequences affect the detection performance. As a result of the experiments, the false detections are reduced compared to cases where behavior sequence is not considered.
- We propose a rule-based behavior detection system based on network traffic analysis. Since the SaaS uses encrypted traffic, the proposed method is performed for encrypted traffic. The proposed method is a signature-based analysis method in a large category. Our proposed method improves detection performance by using multi-modal signatures, defined as a rule. To verify the proposed method, we conducted several experiments. We demonstrated the performance of the proposed method for 5 SaaS applications (i.e., Office 365, Adobe Creative Cloud, Autodesk, Zoom, and Slack). The proposed method shows high detection performance even when using a small amount of data and takes less time in each process.
- The proposed method utilizes various signature extraction algorithms to solve the problems of high SNI dependence and manual rule generation in our previous research. We also propose a rule update mechanism, which is the process of regenerating the rules of the target application by determining when an update is required. Rule update mechanism can solve the performance degradation problem for changing traffic patterns.

The remainder of this paper is organized as follows. Section II describes the related works, and Section III we present the categorization of applications and behavior definition methods suitable for each application type. Section IV explains the rule-based user behavior detection system and mechanism in detail. In Section V, we present the experiments to evaluate the proposed method. Finally, we discuss the concluding remarks and future work in Section VI.

II. RELATED WORK

This section provides related works of encrypted traffic classification and user behavior detection.

A. ENCRYPTED TRAFFIC CLASSIFICATION

There are various types of applications, such as mobile, cloud, and IoT, and most of them use encrypted traffic [36]. Applying the encrypted traffic makes traditional traffic analysis methods inapplicable. Many researchers have utilized learning-based methods such as machine learning and deep learning to solve this problem [30], [31], [32].

In [16], Wang et al. proposed an encryption traffic classification method based on convolutional neural networks. The authors validated with the public ISCX VPN-nonVPN traffic dataset and achieved high performance. In [17], Liu et al. proposed FS-NET (Flow Sequence Network) by applying the RNN and multi-layer encoder-decoder structure. To verify their proposed method, they conducted experiments with their private dataset and achieved a TPR of 99.14% and an FPR of 0.05%. In [18], Lotfollahi et al. proposed Deep Packet, which employed two deep neural network structures, namely stacked autoencoder and convolution neural network. They validated with the public ISCX VPN-nonVPN traffic dataset and achieved an F1 score of 98% in application identification.

The learning-based method significantly improved detection performance, even for encrypted traffic. However, labeled data for network traffic is difficult to obtain, and appropriate features, parameters, model selection, and preprocessing are required for high performance.

In this paper, we employ a multi-modal signature-based traffic analysis method that defines various and diverse common characteristics of the traffic as signatures. The learning-based traffic analysis method and the

TABLE 1. Examples of related work in user behavior detection.

Related Work	Target Application	Type	Defined Behavior
Hou. *et al.* [20]	WeChat	Mobile Messaging	7
Park. and Kim. [21]	KakaoTalk	Mobile Messaging	11
Hua Wu, *et al* [22]	Instagram	SNS	9
Coull. and Dyer. [25]	Apple messaging	Mobile Messaging	5
Conti *et al.* [26]	7 applications	SNS, Mail. etc.	Various

multi-modal signature-based traffic analysis method differ in their approach to behavior detection.

Firstly, the learning-based method involves training models using labeled data. It utilizes pre-labeled traffic data (e.g., normal, and malicious behaviors) to learn behavior patterns through machine learning algorithms. This approach requires a considerable amount of labeled data and can be resource-intensive and time-consuming for training. However, it can be more effective in detecting new patterns or malicious behaviors. Secondly, the multi-modal signature-based method detects behaviors by using predefined signatures for the target traffic. Instead of relying on labeled data, this method defines signatures based on prior knowledge and domain expertise. It does not require extensive training data, making it suitable for scenarios with limited labeled data or privacy concerns. Thirdly, the learning-based method allows for more flexibility and adaptability. It can learn from labeled data and adapt to new patterns, including unknown or emerging behaviors. In contrast, the multi-modal signature-based approach is more deterministic and interpretable. Each signature represents a specific behavior aspect, providing a transparent view of how behaviors are detected.

Overall, the choice between the learning-based method and the multi-modal signature-based method depends on factors such as the availability of labeled data, the desired level of interpretability and adaptability, and the specific requirements of the behavior detection task.

B. USER BEHAVIOR DETECTION

Research on user behavior detection has been conducted for a long time, mainly for network security [20], [29]. In most studies, one application is identified, and the detailed behavior of the application is defined [33], [34]. Each study differs in the application used and the method of defining behavior, as is shown in Table 1.

In [19], Hou et al. defined 7 behaviors for WeChat. The authors conducted experiments to classify defined behaviors using various algorithms, and the Random Forest algorithm showed the best performance. Park and Kim focus on KakaoTalk, a widely used mobile messenger [20]. The authors mentioned network security vulnerabilities for user behavior, although KakaoTalk uses a lightweight proprietary encryption protocol called LOCO. The authors defined 11 user behaviors and detected them with about 99.7% accuracy using Random Forest. Wu et al. used Instagram for user behavior detection and defined 9 behaviors [21]. They used SVM to classify defined behaviors and achieved 99.8% accuracy.

Coull and Dyer [24] propose a method for encrypted traffic analysis targeting Apple's instant messaging service. They used the sizes of the packets and defined 5 user actions such as "start typing", "stop typing", "send text", "send attachment", and "read receipt". The authors also conducted to infer the language (e.g., English, Chinese, French, etc.) and length of the messages.

Some studies conduct behavioral detection studies using multiple applications. Grolman et al. applied transfer learning to identify user behaviors [23]. They conducted experiments for Twitter and Facebook and achieved 0.8 f1-measures. In [25], Conti et al. used 7 applications (i.e., Gmail, Facebook, Twitter, Tumblr, Dropbox, Google+, and Evernote). The authors focus on Android encrypted traffic and define several behaviors of each application. They conducted experiments by collecting traffic to verify their proposed method. In [26], Fu et al. proposed CUMMA, a method for classifying service usage of mobile messaging applications by modeling user behavioral patterns and network traffic characteristics. In [26], the authors specifically target two messaging applications, WeChat, and WhatsApp, and define a set of 8 and 6 user behaviors for each application, respectively. This research is similar to [34] in terms of its approach and methodology. Both studies focus on analyzing user behaviors and network traffic in mobile messaging applications. In [27], Jiang et al. focus on remote desktops and suggest that user behavior information can be exposed even if remote desktop traffic is encrypted.

In [22], Saltaformaggio et al. proposed NetScope using multi-class SVM to classify actions, achieving accuracy and recall of about 78% and 76%. In addition, the authors categorized mobile applications type and defined behavior for each application. Our work is similar to their work in categorizing the applications. However, they focus on only mobile applications and define 1∼3 behaviors for each application. We perform categorization for entire applications and present common and specific behaviors according to the functions and additional given of each application type.

In our previous method [28], we proposed a rule-based user behavior detection and defined 4 user behaviors for Microsoft Office 365. Our proposed method presented high performance. However, generating a rule takes a lot of time and highly depends on SNI information.

III. APPLICATION CATEGORIZATION & USER BEHAVIOR DEFINITION

In this section, we explain a guideline of user behavior definition based on application categorization.

Many researchers use different applications and differently define the application's behaviors. As a result, it becomes challenging to compare the performance of the proposed methods across different studies.

Therefore, our focus has been on simplifying and standardizing the application types and behavior definition methods used for performance comparison among various studies. All applications can be categorized based on their functionalities and services into different types. For example, KakaoTalk and WeChat can be classified as messenger types, and Facebook and Instagram as SNS types. In addition, the behaviors that can be defined for each application type are similar. For example, in the study [25], the authors defined the behaviors such as a post on the wall, open Facebook, open a user profile, and post user status was defined for Facebook, and in [21], the authors defined the behaviors such as posting, repost, browse and favorite are defined. Both researchers defined behavior similarly for SNS-type applications. Therefore, we regard these similar behaviors performed in the same type of application as the same behavior. For example, "post" and "post on the wall" of SNS-type applications are defined as the same behavior corresponding to "posting".

However, application categorization and behavior definition have some considerations. First, it is impossible to categorize all applications because the types of commonly used applications are diverse. In this paper, applications used in previous user behavior detection research are classified into 7 categories (i.e., messenger, document work, design, SNS, remote meeting, mail, and SaaS). Application types can be additionally defined if the target application does not belong to the defined 7 types. For example, if the target application belongs to shopping, 'shopping' can be defined as additional application types. In that case, a new application type can be added, and behavior suitable for the type can be defined.

Second, many applications are multifunctional, encompassing various features. As a result, they can be categorized under more than one type in our research. For instance, Instagram is primarily considered an SNS application, but it also incorporates a messenger function known as Direct Message (DM). Consequently, Instagram falls into two categories: SNS and Messenger types.

Third, even if it is the same type of application, detailed functions or services may differ depending on the application. For example, in this paper, we classify the behaviors that can be commonly defined for each application type and those that can be individually defined. For instance, in a messenger-type application, sending and receiving messages is defined as a common behavior, and an open mini program in WeChat is defined as a specific behavior. In this paper, we classify the behavior into common behavior defined by application type and specific behavior defined individually by application. For example, in a messenger-type application, sending and receiving messages is defined as a common behavior, and an open mini program in WeChat is defined as a specific behavior. We categorized the 7 applications and defined the common and specific behaviors for each application type as shown in Table 2.

In Table 2, the "*application type*" and "*description of application type*" indicate the type of categorized application and the description of each type. The "*application name*" indicates an application example corresponding to the application type, and the "*behavior definition*" indicates examples of common and specific behavior definitions according to categorized application types.

IV. OVERALL DESIGN OF THE PROPOSED SYSTEM

In this section, we describe the overall design of a rule-based user behavior detection system and its detailed mechanism. The entire system structure consists of 3 sub-systems (i.e., rule generation, rule-based user behavior detection, and rule update & re-generation) and is shown in Fig. 2.

A. RULE GENERATION

Before rule generation, the detection behavior for the target application must be defined. As mentioned before, we define 4 behaviors as the target behavior for SaaS applications. The detection rule indicates traffic characteristics for behavior and consists of the application and behavior rules. The overall structure of the detection rule is shown in Fig. 3 according to the behavior sequence. The application rule represents a detection rule for the target application, and the behavior rule represents a detection rule for the target application's behavior. Signatures are classified into Header, SNI, and PSD (Packet Size Distribution) and indicate each type of traffic characteristic. Behavior state information indicates current and preceded behavior information and is determined according to behavior sequence. In other words, an application rule is composed of behavior rules for each behavior and a behavior rule is composed of 3 signatures and status information of the target behavior.

Rule generation is a process of analyzing the traffic of the target application, extracting a signature, and creating a rule by aggregating the signatures. Rule generation is composed of 5 detailed processes (i.e., traffic preprocessing, signature extraction, behavior rule generation, behavior rule verification, and application rule generation).

1) TRAFFIC PREPROCESSING

Traffic pre-processing is performed both for rule creation and behavior detection. Traffic preprocessing in rule generation consists of 3 detailed processes (i.e., traffic collection, flow generation, and GT (Ground Truth) traffic generation).

Fig. 4 shows traffic preprocessing in rule generation. Traffic collection is the process of collecting the traffic of a target application to define the behavior signature. In this paper, we use Wireshark to collect pcap files. Behavior information during the traffic collection is recorded and stored as a log file. The stored log file includes information on the host IP, target application, behavior, and behavior execution time. The log file is used in GT traffic generation of pre-processing

TABLE 2. Categorization of application type and behavior definition based on application type.

Application Type	Description of Application Type	Application Name *(Example)*	Behavior Definition *(Example)*	
Messenger	A software that can **send and receive messages or data in real-time** over the internet	KakaoTalk, WeChat, Slack, Instagram, Facebook, Twitter	**common**	Enter the Chat Room Send a Message / Transfer the File Block / Hide a Friend /
			specific	Open the Mini Game / Transfer the Money / Watch the Advertisement / Voice call
Document Processing	Software that **handles various types of document files**	Office 365, Google Docs, Hancom Office	**common**	Open / Edit / Exit / Save the Document File,
			specific	Use a Detailed Application (e.g., PowerPoint)
Design	Software that provides **specialized functions** for editing and modifying **media such as images or videos**.	Adobe Creative Cloud, AutoCAD,	**common**	Open / Edit / Exit the Media File
			specific	Use advanced tools / Creating Vector Graphics or Illustrations
SNS	Software that provides an **online platform for free communication and information sharing among users**	Instagram, Facebook, Twitter, Tumblr	**common**	Posting / Delete a Post / View the Profile Sending a Message / Reacting to the Post / Sharing the content / Following
			specific	Tagging other users in posts / Creating events / Managing the privacy settings
Remote Meeting	A software that can **conduct remote meetings through voice and video**	Zoom, Webex, Discord	**common**	Join / Terminate the Remote Meeting / Sharing the Screen / Scheduling the Meetings
			specific	Sending a Message / Chatting / Recording the Meeting
Mail	A method of communication in which electronic devices are used to **transmit messages across a computer network**	Gmail	**common**	Send / Receive a Mail
			specific	Creating filters for email sorting / Setting auto-reply to messages
SaaS	**A cloud-based software model that delivers applications to end users** through an internet browser.	Office 365, Zoom, Adobe Creative Cloud, AutoCAD	**common**	Application Start / End, Login, Logout
			specific	

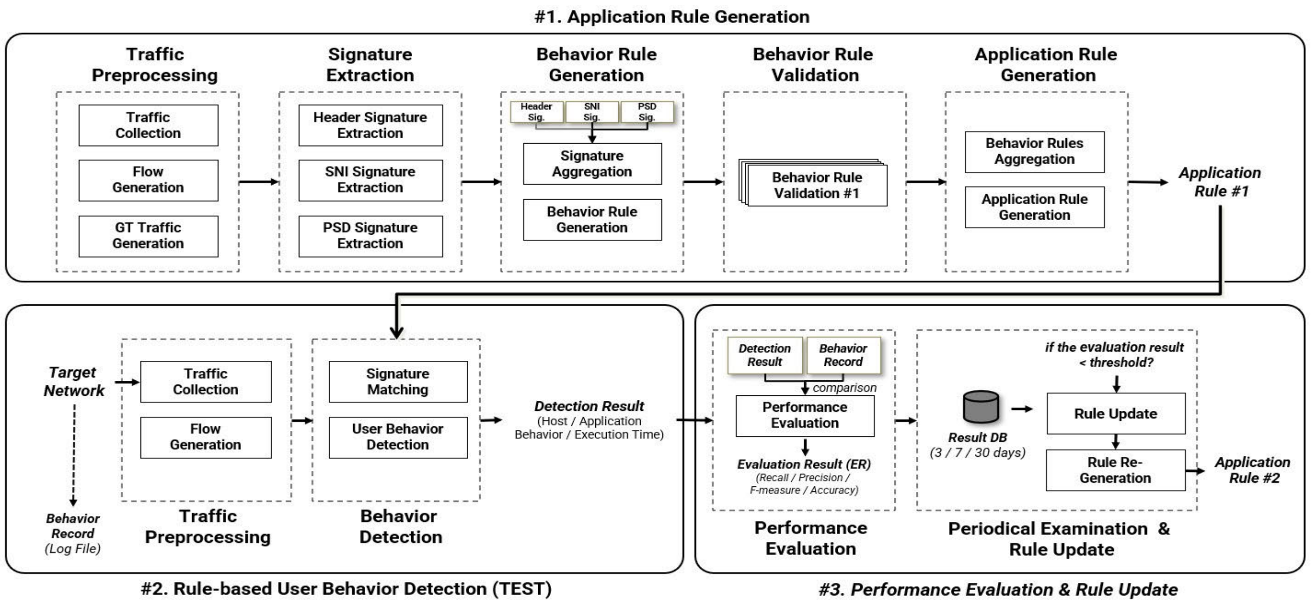


FIGURE 2. Entire structure of proposed method.

and behavior rule verification. Flow generation converts the collected pcap traffic file in packet units into a flow unit traffic file. A flow is a set of packets with the same 5-tuples information (Source IP, Source Port, Protocol, Destination IP, Destination Port). This process applies to TLS packets, and other protocol packets are excluded.

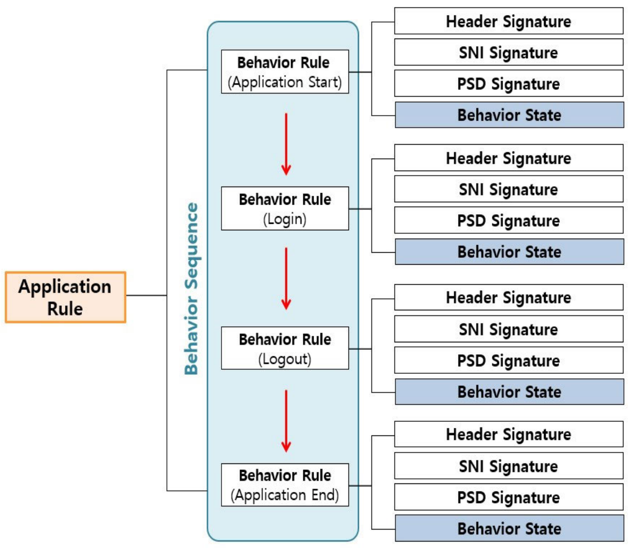


FIGURE 3. The overall structure of the detection rule.

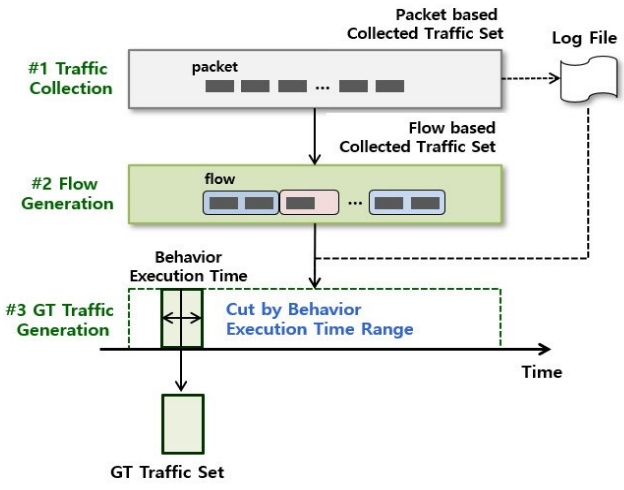


FIGURE 4. A process of traffic preprocessing in rule generation.

GT traffic is generated for the behavior by inputting the converted flow-unit traffic file. GT traffic is required for accurate signature extraction, as common features of traffic for behavior are extracted during the behavior signature generation. Therefore, in the GT traffic generation, the behavior execution time is additionally loaded from the log file, and traffic corresponding to +3 seconds is cut based on the behavior execution time. However, if communication within a behavior takes longer than 3 seconds, the time interval can be set differently for each application. For example, Adobe Creative Cloud takes an average of 10 seconds during the login, and the time interval is set to +10 seconds.

2) SIGNATURE EXTRACTION

Signature extraction takes the collected traffic as input and extracts traffic characteristics from various perspectives. A signature represents a common traffic characteristic and comprises a header, SNI, and PSD information.

Header signature indicates the header information of the flow that occurred when performing a behavior and representatively includes destination IP, port, and protocol. The header signature is defined when a fixed destination IP, port, or protocol is used in the behavior of the target application. SNI signature utilizes the SNI information in the Client Hello packet Extension field during the TLS handshake process. It indicates the URL information about the communicating server. The Client Hello packet is not encrypted because they are sent in the TLS Handshake before performing encrypted communication. For example, when a specific behavior, such as login or logout, is performed, specific SNI information is extracted when visiting or bypassing a specific site. However, since the traffic collected in the real network environment can run many applications on one host, many flows are included, and various SNIs are derived. Therefore, in rule generation, the commonly occurring SNI that occurs in the behavior of the target application is defined as an SNI signature with GT traffic that has undergone preprocessing as an input.

PSD signature indicates a set of packet sizes represented by an integer vector. Packet size information has been used for a long time in many researches. The basic concept of PSD signatures is that there are statistically similar patterns in the traffic when the user behavior is performed. PSD is a set of 1∼ *i*th packet sizes considering packet direction and presented as an integer vector. *i* represent the number of packets to use in PSD, and we set *i* to 5.

In our previous research [28], signatures were extracted manually, consuming much time and effort. To solve this problem, we propose an advanced signature extraction method. The advanced signature extraction method is applied differently depending on the data type. The data type consists of a string and an integer vector according to the kind of signature. String type data includes header and SNI signature, and integer vector type data includes PSD signature.

String-type signatures generally have fixed values. For example, when application *X* acts, the destination IP of *10.x.x.x* and fixed SNI information of *"www.xxx.com"* appear, defined as header and SNI signature, respectively.

$$TraceSet = \{T_1, T_2, T_3 \ldots, T_p\},\quad \text{p = the number of GT traffic traces} \tag{1}$$

$$F_j = \{f_1, f_2, f_3, \ldots, f_n\},\quad \text{j = the target trace number} \tag{2}$$

In the rule generation, GT traffic sets of multiple traces are collected, and the collected traffic traces are composed of several packets and flows. Equation (1) represents the set of collected GT traffic traces, and Equation (2) represents the set of flows in the *j*th trace. Header, SNI, and PSD flow features are derived from a single flow.

$$Flow\_FeatureSet = \begin{cases} Hdr\_Set \\ SNI\_Set \\ PSD\_Set \end{cases} \tag{3}$$

Equation (3) represents the feature set for entire flows in the ith trace, and the feature set consists of Hdr_Set, SNI_Set and PSD_Set.

$$Hdr\_Set = \{H_1, H_2, H_3, \ldots, H_n\}, \quad \text{n = the number of flows} \tag{4}$$

$$SNI\_Set = \{s_1, s_2, s_{3,}, \ldots, s_n\} \tag{5}$$

$$PSD\_Set = \{P_1, P_2, P_{3,}, \ldots, P_n\} \tag{6}$$

In Equation (4), (5) and (6), Hdr_Set, SNI_Set, and PSD_Set represents a set of headers, SNIs and PSDs derived from each flow. Equation (7) indicates the header information of the ith flow (f_i). Equation (8) indicates the PSD which is a sequence of 1~5th packet sizes in ith flow (f_i).

$$H_i = [dst\_ip, dst\_port, prot] \tag{7}$$

$$P_i = [p_1, p_2, p_{3,}p_4, p_5] \tag{8}$$

Header signature is defined when using a fixed server IP, port, and protocol. To set the header signature, destination IP, port, and protocol are derived from all flows in the entire traffic traces. The derived header information is compared for each trace, and common information is defined as a header signature.

$$TS_i = s_1 + i + s_2 + i + s_3 \ldots + s_n \tag{9}$$

$$\text{i = target trace number}$$

$$LS_m = LCS(TS_i, TS_{i+1})$$

$$\text{m = number of LCS algorithms applied} \tag{10}$$

SNI signature is defined when using fixed strings. In this paper, we use the LCS algorithm to extract SNI information automatically. LCS algorithm takes two strings as input and extracts the longest common substring from the two strings. Algorithm 1 indicates the pseudo-algorithm of LCS. Since there are several flows in one GT trace in the collected multiple trace datasets, the LCS (Longest Common Subsequence) algorithm is applied repeatedly multiple times to extract the SNI signature.

Algorithm 2 and Fig. 5 show a process of SNI signature extraction. First, obtain the GT traffic datasets for the specific behavior targeted by the application. Next, extract the SNI for all flows observed in each trace. Subsequently, concatenate the extracted SNIs along with the trace number to form a single string.

This string is referred to as the TS (Trace String) and is represented by Equation (9). For instance, if a particular traffic trace consists of three flows with SNIs "ABC", "def", and "xyz", respectively, and the trace number is "3", the TS would be "abc3def3xyz".

Since the LCS algorithm takes two strings as input, it divides the entire trace set into two traces as T_i, T_{i+1}. S algorithm is repeatedly applied with a pair of TS_i, TS_{i+1}. derived from a pair of trac es as input. Equation (10) shows the string derived when the LCS algorithm is recursively applied m times and is performed until one string is finally obtained.

Algorithm 1 LCS Algorithm

```
Input : string s[1:m], t[1:n]
Output: substring lcs (the longest common substrings)
01: Allocate space L[0:m, 0:n] and max_length = 0
02: L[0, j] = 0 for all
03:   for i = 1 to m do
04:     for j = 1 to n do
05:       if s[i] == t[j] then
06:         if I == 1 or j == 1 then
07:           LCS_Mat[i,j] = 1
08:         else
09:           LCS_Mat[i,j] = LCS_Mat[i-1, j-1] + 1
10:         if LCS_Mat[i,j] > max_length then
11:           max_length = LCS_Mat[i, j]
12:           lcs = s[i-max_length:i-max_length+i]
13:         else if LCS_Mat[I,j] == max_length then
14:           lcs = lcs.append(s[i-max_length:i-max_length+i])
15:       else
16:           LCS_Mat[i, j] = 0
17: return lcs
```

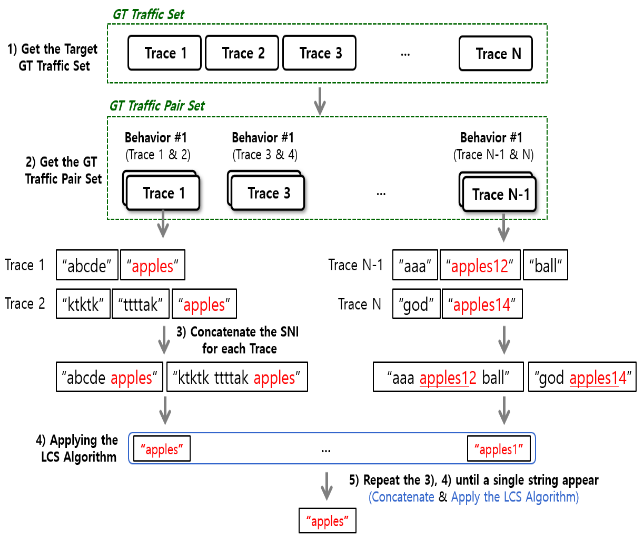

GT Traffic Set
1) Get the Target GT Traffic Set
Trace 1
Trace 2
Trace 3
...
Trace N
GT Traffic Pair Set
2) Get the GT Traffic Pair Set
Behavior #1 (Trace 1 & 2)
Behavior #1 (Trace 3 & 4)
Behavior #1 (Trace N-1 & N)
Trace 1
Trace 3
...
Trace N-1
Trace 1 "abcde" "apples"
Trace 2 "ktktk" "ttttak" "apples"
Trace N-1 "aaa" "apples12" "ball"
Trace N "god" "apples14"
3) Concatenate the SNI for each Trace
"abcde apples"
"ktktk ttttak apples"
"aaa apples12 ball"
"god apples14"
4) Applying the LCS Algorithm
"apples"
...
"apples1"
5) Repeat the 3), 4) until a single string appear
(Concatenate & Apply the LCS Algorithm)
"apples"

FIGURE 5. A process of SNI signature extraction.

In our previous research [28], we relied solely on header and SNI signatures for user behavior detection. However, we encountered a specific scenario where the same signatures were associated with different behaviors, leading to an increased probability of false detection. To address this issue, we present PSD signatures in this paper. Unlike string-based signatures (e.g., header, SNI) that have fixed values, the PSD signature is an integer vector type with variable values. For instance, if the same behavior occurs in different traces, the packet sizes may vary. Therefore, unlike the method used for defining string-based signatures, the PSD signature is defined by setting representative values and thresholds separately.

Fig. 6 illustrates an entire process of extracting header, SNI and PSD signature generation. Initially, the header and SNI signature are sequentially extracted with the GT traffic set as input. Subsequently, the extraction of the PSD signature is

Algorithm 2 SNI Signature Extraction

```
Input: GT Traffic Trace 1~N
Output: OutputSNIList (SNI Signature)
[Notation] N : Entire trace count / M_i : Entire flow count of trace i

// Concatenate the SNI for the target GT traffic traces
01: for i = 1 to N
02:     SNIList = []
03:     for j =1 to M_i
04:         SNIList + = SNI
05:         SNIList + = i
06:     TargetSNIList.append(SNIList)
07: CurrentTraceCount = N
08: TargetListCurrent = []
// SNI signature extraction by using recursive LCS algorithm
09: while CurrentTraceCount != 1: do
10:     for i = 1 to CurrentTraceCount
        // the case of N is odd
11:     if CurrentTraceCount % 2 != 0 then
12:         if i == CurrentTraceCount then
13:             TargetListCurrent. append (TargetSNIList[i])
14:             break
15:     elseifi != CurrentTraceCount then
16:             if i % 2 != 0 then
17:                 A = TargetSNIList[i]
18:                 B = TargetSNIList[i+1]
19:             TargetListCurrent. append (LCS (A, B))
20:             i + = 1
        // the case of N is even
21:     else if CurrentTraceCount % 2 == 0 then
22:         if i % 2 != 0 then
23:             A = TargetSNIList[i]
24:             B = TargetSNIList[i+1]
25:             TargetListCurrent. append (LCS (A, B))
26:             i + = 1
27:   CurrentTraceCount = len(TargetListCurren )
28    TargetSNIList = TargetListCurrent
29:   TargetListCurrent = []
30: OutputSNIList = TargetSNIList
31: return OutputSNIList
```

performed based on two scenarios: the presence or absence of an SNI signature.

Through traffic analysis, we observed that flows with the same SNI signature across different traces exhibited similar PSD patterns. For example, when there is a flow with a unique SNI signature of ''xxx,'' the analysis of its PSD reveals minimal packet size variations among the packets within that flow. As a result, we divide the PSD signature generation into two cases: *i*) when the SNI signature is extracted, and *ii)* when the SNI signature is not extracted.

i) the SNI signature is extracted: If a signature has been extracted, a PSD for a flow, including the corresponding

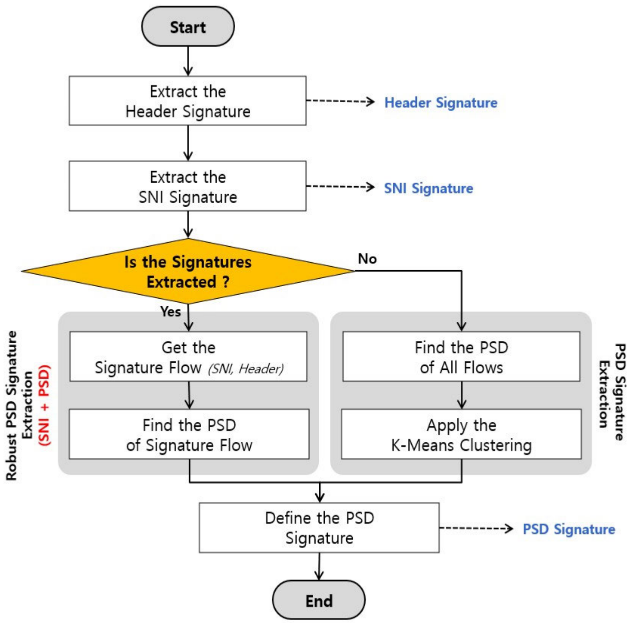

Start
Extract the Header Signature
Header Signature
Extract the SNI Signature
SNI Signature
Is the Signatures Extracted ?
No
Yes
Robust PSD Signature Extraction (SNI + PSD)
Get the Signature Flow (SNI, Header)
Find the PSD of Signature Flow
Find the PSD of All Flows
Apply the K-Means Clustering
PSD Signature Extraction
Define the PSD Signature
PSD Signature
End

FIGURE 6. An entire process of extracting header, SNI, and PSD signature.

signature, is obtained. The PSD for each trace is derived by performing the same operation on various collected traces. Subsequently, the average PSD is computed based on several derived PSDs, as shown in Equations (11) and (12). The difference is then calculated between the average PSD and each individual PSD, as depicted in Equation (13). The threshold is determined as the largest value among these differences. Finally, the average PSD and threshold are defined as the PSD signature.

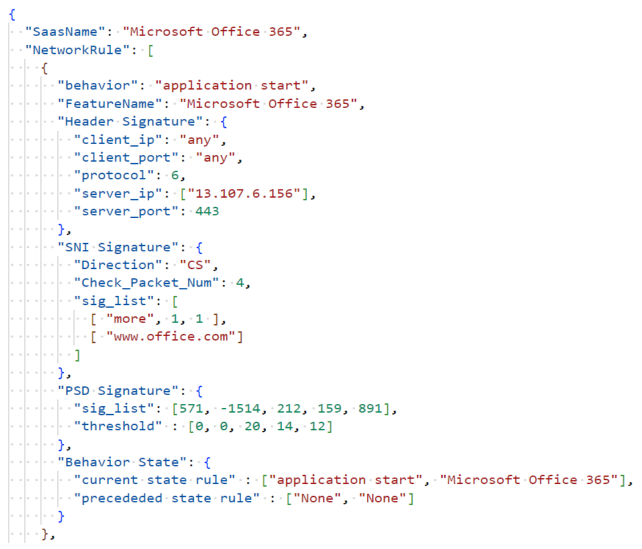
$$AvgPktSet = \{AvgPktSize_1, AvgPktSize_2, \ldots AvgPktSize_5\} \tag{11}$$

$$AvgPktSize_m = \frac{(\sum_{i=1}^{n} p_{i1})}{n}$$
$$\text{n = the number of traffic traces,}$$
$$\text{m = target packet number} \tag{12}$$

$$Thr_m = \max\{|AvgPktSize_m - p_m|\} \tag{13}$$

However, if the difference exceeds the predefined maximum threshold, it is determined that the PSD signature cannot be applied, and only the extracted signature is utilized. For example, if there are 5 traces (n=5) with the same size for the 1st packet in the PSD, the threshold is set to 0. On the other hand, if the sizes of the 2nd packet in the PSD across the 5 traces are 108, 128, 98, 132, and 108, the average packet size for the 2nd packet is 113, and the threshold is set to 19 (132-119), which represents the largest difference from the average value. If the predefined maximum threshold is set to 15, the difference exceeds the predefined maximum threshold. In such cases, statistical significance is not applied, and only header or SNI signatures are utilized.

ii) the SNI signature is not extracted: If there is no extracted signature, a clustering algorithm is used to obtain



```
{
  "SaasName": "Microsoft Office 365",
  "NetworkRule": [
    {
      "behavior": "application start",
      "FeatureName": "Microsoft Office 365",
      "Header Signature": {
        "client_ip": "any",
        "client_port": "any",
        "protocol": 6,
        "server_ip": ["13.107.6.156"],
        "server_port": 443
      },
      "SNI Signature": {
        "Direction": "CS",
        "Check_Packet_Num": 4,
        "sig_list": [
          [ "more", 1, 1 ],
          [ "www.office.com"]
        ]
      },
      "PSD Signature": {
        "sig_list": [571, -1514, 212, 159, 891],
        "threshold" : [0, 0, 20, 14, 12]
      },
      "Behavior State": {
        "current state rule" : ["application start", "Microsoft Office 365"],
        "precededed state rule" : ["None", "None"]
      }
    },
```

FIGURE 7. An example of behavior rule of the application start of Microsoft Office 365.

TABLE 3. An example of signature format of behavior rule.

Behavior Rule			
Target Application		Microsoft Office 365	
Target Behavior		Application Start	
Signatures	Header Sig.	Server IP	"13.107.6.156"
		Server Port	443
		Protocol	TCP
	SNI Sig.	"www.office.com"	
	Statistical Sig.	[571/0, -1514/0, 212/20, 159/14, 891/12]	
	Behavior State	Current	Application Start
		Preceded	None

the PSD signature. First, as shown in Equation (6), the PSD of all flows in the collected GT traffic are derived. After that, we apply the k-means clustering algorithm to the derived PSD. Each cluster is identified while changing k, and the minimum k containing all GT traffic traces in the cluster is obtained. When k is derived, the threshold is defined as the maximum distance between the centroid vector and the cluster. Finally, each cluster's centroid vector and threshold are defined as the PSD signature.

3) BEHAVIOR RULE GENERATION

Behavior rule generation is a process of creating a detection rule for one behavior in a target application, and it consists of signature aggregation and behavior rule generation. Signature aggregation is a process of collecting the header, SNI, and PSD signatures derived from the previous process. Behavior rule generation combines the collected signatures and behavior state information of the target behavior and derives them as rules. An example of a defined behavior rule is shown in Table. 3 and Fig. 7.

Table 3 shows rules for applications start with Microsoft Office 365. It exhibits a fixed server IP of "13.107.6.156" and an SNI of "www.office.com". The PSD signature is defined as [571/0, −1514/0, 212/20, 159/14, 891/12], while [571, −1514, 212, 159, 891] represents the PSD values, and [/0, /10, /5, /16, /12] indicates the threshold for each packet. The state information represents the currently detected behavior and the preceding behavior. Taking into account the sequence of behaviors, the current and preceding behaviors mentioned in the state information cannot be detected in the subsequent behavior detection. For instance, a login action is not detected unless the application start is detected before considering the behavior sequence. In Table 3, the current behavior is application start, and there is no preceding behavior.

Fig. 7 shows the behavior rules for application start in Microsoft Office 365 by utilizing Table 3. The contents of each signature are stored according to the JSON format.

4) BEHAVIOR RULE VALIDATION

In behavior rule verification, the validation process involves checking the accurate detection of behaviors by inputting the behavior rule and the GT traffic, which was not used in previous signature extraction. The validation is conducted on two types of traffic: *i*) traffic with the same behavior as the target behavior of the application, and *ii)* traffic with different behaviors of the same application. If the generated behavior rules result in false detections, an analysis is conducted to determine the cause of the false detections.

In the first case, if patterns are observed that differ from the defined signatures, new signatures are added. For example, if signature #1 is derived from 1 to 8 traces, but in 2 traces, signature #1 is not derived while signature #2 is derived instead, signature #2 is added. In this case, both signatures #1 and #2 are used to detect the behavior. On the other hand, in the second case, if false detections occur, the corresponding signature is considered as not being a unique signature for detecting the behavior and is discarded.

5) APPLICATION RULE GENERATION

In the process of application rule generation, the rules defined for each behavior, which have undergone the verification process, are consolidated into a single application rule. The generated application rule is then used to assess the detection performance by detecting user behaviors in the actual network environment.

B. RULE-BASED USER BEHAVIOR DETECTION

Rule-based behavior detection is a process of detecting user behavior in a real network environment using generated application rules. The real network environment represents a network with multiple hosts, such as businesses or university network environments. User behavior detection is composed of 4 detailed processes (i.e., traffic preprocessing, behavior detection, performance validation, and rule update)

1) TRAFFIC PREPROCESSING

Traffic preprocessing in behavior detection involves two detailed processes: traffic collection and flow generation. Traffic collection is conducted within the target network in a multi-host environment. In our proposed system, we collect the traffic by employing packet mirroring on the campus network of Korea University. The collected traffic encompasses packets from various applications generated by multiple users. The flow generation process follows the same preprocessing method as in rule generation. In our proposed system, user behaviors are recorded to evaluate the detection results. These behavior records are utilized in the performance evaluation process.

2) BEHAVIOR DETECTION

Behavior detection aims to verify the accurate detection of the target application's behavior using preprocessed flows and the generated application rule as input. Our goal is to precisely detect user(host), application, behavior, and behavior execution time utilizing the generated application rules.

The method of applying header, SNI, and PSD signatures in the application rule for behavior detection is as follows. Firstly, the header signature in the rule checks if the collected traffic contains the defined IP, Port, and Protocol. If a matching flow is found, the behavior defined in the rule is detected. Secondly, the SNI signature in the rule checks if the defined SNI is present in the collected traffic. When a matching flow is identified, the corresponding behavior is detected. Thirdly, the application of PSD signatures within the rules differs based on two scenarios. If there is a header or SNI signature, a PSD is derived for a flow that matches the signature. The derived PSD is compared with the defined signature, including the PSD and thresholds of each packet. If the difference between the signature and the derived PSD is smaller than the threshold, the behavior is detected. On the other hand, if there is no header or SNI signature, PSDs are derived for the entire flow. The derived PSD is compared with the defined signature, including centroid vectors and thresholds. If the distance between the derived PSD and centroid vectors is smaller than the threshold, the behavior is detected. As a result of behavior detection, user, application, behavior, and behavior execution time information is derived.

C. PERFORMANCE EVALUATION & RULE UPDATE

In this process, performance evaluation and rule update for detection results are performed. Performance evaluation is performed by comparing the behavior record with the detection result, and the evaluation result is stored in the result DB. A rule update is performed in the case of performance degradation for a certain period in the stored results.

1) PERFORMANCE EVALUATION

In performance evaluation, the evaluation result is obtained by comparing the detection result with behavior records. There are two methods of behavior recording: manual behavior recording and agent-based behavior recording. In the manual recording method, the user directly records the host, application, behavior, and behavior execution time. In the agent-based recording method, the host, application, behavior, and behavior execution time are recorded using process APIs. This method requires an agent to be installed on the host of the detection target. The agent operates based on the API calls triggered when an application behavior is performed on the host.

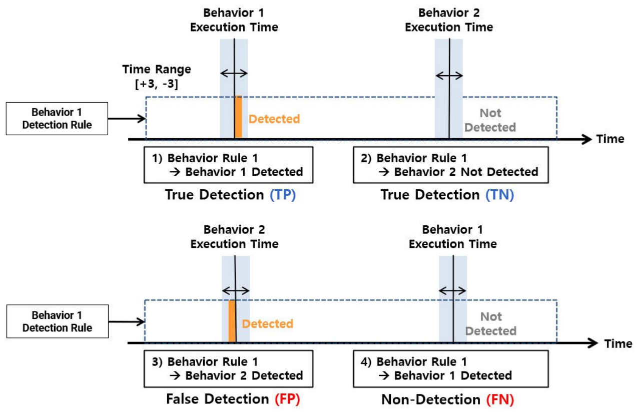

Behavior 1 Execution Time
Behavior 2 Execution Time
Time Range [+3, -3]
Behavior 1 Detection Rule
Detected
Not Detected
Time
1) Behavior Rule 1 → Behavior 1 Detected
True Detection (TP)
2) Behavior Rule 1 → Behavior 2 Not Detected
True Detection (TN)
Behavior 2 Execution Time
Behavior 1 Execution Time
Behavior 1 Detection Rule
Detected
Not Detected
Time
3) Behavior Rule 1 → Behavior 2 Detected
False Detection (FP)
4) Behavior Rule 1 → Behavior 1 Detected
Non-Detection (FN)

FIGURE 8. An example of behavior detection result.

The behavior detection result is categorized into true detection (TP, TN), false detection (FP), and non-detection (FN), as illustrated in Fig. 8. We focus on detecting the target behavior; therefore, we do not consider other behaviors from the perspective of the target behavior. True Positive (TP) refers to the case where the host IP, application, and behavior information all match, and the detected time is within the range of the behavior execution time. True Negative (TN) indicates the case where a different behavior is performed and correctly not detected. False Positive (FP) corresponds to cases where a different behavior is mistakenly detected as the target behavior. False Negative (FN) represents cases where the target behavior is performed but not detected.

The detection result calculates the Recall, Precision, and F1 measure by using TP, TN, FP, and FN. Equation (14), (15), (16), and (17) indicates the formula of Recall, Precision, F1-measure, and Accuracy.

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$Precision = \frac{TP}{TP + FP} \tag{15}$$

$$F1 - measure = \frac{2 \times Precision \times Recall}{(Recall + Precision)} \tag{16}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{17}$$

2) RULE UPDATE

In real network environments, SaaS applications often encounter changes in traffic patterns due to reasons such as security updates and application version upgrades. A limitation of the signature-based analysis method is that predefined

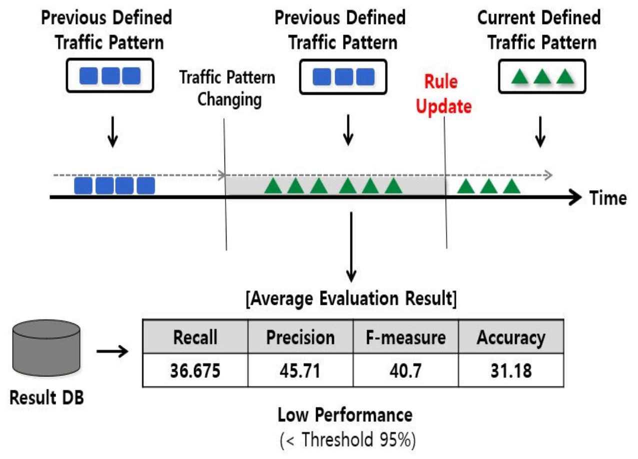

Previous Defined Traffic Pattern
Traffic Pattern Changing
Previous Defined Traffic Pattern
Rule Update
Current Defined Traffic Pattern
Time
[Average Evaluation Result]
Recall
Precision
F-measure
Accuracy
36.675
45.71
40.7
31.18
Result DB
Low Performance (< Threshold 95%)

FIGURE 9. A process of rule update mechanism.

signatures cannot be applied when there are changes in traffic patterns or characteristics. As the proposed method is also a type of signature analysis, it is susceptible to changes in traffic patterns.

To address this limitation, we have designed a mechanism for rule update and regeneration. The rule update process determines when a change in traffic pattern occurs, while the rule regeneration process executes the rule generation procedure when it is deemed necessary.

The rule update process assesses the performance of the rule by collecting behavior detection results. As mentioned earlier, user behavior detection is conducted in a multi-host environment. The traffic collected through packet mirroring is stored every minute. Behavior detection is performed on this collected traffic, and detection results are generated at one-minute intervals. These detection results are aggregated in units of 1, 6, 12 hours, and 1-, 3-, and 7-day intervals.

Fig. 9 illustrates the mechanism of rule update, showcasing how a threshold is set for detection results to assess the degradation of detection performance caused by changes in traffic patterns. If the detection result falls below the threshold, the average detection result is calculated for 30 minutes, 1 hour, and 3 hours from the corresponding time point. If the derived average detection result remains below the threshold, it indicates a change in the traffic pattern of the target application.

When conducting behavior detection experiments, we have set the detection result threshold to 0.95, taking into account that the majority of applications demonstrate an average detection performance of 95% or higher. In these experiments, F-measure and Accuracy are used as the evaluation metric to assess the overall detection performance. In cases where over 10 user behaviors occur within a specified period, and the average detection performance falls below 0.95, a rule update is performed.

3) RULE RE-GENERATION

Rule regeneration performs the rule generation process again, targeting application behaviors that require rule update. In rule regeneration, the behavior and application rule generation are performed in the same way.

V. EVALUATION OF THE PROPOSED METHOD

In this section, we conduct experiments to verify our proposed method. We implement the proposed system through Python code on Linux. Experiments are performed on a server with Intel(R) Core (TM) i7-4770K CPU, 32GB Memory, and CentOS Linux release 7.9.2009 operating system.

We use Wireshark to collect the target application traffic for rule generation. The system extracted one JSON file for each application containing the information of header, SNI, PSD signatures, and behavior sequence, as previously shown in Table 3. We performed 4 experiments with traffic collected from multi-host network environments and generated rule files as input.

In the first experiment, we evaluate the detection performance for 5 applications in a multi-host environment. We used packet mirroring within Korea University to implement a multi-host environment and collected the campus network traffic. In the second experiment, we conduct detection experiments targeting other types of applications besides SaaS. The target application was selected as Microsoft Office 365 and Zoom. Microsoft Office 365 and Zoom are SaaS types, but Microsoft Office 365 also belongs to the document processing type, and Zoom belongs to the remote meeting type. We defined the 3 new target behaviors as ‘‘use PowerPoint, Word, and Excel’’ in Microsoft Office 365 and 2 new target behaviors as ‘‘remote meeting’’ and ‘‘terminate the remote meeting’’ in Zoom. In the third experiment, we conducted several comparison experiments.

The detection performance is compared according to the signature application method. In addition, we compare the detection performance with and without the behavior sequence to verify the effect of the behavior sequence. In the fourth experiment, the time consumed in the rule generation is compared with the proposed method and our previous method [28].

A. DATASET

We selected 5 SaaS applications (e.g., Microsoft Office 365, Adobe Creative Cloud, Autodesk, Slack, and Zoom) for the experiment and used the privately collected dataset for each application. For the dataset, 30 traces were collected for each application, of which 20 traces were used for rule generation and 10 traces for behavior detection. Information on the dataset is shown in Table 4 and 5.

In SaaS-type applications, it is important to select an appropriate license because the services and behaviors provided may vary depending on the license type. We selected widely used licenses with common functions for each application. An educational license provided for the university is used in Office 365, Adobe Creative Cloud, and Zoom. The AutoCAD LT 2023 license is used in Autodesk, and the free license is used in Slack.

TABLE 4. Information of traffic dataset in rule generation.

Rule Generation: Traffic Information (for SaaS type)			
Application	**Behavior**	**Flow**	**Packet**
Microsoft Office 365	App. Start	5,523	185,441
	Login	5,778	230,109
	Logout	6,120	100,279
	App. End	4,121	98,118
Adobe Creative Cloud (i.e., Adobe CC)	App. Start	6,170	180,214
	Login	5,121	164,114
	Logout	7,941	242,253
	App. End	3,024	156,211
Autodesk	App. Start	5,018	170,112
	Login	5,121	146,525
	Logout	3,154	94,565
	App. End	2,858	80,112
Slack	App. Start	3,021	77,858
	Login	5,475	90,141
	Logout	5,874	60,114
	App. End	5,928	60,358
Zoom	App. Start	5,852	101,225
	Login	5,787	128,221
	Logout	4,054	146,855
	App. End	5,995	98,996

Table 4 shows the traffic data used to generate the rule and shows the average flow and number of packets for each trace. In rule generation, the target behavior was performed 5 times in one trace, and 20 traces were used for each behavior. For the better performance, we collected GT traffic of each behavior. To collect GT traffic for each application's behavior, the collection environment was built as an isolated network environment and collected in a single host environment. TLS is widely used for encrypted communication, and among the five applications used in our experiments, TLS was the most commonly utilized method. Across all applications, TLS 1.2 packets appeared with the highest frequency. TLS 1.3 followed as the second most prevalent version. Microsoft Office 365 also exhibited UDP-based encryption using QUIC. Additionally, TLS 1.1 packets were sporadically observed in a few applications.

Table 5 shows each traffic information collected in a multi-host environment, and 10 traces were collected for each application. Since we collected the traffic in a multi-host environment, the traffic includes the target host's application behavior and various traffic from other hosts. *Flow* and *Packet* represent the number of flows and packets in the traffic, respectively, and *Host* represents the total number of hosts in a multi-host environment. For each trace, 4 behaviors were repeated 10 times for each application.

We recorded the host IP, application name, behavior, and behavior execution time information for the collected dataset, and the recording process was performed manually in the experiment.

B. PERFORMANCE EVALUATION

In this section, we describe the results of 4 experiments. Behaviors for each experiment were recorded manually, and evaluation metrics were calculated by comparing them with behavior detection results.

1) USER BEHAVIOR DETECTION IN A MULTI-HOST ENVIRONMENT

The first experiment focuses on behavior detection in a multi-host environment, specifically aiming to accurately detect behaviors defined for each application. The results of this experiment are presented in Table 6.

In Table 6, the 4 behaviors are labeled as behaviors #1(application start), #2(login), #3(logout), and #4(application end), respectively. The experiment resulted in an average F-measure ranging from 92∼ 99% and an accuracy ranging from 89∼99% across the five applications. Most of the applications achieve an F-measure and accuracy in the range of approximately 95∼99%, except Slack. Among the 5 applications, Microsoft Office 365 exhibits the highest performance, achieving an F-measure and accuracy of 99%.

Slack is classified as both a SaaS and messenger-type application, and its login and logout processes differ from those of other applications. Messenger-type applications can run in the background and utilize tokens for identification and authentication during the login and logout procedures. Due to the token-based authentication, where users can access applications without additional credentials within the token's validity period, there are instances where traffic patterns may not be clearly observed during login and logout. This can result in higher rates of false detection and non-detection, mainly caused by inaccurate signature definitions for these specific stages. This challenge poses difficulties in the field of network traffic analysis, and we plan to explore methods for detecting behaviors in applications that utilize token-based authentication in future research.

2) USER BEHAVIOR DETECTION FOR OTHER APPLICATION TYPE

We conducted an experiment applying the proposed method to detect the behavior of different types of applications.

We defined 3 behaviors of the document processing type application and 2 behaviors of the remote meeting type application, instead of the 4 behaviors defined for the SaaS type. As the behaviors defined by Microsoft Office 365 and Zoom differ from the 4 behaviors of the SaaS type, we created new rules specifically for these behaviors.

Table 7 represents information about collected traffic for rule generation and behavior detection for each action. It provides details about flows and packets. However, since our emphasis was on SaaS-type application behavior detection, the second experiment was carried out using only 5 traces for each application. Table 8 shows the detection experiment results for different types of application behaviors. In Table 8, the 3 behaviors of Microsoft Office 365 (i.e., use PowerPoint,

TABLE 5. Information of traffic dataset in a multi-host environment.

Traffic Information (for SaaS type)											
Application	**Traffic**	**Trace**									
		#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**
Microsoft Office 365	**Flow**	1,523	1,622	3,588	2,619	2,605	3,768	4,705	7,545	6,608	6,550
	Packet	124,449	100,290	119,622	228,166	215,259	223,823	419,659	414,095	316,055	216,295
	Host	3	3	3	5	5	5	10	10	10	10
Adobe Creative Cloud	**Flow**	2,594	1,627	1,625	2,520	2,820	3,616	4,653	6,540	6,625	6,804
	Packet	98,937	83,598	124,562	313,784	235,441	219,964	430,121	219,753	316,873	135,743
	Host	3	3	3	5	5	5	10	10	10	10
Autodesk	**Flow**	1,611	1,538	1,014	1,685	1,626	4,842	3,763	6,851	5,628	7,743
	Packet	135,516	84,956	95,650	114,658	151,220	390,611	165,612	356,190	3557,89	363,467
	Host	3	3	3	5	5	5	10	10	10	10
Slack	**Flow**	1,762	1,598	1,614	2,500	2,503	2,715	4,649	7,606	6,503	6,568
	Packet	91,890	121,473	77,734	222,468	212,412	395,528	216,353	453,651	339,436	212,355
	Host	3	3	3	5	5	5	10	10	10	10
Zoom	**Flow**	829	1,684	904	3,678	3,635	2,746	4,473	6,489	5,507	5,496
	Packet	97,872	117,027	96,312	266,372	226,031	306,863	235,439	335,321	235,577	225,167
	Host	3	3	3	5	5	5	10	10	10	10

TABLE 6. Result of behavior detection for 5 SaaS applications.

	Detection Result (%)				
	Beh.	**Recall**	**Precision**	**F-measure**	**Acc.**
Microsoft Office 365	#1	100	100	100	100
	#2	100	98.25	99.12	99.56
	#3	99.28	100	99.63	99.69
	#4	97.68	100	98.83	98.92
	Avg.	**99.24**	**99.56**	**99.27**	**99.49**
Adobe Creative Cloud	#1	96.54	94.33	95.42	94.10
	#2	100	92.18	95.93	95.68
	#3	94.35	96.82	95.57	95.93
	#4	95.61	97.78	96.68	98.01
	Avg.	**96.63**	**94.53**	**95.57**	**95.93**
Autodesk	#1	100	97.17	98.56	98.66
	#2	98.28	94.10	96.14	95.71
	#3	100	93.18	96.47	96.30
	#4	100	94.58	97.21	97.25
	Avg.	**99.57**	**94.76**	**97.10**	**96.98**
Slack	#1	94.18	96.57	95.36	96.38
	#2	93.29	87.38	90.24	89.17
	#3	91.58	84.69	88.00	87.82
	#4	93.69	97.29	96.48	94.11
	Avg.	**93.18**	**91.48**	**92.52**	**89.37**
Zoom	#1	100	94.18	97.00	95.97
	#2	100	92.60	96.16	94.61
	#3	94.18	95.71	94.94	95.18
	#4	98.68	94.28	96.43	96.02
	Avg.	**98.21**	**94.19**	**96.13**	**95.45**

TABLE 7. Information of traffic dataset for other type application.

Rule Generation: Traffic Information (for other type)			
Application	**Behavior**	**Flow**	**Packet**
Microsoft Office 365	**Use a PowerPoint**	3,127	100,244
	Use a Word	2,703	98,122
	Use an Excel	3,877	112,139
Zoom	**Start Remote Meeting**	4,580	158,280
	Terminate the Remote Meeting	3,012	100.224
Experiment #2: Traffic Information (for other type)			
Application	**Trace**	**Flow**	**Packet**
Microsoft Office 365	**#1**	523	54,449
	#2	724	54,112
	#3	844	66,258
	#4	887	50,114
	#5	984	77,245
Zoom	**#1**	1,024	90,345
	#2	917	87,001
	#3	887	123,157
	#4	1,125	104.019
	#5	974	70,885

Word, and Excel) are represented as behaviors #o1, #o2, and #o3, 2 behaviors of Zoom (i.e., remote meeting and terminate the remote meeting) are represented as behaviors #z1 and #z2.

As a result of the experiment, an average of 98∼100% F-measure and 98∼ 100% accuracy was shown in 2 applications. Microsoft Office 365 shows clear SNI signatures for all 3 behaviors. For example, the SNI of "*powerpoint.office.com*" is derived using PowerPoint. Zoom shows

TABLE 8. Result of behavior detection for other type applications.

	Detection Result (%)				
	Beh.	Recall	Precision	F-measure	Acc.
Microsoft Office 365	#o1	100	100	100	100
	#o2	100	100	100	100
	#o3	100	99.58	99.79	99.70
	Avg.	**100**	**99.86**	**99.93**	**99.90**
Zoom	#z1	98.02	97.94	97.98	98.03
	#z2	98.45	99.34	98.89	98.61
	Avg.	**99.22**	**98.64**	**98.43**	**98.32**

TABLE 9. Comparison result of the behavior rule generation time.

	Application Rule-Generation Tim (minute)	
	Previous Method [29]	Proposed Method
Application Start	184	45
Login	200	42
Logout	191	58
Application End	164	39
Rule Aggregation	20	1
Total	759	185
Average (Except for the rule aggregation)	184.75	46

TABLE 10. Comparison result of the applkication rule generation time.

		Behavior Rule-Generation Time (minute)	
Method		Previous Method [29]	Proposed Method
Traffic Preprocessing		10	10
Signature Generation	**Header**	20	20
	SNI	120	5
	PSD	-	5
Signature Aggregation (Behavior Rule Generation)		20	1
Total		180	41

a similar pattern in the PSD or each behavior. PSDs derived for each activity show a similar pattern. The results show that a clear common signature can be derived from different types of application behaviors, and the proposed method can be applied to other types of application behaviors.

3) COMPARISON EXPERIMENT (1)

In the third experiment, we conducted performance comparison experiments for the signature application method. Microsoft Office 365 was selected as the target application, and the experiment environment was set to be identical. We compared the performance of 5 signature methods: *i)* header, *ii)*SNI, *iii)*PSD, *iv)*All, and *v)* All (BS). Among these methods, *i*), *ii),* and *iii)* apply individual methodologies, *iv)* All represents a method that utilizes all 3 signatures in the proposed approach but does not consider behavior sequences. On the other hand, *v)*All (BS) represents a method that uses all 3 signatures and takes behavior sequences into consideration.

If a corresponding signature is not present in the target behavior, it is considered as a non-detection since it cannot be detected. For example, the header signature exists in the application start of Microsoft Office 365 but does not exist in the login behavior. In such cases, the performance of the header signature in login is evaluated as non-detection.

Fig. 10 shows the detection performance according to the applied signature. Since Microsoft connects to a specific site when starting the application, information about the specific site is defined as a header signature. Therefore, the application start of Microsoft Office 365 can be detected with 100% accuracy based on this information. However, the header signature is derived only at the application start, it cannot be detected in other behaviors.

The SNI signature demonstrated 100% accuracy in application start and approximately 95-97% accuracy in login and logout. However, for the application end, the accuracy dropped to around 80-82%. The application end is detected when the process's end button is pressed. During this event, the SNI signature for the application end does not have a fixed SNI but varies depending on the traces. As a result, false detections and non-detections occurred relatively more frequently compared to other behaviors.

On the other hand, the PSD signatures achieved approximately 93-98% detection accuracy across the four behaviors, exhibiting the highest individual signature detection performance. However, false detections and non-detections were more likely to occur due to slight variations in threshold values and distances.

In the case of All, it is detected with an accuracy of about 97∼100%, and the overall detection performance is improved compared to when an individual signature is applied. However, since the behavior sequence is not considered, duplicate detection occurs when the same pattern occurs even if the target behavior is detected.

Duplicate detection is judged as a false detection (FP) in the detection criteria, and it mainly occurs in the login. All (BS) can solve this problem because it considers the sequence of behaviors. Considering the sequence of behaviors, login is only detected after the application start. In addition, after being detected, it is not detected until a logout occurs. Therefore, All (BS) showed about 99∼100% detection accuracy in 4 behaviors,

Overall, the detection performance can be limited when relying solely on individual signatures. This is because some traffic information used for detecting specific behaviors can be effectively captured by signatures, while others may lack clear patterns for accurate signature definition. For instance,

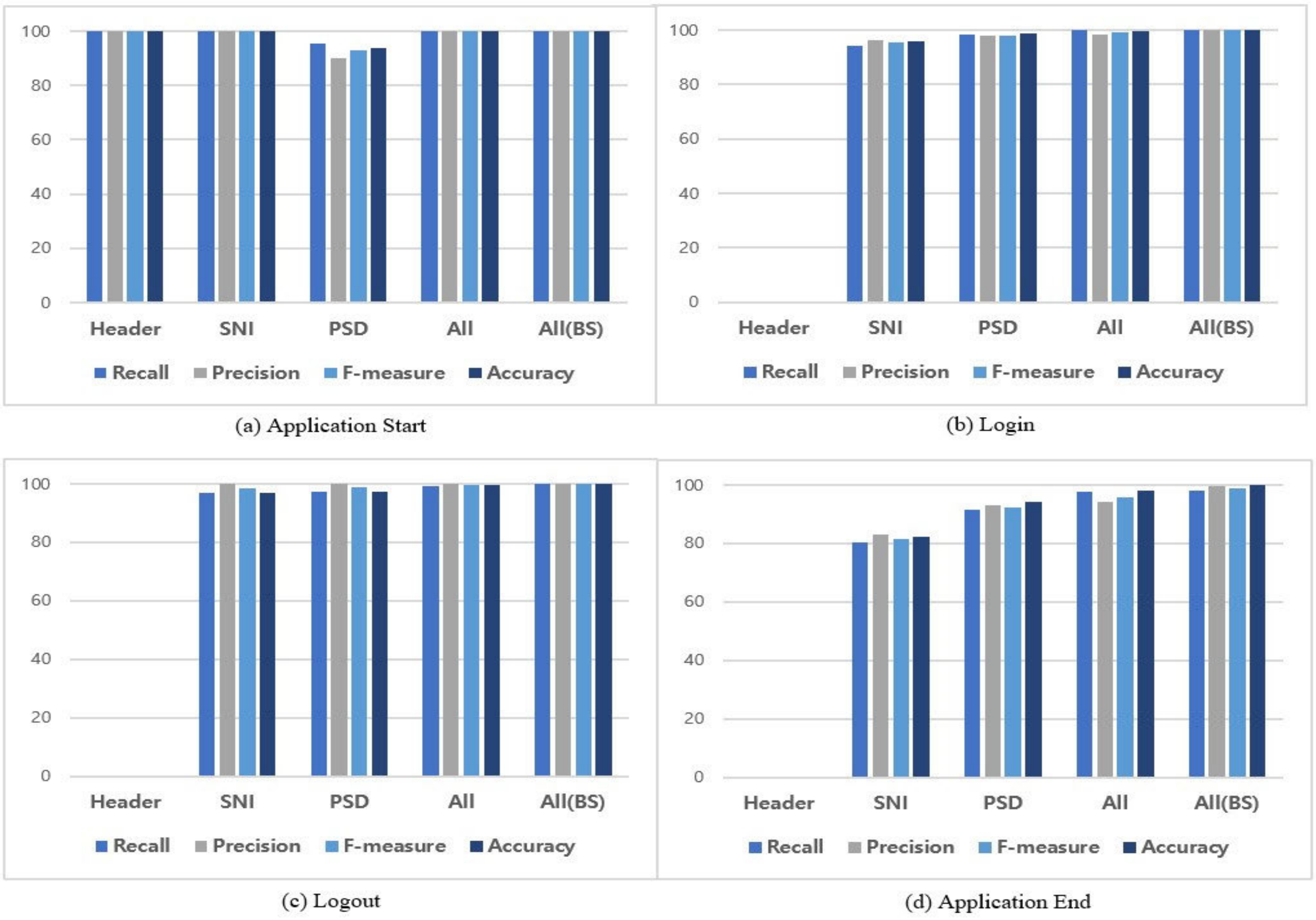


FIGURE 10. Performance comparison for the signature application method in Microsoft Office 365.

in Fig. 10, the application end behavior in Microsoft Office 365 does not exhibit a distinct SNI pattern, resulting in an SNI accuracy of approximately 80%. On the other hand, PSD signatures demonstrate relatively clearer patterns, achieving a detection performance of around 90%. However, in practical detection, it is not solely dependent on individual signatures but also incorporates the three signatures and behavior sequences for a comprehensive approach. Consequently, even for the application end behavior in Microsoft Office 365, a detection performance of over 98% is achieved. This aspect can be regarded as another contribution of this paper, addressing the limitations of existing signature-based analysis methods that rely solely on individual signatures.

4) COMPARISON EXPERIMENT (2)

To validate the proposed method, a comparative experiment with other methods is necessary. However, as mentioned earlier, comparing the objective performance in user behavior detection research is challenging due to variations in the target application and behavior definition methods used in each research.

In the fourth experiment, we conducted comparison experiments with our previous research [28]. We compare the rule generation time to verify the efficiency of the automatic rule generation method of the proposed method.

Table 9 shows the behavior rule generation time for each method, and we use the Autodesk login for the target. The traffic collection and pre-processing process is the same for both methods, and it takes 10 minutes. The header signature is the process of extracting a fixed IP and port if there is one, and the two methods perform the same. There are differences in the two methods of the SNI signature generation process. In the previous research, the SNI signature was used to derive commonly occurring SNI information by checking the SNI for all flows for each trace, and it took about 120 minutes.

However, the proposed method automatically extracts a common SNI signature with all trace traffic as input and takes within 5 minutes. Since PSD signatures were not applied in the previous method, we excluded the PSD signatures. The proposed method extracts signatures within 5 minutes. Signature aggregation is a process of generating a single behavior detection rule by collecting several generated signatures. The previous method was performed manually, and it took about 20 minutes for the procedure. On the other hand, the proposed method performs this process automatically, and each takes within 1 minute. Comparing the total time, the previous method consumes 180 minutes, excluding PSD signatures,

TABLE 11. An example of calculating the computational complexity for the behavior detection.

	Information
Input Traffic	Network Traffic Collected for 1 minute
Trace Count	1
Entire Flow Count	N
SNI Signature Matched Flow Count	T (T<N)
Matching Method	Linear (Signature: Flow = 1: N)
Signature Count	Header :1
	SNI :1
	PSD: m
Time Complexity *(Best Case)*	Header :O(1)
	SNI :O(1)
	SNI+PSD :O(1)
	PSD :O(1)
Time Complexity *(Worst Case)*	Header :O(N)
	SNI :O(N)
	SNI+PSD :O(T)
	PSD :O(m × N)

and the proposed method consumes about 90 minutes. The time difference will likely become larger when there are more traffic traces.

Table 10 shows the application rule generation time for each method, and we use Autodesk for the target. In the case of the previous method, it takes about 759 minutes to generate the entire application rule and an average of 184.75 minutes for one behavior, excluding the rule aggregation. In the case of the proposed method, it takes about 185 minutes to generate the entire application rule and an average of 46 minutes for one behavior, excluding the rule aggregation. Comparing the rule generation time of the two methods, the proposed method can significantly reduce both the behavior and application rule generation time.

Although our proposed method's automatic rule generation significantly reduces the time compared to the manual rule generation in previous studies, it is still time-consuming task. To address this issue, we have divided our system into two main parts: rule generation and behavior detection. The rule generation part can operate offline once sufficient traffic data sets for the target application have been collected. Additionally, similar to the training time in the learning-based analysis method, the rule generation time may vary based on the volume of collected traces. Nevertheless, it is possible to further decrease the rule generation time using a GPU, as demonstrated in the comparison experiment (2).

VI. DISCUSSION

A. COMPUTATIONAL COMPLEXITY

The computational complexity can vary depending on different situations and environments. In this section, we explain various factors that influence computational complexity. As we mentioned in Section IV, the proposed system consists of 3 main parts: rule generation, rule validation, and behavior detection, all of which are affected by hardware specifications.

In the rule generation part, multiple GT traffic traces are taken as input to derive the three signatures, and the aggregation process is performed. Therefore, the computational complexity of rule generation is primarily influenced by the amount of traffic data (e.g., GT traffic trace count, the number of flows in each trace), and the time complexity increases linearly with the increase in data quantity. Similarly, in the rule validation part, the computational complexity is also affected by the amount of traffic data (e.g., GT traffic trace count, the number of flows in each trace) and the time complexity increases linearly with the increase in data quantity.

Behavior detection involves deriving detection results through signature matching with the input traffic data collected over one minute through mirroring. Since behavior detection utilizes data preprocessed at the flow level, it is less affected by the packet quantity and more influenced by the number of flows. Furthermore, the final number of rules and signatures obtained during the rule generation and validation processes affects the matching frequency, resulting in a linear increase in time complexity with the number of signatures.

Table 11 provides an example of computational complexity calculation for behavior detection regarding a single behavior in one application. The input traffic represents collected network data, and the number of flows in the traffic is denoted as N, with T representing the number of flows matched by the SNI signature. The matching method indicates a one-to-one correspondence between a flow and a signature. For example, a single SNI signature for a target behavior will undergo matching with all collected flows until the target behavior is detected.

In this case, the time complexity is represented as follows for each signature. The time complexities in Table 11 are provided as examples for specific scenarios and may vary depending on the other various factors explained earlier.

B. LIMITATIONS OF THE PROPOSED METHOD

Our proposed method has several limitations. Firstly, it incorporates a rule update mechanism to address the challenge of traffic pattern changes in the signature-based method. While the focus of this paper is on the overall system architecture and signature generation, it is essential to validate the effectiveness of the rule update mechanism. Nonetheless, the rule update mechanism is expected to address the limitations of the signature-based method in response to traffic pattern changes. Since the rule generation is the same, enabling the system to autonomously assess the need for rule updates based on performance degradation by adjusting the threshold. This approach is anticipated to effectively overcome the drawbacks of the signature-based method caused by changes in traffic patterns.

Secondly, the proposed method exhibits high detection performance for the majority of SaaS applications. However, it exhibits lower detection performance when applied to applications that employ token-based authentication methods like Slack. Token-based authentication involves exchanging authentication information during the initial authentication, which allows for behavior detection through the generated traffic. However, after the initial authentication, the client holds the authentication information in the form of tokens, which remain valid until their expiration. Subsequent authentication actions, such as login and logout, only require checking the validity of the tokens. In this case, the minimal packet generation, unlike session-based authentication, poses a challenge for detecting behaviors through traffic analysis, not only in the proposed method but also in traffic analysis research.

Thirdly, while we have conducted several experiments on the proposed method, it is crucial to validate its effectiveness through performance comparisons with other research. In particular, this paper acknowledges the need for performance comparisons with recent studies in the field of user behavior detection research. As mentioned earlier, comparing performance with other studies can be challenging due to variations in the applications used and behavior definitions. However, in the second experiment, we provide preliminary evidence that the proposed method can be applied to behaviors of applications used in other studies, allowing for performance comparisons. This is expected because the proposed method consistently demonstrates high performance, even when applied to different types of applications with varying behavior definitions used in other studies.

Lastly, the proposed method cannot directly apply the header and SNI signatures to other encryption protocols such as IPsec and QUIC. However, by utilizing the PSD signature, the proposed method can be partially applied to these protocols. It is important to note that relying solely on the PSD signature may result in performance degradation. Therefore, further research is needed to explore advanced techniques to address this limitation.

Nevertheless, the proposed method exhibits high detection performance by leveraging multi-modal signatures for encrypted traffic. As a result, the proposed method can be applied to diverse domains that involve encrypted traffic, including user behavior detection and malicious behavior detection.

C. FEASIBILITY AND UTILIZATION OF THE PROPOSED METHOD

The primary objective of the proposed method is to accurately detect user behavior using Multi-Modal Signatures. It seamlessly integrates with existing traffic analysis systems and operates effectively on the internet. The method comprises two components: rule generation and behavior detection, each offering distinct advantages depending on its specific application.

The proposed method can be applied in diverse scenarios, with two prominent examples. Firstly, it can be integrated into enterprise or organizational network traffic analysis and management systems. When deployed within an organization's network, the system conducts comprehensive traffic analysis, providing profound insights into user activities and application usage. We have successfully implemented this approach on Korea University's campus network.

Secondly, the method shows potential for network security. Though network security is not the primary focus of this paper, the proposed method is expected to demonstrate promising results in efficiently and accurately identifying malicious activities. Leveraging its multi-modal approach, the system effectively recognizes patterns indicative of various cyber threats, including malware infections, intrusion attempts, and anomalous activities. This capability could strengthen network security and enable a proactive response to potential cybersecurity incidents. Moreover, the method's adaptability allows for continuous rule updates and refinements, potentially enhancing its effectiveness against evolving cyber threats over time.

However, it is important to acknowledge that the proposed method may take longer processing times in certain network environments, especially with large-scale traffic. Additionally, creating signatures for target applications beforehand may limit its response to novel types of applications or attack behaviors. Nonetheless, the proposed system holds potential for application in various fields, such as network monitoring, intrusion detection, and traffic analysis, where its adaptability and multi-modal approach could be advantageous.

VII. CONCLUSION

We have discussed the research on user behavior detection and highlighted several limitations in existing studies. These studies often differ in terms of the target application and behavior definition methods used, making it challenging to objectively compare performance.

In this paper, we address the challenge of achieving objective performance comparison in user behavior detection research. To address this issue, we propose a method for categorizing application types, which serves as a guideline for consistently defining behavior based on the application type. This approach significantly improves the comparability of research performance.

As part of our contributions, we propose a rule-based user behavior detection method. This signature-based traffic analysis method utilizes multi-modal signatures, including header, SNI, and PSD signatures, and is capable of effectively handling encrypted traffic. Our proposed system consists of 3 main modules: rule generation, behavior detection, and rule update.

In the rule generation phase, we address the challenges of manual traffic analysis and rule generation by applying an automatic signature extraction algorithm for each signature. The behavior detection module is responsible for detecting user behavior in a real network, providing detection results in terms of recall, precision, F-measure, and accuracy. To ensure the method's adaptability to changing traffic patterns, we also introduce a rule update mechanism that mitigates performance degradation. With this mechanism, the detection rules can be adjusted to accommodate variations in traffic patterns, maintaining a high level of accuracy over time.

To verify the proposed system, we conducted several experiments for 5 SaaS applications. In the first experiment, we evaluated the detection performance in a multi-host environment. The results demonstrated that the proposed method achieved high detection performance except for Slack, which utilizes a token-based authentication method. In the second experiment, we evaluated the detection performance for applications of other types, and the proposed method exhibited consistently high performance.

In the third experiment, we conducted a comparative analysis of the detection performance for each signature and examined the impact of behavior sequences on the overall performance. In our experiments, we compared the detection performance of each signature, and we found that in most applications, only SNI and PSD signatures were used, while header signatures were rarely utilized. Header signatures are primarily applicable when fixed IP and Port are used. However, in most current applications, fixed IP and port are not employed. Therefore, we plan to conduct future research to enhance header signatures by utilizing IP ranges instead of fixed IPs. The results highlighted the effectiveness of the proposed method in analyzing behavior sequences, further enhancing the detection performance.

Lastly, in the fourth experiment, we compared the rule generation time with our previous research [28]. The proposed method significantly reduced the rule generation time compared to the manual rule generation approach employed in the previous research. This demonstrates the efficiency and effectiveness of the proposed automated rule generation process.

In the future, we plan to improve our proposed method through five additional research directions. Firstly, we aim to enhance the signature generation process with more sophisticated algorithms to further increase the detection performance. This improvement is expected to enhance the results of the first experiment. Secondly, we will conduct additional traffic analysis to address the low detection performance observed in token-based applications. Thirdly, we intend to use the application type categorization and behavior definition method presented in this paper to perform performance comparisons with other studies. Fourthly, considering the significance of user behavior in network management and security, we will explore applying our proposed method to detect malicious traffic in the field of network security. Lastly, we are planning to review and partially share a portion of the traffic dataset used for rule generation in this paper. These future research endeavors will help further refine and validate the effectiveness of our proposed approach.

REFERENCES

[1] A. Callado, C. Kamienski, G. Szabo, B. P. Gero, J. Kelner, S. Fernandes, and D. Sadok, “A survey on internet traffic identification,” *IEEE Commun. Surveys Tuts.*, vol. 11, no. 3, pp. 37–52, 3rd Quart., 2009.

[2] A. Dainotti, A. Pescape, and K. C. Claffy, “Issues and future directions in traffic classification,” *IEEE Netw.*, vol. 26, no. 1, pp. 35–40, Jan. 2012.

[3] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, “Network traffic classification using correlation information,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 104–117, Jan. 2013.

[4] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Müller, and K. Hanssgen, “A survey of payload-based traffic classification approaches,” *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1135–1156, 2nd Quart., 2014.

[5] A. Shahraki, M. Abbasi, A. Taherkordi, and A. D. Jurcut, “Active learning for network traffic classification: A technical study,” *IEEE Trans. Cogn. Commun. Netw.*, vol. 8, no. 1, pp. 422–439, Mar. 2022.

[6] A. Madhukar and C. Williamson, “A longitudinal study of P2P traffic classification,” in *Proc. 14th IEEE Int. Symp. Modeling, Anal., Simulation*, Monterey, CA, USA, 2006, pp. 179–188.

[7] Y.-H. Goo, K.-S. Shim, S.-K. Lee, and M.-S. Kim, “Payload signature structure for accurate application traffic classification,” in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Kanazawa, Japan, Oct. 2016, pp. 1–4.

[8] F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus, “Lightweight, payload-based traffic classification: An experimental evaluation,” in *Proc. IEEE Int. Conf. Commun.*, Beijing, China, May 2008, pp. 5869–5875.

[9] J.-S. Park, S.-H. Yoon, and M.-S. Kim, “Performance improvement of payload signature-based traffic classification system using application traffic temporal locality,” in *Proc. Asia–Pacific Netw. Oper. Manage. Symp.*, 2013, pp. 1–6.

[10] X. Feng, X. Huang, X. Tian, and Y. Ma, “Automatic traffic signature extraction based on smith-waterman algorithm for traffic classification,” in *Proc. 3rd IEEE Int. Conf. Broadband Netw. Multimedia Technol. (IC-BNMT)*, Beijing, China, Oct. 2010, pp. 154–158.

[11] H. M. An, S. K. Lee, J. H. Ham, and M. S. Kim, “Traffic identification based on applications using PSD signature free from abnormal TCP behavior,” *J. Inf. Sci. Eng.*, vol. 31, no. 5, pp. 1669–1692, Sep. 2015.

[12] C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21954–21961, 2017.

[13] P. Wang, X. Chen, F. Ye, and Z. Sun, “A survey of techniques for mobile service encrypted traffic classification using deep learning,” *IEEE Access*, vol. 7, pp. 54024–54033, 2019.

[14] M. J. de Lucia and C. Cotton, “Detection of encrypted malicious network traffic using machine learning,” in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Norfolk, VA, USA, Nov. 2019, pp. 1–6.

[15] T. T. T. Nguyen and G. Armitage, “A survey of techniques for internet traffic classification using machine learning,” *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.

[16] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, “End-to-end encrypted traffic classification with one-dimensional convolution neural networks,” in *Proc. IEEE Int. Conf. Intell. Secur. Informat. (ISI)*, Jul. 2017, pp. 43–48.

[17] C. Liu, L. He, G. Xiong, Z. Cao, and Z. Li, “FS-Net: A flow sequence network for encrypted traffic classification,” in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Paris, France, Apr. 2019, pp. 1171–1179.

[18] M. Lotfollahi, M. J. Siavoshani, R. S. H. Zade, and M. Saberian, “Deep packet: A novel approach for encrypted traffic classification using deep learning,” *Soft Comput.*, vol. 24, no. 3, pp. 1999–2012, Feb. 2020.

[19] C. Hou, J. Shi, C. Kang, Z. Cao, and X. Gang, “Classifying user activities in the encrypted WeChat traffic,” in *Proc. IEEE 37th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Nov. 2018, pp. 1–8.

[20] K. Park and H. Kim, “Encryption is not enough: Inferring user activities on KakaoTalk with traffic analysis,” in *Proc. Int. Workshop Inf. Secur. Appl. (WISA)*. Cham, Switzerland: Springer, 2015, pp. 254–265.

[21] H. Wu, Q. Wu, G. Cheng, and S. Guo, “Instagram user behavior identification based on multidimensional features,” in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Toronto, ON, Canada, Jul. 2020, pp. 1111–1116.

[22] B. Saltaformaggio, H. Choi, K. Johnson, Y. Kwon, Q. Zhang, X. Zhang, D. Xu, and J. Qian, “Eavesdropping on fine-grained user activities within smartphone apps over encrypted network traffic,” in *Proc. Workshop Offensive Technol. (WOOT)*, 2016, pp. 69–78.

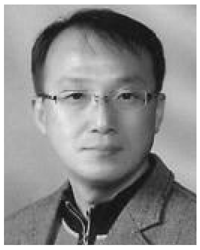
[23] E. Grolman, A. Finkelshtein, R. Puzis, A. Shabtai, G. Celniker, Z. Katzir, and L. Rosenfeld, “Transfer learning for user action identication in mobile apps via encrypted trafc analysis,” *IEEE Intell. Syst.*, vol. 33, no. 2, pp. 40–53, Mar. 2018.

[24] S. E. Coull and K. P. Dyer, "Traffic analysis of encrypted messaging services: Apple iMessage and beyond," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 5–11, Oct. 2014.
[25] M. Conti, L. V. Mancini, R. Spolaor, and N. V. Verde, "Analyzing Android encrypted network traffic to identify user actions," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 1, pp. 114–125, Jan. 2016.
[26] Y. Fu, H. Xiong, X. Lu, J. Yang, and C. Chen, "Service usage classification with encrypted internet traffic in mobile messaging apps," *IEEE Trans. Mobile Comput.*, vol. 15, no. 11, pp. 2851–2864, Nov. 2016.
[27] M. Jiang, G. Gou, J. Shi, and G. Xiong, "I know what you are doing with remote desktop," in *Proc. IEEE 38th Int. Perform. Comput. Commun. Conf. (IPCCC)*, London, U.K., Oct. 2019, pp. 1–7.
[28] J.-T. Park, U.-J. Baek, M.-S. Kim, M.-S. Lee, and C.-Y. Shin, "Rule-based user behavior detection system for SaaS application," in *Proc. 23rd Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2022, pp. 1–4.
[29] A. Subahi and G. Theodorakopoulos, "Detecting IoT user behavior and sensitive information in encrypted IoT-app traffic," *Sensors*, vol. 19, no. 21, p. 4777, Nov. 2019.
[30] X. Xiao, W. Xiao, R. Li, X. Luo, H. Zheng, and S. Xia, "EBSNN: Extended byte segment neural network for network traffic classification," *IEEE Trans. Depend. Sec. Comput.*, vol. 19, no. 5, pp. 3521–3538, Sep. 2022.
[31] J. Li, S. Wu, H. Zhou, X. Luo, T. Wang, Y. Liu, and X. Ma, "Packet-level open-world app fingerprinting on wireless traffic," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2022, pp. 1–18.
[32] J. Luxemburk and T. Čejka, "Fine-grained TLS services classification with reject option," *Comput. Netw.*, vol. 220, Jan. 2023, Art. no. 109467.
[33] A. Bozorgi, A. Bahramali, F. Rezaei, A. Ghafari, A. Houmansadr, R. Soltani, D. Goeckel, and D. Towsley, "I still know what you did last summer: Inferring sensitive user activities on messaging applications through traffic analysis," *IEEE Trans. Depend. Sec. Comput.*, vol. 20, no. 5, pp. 4135–4153, Sep./Oct. 2023.
[34] Z. Erdenebaatar, R. Alshammari, N. Zincir-Heywood, M. Elsayed, B. Nandy, and N. Seddigh, "Analyzing traffic characteristics of instant messaging applications on Android smartphones," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Miami, FL, USA, May 2023, pp. 1–5.
[35] R. Han, K. Kim, B. Choi, and Y. Jeong, "A study on detection of malicious behavior based on host process data using machine learning," *Appl. Sci.*, vol. 13, no. 7, p. 4097, Mar. 2023.
[36] M. H. Pathmaperuma, Y. Rahulamathavan, S. Dogan, and A. M. Kondoz, "Deep learning for encrypted traffic classification and unknown data detection," *Sensors*, vol. 22, no. 19, p. 7643, Oct. 2022.
[37] A. H. Celdrán, J. von der Assen, K. Moser, P. M. S. Sánchez, G. Bovet, G. M. Pérez, and B. Stiller, "Early detection of cryptojacker malicious behaviors on IoT crowdsensing devices," in *Proc. IEEE/IFIP Netw. Oper. Manage. Symp. (NOMS)*, Miami, FL, USA, May 2023, pp. 1–8.

JEE-TAE PARK was born in Busan, South Korea, in 1993. He received the B.S. degree in computer and information science from Korea University, South Korea, in 2016, where he is currently pursuing the Ph.D. degree (integrated program). His current research interests include internet traffic classification, internet security, and network management.

CHANG-YUI SHIN received the B.S. degree in operating analysis from the Korea Military Academy, Seoul, in 2003, the M.S. degree in electronic computer engineering from Korea University, Seoul, in 2007, and the Ph.D. degree from Korea University, Sejong. Since being commissioned as an Army Officer, in 2003, his service department is Signal in Korea Army. At the time, he had researched the field of mobile ad hoc networks. After that, he became interested in feasible research, such as weapon system planning, interoperability, development quality management, and actual operation while working in various organizations. During the Ph.D. degree, he researches internet traffic classification, network management, and AI.

UI-JUN BAEK was born in Seoul, South Korea, in 1993. He received the B.S. degree in computer and information science from Korea University, South Korea, in 2018, where he is currently pursuing the Ph.D. degree (integrated program). His current research interests include blockchain transaction monitoring, network management, and internet security.



MYUNG-SUP KIM was born in Gyeongju, South Korea, in 1972. He received the B.S., M.S., and Ph.D. degrees in computer science and engineering from POSTECH, South Korea, in 1998, 2000, and 2004, respectively. From September 2004 to August 2006, he was a Postdoctoral Fellow with the Department of Electrical and Computer Engineering, University of Toronto, Canada. Since 2006, he has been a Full Professor with the Department of Computer Convergence Software, Korea University, South Korea. His current research interests include internet traffic monitoring and analysis, service and network management, future internet, and internet security.