Hindawi
Security and Communication Networks
Volume 2018, Article ID 9706706, 24 pages
https://doi.org/10.1155/2018/9706706

Research Article

Abnormal Behavior Detection to Identify Infected Systems Using the *APChain* Algorithm and Behavioral Profiling

Jungwoo Seo and Sangjin Lee

Graduate School of Information Security, Korea University, Republic of Korea

Correspondence should be addressed to Jungwoo Seo; korea002@korea.ac.kr

Received 28 January 2018; Revised 20 May 2018; Accepted 6 August 2018; Published 4 September 2018

Academic Editor: Petros Nicopolitidis

Recent cyber-attacks have used unknown malicious code or advanced attack techniques, such as zero-day attacks, making them extremely difficult to detect using traditional intrusion detection systems. Botnet attacks, for example, are a very sophisticated type of cyber-security threat. Malicious code or vulnerabilities are used to infect endpoints. Systems infected with this malicious code connect a communications channel to a command and control (C&C) server and receive commands to perform attacks on target servers. To effectively protect a corporate network's resources against such threats, we must be able to detect infected systems before an attack occurs. In this paper, an attack pattern chain algorithm (*APChain*) is proposed to identify infected systems in real-time network environments, and a methodology for detecting abnormal behavior through network-based behavioral profiling is explained. *APChain* analyzes the attribute information of real-time network traffic, connects chains over time, and conducts behavioral profiling of different attack types to detect abnormal behavior. The dataset used in the experiment employed real-time traffic accumulated over a period of six months, and the proposed algorithm was developed into a prototype for the experiment. The C&C channel detection accuracy was measured at 0.996, the true positive rate at 1.0, and the false positive rate at 0.003. This study proposes a methodology that can overcome the limitations of conventional security mechanisms and suggests an approach to the detection of abnormal behavior in a real-time network environment.

1. Introduction

According to the 2016 Internet Security Report [1], targeted attacks, such as spear phishing, increased by 55% in 2015 compared to the previous year. Notably, in 2015, attacks using zero-day vulnerabilities increased by 125%. Every year, more than 10 new zero-day vulnerabilities are reported. Moreover, 430 million new pieces of malicious code were discovered in 2015, a 36% increase from the year before. New vulnerabilities include targeted attacks, smartphone threats, social media scams, and Internet of Things (IoT) vulnerabilities.

In the past, attacks tended to extensively infect nonspecific systems with malicious code; however, cyber-attacks that attack targets with specific objectives, such as leaking important information or destroying the system, are becoming more common [2]. Attackers distributing malicious code can control a remote host through a command and control (C&C) channel and connect to backdoor networks to perform its attacks. An effective way to detect hosts that are infected by a botnet is to detect the C&C channel and eliminate the infected system before the attack can be launched. However, a number of major challenges exist in detecting C&C channels. For example, attackers may regularly change the address of the C&C server or use evasive methods such as proxy server traffic redirection. Additionally, because C&C channels use HTTP or HTTPS protocols to communicate, they are difficult to distinguish from general web traffic, making it challenging to establish a definitive countermeasure.

The research in [3, 4] identified botnet C&C channels in an internal network without any prior information. However, as mentioned in [5], because botnet attacks have distinguishable characteristics, an improved detection algorithm is needed [6]. For example, the research in [3] detected C&C channels by checking the active responses from a host group at regular time intervals. However, to communicate with the C&C server, a botnet attempts to make contact in irregular connection cycles, presenting a problem for existing methodologies employed to detect C&C channels. For that

reason, recent studies, including [4, 7–9], have switched their focus to detecting abnormal behaviors of infected systems.

The effective detection of a botnet requires a detailed understanding of the internal network environment and the information service, as well as the configuration of multiple network monitoring environments such as log analysis, file integrity checking, registry monitoring, and rootkit detection. However, when configuring a host-based and network-based consolidated monitoring environment in a corporate network environment, resource utilization and performance limitations are typically encountered. Setting up many rule sets and excessive anomaly detection protocols to identify botnets in a large volume network environment will increase resource inefficiency and, in some cases, threaten the operability of the internal network. Because of these problems, companies limit the rule sets for their information security system or shut down their detection function. Therefore, understanding the characteristics of a botnet, improving resource efficiency, and providing stable detection performance are important elements of any response to intrusions.

In this paper, in order to detect botnets, the attribute information of network traffic is used to construct an attack pattern chain algorithm (*APChain*) over time, and behavioral profiling is conducted to detect abnormal activity. With this method, real-time network traffic analysis, optimal resource utilization, and encrypted packet attack detection are possible.

The remainder of the paper is structured as follows. Section 2 reviews related work and current challenges, while Section 3 presents a conceptual overview of the proposed approach. Section 4 describes the system model for the proposed algorithm's architecture. Section 5 presents an experimental evaluation of the approach using a real-time traffic dataset. Finally, conclusions are drawn in Section 6.

2. Related Work

Botnets have become a major threat on the Internet and extensive research has been conducted over the past several years to detect them. Botnet detection can be classified into two main types: vertical correlation [10] and horizontal correlation [3, 11].

BotHunter [10] is an example of detection based on vertical correlation. It observes a single machine and compares its behavior with a model of bot behavior. It recognizes correlated dialog trails consisting of multiple stages and representing successful bot infection. Therefore, this strategy is also referred to as "dialog correlation." BotHunter is designed to track two-way communication flows between internal assets and external entities and consists of a correlation engine driven by several malware-focused network detection sensors. The BotHunter correlator links the dialog trail of inbound intrusion alarms with those outbound communication patterns to detect infected local hosts. When a sequence of evidence matches BotHunter's infection dialog model, a report is produced that captures all of the events relevant to the local host's role in the infection process.

BotHunter has some important limitations. For example, it is restricted to the life cycle of the predefined infection model, and some stages, such as C&C communication, provide only signature-based sensors. Thus, BotSniffer and BotMiner are often used to complement BotHunter. They do not necessarily require the observation of multiple different stages within an individual host, nor do they require botnet-specific signatures.

BotSniffer [3] and BotMiner [11] operate on the principle of horizontal correlation by observing correlations and similarity across multiple hosts. Because horizontal correlation detection strategies conduct preprogrammed activities related to the C&C channel under the control of a botmaster, bots within the same botnet will exhibit spatial-temporal correlation and similarity. BotSniffer is designed to primarily detect centralized C&C channels, and it monitors multiple rounds of spatial-temporal correlation and the similarity in activity responses from a group of hosts that share a common centralized server connection such as IRC or HTTP. BotSniffer can achieve the theoretical bounds for false positive and false negative rates within a reasonable detection time using statistical algorithms. BotMiner presents a more general detection framework that is independent of botnet C&C protocols and structure. It clusters together similar communication traffic and similar malicious traffic and performs cross-cluster correlation to identify hosts that share similar communication and malicious activity patterns. Therefore, these hosts are considered bots within the monitored network. BotSniffer has an important limitation in that it is restricted to the detection of botnets that primarily use centralized C&C channels. With horizontal correlation, it is difficult to detect a botnet in a small network environment and to classify hosts that are infected with different characteristics of the botnet within the same network range. In addition, BotHunter, BotSniffer, and BotMiner all usually require a relatively long time to observe multiple stages of botnet communication.

Other strategies for botnet detection have also been proposed. In [12], specific traffic information is extracted to run a learning module that can detect a C&C channel. NetFlow's records (flow size, client access patterns, and temporal behavior) are used to identify the characteristics of the traffic; using the learning module, reference values are created to match the C&C channel characteristics. A detection module then conducts matching to detect the botnet C&C channel. In [13], attacks are detected by analyzing ordinary HTTP requests and C&C channel characteristics to detect HTTP-based C&C channels. To precisely differentiate between C&C and legal domains, a CODDs-defined approach is proposed. The proposed algorithm analyzes the DNS information requests corresponding to these domains during a particular time window.

In this paper, the *APChain* algorithm is proposed to detect abnormal behavior. Abnormal behavior traffic is detected in the initial stages with the goal of quickly eliminating systems infected by malware. We configure port mirroring on the backbone switch to create *APChain* in a real-time network. Over a period of time, attribute information is linked to the chain, and the results from *APChain* are then used to conduct behavioral profiling to detect abnormal behavior.

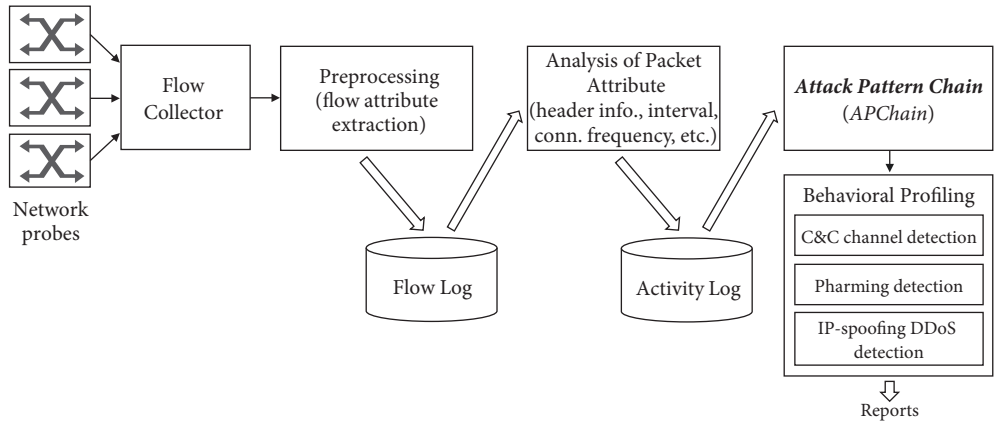


Figure 1: Conceptual diagram of the proposed methodology.

This study makes three main contributions. First, this paper proposes an approach that differs from traditional methods of botnet detection. The proposed methodology constructs *APChain* using traffic attribute information and detects abnormal behavior such as botnets through behavioral profiling. Our system is able to detect new types of botnet and can improve its accuracy by modifying its behavioral profiling algorithm. Second, botnets can have very flexible C&C channels. They can use different protocols such as IRC and HTTP and can encrypt content for C&C communication [3, 10, 11, 14]. This paper proposes a detection scheme that groups hosts with similar behavior over time. We are able to determine the IP address of the C&C server, the connection time, and connection count, among other parameters. As a result, the proposed approach detects variable C&C channels by tracking communication with the C&C server. Finally, current detection techniques are based on the inspection of network traffic. However, recent malware uses encrypted C&C traffic or code obfuscation to evade these detection techniques. The proposed methodology utilizes the attribute information of the protocol header to overcome problems associated with analyzing the payload of network traffic. Therefore, it is possible to detect encrypted C&C channels and to detect the abnormal behavior of obfuscated packets.

3. System Overview

In this section, we investigate the overall concept of the proposed methodology, the configuration of *APChain* in detecting abnormal behavior, and the behavioral profiling process.

3.1. Overview of the Proposed Methodology. Recent cyberattacks are initiated by using advanced social engineering or by sending a targeted e-mail to attack targets [15]. If a host is infected by malicious code, an attempt to communicate with a C&C server is made and a communications channel is formed using Internet Remote Chat (IRC) or an HTTP protocol. When a communications channel is established, the host receives an attack command from the C&C server or updates the binary file. At this time, *APChain* is configured to monitor real-time traffic and detect abnormal behavior, followed by behavioral profiling.

The methodology proposed in this paper is presented in Figure 1. It consists of a five-step process to detect abnormal behavior. First, real-time traffic is collected by the flow collector and attribute information from the collected traffic is extracted. Second, this traffic attribute information is analyzed in order to configure *APChain*. *APChain*'s role is to connect the features of the traffic attribute information into a chain and to match this to abnormal behavior. Third, behavioral profiling based on *APChain* is conducted. Fourth, suspicious patterns are categorized and abnormal behavior is detected. Finally, intrusion responses against the abnormal behavior are executed, and network forensics for an audit trail are conducted.

3.2. Attack Pattern Chain Algorithm (APChain). In order to detect abnormal behavior, attribute information is extracted from the DMZ or the internal backbone switch, and a chain is configured that contains the attribute information over time.

3.2.1. Attribute Information for APChain. Network traffic headers contain a variety of information such as the IP addresses and protocols of the origin and destination. The standardized extraction defined by the communication protocol enables efficient analysis. An 8-tuple {source IP, source port, destination IP, destination port, protocol, access time, MAC, URL}, which contains the fundamental attribute information required to create *APChain*, is extracted from the traffic header and payload, and a 10-tuple {*SN*, *SGN*, *TIN*, *CND*, *RATD*, *CNTI*, *RATD*, *CNTI*, *ACTTI*, *SCTTI*} is created, which provides additional attribute information based on the extracted 8-tuple. The 8-tuple defines the network connection between the origin and the destination systems, and the 10-tuple is utilized as analysis data for the detection of a host's abnormal behavior.

3.2.2. Execution Results of APChain. The 8-tuple {source IP, source port, destination IP, destination port, protocol, time, MAC, URL}, collected through the network switch of the experimental environment depicted in Figure 10, is

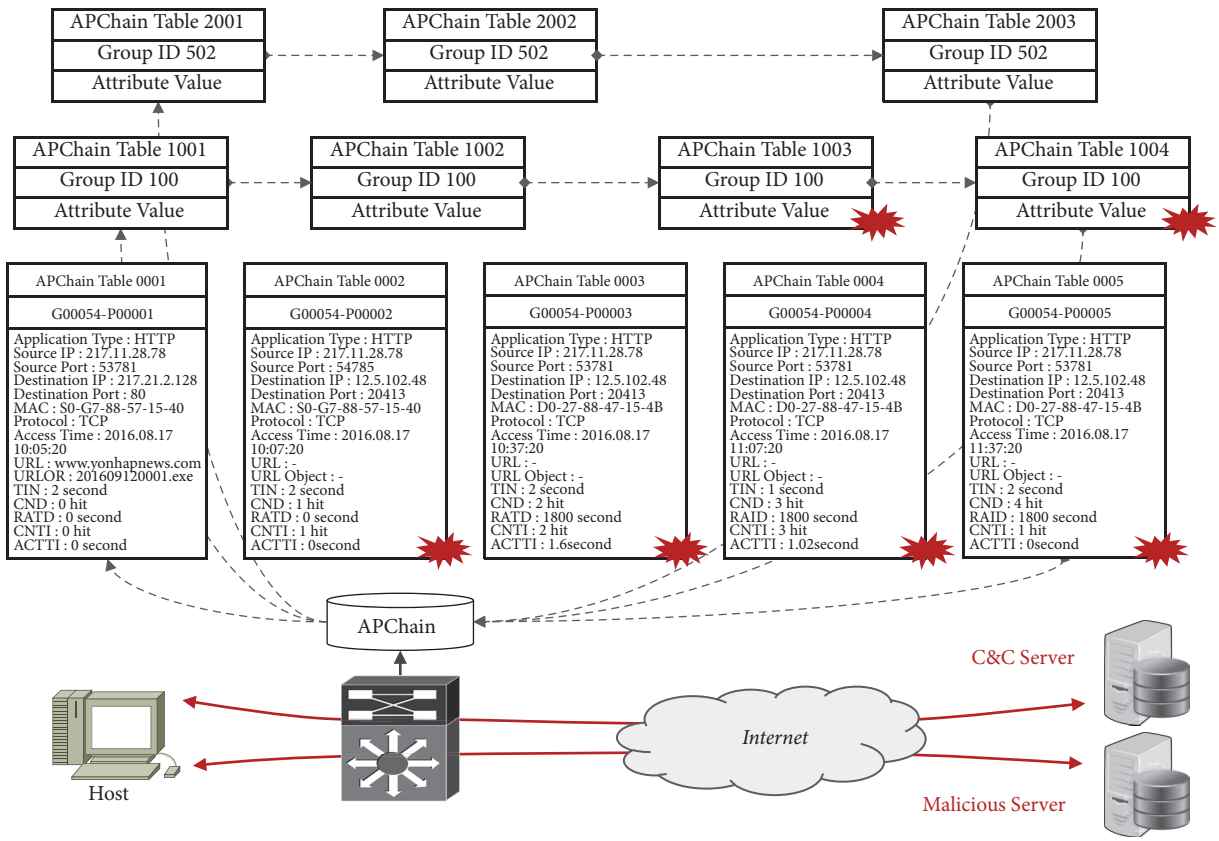


Figure 2: Conceptual diagram of *APChain* configuration.

used as input to create *APChain*. The results using the *APChain* algorithm in Table 3 are expressed as Output=Input ∪ {*SN*, *SGN*, *TIN*, *CND*, *RATD*, *CNTI*, *RATD*, *CNTI*, *ACTTI*, *SCTTI*}. Table 1 presents the *APChain* table created using the traffic attribute information.

Figure 2 shows an example of *APChain* configuration for the abnormal traffic behavior of a host infected by malicious code.

For example, the intrusion server (217.11.xxx.78) connects to the malicious server (12.5.xxx.48), which distributes malicious code, infecting the host. The infected host periodically communicates with the C&C server (106.23.xxx.129) and updates the binary malicious code or receives an attack command. At this time, to configure *APChain* to detect abnormal behavior, port mirroring is configured on the backbone switch, and network traffic attribute information is extracted to create *APChain*. This traffic attribute information is utilized to calculate additional attributes and added to *APChain*.

When connections are made to a web server with the same target address, they are defined in *APChain* by the same group name, "GXXX53-PXXXX," and, after they are interconnected in the chain, behavioral profiling is conducted to detect abnormal behavior.

3.3. Behavioral Profiling. Behavioral profiling analyzes attack types based on their characteristics and categorizes any traffic that exhibits abnormal behavior. Table 2 shows the behavioral profiling algorithm using *APChain*.

The behavioral profiling algorithm in Table 2 consists of case studies for three representative attack types. The first case analyzes the *RATD*, *CNTI*, *ACTTI*, and *SCTTI* field values of *APChain* and confirms whether there is communication with a C&C server to detect a C&C channel. The second case analyzes the *URL*, *RATD*, and *CNTI* fields of *APChain* and checks for website tampering in order to detect pharming attacks. The third case analyzes the *RATD*, *CNTI*, *ACTTI*, and *MAC* fields of *APChain* and checks for IP-spoofing by calculating the traffic frequency in order to detect IP-spoofing DDoS botnets. Figure 3 presents the abnormal behavior detection process of *APChain*-based behavioral profiling in a real-time network environment.

3.4. Elimination of Whitelist-Based False Positives. In this paper, we target the early detection of abnormal behavior using *APChain* and behavioral profiling. However, because the proposed methodology utilizes the attribute information of network traffic, false positives may occur when normal traffic is included in the detection results. For this reason, the proposed algorithm exchanges data with the external system or categorizes normal traffic such as web service calls based on a whitelist and eliminates them from the analysis [16, 17].

The IP addresses registered on the whitelist are classified into two types. The first type is those that are part of the *Internal Whitelist,* which has packet attributes (e.g., Sip, Dip,

Table 1: *APChain* table.

Attribute	Type	Description
SN	*Varchar*	*Sequence number*
SGN	*Varchar*	*Sequence group number with the same destination IP*
Sip	*Varchar*	*Source IP*
Sport	*Integer*	*Source port*
Dip	*Varchar*	*Destination IP*
Dport	*Integer*	*Destination port*
MAC	*Varchar*	*Media Access Control address*
PT	*Varchar*	*Protocol*
TS	*Date*	*Time stamp*
URL	*Varchar*	*Uniform Resource Locator address*
TIN	*Integer*	*TIN subtracts the previous access time from the current access time*
CND	*Integer*	*CND is the cumulative number of times a connection is made to the same target IP address*
RATD	*Integer*	*RATD is the connection time interval to the same target IP address*
CNTI	*Integer*	*CNTI is the cumulative number of times a connection is made to the same target IP address over 30 minutes*
ACTTI	*Integer*	*ACTTI is the average connection time interval to the same target IP address over 30 minutes*
SCTTI	*Integer*	*SCTTI is the standard deviation of the connection time interval to the same target IP address over 30 minutes*

Table 2: Behavioral profiling algorithm.

Algorithm 1. Behavioral Profiling

Input: Result of Function ***APChain*** **(T)**: where T is a collection of network traffic.
Output: Results of abnormal behavior
Function Behavioral_Profiling (T):
$H = \{h_1, \cdots, h_n, h_m\}$, where h is the host infected by malware
$D = \{d_1, \cdots, d_n, d_m\}$, where d represents destination servers
$R \longleftarrow$ *RATD* field value of *APChain*
$C \longleftarrow$ *CNTI* field value of *APChain*
$A \longleftarrow$ *ACTTI* field value of *APChain*
$S \longleftarrow$ *SCTTI* field value of *APChain*
while (not stop condition) **do**
  if Abnormal_Behavior *C&C* **then**
    $M = \{m_1, \cdots, m_n, m_m\}$, where m is the C&C server
    /*the host attempts to connect to the C&C server*/
    interval$(H \longrightarrow M) \in (A \pm S)$
    frequency$(H \longrightarrow M) > C$
  if Abnormal_Behavior (*Pharming*) **then**
    $M = \{m_1, \cdots, m_n, m_m\}$, where m is the fake website
    /*the host connects to the fake website*/
    interval$(H \longrightarrow M) < \theta_i$
    frequency$(H \longrightarrow M) > \theta_j$
    $(d_{n.Dip} = d_{m.Dip}) \cap (d_{n.url} \neq d_{m.url})$
  if Abnormal_Behavior (*DDoS*) **then**
    $M = \{m_1, \cdots, m_n, m_m\}$, where m is the victim system
    /*the host executes a DDoS attack*/
    interval$(H \longrightarrow M) \cong 0$
    frequency$(H \longrightarrow M) > \theta_m$
  if Abnormal_Behavior (*IP-spoofed DDoS*) **then**
    $(h_{n.mac} = h_{m.mac}) \cap (h_{n.Sip} \neq h_{m.Sip})$
      frequency$(H \longrightarrow M) > \theta_n$

Table 3: Attack pattern chain (*APChain*) algorithm.

Algorithm 2. Attack pattern chain (*APChain*)

/* T ⟵ set of packets */

Function APChain (T):
While (not stop condition) **do**
C ⟵ ∅ /* Array for record of *APChain* */
for T is not Ø **do**
$C_{(n).sn}$ ⟵ *sequence number* /* *S00001* */
$C_{(n).sgn}$ ⟵ *sequence group number* /* *G00001* */
$C_{(n).Sip}$ ⟵ *source IP* /* *58.203.xxx.xxx* */
$C_{(n).Sport}$ ⟵ *source port* /* *port number* */
$C_{(n).Dip}$ ⟵ *destination IP* /* *211.106.xxx.xxx*x*/
$C_{(n).Dport}$ ⟵ *destination port* /* *port number* */
$C_{(n).mac}$ ⟵ *MAC address* /* *D0-27-88-47-15-4B* */
$C_{(n).ts}$ ⟵ *access time* /* *2016.08.12 07:05:28* */
$C_{(n).url}$ ⟵ *URL* /* *www.cnn.com* */
$C_{(n).tin}$ ⟵ $(C_{(n).at} - C_{(n-1).at})$ /* the cycle interval between the previous packet and the current packet */
$C_{(n).cnd}$ ⟵ *Call Function CND*(C, T)
$C_{(n).ratd}$ ⟵ *Call Function RATD*(C, T)
$C_{(n).cnti}$ ⟵ *Call Function CNTI*(C, T)
$C_{(n).actti}$ ⟵ *Call Function ACTTI*(C, T)
$C_{(n).sctti}$ ⟵ $Sqrt((C_{(n).ratd} - C_{(n).actti})^2/C_{(n).cnti})$

Function CND (C,T):
for T is not null **do**
if $(C_{(n).srcIP} = C_{(n-1).srcIP}) \cap (C_{(n).dstIP} = C_{(n-1).dstIP})$ **then**
accVal + = 1
return *accVal*

Function RATD (C,T):
if $C_{(n).srcIP} = C_{(n-1).srcIP} \cap C_{(n).dstIP} = C_{(n-1).dstIP}$ **then**
interVal = $C_{(n).at} - C_{(n-m).at}$
return *interVal*

Function CNTI (C,T):
for T is not null $\cap (C_{(n).at} - C_{(n-m).at}) \leq 30$(minute) **do**
if $C_{(n).srcIP} = C_{(n-1).srcIP} \cap C_{(n).dstIP} = C_{(n-1).dstIP}$ **then**
accVal + = 1
return *accVal*

Function ACTTI (C,T):
for T is not null $\cap (C_{(n).at} - C_{(n-m).at}) \leq 30$(minute) **do**
if $C_{(n).srcIP} = C_{(n-1).srcIP} \cap C_{(n).dstIP} = C_{(n-1).dstIP}$ **then**
interVal + = $C_{(n).at} - C_{(n-m).at}$
accVal + = 1
return *interVal*/*accVal*

and interval) regularly exchanged between the internal and external environments. When batch scripts such as crontab are used in the communication with the external environment for data exchange, the *Internal Whitelist* compares and analyzes the IP address, port, and execution cycle and saves the analysis in a file.

However, it is difficult to detect abnormal behavior when it is registered with the *Internal Whitelist* because it is

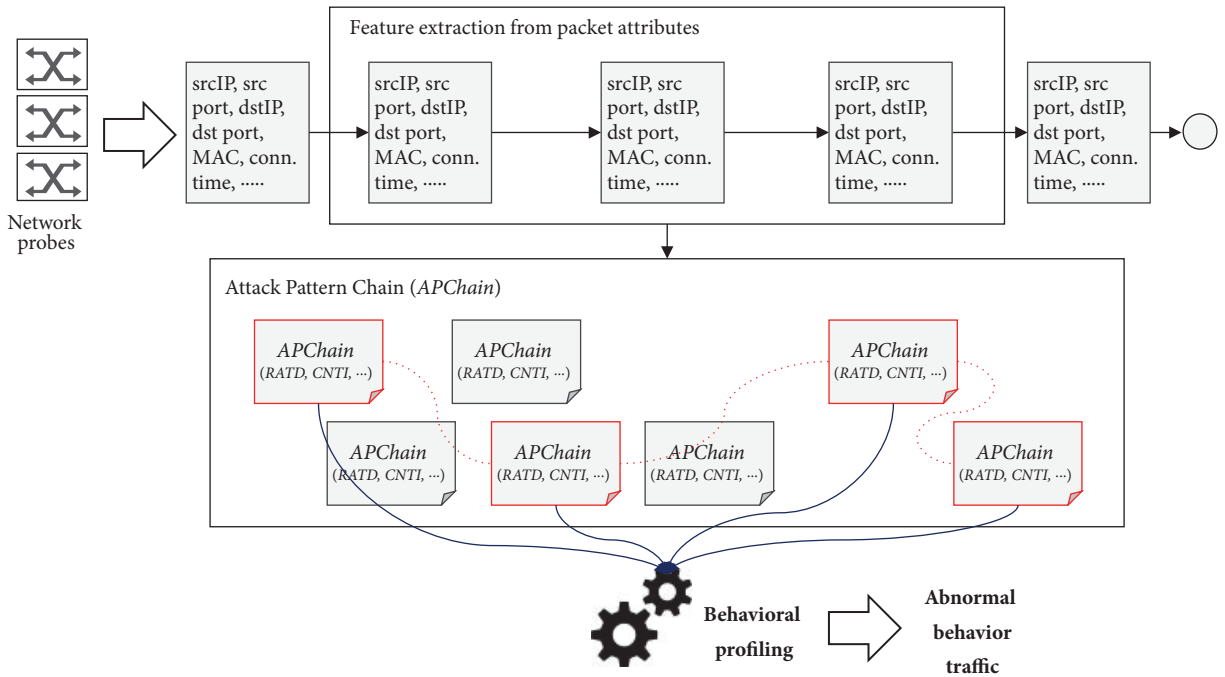


Figure 3: Abnormal behavior detection using *APChain* and behavioral profiling.

categorized as trustworthy communication. Therefore, the condition of (1) is periodically checked to detect the abnormal behavior of the hosts registered in the *Internal Whitelist*; any abnormal behavior detected will be excluded from the *Internal Whitelist*.

$$\begin{aligned} &if \quad interval\left(Server_{src} \longleftrightarrow Server_{dst}\right) \equiv \theta \\ &then \quad normal\ network\ traffic \\ &otherwise \quad abnormal\ network\ traffic \end{aligned} \tag{1}$$

The second type of IP address is those associated with sites on well-known *Global Whitelists,* such as those compiled by antivirus software companies. Through the continuous updating of the *Global Whitelist*, new C&C IP addresses are added, and IP addresses and domain names are included. Figure 4 outlines the elimination of whitelist-based normal IP addresses from the behavior profiling process.

The elimination process for whitelist-based false positives is shown in Figure 4. The traffic from January to June 2017 collected in the experimental environment presented in Figure 10 is analyzed and whitelist-based IP addresses involved in normal communication are eliminated from the analysis.

3.5. Characteristics of C&C Channels and Their Detection Method

3.5.1. Characteristics of C&C Channels. Attackers use either an encrypted communications channel or an alternative communication method to hide C&C channels. When hosts infected with malicious code establish an Internet-enabled communications environment, a channel is created to communicate with a C&C server and, through this, the infected host receives an attack command from the C&C server or extracts vital data from the host on its internal network [7, 18–20]. A C&C server can have at least 1 and up to N number of C&C channels with hosts and involves repeated connection and standby requests. Because the communication cycles of C&C channels differ depending on the characteristics of the malicious code, it is impossible to detect all C&C channels in categorized regular time intervals. Even though a C&C channel may exist between a host and a C&C server according to the configuration conditions, this does not mean that communication will always occur on a regular cycle. Moreover, a host infected with malicious code downloads the IP address of a new C&C server from the original C&C server and creates a communication channel with this new server. Figure 5 shows the connection and standby cycles of a C&C channel connection to a Linux/Xor.DDoS botnet.

3.5.2. C&C Channel Detection Method. Analysis of the real-time traffic occurring in the experimental environment allowed us to categorize C&C channels into three types. The first type either engages in socket communication with the external server via the Internet to exchange data or uses a file transfer program such as the File Transfer Protocol (FTP) to exchange data with anonymous or authenticated users. In instances such as this, in which data is exchanged with the external system, a high connection frequency is observed at regular time intervals. The second type is when a connection is made to the website through browsers to surf the web. The host connects to a specific website and regularly requests a service or requests a random service from an unspecified website. In the case where a service is requested via a website, the connection frequency has significant variation at random time intervals. The third

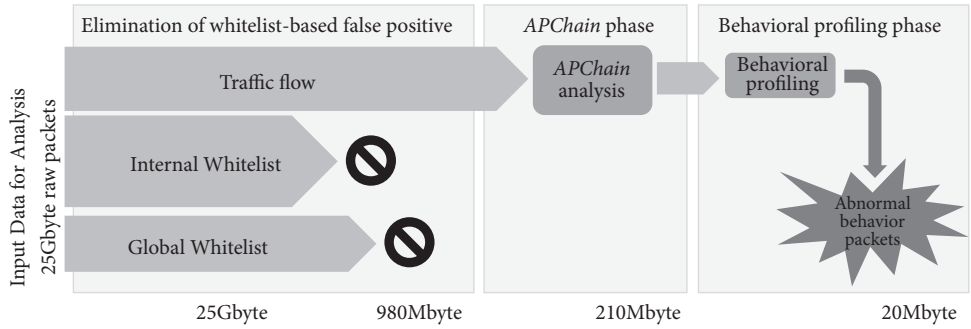


Figure 4: Elimination of whitelist-based false positives.

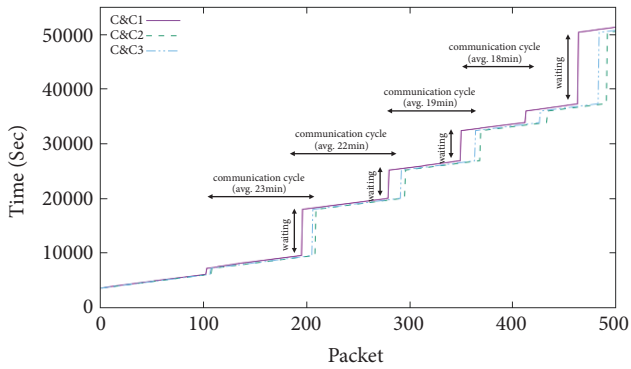


Figure 5: Characteristics of a Linux/Xor.DDoS botnet C&C channel.

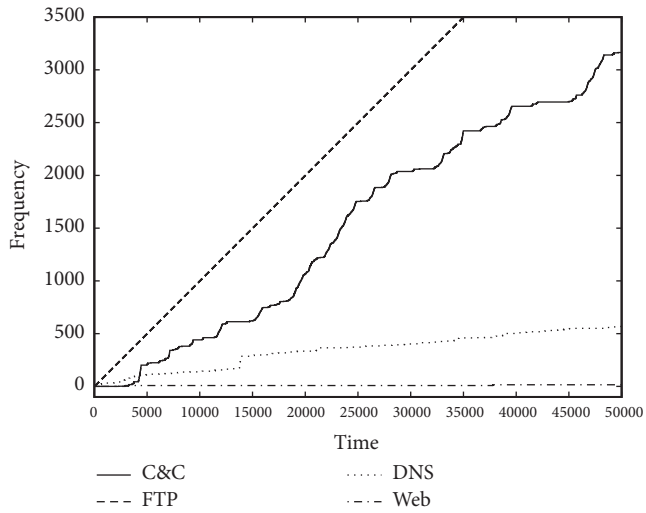


Figure 6: Analysis of network flow characteristics over time.

type is when a host is infected by malicious code upon connecting to a botnet or malicious code distribution site and attempts to communicate with an external network. The infected host configures a communication channel with the C&C server and either receives an attack command or updates the configuration file. When communication with a C&C server occurs, the connection cycle varies according to the characteristics of the malicious code and exhibits a high connection frequency.

Further analysis of the network traffic characteristics over time confirms that, according to the traffic type, a difference occurs between the connection frequency and connection time intervals. Figure 6 presents the connection frequency and intervals for three types of traffic: a C&C channel used to communicate with a C&C server, communication with an external file exchange system to exchange data, and website service requests made upon connecting to a website. If it were possible to analyze the traffic collected in real time, as was done in the experimental environment, it would be possible to detect attacks that have singularities such as those of the C&C channel.

In this paper, the following attributes are used to analyze the characteristics of a C&C channel and its connection frequency to configure *APChain*: *CND*, *RATD*, *ACTTI*, and *SCTTI*. The time interval for the analysis of the traffic attributes is set to 30 minutes. The reason for this is because the average lead time for a host infected by malicious code to perform malicious activity is three hours. Therefore, the goal is to detect abnormal behavior using the proposed algorithm before the infected host can initiate malicious activities.

3.6. Characteristics of Pharming. Pharming is a type of cyber-attack in which specific domain names such as a cache DNS server are configured to use a forged IP address. Users input a normal website address to request a web service but are instead connected to a fake site created by the attacker. The fake site has the same form as the normal website and obtains personal information such as a user's login credentials and account information. Therefore, an effective method to detect pharming attacks is to identify real websites and detect whether the DNS cache information has been falsified or not.

The proposed methodology analyzes the traffic attribute information over time and categorizes pharming attacks. For example, when a host is infected by a pharming attack, even if it were to request a normal website, as in Figure 7, a fake website is delivered with a falsified domain name and, as (2) shows, the number of calls to the fake website increases at a specific point in time.

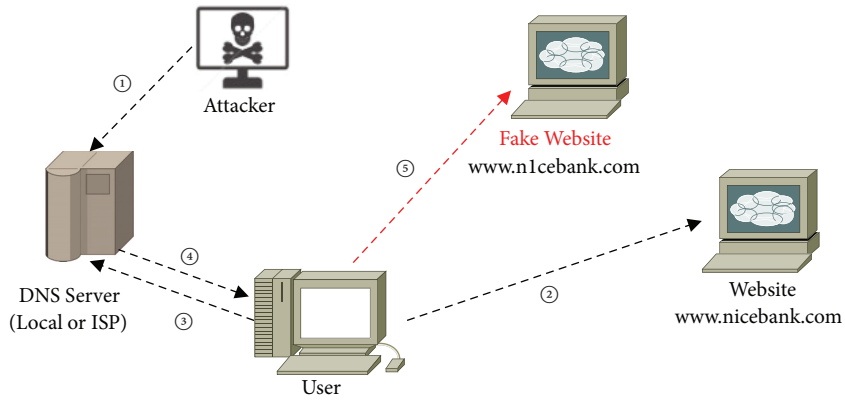


Figure 7: Flowchart of a pharming attack.

$$H = H_1, H_2, H_3$$
$$(H_1 = kbank.com, H_2 = hbank.com, H_3 = bbank.com) \quad (2)$$
$$P = Pharming\ Site$$
$$P \longleftarrow H$$

In this paper, to detect pharming attacks, the URL and IP address of the website the host is connecting to and the connection time are analyzed to configure *APChain*. *APChain*'s *URL*, *Dip*, *CND*, *RATD*, and *CNTI* are used as key elements in behavioral profiling to detect pharming attacks.

3.7. Characteristics of IP-Spoofing DDoS Botnets. Because of the increase in network bandwidth and the development of hardware, recent DDoS attacks are generating larger volumes of traffic incomparable to what has been seen in the past [21, 22]. Malicious code such as IP-spoofing DDoS botnets Linux.Shelldos and Linux.Xor.DDoS receives attack commands from a C&C server and generates large volumes of packets threatening to paralyze communications.

Although security systems such as firewalls are set up to detect and block this type of attack, if a host infected by malicious code were to manifest large numbers of DDoS attacks, then even if a security system were to detect the DDoS attacks and block them, the internal network could be paralyzed due to the large volume of traffic. Therefore, if a host infected by an IP-spoofing DDoS attack is not detected and eliminated quickly, the internal network will be subject to the threat for a very long period of time.

Figure 8(a) shows that the internal network is affected by a large amount of traffic originating from a host infected with a DDoS botnet. Figure 8(b) presents the traffic from the internal network using PRTG software. The infected host generates a large volume of traffic from the host to the external server and failures occur on the internal network. Figure 9 shows the packets generated from a host infected by an IP-spoofing DDoS botnet.

In this paper, to detect IP-spoofing DDoS botnets, the attributes of outbound traffic are analyzed, and the following values are used to configure *APChain*: *Sip*, *Sport*, *MAC*, *CND*, *RATD*, *CNTI*, *ACTTI*, and *SCTTI*. Additionally, behavioral profiling is used to confirm whether IP-spoofing has occurred or not and to identify the host IP infected by the botnet.

4. System Model

In this section, we outline the proposed algorithm for detecting abnormal behavior and examine the abnormal behavior detection methods according to attack type in the form of case studies.

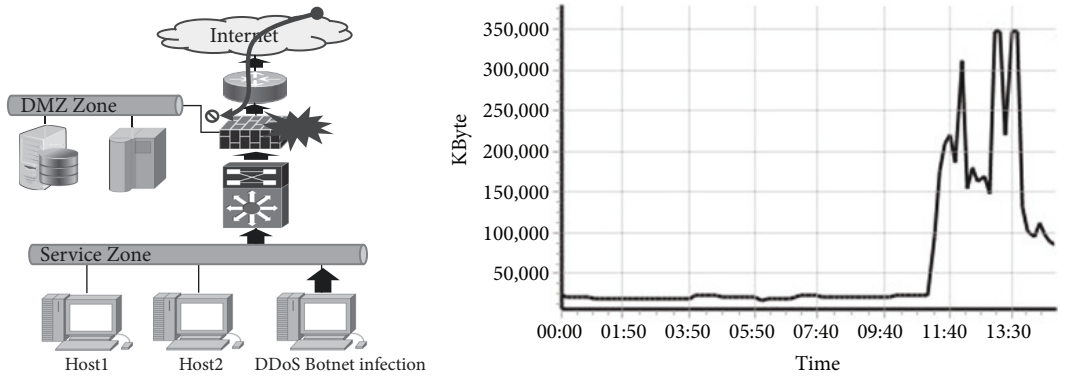
4.1. Collection of Network Traffic. Port mirroring is configured to the backbone switch for the collection of real-time network traffic. Port mirroring is set up to collect all of the traffic routed from the backbone switch to analyze the attribute information. Figure 10 presents a diagram outlining the collection of real-time network traffic.

4.2. Extraction of Attribute Information. Attribute information extracted from real-time traffic in the experimental environment presented in Figure 10 is stored in the flow log database (FLDB). The FLDB stores the features of the attributes extracted from the traffic headers, both normal and malicious traffic (e.g., Trojans, botnets, etc.) is included.

The collection of network traffic collected at the backbone switch is represented as $T=\{t_1, \cdots, t_{n-1}, t_n\}$, and the traffic attribute information is defined as $t=\{t_{ip}, t_{port}, t_{time}, t_{mac}, \cdots, t_{url}\}$. The size of the attribute information stored in the FLDB is $F = \sum_{i=1}^{n} 1$ and is cumulatively calculated if new attribute information is stored in the FLDB. The protocol stack for extracting packet attribute information is shown in Figure 11.

4.3. Attack Pattern Chain (APChain) Creation. The *APChain* algorithm configures a chain of traffic attribute information over time. Behavioral profiling using *APChain* is then conducted to detect abnormal behavior. Table 3 shows the algorithm used to configure *APChain* using traffic attribute information extracted from the protocol stack (Figure 11).

The source IP, source port, destination IP, destination port, access time, and *MAC* value are extracted from the network traffic, and the URL information from the payload is analyzed to configure *APChain*. Additionally, the connection



(a) IP-spoofing DDoS traffic flowchart (b) Network bandwidth usage

Figure 8: Analysis of DDoS characteristics.

Attack Name	Date	Src IP	Src Port	Dst IP	Dst Port	MAC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.73	TCP/5761	59.36.97.66	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.102	TCP/46000	59.36.97.21	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.107	TCP/63500	59.36.97.66	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.248	TCP/4189	59.36.97.46	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.174	TCP/17611	59.36.97.24	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.104	TCP/63449	59.36.97.62	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.186	TCP/37897	59.36.97.62	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.196	TCP/51217	59.36.97.46	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.216	TCP/49909	59.36.97.21	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.66	TCP/64213	59.36.97.46	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.115	TCP/5997	59.36.97.66	TCP/3389	2C-44-FD-7D-B2-AC
tcp syn flooding	2016/01/19 11:17:45	203.2 7.66.233	TCP/23442	59.36.97.66	TCP/3389	2C-44-FD-7D-B2-AC

Figure 9: Traffic of a host infected by an IP-spoofing DDoS botnet.

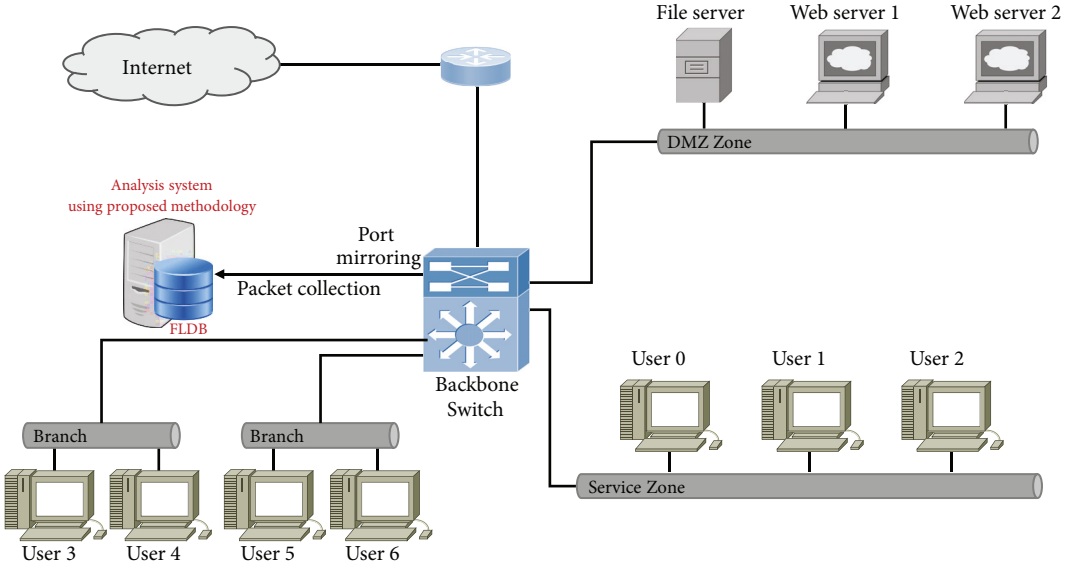


Figure 10: Diagram of the experimental environment.

time and connection frequency of traffic that have the same target IP address are calculated and stored within *APChain*.

APChain record is 102 Bytes in size, and the same number of *APChain* records is created as the number of transfers inbound to outbound. Figure 12 shows the results of *APChain* creation using attribution information from the collected traffic.

4.4. Abnormal Behavior Detection Using Behavioral Profiling. In this section, we will review the process and algorithm for behavior profiling using *APChain* and investigate the detection of abnormal behavior using three case studies.

4.4.1. C&C Channel Detection [Case Study A]. Hosts are infected with malicious code by malicious code distribution

Table 4: The hypothesis for C&C channel detection.

Hypothesis 1. C&C channel detection
Given an environment:
Let $T = \{t_1, \cdots, t_{n-1}, t_n\}$, where t represents the network traffic, and T_n is the traffic currently being analyzed.
Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h represents a host infected by malicious code.
Let $S = \{s_1, \cdots, s_{n-1}, s_n\}$, where s represents a C&C server.
$H \Longrightarrow S$, the infected host attempts to connect to the C&C server.
A host infected with a botnet creates a C&C channel in order to communicate periodically with the C&C server. As a result, the frequency that a host connects to a particular system increases, and if this pattern is repeated often enough, it can be considered to be unusual traffic. At this point, the host and the C&C server receive attack commands or update binary files while repeating the connection requests and responses.
Therefore, the set (G) of C&C channels can be formed by detecting and grouping the requests from hosts connecting to the C&C server.
$G = \{H_1 \Longrightarrow S_1, H_2 \Longrightarrow S_2, \cdots, H_{n-1} \Longrightarrow S_{n-1}, H_n \Longrightarrow S_n\}$

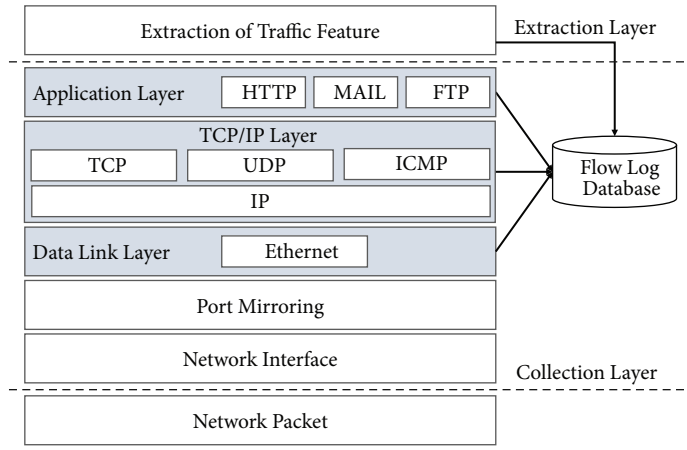


Figure 11: Protocol stack for feature vector extraction.

sites, phishing emails, or social networks. Infected hosts attempt to connect to receive an attack command from a C&C server, to update the C&C server list, or to update the binary file. At this time, the C&C channel uses random ports higher than the known ports and configures a channel with at least one C&C server. Therefore, one method to effectively block botnet attacks is to detect the communication channel with the C&C server and eliminate it before the host infected by malicious code can manifest an attack. The hypothesis that we established for C&C channel detection is presented in Table 4. C&C channel detection results are verified in the experiments.

The commands used to trigger communication between the host and the C&C server are included in a botnet, and the communication method for the creation of the C&C channel and connection cycle are configured. Therefore, if we can detect hosts infected by malicious code connecting to a C&C server, then we can block the attack before the actual attack can be manifested.

In this paper, the following process is followed to detect a C&C channel between a host infected by a botnet and a C&C server. First, it confirms if the traffic to be analyzed contains an IP address included as part of the whitelist-based false positive elimination process. If the IP address is included on the list, it is categorized as normal traffic, and if not, it is analyzed to determine abnormal behavior. Second, it calculates the frequency (*CNTI*) of the host connecting to the target system and compares it with the threshold value. If the *CNTI* value is higher than the threshold value, then it concludes that communications are occurring regularly and moves on to the next step. Third, it compares the connection time interval (*RATD*) for traffic that has the same target IP address with the average connection time interval calculated every 30 minutes. At this time, the standard deviation of the average connection time interval (*SCTTI*) is also calculated, and if the connection time interval (*RATD*) is $\{(ACTTI - SCTTI) \pm \theta_i \leq (RATD) \geq (ACTTI + SCTTI) \pm \theta_j\}$, then the corresponding traffic is suspected to be a C&C channel. Figure 13 displays the distribution of the connection frequencies and connection cycles of traffic suspected to be C&C channels. Finally, if the connection frequency of a bot to a suspected C&C channel exceeds the threshold value, it is categorized as abnormal traffic behavior.

Normal packets and abnormal packets can be categorized by analyzing the network traffic collected in the experimental environment shown in Figure 10, and traffic suspected as being a C&C channel can be categorized according to the hypothesis defined in Table 4.

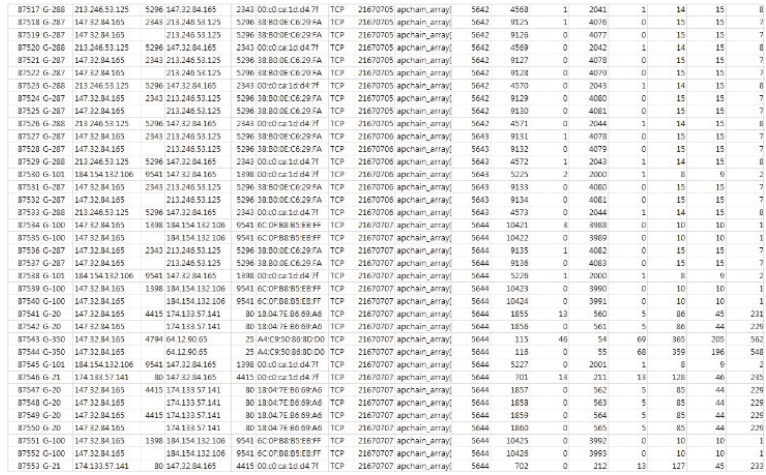


Figure 12: Results of the execution of the *APChain* algorithm.

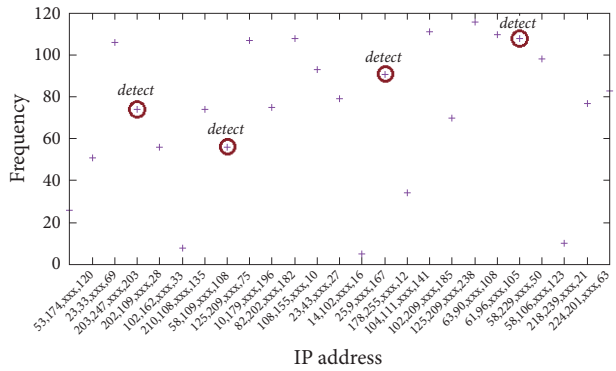


Figure 13: Categorization of abnormal traffic behavior suspected of being a C&C channel.

Figure 13 presents the connection frequency between a host and a suspected C&C channel. If *APChain*'s *ACTTI* is *A* and *SCTTI* is *S* with *APChain* connection frequency as *x*, then, for the traffic suspected as being a C&C channel, *C* is $C=\{x \mid \exists x, (A-S) \pm \theta_i \leq x \geq (A+S) \pm \theta_j\}$. Here, *ACTTI* is used as the reference value for C&C channel detection. A fixed value is not used; it varies according to the connection cycle of the target IP address. Table 5 presents the algorithm used to detect a C&C channel. The proposed algorithm was developed into a prototype, and its performance is verified in the experiment and evaluation sections of this paper.

4.4.2. Pharming Attack Detection [Case Study B]. Pharming attacks target specific individuals or organizations and infect them via spear phishing or malicious code distribution sites. Pharming attacks falsify Windows environment files or DNS addresses, so even if a host infected by pharming were to call a fake site, the intrusion response system would categorize the corresponding traffic as normal. The hypothesis for detecting a pharming attack is shown in Table 6.

The algorithm to detect a pharming attack proposed in this paper analyzes whether the site that the host connects to is fake or not. For example, when the host attempts to connect to a website with a specific IP address through different domain names, it can be suspected of being a pharming attack. The average time interval of the connection to a fake website is *RATD*, and the frequency is *CNTI*. When the threshold is *y*, traffic suspected as pharming, *P*, is represented as $P = \{x \mid x \in threshold(\mathrm{y}), (RATD \leq x) \cap (CNTI \geq x)\}$. Table 7 presents the algorithm used to detect a pharming attack through behavioral profiling.

4.4.3. IP-Spoofing DDoS Botnet Detection [Case Study C]. As advances in network infrastructure and hardware have progressed, the large volume of traffic generated from a small number of systems has become the main threat to the stability of an internal network. In particular, if malicious

Table 5: C&C channel detection algorithm.

Algorithm 3. C&C channel detection

/* T ⟵ set of packets */
/* C ⟵ set of *APChain* fields */

Function Behavioral_Profiling (T):
  While (not stop condition) **do**
    C ⟵ Call Function APChain (T)

  for T is not null **do**
    if $(C_{.cnti} \geq \theta_i) \cap (C_{.cnti} \leq \theta_j)$ **then**
      ∃T(T is in the candidate group for abnormal traffic)
    if $(|C_{.ratd} - (C_{.actti} \pm C_{.sctti})| \leq \theta_x)$ **then**
      ∀T(T is abnormal traffic)
      accVal + = 1
      if accVal > θ_y **then**
        ∀T(T is in the candidate group for a C&C channel)
        if T ∉ *Internal Whitelist* (T) **then**
          ∀T(T is a C&C channel)
  return T is a C&C channel

Table 6: Hypothesis for detecting a pharming attack.

Hypothesis 2. Pharming attack detection
Given an environment:
Let $T = \{t_1, \cdots, t_{n-1}, t_n\}$, where t represents the network traffic, and T_n is the traffic currently being analyzed.
Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h is a host infected with malicious code.
Let $S = \{s_1, \cdots, s_{n-1}, s_n\}$, where s is a fake website.
$H \Longrightarrow S$, the infected host connects to the fake website.
The user requests a legitimate website, but a host infected with pharming redirects the connection to a fake website. At that time, when requesting the URL of the legitimate website, a connection to a specific destination IP address may be suspected as a pharming attack.
Therefore, we analyze requested URLs from hosts and compare them to the IP addresses of destination websites. The set (G) contains hosts that request the same destination IP addresses.
$G = \{H_1 \Longrightarrow S_1, H_2 \Longrightarrow S_2, \cdots, H_{n-1} \Longrightarrow S_{n-1}, H_n \Longrightarrow S_n\}$

Table 7: Pharming attack detection algorithm.

Algorithm 4. Pharming attack detection

```
/* T ⟵ set of packets */
/* C ⟵ set of APChain fields */

Function Behavioral_Profiling (T):
  while (not stop condition) do
    C ⟵ Call Function APChain (T)
    for T is not null do
      if(C_(n).Dip = C_(m).Dip) ∩ Call Func n_gram (C) then
        ∃T(T is in the candidate group for pharming)
        accVal+ = 1
        if (accVal ≥ θ_i) ∩ (C_.cnti ≥ θ_j) ∩ (C_.ratd ≥ θ_k) then
          /* Analysis of connections to websites suspected as
          pharming and connections to a specific target
          IP address */
          ∀T(T is pharming)
      return pharming

Function n-gram(C):
      if (n_gram(C_(n).url) ∈ C_(n−m).url) then
      /* Accuracy is increased using the n-gram algorithm */
        return true
```

codes such as Windows or Linux-related DDoS botnets are not detected in time, then large volumes of traffic would be generated on the internal network, and we would face the threat of the entire network shutting down. Therefore, it is imperative to detect and effectively eliminate hosts infected with malicious code. However, it is not easy to detect hosts that manifest IP-spoofing DDoS attacks, and related research is limited. In this paper, *APChain* is configured, and we propose a method to detect IP-spoofing DDoS botnets through behavioral profiling. To maintain a communication channel with a C&C server, a host infected with an IP-spoofing DDoS botnet attempts to connect and, if successful, moves to a standby state to receive attack commands. If the host receives an attack command from the C&C server, it generates a large volume of dummy packets to send to the attack target destination. At this time, the source IP address of the packet uses a spoofed IP address and dummy data included in the payload and transfers them to the attack target system. The proposed methodology uses *APChain*'s connection frequency (*CND*), connection time interval, and average connection time interval (*ACTTI*) values to conduct behavioral profiling and detect DDoS attacks. The hypothesis employed to detect IP-spoofing DDoS botnets is shown in Table 8.

The algorithm proposed to detect IP-spoofing DDoS botnets detects hosts that have different source IP addresses but that have the same MAC address. For example, when the traffic frequency for a host connecting to the target IP address is *CNTI* and the average connection cycle is *ACTTI*, if the threshold value is y, then the traffic is suspected to be a DDoS attack, D, where $D = \{x \mid x \in \text{threshold}, (CNTI \geq \theta) \cap (ACTTI \approx 0)\}$. At this time, if the traffic has different source IP addresses but the same MAC address, it is presumed to be an IP-spoofing DDoS botnet.

The method used to detect an infected host is to search for the MAC address suspected to be a spoofed IP address in the *MAC* field of the *APChain* table and to extract the record that was first registered in the *APChain* table to confirm the source IP address. It is assumed that the host with the same IP address as the source IP address is a botnet. Table 9 summarizes the IP-spoofing DDoS botnet detection algorithm using behavioral profiling.

5. Experimental Evaluation

The test bed was prepared as shown in Figure 10, and a prototype was developed to evaluate the performance of the proposed algorithm. Real-time network traffic collected from the test-bed environment and datasets downloaded from the Malware Capture Facility Project (MCFP) were used as data for the experiment. The dataset included botnets that create and utilize C&C channels.

In this section, the following four key aspects are covered:

(i) Explanation of the datasets used in the experimental environment

(ii) Evaluation of the accuracy of the proposed algorithm

Table 8: Hypothesis for detecting IP-spoofing DDoS botnets.

Hypothesis 3. IP-spoofing DDoS botnet detection
Given an environment:
Let $T = \{t_1, \cdots, t_{n-1}, t_n\}$, where t represents the network traffic, and T_n is the traffic currently being analyzed.
Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h is a host infected with malicious code.
Let D = $\{d_1, \cdots, d_{n-1}, d_n\}$, where d represents a target system for DDoS attack.
$H \Longrightarrow D$, the infected host executes a DDoS attack on the target system.
A host infected with an IP-spoofing DDoS botnet receives an attack command from the C&C server and implements a DDoS attack, and the origin IP address of the host attacking with DDoS is modified. At that time, a DDoS attack can be suspected if the host sends large amounts of traffic to the destination system. Also, an IP-spoofing DDoS botnet can be categorized if a particular host has a different origin IP address but the same MAC address.
Therefore, the set (G) consists of hosts that perform IP-spoofing DDoS attacks.
G = $\{H_1 \Longrightarrow D_1, H_2 \Longrightarrow D_2, \cdots, H_{n-1} \Longrightarrow D_{n-1}, H_n \Longrightarrow D_n\}$

Table 9: IP-spoofing DDoS botnet detection algorithm.

Algorithm 5. IP-spoofing DDoS botnet detection

/*T ⟵ set of packets */
/* C ⟵ set of *APChain* fields */

Function Behavioral_Profiling (T):
  while (not stop condition) **do**
    C⟵ Call Function APChain (T)
    for T is not null **do**
      if $C_{.cnti} \geq \theta_i$ **then**
        if $(C_{.ratd} \cong 0) \cap (C_{.actti} \cong 0)$ **then**
          ∃T(T is in the candidate group for DDoS)
          accVal1 += 1
          if accVal1 $\geq \theta_j$ **then**
            ∀T(T is a DDoS attack)
            if $(C_{(n).Sip} \in C_{(n+m).Sip}) \cap (C_{(n).mac} \in C_{(n+m).mac})$ **then**
              ∃T(T is IP spoofing)
              accVal2+ = 1
              if accVal2 $\geq \theta_j$ **then**
                ∀T(T is an IP spoofing DDoS attack)
              return Call Function infection_host ($C_{(n).mac}$, C)

Function infection_host ($C_{.(n).mac}$, C)
  while (not stop condition) **do**
    if $(C_{(n).mac} = C_{.mac})$ **then**
      host_info ⟵ $C_{(n).Sip}$
return host_info

(iii) Measurement of the performance of the developed prototype

(iv) Effectiveness and accuracy of the experimental results

5.1. Experimental Environment and Performance. The experimental test environment is divided into a server farm domain, a user domain, and a branch office. Real-time traffic is collected from the server farm domain and the user domain. The intranet bandwidth is 10 Gbps, and the Internet environment bandwidth is 2 Gbps. The experimental environment for performance evaluation is presented in Table 10.

Table 10: Experimental environment.

Environment	Description
Servers	≒ 100 active servers in a server farm
Host	≒ 1,500 active hosts in the internal network
Bandwidth	10 Gbps internal network and 2 Gbps external network
Traffic flow	≒ 530 GB of data monthly

Table 11: Characteristics of *Dataset_1*.

Environment	Description
Period of traffic collection	From January to June 2017
Number of records	≅ 150,000,000
Collected record size	≅ 3.185 TB
Servers and Hosts	≅ 100 servers, 1,500 hosts
Flow type	Botnet, normal

The prototype was implemented in an environment consisting of an Intel i7 8-core CPU, 16 GB of RAM, and an 8 TB HDD with Java used as the programming language.

5.2. Test Dataset. The datasets used in the experiment are network traffic collected in real-time from the test-bed environment and traffic representing malicious behavior from the MCFP [23, 24]. The datasets are explained below.

Dataset_1 collects traffic in real-time from the experimental environment shown in Figure 10 and analyzes it. The experimental environment comprises 100 servers and 1,500 hosts; the collected data includes network traffic from the server farm and user domains. The dataset stored collected traffic flow for a period of six months, from January 2017 to June 2017. It includes malicious code that corresponds to the attack types specified in the System Model section. The traffic flow consists of 150,000,000 records and is 3.185 TB in size; the characteristics are presented in Table 11.

The scenarios used in *Dataset_1* are summarized in Table 12. *Dataset_1* includes header information (source IP,

Table 12: *Dataset_1*.

Scenario	Capture name	Size	Threat type
S01	KU-Malware-01	2.6 GB	Bot
S02	KU-Malware-02	4.3 GB	Bot
S03	KU-Malware-03	1.4 GB	Bot
S04	KU-Malware-04	412 MB	Bot
S05	KU-Malware-05	1.3 GB	Pharming
S06	KU-Malware-06	5.4 GB	Pharming
S07	KU-Malware-07	354 MB	Pharming
S08	KU-Malware-08	4.3 GB	DDoS bot
S09	KU-Malware-09	3.8 GB	DDoS bot
S10	KU-Malware-10	2.8 GB	DDoS bot

Table 13: Dataset_2.

Scenario	Capture name	Size	Bot
S11	CTU-Malware-13	305 MB	Murlo
S12	CTU-Malware-42	5.75 GB	Neris
S13	CTU-Malware-44	4.78 GB	Rbot
S14	CTU-Malware-46	371 MB	Virut
S15	CTU-Malware-47	3.05 GB	Menti
S16	CTU-Malware-78	6.3 GB	Zeus
S17	CTU-Malware-116	317 MB	Kazy

Destination IP, Protocol, etc.), MAC address, and URL of packets extracted from the network traffic. It includes both normal and malicious traffic.

Scenarios S01 and S02 are packets installed with malicious code through a backdoor from a seized account in a system located in the DMZ that includes a communications channel with a C&C server. S03 has a web application system (WAS) administrator account set up with initial values. After the system is seized by an attacker, a webshell is uploaded to install malicious code. S04, S05, S06, and S07 are a scenario in which a host connected to a malicious code distribution sites is infected with a malicious code. Scenarios S08, S09, and S10 involve a host infected with malicious code which is maintaining a communication channel with a C&C server and conducting an IP-spoofing DDoS attack [25].

Dataset_2 is malicious code traffic distributed by the MCFP [23]. The MCFP is a research project created by the Czech Technical University ATG Group with the objective of capturing, analyzing, and distributing malicious code traffic, and it distributes datasets to assist in the development of various detection methods. The malicious code traffic distributed by the MCFP can be downloaded from its website [1] and includes a PCAP file, netflow file, and readme file. However, because the PCAP file contains personal information, it does not provide all data. The PCAP file used in the experiment targets traffic that contains information about the C&C server. The MFCP dataset used in the experiment is summarized in Table 13.

Dataset_2 in Table 13 is a Trojan horse that is installed via user e-mail, messenger applications, or reference libraries. Scenarios S12 and S13 are Neris bot and Rbot bot, and scenarios S14 and S15 are Virut bot and Menti bot. The PCAP file used for the analysis contains both normal and botnet traffic and was captured on the main router of the university network.

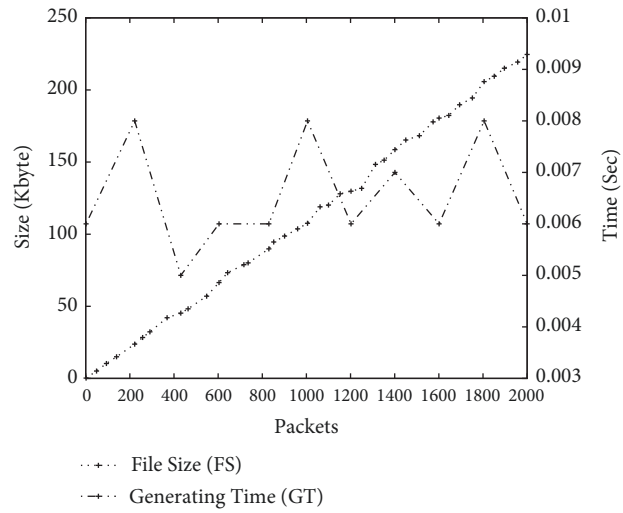


Figure 14: Time and size for *APChain* table generation.

5.3. APChain Creation. The configuration of *APChain* is critical to detecting abnormal behavior and becomes an important factor with regard to detection speed and accuracy. *APChai*n extracts attribute information from real-time traffic, performs additional analysis, and links attribute information over time. Figure 14 shows the time (GT) to

Table 14: Proof for the detection of C&C channels.

Proof 1. Detection of C&C channels

Given an environment,

Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h is a host infected with malware.

Let $S = \{s_1, s_2, \cdots, s_{n-1}, s_n\}$, where s is a C&C server.

If the connection cycle from the host to the destination system satisfies condition (x), then calculate the accumulated count (ac), and if the accumulated count satisfies threshold (θ), then this is defined as a C&C channel.

$x = [(\mu - \sigma) \pm \theta_i] \leq R \cap [(\mu + \sigma) \pm \theta_j] \geq R$ (where R is *RATD* field value of *APChain*)

$\mu = \sum_{i=1}^{n=\gamma} (t_i - t_{i-1})/\gamma, \sigma = \sqrt{\sum_{k=1}^{n=\gamma} (t_k - \mu)^2/n}$

if x=true, then $ac = \sum_{k=1}^{n=\gamma} 1$ (where γ is the record number of *APChain*)

$$\text{if } ac \geq \theta_k, \begin{cases} \textit{true: Command and Control channel} \\ \textit{false: normal network traffic} \end{cases}$$

At this point, if the host is not communicating with the C&C server, the host is not infected with a botnet. Accordingly, a host infected with a botnet will maintain the C&C channel while periodically connecting to the C&C server.

Therefore, a C&C channel can be categorized if false positives are excluded from the set (C) of hosts suspected to be a C&C channel.

$C = \{h_n, \cdots, h_m\} \cap \textit{false positive} \notin C$

Table 15: Experimental results for C&C channel detection.

Scenario	Accuracy	Precision	Recall	TP	TN	FP	FN
S01	1.0	1.0	1.0	3	431	0	0
S02	0.993	0.5	1.0	4	537	4	0
S03	0.996	0.6	1.0	3	642	2	0
S04	1.0	1.0	1.0	1	102	0	0
S11	1.0	1.0	1.0	1	12	0	0
S12	0.993	0.03	1.0	1	4174	27	0
S13	0.999	0.5	1.0	1	9988	1	0
S14	0.994	0.1	1.0	1	1536	9	0
S15	0.994	0.09	1.0	1	1535	10	0
S16	0.950	0.5	1.0	1	18	1	0
S17	0.922	0.1	1.0	1	94	8	0

generate *APChain* records by packet and the size (FS) of *APChain* table.

According to the experimental results in Figure 14, *APChain* algorithm requires a minimum storage capacity of 102 Bytes to 115 Bytes per packet to generate *APChain* table, and 100,000 packets require a storage capacity of 10.9 Mbytes. A minimum of 5 milliseconds and maximum of 8 milliseconds are needed to analyze the attribute information of the packets and to generate record of *APChain* table.

As a result, if it takes 6 milliseconds overall to generate *APChain* record, *ACTTI* and *SCTTI* fields account for 3 milliseconds and 2 milliseconds. Therefore, in order to apply the proposed methodology to a real-time network environment, the calculation of the *ACTTI* and *SCTTI* fields of *APChain* algorithm should be done effectively.

5.4. Performance Evaluation. In this section, the proposed algorithm is applied to the attack types defined in the system model in order to detect these attacks and evaluate its performance.

5.4.1. C&C Channel Detection [Case Study A]. Hosts infected by a botnet maintain a communication channel with a C&C server, receiving either an attack command or updating the configuration before the attack is initiated. Therefore, if it were possible to detect the channel between the host infected by malicious code and the C&C server before important information is breached, it would then be possible to detect or block attacks within the lead time needed for attacks to succeed. According to this, the hypothesis used to detect C&C channels is defined in Table 4, and the proof for the hypothesis is provided in Table 14. The performance of the proposed algorithm is verified using experiments, the results of which are presented in Table 15.

The datasets used in detecting C&C channels employed the scenarios S01, S02, S03, and S04 in Table 12 and S08, S09, S10, S11, S12, S13, and S14 in Table 13. The C&C channel detection algorithm defined in the System Model section was implemented as a prototype to evaluate performance. The goal of the proposed methodology is to detect C&C channels using the *APChain* algorithm and behavioral

Table 16: Experimental results for C&C channel detection according to the threshold value.

Category	Accuracy	Precision	Recall	FP rate
Threshold A (0, 240) $\{x \mid 0 \leq x \geq 240, x\text{=frequency}\}$	0.0156	0.0006	1.0	0.9850
Threshold B (30, 240) $\{x \mid 30 \leq x \geq 240\}$	0.9998	0.8571	1.0	0.0001
Threshold C (30, 720) $\{x \mid 30 \leq x \geq 720\}$	0.9856	0.0428	1.0	0.0143
Threshold D (90, 720) $\{x \mid 90 \leq x \geq 720\}$	0.9854	0.0289	0.6667	0.0143
Threshold E (0, ∞) $\{x \mid 0 \leq x \geq \infty\}$	0.0006	0.0006	1.0	1.0

profiling regardless of any changes to the C&C channel's communication cycle.

One of the steps in the life cycle of a botnet attack is the creation of a communication channel with a C&C server. The proposed algorithm analyzes inbound to outbound traffic to configure *APChain*, and behavioral profiling is conducted according to the attack type to detect attacks. At this time, *APChain*'s *SGN* field is used as a reference value that is applied to configure the same target IP addresses as a group and link them into a chain. According to the experimental results in Table 15, by using the proposed algorithm, the C&C channel's true positive rate is 1.0 and the false positive rate, which indicates normal network traffic that is classified as a C&C channel, is 0.003.

A true positive is when a C&C channel is detected and determined not to be normal traffic and a true negative is when normal traffic is detected and determined not to be a C&C channel. A false positive detects normal traffic as a C&C channel, and a false negative is when a C&C channel is detected as normal traffic. In such instances, the accuracy is 0.996, the precision is 0.225, and the recall is 1.0.

Although the detection of the configuration of a communication channel between a host infected by malicious code and a C&C server has been the focus of several studies, including [3, 7, 26], these detection methods may encounter an increase in false positives or false negatives when the C&C channel occurs irregularly. The methodology here proposes the *APChain* algorithm to alleviate the existing problems by using connection time intervals, frequencies, and standard deviations to perform behavioral profiling and thus detect abnormal behavior.

According to the test results displayed in Table 15, accurate detection was made in scenarios S01, S04, and S11 while detection errors occurred in S02, S03, S12, S13, S14, S15, S16, and S17. The detection errors were false positives, in which normal traffic was detected as C&C traffic. The reason why these false positives occurred was because traffic inspecting the service between the host and the external system was present. However, false positives can be categorized as normal traffic through the elimination of whitelist-based false positives (Section 3.4). We thus applied the *Internal Whitelist* from Section 3.4 to S12, and the false positive rate decreased by 56% from 27 to 15.

The experimental results for detecting a C&C channel according to the threshold value are presented in Table 16.

In Table 16, category x indicates the frequency of packets suspected of being a C&C channel. When the cumulative frequency resides within the range of the minimum and maximum values of the threshold given in Table 16, the accuracy, precision, recall, and FP rate are calculated. The accuracy for Threshold B (30, 240) is 0.9998, the precision is 0.8571, and the recall is 1.0. The accuracy of Thresholds C (30, 720) and D (90, 720) are similar to that of Threshold B (30, 240), but the recall and precision are lower than Threshold A (0, 240). For Threshold C (30, 720), the accuracy, which represents the accuracy of C&C channel detection, is higher. However, the false positive rate, which indicates the rate at which normal traffic is categorized as a C&C channel, also increases. For Threshold D (90, 720), C&C channels were not accurately detected, resulting in false negatives, and normal traffic was categorized as a C&C channel, causing the false positive rate to also increase. Consequently, the C&C channel detection accuracy was the highest when the conditions for Threshold B were selected; when the conditions for Thresholds C (30, 720) and D (90, 720) were applied, both the false negative and false positive rates increased.

Based on the monitoring of the connection cycle and frequency between a host infected by malicious code and a C&C server, it is found that the connection cycle varies and the cumulative connection frequency is linear. Consequently, detecting a C&C channel with the proposed algorithm reduces detection errors while presenting an effective method for detecting attacks.

5.4.2. Pharming Attack Detection [Case Study B]. Pharming attacks either falsify a Domain Name Service (DNS) server address or falsify a DNS cache file; in an attempt to connect to a normal website, users instead are connected to a fake website because of the falsified DNS server. Pharming attacks can be detected by categorizing whether the host is connecting to a normal site or to a fake site. Accordingly, the hypothesis defined in Table 6 is used to detect a pharming attack, and the proof of this hypothesis is provided in Table 17. The performance of the proposed algorithm is verified experimentally and the results are presented in Table 18.

Table 17: Proof for the detection of a pharming attack.

Proof 2. Pharming attack detection

Given an environment:

Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h is a host infected by malware.

Let $S = \{s_1, s_2, \cdots, s_{n-1}, s_n\}$, where s is a web server.

Calculate the accumulated count if the destination system has the same IP address but different URLs when connecting from the host to the destination system. At this point, if the accumulated count satisfies threshold (θ), then it is defined as a pharming attack.

$x = (h_{\text{n.}url} \longrightarrow S_{\text{n}}) \cap (h_{\text{m.}url} \longrightarrow S_{\text{n}})$

if $x = \text{true} \cap (h_{\text{n.}url} \neq h_{\text{m.}url})$, then $ac = \sum_{i=1}^{n=\gamma} 1$

$$\text{if } ac \geq \theta, \begin{cases} \textit{true: pharming attack} \\ \textit{false: normal network traffic} \end{cases}$$

At this point, if the URL and IP address requested by the hosts connecting to the destination system are different, then this is considered normal service. However, there are some cases where the hosts infected with pharming request different URLs from the server but they have the same IP address.

Therefore, the set (C) consisting of hosts requesting the same IP address but different URLs from the web server can be categorized as being infected with pharming.

C = $\{h_n, \cdots, h_m\}$, where h is a host infected by pharming

Table 18: Experimental results for pharming attack detection.

Scenario	Accuracy	Precision	Recall	TP	TN	FP	FN
S05	0.994	0 .6	1.0	3	382	2	0
S06	1.0	1.0	1.0	4	279	0	0
S07	0.996	0.4	1.0	4	1244	5	0

The datasets used in the experiment are applied to scenarios S05, S06, and S07 in Table 12, and the algorithm defined in Table 7 is implemented as a prototype. *APChain* field values, *Sip*, *Sport*, *Dip*, *Dport*, *URL*, *CND*, *RATD*, and *ACTTI,* are analyzed, and the characteristics of the host infected by pharming are detected through behavioral profiling.

To detect a pharming attack, we conduct behavioral profiling when *APChain*'s *ACTT*I value is higher than the *RATD* value and the connection frequency, *CND*, increases.

In regard to the outbound traffic collected at the backbone switch, the connection traffic from the host to the website is analyzed, and traffic suspected to be exhibiting abnormal behavior is categorized into a candidate group. The characteristics of the pharming attack are analyzed, and behavioral profiling is conducted to detect an attack.

According to the experimental results shown in Table 18, the ratio for accurately detecting a host infected by a pharming attack connecting to a fake website is 99.6%, with three cases in which a host connecting to a normal website is categorized as a host infected by pharming. To reduce detection errors, URL comparisons are done using n-gram indexing; when the URL address matching accuracy exceeds 80%, the address is categorized as the same URL address. Table 19 shows the detection accuracy rates when n-gram indexing and string matching are used.

5.4.3. IP-Spoofing DDoS Botnet Detection [Case Study C]. Hosts infected with malicious code such as the Linux-related Xor.DDoS receive commands from a C&C server to perform an IP-spoofing DDoS botnet attack on target systems in the internal network. Hosts infected with malicious code generate large numbers of falsified source IP addresses and attempt to connect to an external attack system. At this time, even if the internal network firewall detects a large volume of SYN flooding attacks occurring outbound to inbound and closes the port, if the hosts infected by the malicious code cannot be eliminated, the bandwidth will be exhausted by a large volume of packets. Therefore, the proposed methodology can quickly detect DDoS attacks occurring internally and identify and remove hosts that are part of an IP-spoofing DDoS botnet. The proof for the hypothesis used to detect an IP-spoofing DDoS botnet is provided in Table 20. The performance of the proposed algorithm is verified experimentally, with the results summarized in Table 21.

Scenarios S08, S09, and S10 were used from the experimental dataset for the detection of IP-spoofing DDoS botnets and the prototype algorithm was implemented for the experiment.

For the outbound traffic collected from the backbone switch, the traffic that leads to the DDoS attack on the host is analyzed and the traffic suspected of abnormal behavior is categorized as part of the candidate group. We link *APChain* records over time and conduct behavioral profiling to detect IP-spoofing DDoS botnet. At this time, if a DDoS attack is detected, the *Sip* and *MAC* fields of *APChain* are analyzed to find the host infected with the IP-spoofing DDoS botnet.

A true positive is the detection of an IP-spoofing DDoS botnet and a true negative is the detection of normal traffic. A false positive is erroneously detecting normal traffic as an IP-spoofing DDoS botnet and a false negative is erroneously detecting an IP-spoofing DDoS botnet as normal traffic. As shown in Table 21, the detection of IP-spoofing DDoS botnets

Table 19: Accuracy comparison between n-gram indexing and string matching.

Condition	Accuracy	Precision	TP Rate	FP Rate
n-Gram indexing	0.996	0.611	1.0	0.003
String matching	0.84	0.034	1.0	0.160

Table 20: Proof of the detection of an IP-spoofing DDoS botnet.

Proof 3. IP-spoofing DDoS botnet detection

Given an environment:

Let $H = \{h_1, \cdots, h_{n-1}, h_n\}$, where h is a host infected by malware.

Let $S = \{s_1, \cdots, s_{n-1}, s_n\}$, where s is a web server.

Calculate the accumulated count if the connection cycle from the host to the destination system satisfies condition (x), and it is defined as DDoS if the accumulated count (ac) satisfies threshold (θ). If the IP address of the origin host is modified via forgery, it is detected as an IP-spoofing botnet.

$x = (h_{\text{n.}time} - h_{\text{m.}time} \cong 0) \cup (s_{\text{n.}time} - s_{\text{m.}time} \cong 0)$

if x=true, then $ac = \sum_{k=1}^{n=\gamma} 1$

$$\text{if } ac \geq \theta_i, \begin{cases} true: DDoS\ attack \\ false: normal\ network\ traffic \end{cases}$$

$y = (h_{\text{n.}Sip} \neq h_{\text{m.}Sip}) \cap (h_{\text{n.}mac} = h_{\text{m.}mac})$

$$\text{if } y \geq \theta_j, \begin{cases} true: IP\ spoofing \\ false: normal\ IP \end{cases}$$

However, if DDoS occurs, the connection cycle is close to 0, and there is a large amount of network traffic. Therefore, calculate the accumulated count threshold (θ) when the traffic occurrence cycle is equal or similar to 0, or when the connection cycle of a particular web server (S) is equal or similar to 0. On this occasion, if it satisfies the condition ($ac \geq \theta$), then it is categorized as a DDoS attack.

To detect an IP-spoofing DDoS botnet, the set (C) of the hosts with the same MAC address but different origin IP address is categorized as the one infected with a botnet.

$C = \{h_n, \cdots, h_m\}$, where h is a host infected with an IP-spoofing DDoS botnet.

Table 21: Experimental results for IP-spoofing DDoS botnet detection.

Scenario	Accuracy	Precision	Recall	TP	TN	FP	FN
S08	1.0	1.0	1.0	4	457	0	0
S09	1.0	1.0	1.0	3	526	0	0
S10	1.0	1.0	1.0	3	1065	0	0

shows an accuracy and precision rate of 1.0 for scenarios S08, S09, and S10.

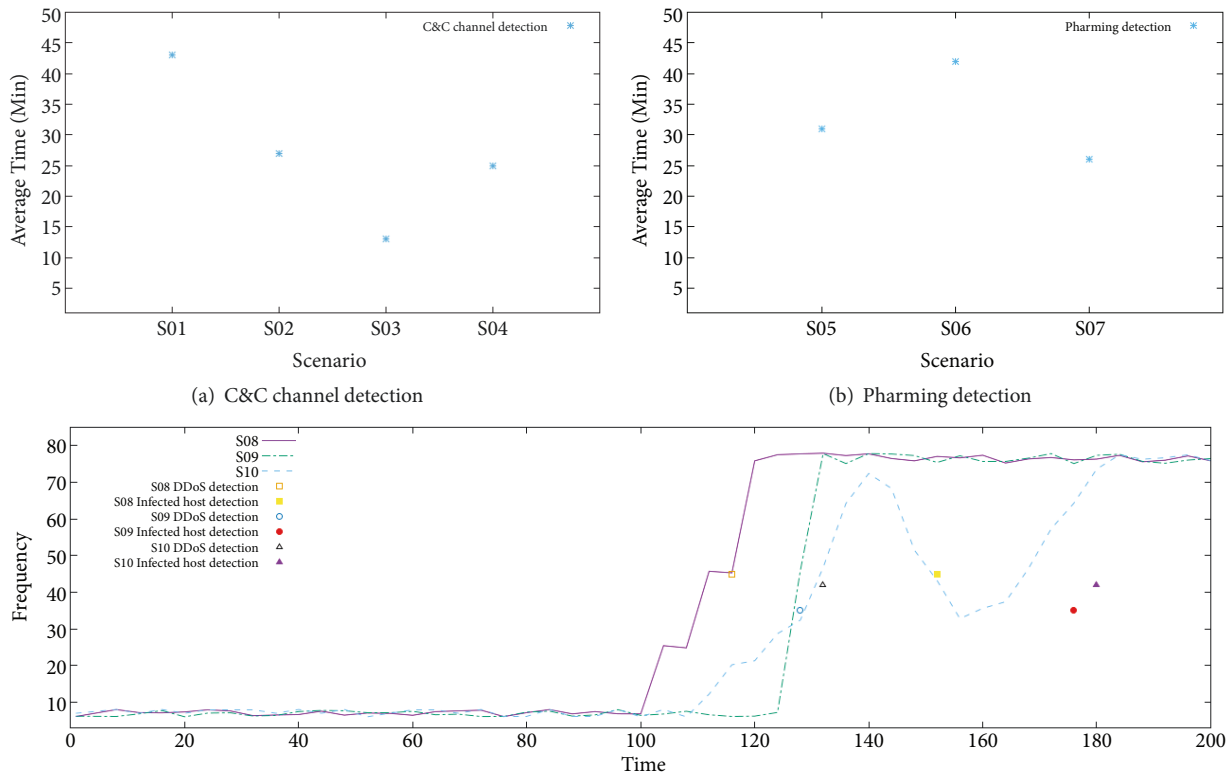
According to the experimental results, the host infected with malicious Xor.DDoS code generates more than one million SYN flooding packets per minute and transmits about 64 GB of dummy traffic to the network.

5.5. Performance Ability. In this section, we measure the detection time for the three types of attack. The datasets used in the experiment are defined in Tables 12 and 13.

Figure 15(a) shows that S01 is the scenario in which the most time is used to detect a C&C channel; the C&C channel and the infected host were detected within an average time of 43 minutes. In S03, it takes an average of 13 minutes to detect the C&C channel in the infected host. Based on these results, it is possible to detect the C&C channel within the lead time, after which the host can be infected with malicious code and threatened.

In Figure 15(b), the time between the host that has been infected with malicious code connecting to the fake site and detecting the pharming attack is 31 minutes on average for S05. For S06, it takes 42 minutes on average to detect the pharming attack because the time interval for the host to call the fake site is large. The experiment indicates that, when the host infected by a pharming attack connects to a fake site, detection is possible with the proposed algorithm and, because the IP address of the target website can be changed, detection accuracy increases when abnormal behavior is detected more than three times.

According to Figure 15(c), when a host infected with malicious code develops a DDoS attack, the generated outbound traffic rapidly increases and the bandwidth of the internal network is exhausted. In S08, the detection of a DDoS attack is detected within 9 seconds, and IP-spoofing is detected within 60 seconds. In the test results for scenario S09, a DDoS attack is detected when the host's malicious code produces a large amount of traffic, and IP-spoofing was detected within 75 seconds. The proposed methodology thus exhibits a DDoS attack detection performance similar to that of an intrusion detection system, and it is able to effectively detect a host infected with IP-spoofing.



(a) C&C channel detection

(b) Pharming detection

(c) IP-spoofing DDoS detection

Figure 15: Measurement of case study detection times.

In order to apply the proposed approach to a large-scale network environment, it is necessary to efficiently implement *APChain* algorithm and ensure sufficient hardware performance to process a large number of packets. For example, approximately 3.4% of the IP packets collected in al real-time network are TCP traffic [27], and packets collected in the experimental environment of Section 5.1 process traffic of up to 100 TCPs per second. The proposed algorithm can be applied to a real-time network environment, because it requires 7 milliseconds per packet to generate one record of *APChain* table.

In this paper, we exclude known communications that are to send and receive messages by the protocols defined by the FTP or Socket program to increase detection performance and minimize resource consumption. Predefined known communication is registered and managed in the *Whitelist* in Section 3.4.

6. Conclusion and Future Work

Botnets are still a serious threat when it comes to cyber-attacks, and attacks on specifically targeted systems go beyond simple hacking. Attackers are employing a variety of attack methods that utilize, for example, malicious code, software vulnerabilities, and social engineering techniques and, by introducing new attack techniques that bypass security systems, they are making it difficult to detect attacks on existing systems.

Because signature-based detection is not up to the task of deterring new attack techniques, research on abnormal behavior detection through behavior analysis and the detection of malicious code based on virtual sandboxes is underway. However, when traffic volume increases in a large network environment, sandbox-based malicious code detection becomes time consuming and increases resource inefficiency. It is also not easy to detect and respond to attacks when malicious code is obfuscated or when encrypted communication is used with an external attacker. Therefore, it is necessary to minimize the consumption of resources in a large-scale network environment and to construct an environment capable of effective infringement response even when facing encrypted communication.

In this study, to detect abnormal behavior, we analyze traffic attribute information, construct an attack pattern chain algorithm (*APChain*), and conduct behavioral profiling. The configuration of *APChain* analyzes the attribute information of traffic collected in a real-time network environment and stores it in a database. The attribute values stored in the *APChain* table are organized into chains according to the passage of time, and abnormal behavior profiling is then used to detect malicious behavior.

The performance evaluation of the proposed algorithm is analyzed with respect to three types of botnet attack. First, C&C communication is detected through network-based behavioral profiling, which is essential in the botnet life cycle. The proposed methodology improves the limitations of

C&C channel detection through existing behavior analysis, proposes a methodology that can be effectively applied to a large-scale network environment, and detects C&C channels with an accuracy rate of 99.6%.

Second, by analyzing a pharming attack's characteristics, we construct *APChain* for real-time traffic and behavioral profiling for abnormal behavior is conducted to detect an attack. Pharming attacks are difficult to detect in traditional security devices because the host sends a connection request to a normal website and receives a service response, which is not easy to detect with an endpoint system such as a vaccine or a zombie PC detection solution. In this paper, the experimental results show pharming attacks are accurately detected using the proposed algorithm.

Third, it detects IP-spoofing DDoS botnets and discovers and eliminates hosts that generate IP-spoofing. An IP-spoofing DDoS botnet performs a DDoS attack against a falsified source IP address of the host, and intrusion prevention systems have difficulty blocking the relevant port. If the wrong IP address is blocked, normal service may be disrupted. Therefore, it is important for service continuity to detect and eliminate a host infected with malicious code. In this study, we analyze DDoS attacks by analyzing *APChain* field values and categorize hosts infected by malicious code by analyzing the *APChain* for the IP-spoofing of the hosts. In the experiment using the proposed algorithm, the DDoS attack was detected within 10 seconds, and the detection of the host infected with the malicious code was possible within 60 seconds.

We propose the *APChain* algorithm and a behavioral profiling algorithm to overcome the limitations of existing botnet detection techniques. It proposes an effective abnormal behavior detection method that reduces the number of false positives and false negatives while using limited resources and without having to change the configuration of network resources.

In the paper, the proposed C&C channel, pharming, and IP-spoofing DDoS botnet detection are based on the network traffic between hosts and botnets. However, more experiments and verification are needed to detect advanced persistent threat (APT) attacks using C&C channel detection. As part of our future work, we will conduct further studies to detect the C&C channels for APT attacks and IoT botnets using the proposed algorithm. For example, IoT devices infected with malicious code through an IoT botnet will communicate with the C&C server to receive an attack command or download a new binary file. At this time, it will be possible to detect the C&C channel using the proposed methodology.

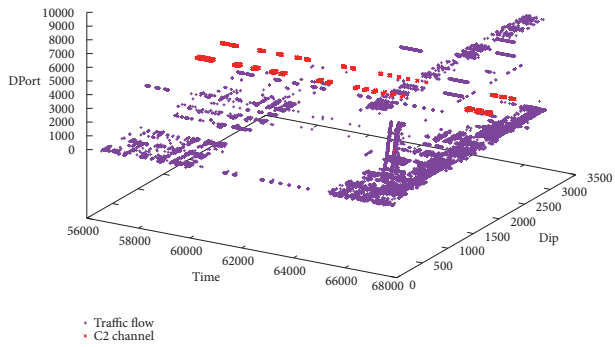


Figure 16: Analysis of behavior model of C&C channels.

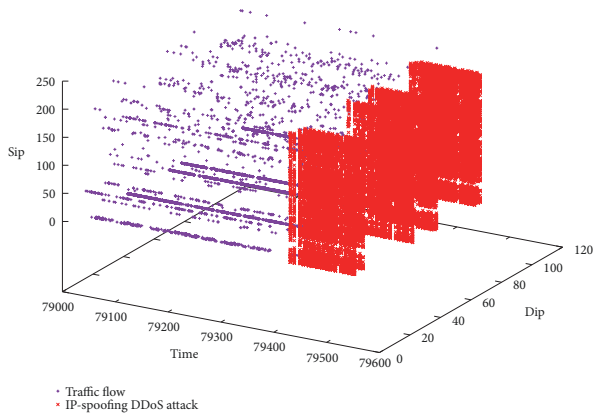


Figure 17: Analysis of behavior model of IP-spoofing DDoS attack.

Appendix

A. The Behavior Model of Network Traffic Using *APChain*

This appendix describes the characteristics of C&C channels and IP-spoofing DDoS attacks using *APChain* for the datasets defined in Section 5.2. The goal of this section is to analyze the behavioral model of botnet attacks and present the direction of research into the proposed methodology, which detects C&C channels and IP-spoofing DDoS attack through *APChain* algorithm and behavior profiling.

In Figure 16, each point represents the traffic flow between the host and destination system. The x-axis represents the flow of time at which the network traffic is collected to configure *APChain*, the y-axis represents the IP address of the target system, and the z-axis represents the port for connection from the host to the destination system. The traffic flow represents the normal traffic collected on the protocol stack in Figure 11 and the C2 (Command and Control) channel represents the C&C communication between the host and the C&C server.

In Figure 17, the traffic flow represents normal traffic collected on the protocol stack in Figure 11 before the IP-spoofing DDoS attack occurs. The red point represents the result of detecting an IP-spoofing DDoS attack, and the IP address of the host is IP spoofed to conduct a DDoS attack. At this time, the host infected with the bot attacks multiple target systems.

B. *APChain* Implementation

(1) # Import packets from backbone switch

(2) # Process pre-processing for behavior profiling

(3)

(4) THRESHOLD = 10.0

```
(5)  ANALYSIS_TIME = 1800
(6)  24H_ANALYSIS_TIME = 86400
(7)
(8)  Function Main {
(9)   attributes[] ⟵ packet
(10)    tuples = []
(11)
(12)    for i in rang(1, len(attributes)):
(13)     # Set the basic attribute information of packets
(14)     sn = Integer.toString(i);
(15)     sgn = Call Function SGN(attributes)
(16)     sip = source_ip in attributes[]
(17)     sport = source_port in attributes[]
(18)     dip = destination_ip in attributes[]
(19)     dport = destination_port in attributes[]
(20)     mac = local_host_mac in attributes[]
(21)     pt = protocol in attributes[]
(22)     ts = time_stamp in attributes[]
(23)     url = destination_url in attributes[]
(24)
(25)     # Calculate additional information using attribute information
(26)     tin = ts-previous_time_stamp
(27)     cnd = Call Function CND(attributes)
(28)     ratd = Call Function RATD(attributes)
(29)     cnti = Call Function CNTI(attributes)
(30)     actti = Call Function ACTTI(attributes)
(31)     sctti = Call Function SCTTI(attributes)
(32)
(33)     # Insert tuples_values
(34)     tuples_values.append(sn, sgn, sip, sport, dip, dport, mac, pt, ts, url)
(35)     tuples_values.append(tin, cnd, ratd, cnti, actti, sctti)
(36)
(37)     # Insert tuples to APChain DB
(38)     APChain ⟵ tuples
(39)  }
(40)
(41)  Function SGN(String[] attribute) {
(42)   Get the same group ID, if the destination IP of packets is the same
(43)    : param dataset: attribute information of packets
(44)    : rtype: string
(45)    : return: Group ID such as "G-XXXX"
(46)    Extract the attribute_dip from attribute
(47)    Extract the apchain_dip from APChain table
(48)    if attribute_dip == apchain_dip && attribute_sip == apchain_sip:
(49)      return existing_SGN_value
(50)    else:
(51)      return new_SGN_value
(52)  }
(53)
(54)  Function CND(String[] attribute) {
(55)   Get the cumulative counts for packets of the same SGN
(56)    : param dataset: attribute information of packets
(57)    : rtype: int
(58)    : return: The number of accesses
(59)    Extract the apchain_attribute from APChain table
(60)    time_stamp ⟵ attribute.time
(61)    for i in range(time_stamp, time_stamp-24H_ANALYSIS_TIME):
(62)      if attribute.sgn == apchain_attribute.sgn:
(63)        return (apchain_attribute.cnd + 1)
(64)    return (initialization_value(1))
(65)  }
(66)
(67)  Function RATD(String[] attribute) {
(68)   Get the time interval for packets of the same SGN
(69)    : param dataset: attribute information of packets
(70)    : rtype: date
(71)    : return: The time interval of accesses
(72)    Extract the apchain_attribute from APChain table
(73)    time_stamp ⟵ attribute.time
(74)    for i in range(time_stamp, time_stamp-ANALYSIS_TIME):
(75)      if attribute.sgn == apchain_attribute.sgn:
(76)        return (time_stamp-apchain_attribute.time)
(77)    return initialization_value(time_stamp)
(78)  }
(79)
(80)  Function CNTI(String[] attribute) {
(81)   Get the cumulative counts for packets of the same SGN
(82)    : param dataset: attribute information of packets
(83)    : rtype: int
(84)    : return: The number of accesses over a period of time
```

```
(85)  Extract the apchain_attribute from APChain table
(86)  time_stamp ⟵ attribute.time
(87)  for i in range(time_stamp, time_stamp-ANALYSIS_TIME):
(88)    if attribute.sgn == apchain_attribute.sgn:
(89)      return (apchain_attribute.cnd + 1)
(90)  return initialization_value(1)
(91)  }
(92)
(93)  Function ACTTI(String[] attribute) {
(94)  Get the average of access time for packets of the same SGN
(95)  : param dataset: attribute information of packets
(96)  : rtype: float
(97)  : return: The average of accesses time over a period of time
(98)  Extract the apchain_attribute from APChain table
(99)  time_stamp ⟵ attribute.time
(100) apchain_ratd ⟵ apchain_attribute.ratd
(101) for i in range(time_stamp, time_stamp-ANALYSIS_TIME):
(102)   if attribute.sgn == apchain_attribute.sgn:
(103)     if apchain_ratd >= THRESHOLD:
(104)       Calculate the accumulated RATD (acc_ratd) from apchain_attribute
(105)       Calculate the number of times (acc_cnt) from acc_ratd
(106)       return (acc_ratd / acc_cnt)
(107)   return initialization_value(access_time)
(108) }
(109)
(110) Function SCTTI(String[] attribute) {
(111)  Get the standard deviation of access time for packets of the same SGN
(112)  : param dataset: attribute information of packets
(113)  : rtype: float
(114)  : return: The standard deviation of accesses time over a period of time
(115)  Extract the apchain_attribute from APChain table
(116)  attribute_actti ⟵ attribute.actti
(117)  apchain_ratd ⟵ apchain_attribute.ratd
(118)  for i in range(time_stamp, time_stamp-ANALYSIS_TIME):
(119)   if attribute.sgn == apchain_attribute.sgn:
(120)     if apchain_ratd >= THRESHOLD:
(121)       sum += Math.pow(apchain_ratd-attribute_actti, 2)
(122)       Calculate the number of times (acc_cnt)
(123)  return Math.sqrt(acc_ratd / acc_cnt)
(124)  }
```

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors wish to thank their colleagues for their insightful discussion and their feedback at various stages of the research.

References

[1] Symantec, Internet Security Threat Report, vol. 21, 2016.

[2] S. Le Blond, D. Choffnes, W. Zhou, P. Druschel, H. Ballani, and P. Francis, "Towards efficient traffic-analysis resistant anonymity networks," in *Proceedings of the SIGCOMM*, ACM, 2013.

[3] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[4] A. Oprea, Z. Li, T. Yen, S. H. Chin, and S. Alrwais, "Detection of early-stage enterprise infection by mining large-scale log data," in *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 45–56, June 2015.

[5] C. Terry, "The Anatomy of an Advanced Persistent Threat," 2010, http://www.securityweek.com/anatomy-advanced-persistent threat.

[6] Y. Masahiro, M. Masanobu, U. Yuki, T. Satoru, and T. Masahiko, "RAT-based malicious activities detection on enterprise internal networks," in *Proceedings of the International Conference for Internet Technology and Secured Transactions 2015*, pp. 321–325, 2015.

[7] M. N. Sakib and C.-T. Huang, "Using anomaly detection based techniques to detect HTTP-based botnet C&C traffic," in *Proceedings of the IEEE International Conference on Communications (ICC '16)*, May 2016.

[8] K. Wang, C.-Y. Huang, L.-Y. Tsai, and Y.-D. Lin, "Behavior-based botnet detection in parallel," *Security and Communication Networks*, vol. 11, no. 7, 2014.

[9] O. A. Osanaiye and M. Dlodlo, "TCP/IP header classification for detecting spoofed DDoS attack in Cloud environment," in *Proceedings of the International Conference on Computer as a Tool, IEEE EUROCON 2015*, September 2015.

[10] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of the 16th USENIX Security Symposium (Security'07)*, 2007.

[11] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th conference on Security symposium*, pp. 139–154, 2008.

[12] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC 2012*, pp. 129–138, December 2012.

[13] Y. Wang, M.-Z. Hu, B. Li, and B.-R. Yan, "Tracking anomalous behaviors of name servers by mining DNS traffic," *Lecture Notes in Computer Science*, vol. 4331, pp. 351–357, 2006.

[14] F. Tegeler, X. Fu, G. Vigna, and C. Kruegel, "BotFinder: Finding bots in network traffic without deep packet inspection," in *Proceedings of the 8th ACM International Conference on Emerging Networking EXperiments and Technologies, CoNEXT 2012*, pp. 349–360, France, December 2012.

[15] C. Tankard, "Advanced Persistent threats and how to monitor and deter them," *Network & Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[16] T.-F. Yen, A. Oprea, K. Onarlioglu et al., "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference, ACSAC 2013*, pp. 199–208, December 2013.

[17] X. Hu, J. Jang, M. P. Stoecklin et al., "Baywatch: robust beaconing detection to identify infected hosts in large-scale enterprise networks," in *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 479–490, July 2016.

[18] T. Hyslip and J. Pittman, "A survey of botnet detection techniques by command and control infrastructure," *Journal of Digital Forensics, Security and Law*, vol. 10, no. 1, pp. 7–26, 2015.

[19] J. Gardiner, M. Cova, and S. Nagaraja, "Command & Control: Understanding, Denying and Detecting," *CoRR*, vol. abs/1408.1136, 2014.

[20] D. Zhao, I. Traore, B. Sayed et al., "Botnet detection based on traffic behavior analysis and flow intervals," *Computers & Security*, vol. 39, pp. 2–16, 2013.

[21] Y. Zhou, C. Jiao, H. Chen, L. Ma, and G. Hu, "Traffic behavior feature based DoS&DDoS attack detection and abnormal flow identification for backbone networks," *Journal of Computer Applications*, vol. 10, no. 33, 2013.

[22] E. C. Sasu and O. Prostean, "Using constant traffic to specific IP destinations for detecting spoofed MAC addresses in local area networks," in *Proceedings of the 2010 International Joint Conference on Computational Cybernetics and Technical Informatics*, 2010.

[23] S. García, A. Zunino, and M. Campo, "Survey on network-based botnet detection methods," *Security and Communication Networks*, vol. 5, no. 7, pp. 878–903, 2013.

[24] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Journal of Computers and Security*, vol. 45, pp. 100–123, 2014.

[25] Malware File, http://malware-traffic-analysis.net.

[26] B. Soniya and M. Wilscy, "Detection of randomized bot command and control traffic on an end-point host," *Alexandria Engineering Journal*, vol. 3, no. 55, 2016.

[27] J. Charzinski, "HTTP/TCP connection and flow characteristics," *Performance Evaluation*, vol. 42, no. 2, pp. 149–162, 2000.