Hindawi
Mobile Information Systems
Volume 2022, Article ID 6743070, 13 pages
https://doi.org/10.1155/2022/6743070

Research Article

Implementation and Design of a Zero-Day Intrusion Detection and Response System for Responding to Network Security Blind Spots

Won-Seok Choi,[1] **Si-Young Lee**,[2] **and Seong-Gon Choi**[3]

[1]*Research Institute for Computer and Information Communication (RICIC), Chungbuk National University, Chungdae-ro 1, Seowon-Gu, Cheongju, Republic of Korea*
[2]*Xabyss Corp., Seongnam-si, Republic of Korea*
[3]*College of Electrical and Computer Engineering, Chungbuk National University, Chungdae-ro 1, Seowon-Gu, Cheongju, Republic of Korea*

Correspondence should be addressed to Seong-Gon Choi; choisg@chungbuk.ac.kr

Received 25 January 2022; Revised 10 March 2022; Accepted 17 March 2022; Published 8 April 2022

Academic Editor: Kuo-Hui Yeh

We propose a zero-day intrusion detection and response system (ZDRS) for responding to network security blind spots. An existing detection and response system for the analysis of zero-day attacks uses a full-packet storage method; thus, the longer the time required to recognize a zero-day attack, the higher is the packet storage capacity and inspection cost. To solve the storage capacity and inspection cost problems, we design an architecture for ZDRS for a retroactive security check (RSC) using a first-N packet storage method. For fast verification of the RSC result, we propose a drill-down session metadata searching algorithm using session and flow metadata. The ZDRS comprises a network processing unit and a security processing unit. The ZDRS network processing unit generates metadata for the RSC verification and efficiently stores packets using the first-N packet storage method. The ZDRS security processing unit performs the RSC and RSC verification using the drill-down session metadata searching algorithm. For ZDRS performance analysis, we implemented ZDRS and analyzed the storage efficiency, detection efficiency, and detection speed of ZDRS at the campus level. As a performance analysis result of implementation, the amount of data storage decreased from 3.4 terabyte to 62 gigabyte compared to the full-packet storage method by 1.82%, and storage efficiency increased by 54.84 times. Furthermore, the detection rate of 99.55% based on the first 5-kilobyte size compared to the full-packet storage method was confirmed.

1. Introduction

Information and communication technology (ICT) has been rapidly developing into new network types such as digital twin (DT), Internet of Things (IoT), cloud computing, and vehicle networks. The appearance of new networks and technologies creates new applications and increases network traffic. With developments in ICT, new security requirements are required, and new types of attacks have also increased. However, the response to new and unknown network attacks has no choice but to react reactively [1–8].

Keeping up with hackers' efforts for attacks that mimic normal behavior is difficult. There is no perceptible manner to prevent hackers from attacking blind spots before the security detection rule is implemented. Therefore, to respond to such blind spots, a general procedure for recognizing, analyzing, and responding to hacker attacks, creating rules, and applying rules is necessary [9]. However, the application of the new detection rule only responds to attacks after that point in time, and checking for the attacks

previously received is not possible. For example, for an attack that leaks user information, analyzing which user information has been leaked is necessary in order to prepare future countermeasures. Therefore, analyzing the data from the point of attack until the detection rule is applied is necessary.

As mentioned above, a zero-day attack refers to an attack that occurs during the time from the creation of a new/variant security threat until the detection technology for blind spots is provided. When a new security threat occurs, it takes physical time for a new security rule or technology to be recognized and dealt with. This time is called the zero-day attack period, and a method that can quickly and efficiently detect and analyze attacks during the zero-day attack period is required.

Several studies have been conducted to counter zero-day attacks. Most research on zero-day attacks has focused on detecting security threats in the past. It can be classified into provenance tracking methods [10], detection method for software defined network (SDN) [11], signature-based detection methods [12, 13], machine learning-based method [14–16], anomaly detection methods [14–20], and specification-based detection methods [20–23]. However, the detection of a zero-day attack requires stored data during the zero-day attack period.

The detection of the zero-day period is performed through forensic analysis of all packets stored during the zero-day period. It stores all incoming packets for a certain period and performs forensic analysis on all stored packets using the abovementioned detection method-related studies [24–27].

For blind spots, inspection through forensic data analysis requires storing all packets during the zero-day attack period and subsequently checking and analyzing all packets. However, as the zero-day attack period increases, the storage capacity and cost increase rapidly. In addition, the analysis time increased.

Therefore, in this study, we classify and define blind spots for zero-day attacks. We propose a zero-day intrusion detection and response system (ZDRS) for coping with blind spots that can be solved. The ZDRS solves the problem of existing forensic storage by drastically reducing the storage cost through the first-N packet storage method.

We propose a recheck solution for blind spots using a system architecture for a regression security check, a metadata generation algorithm, and a metadata search algorithm. We solve the problem of existing forensic storage by drastically reducing the storage cost through the first-N packet storage method in the ZDRS. In addition, we analyze the number of detections, detection rate, storage efficiency, and detection speed through a campus-level implementation on the ZDRS.

The main contributions of this paper are as follows:

(i) We classify and define the type of blind spot

Existing studies have considered the security blind spot as one area without classification. However, we classified them into three categories to respond to more precise security blind spots. In particular, security blind spots were classified into three types: operational security blind spots, temporal security blind spots, and unknown security blind spots according to the timing of security responses. Existing studies have tried to solve the unknown security blind spot. However, we introduce the threats of operational security blindness and time security blindness and present solutions for resolving operational security blindness and security blindness in time

(ii) We present solutions for resolving operational security blind spots and time security blind spots

One problem of the solution to solve the operational security blind spot and temporary blind spot is packet storage issue. In existing researches, packets are inspected through a forensic method, so all packets for a certain period are required. However, the forensic method increases cost for packet storage because the longer the period to be inspected and the greater the flow of data, the more packets need to be stored. To solve this problem, we design an architecture for the ZDRS using the first-N packet storage method. Moreover, we propose the RSC method using the RSC algorithm for intrusion detection in a zero-day period and propose a drill-down session metadata searching algorithm and metadata generation method to support the correct verification of the RSC.

(iii) We verify the proposed ZDRS through implementation test

In real network environments where various protocols coexist, implementation is performed in a campus network to analyze the accuracy and efficiency of the ZDRS. In order to test in the real network environment, it was approved by Chungbuk National University, and we collected data at the egress point of the Chungbuk National University network. We also analyze the ZDRS test results based on the collected data.

The remainder of this paper is organized as follows. We review the related work in Section 2. We classified the blind spots in Session 3. The ZDRS is introduced in Section 3.2. We provide the implementation results of the ZDRS in Section 4, , and the concluding statements are presented in Section 5.

2. Related Work

In this section, we introduce research related to zero-day intrusion detection as shown in Table 1, such as the provenance tracking method, detection method of a software-defined network, signature-based detection method, anomaly detection method, and specification-based detection method.

Sun et al. proposed a provenance tracking method that is a probabilistic approach and implemented a prototype system ZePro for zero-day attack path identification. They used Bayesian networks to identify the zero-day attack paths. To capture the zero-day attack, a dependency graph called an

Table 1: Classification of related work on zero-day attacks.

Category	Reference
Provenance tracking method	[10]
Detection architecture of a software-defined network	[11]
Signature-based detection method	[12, 13]
Machine learning-based detection method	[14–16]
Anomaly detection method	[14–20]
Specification-based detection method	[21–23]

object instance graph is first built as a supergraph by analyzing system calls. To further reveal the zero-day attack paths hidden in the supergraph, the system builds a Bayesian network based on the instance graph [10].

Al-Rushdan et al. proposed a detection method designed and implemented for SDN using a modified sandbox tool called Cuckoo. The provenance tracking method is a probabilistic approach that implements a prototype system ZePro for zero-day attack path identification [11].

Hindy et al. and Bherde and Pund proposed a signature-based detection method [12, 13].

Hindy et al. proposed an autoencoder implementation for detecting zero-day attacks. CICIDS2017 and NSL-KDD were used for the evaluation. Hindy et al.'s model benefits greatly from autoencoder encoding–decoding capabilities [12].

Bherde and Pund proposed a technique for detecting zero-day attacks by using signature-based and knowledge-based methods. This system is a combination of signature-based and knowledge-based systems. [13].

Patidar and Khanderwal, Mirsky et al., and Bovenzi et al. proposed machine learning-based detection method [14–16]. In general, machine learning-based detection methods are included in anomaly detection methods that use machine learning as a method to find anomaly packets.

Patidar and Khanderwal proposed a solution based on the detection of zero-day malware using machine learning and artificial intelligence. They made an idea-level proposal, and the performance of the proposed method has not verified [14].

Mirsky et al. introduce a kitsune and evaluate its performance. The kitsune is a neural network-based network intrusion detection system which has been designed to be efficient and plug-and-play. It accomplishes this task by efficiently tracking the behavior of all network channels,and by employing ensemble of autoencoders for anomaly detection. They discussed the framework's online machine learning process in detail and evaluated it in terms of detection and runtime performance [15].

Bovenzi et al. proposed a hierarchical hybrid intrusion detection (H2ID), a two-stage hierarchical network intrusion detection approach. The H2ID performs anomaly detection via a novel lightweight solution based on a multimodal deep autoencoder and attack classification, using soft-output classifiers. The attack classification can use any machine/deep learning-based supervised classifier [16].

Patidar and Khanderwal, Mirsky et al., Bovenzi et al., Innab et al., Duessel et al., Aygun and Yavuz, and Bostani and Sheikhan proposed an anomaly detection method [14–20].

Innab et al. proposed a hybrid system between an anomaly-based detection system and a honeypot to detect a zero-day attack, consequently to integrate both approaches in one hybrid model as enhanced solution of detecting the zero-day attack that may occur in the system [17].

Duessel et al. proposed a detection method for zero-day attacks using context-aware anomaly detection at the application layer. They presented a new data representation, called cn-grams, that allows to integrate syntactic and sequential features of payloads in a unified feature space and provides the basis for context-aware detection of network intrusions. Also, they conducted experiments on both text-based and binary application-layer protocols which demonstrate superior accuracy on the detection of various types of attacks over regular anomaly detection methods [18].

Aygun and Yavuz proposed a network anomaly detection method with stochastically improved autoencoder-based models, to detect zero-day attacks with high accuracy. The key factor of their models is the threshold value which was determined using a stochastic approach rather than the approaches available in the current literature. Their models were tested using the KDDTest+ dataset contained in NSL-KDD, and their research achieved an accuracy of 88.28% and 88.65%, respectively [19].

Bostani and Sheikhan proposed a hybrid of anomaly-based and specification-based IDS for Internet of Things using unsupervised OPF based on the MapReduce approach. Specifically, the specification-based intrusion detection agents that are located in the router nodes analyze the behavior of their host nodes and send their local results to the root node through normal data packets. In addition, an anomaly-based intrusion detection agent that is located in the root node employs the unsupervised optimum-path forest algorithm for projecting clustering models by using incoming data packets. In experimental results, their method showed achieve true positive rate of 76.19% and false positive rate of 5.92% when both sinkhole and selective-forwarding attacks were launched [20].

Bostani and Sheikhan, Siu and Panda, Althubaity et al., and Altaf and Majeed proposed a specification-based detection method [20–23].

Siu and Panda proposed a specification-based detection method for attacks in a multiarea system. They described the implementation of a three-area system model. Also, they assessed the risk and devise several intrusion scenarios. Specifically, they injected false data into the frequency measurement and Automatic Generation Control (AGC) signals. And then, they developed a rule-based method to detect anomalies at the system-level [21].

Althubaity et al. proposed a hybrid specification-based intrusion detection system for rank attacks in 6TiSCH networks. Specifically, they proposed a hybrid specification-based intrusion detection system (IDS) that consists of centralized and distributed modules installed on the sink and RPL nodes, respectively, to prevent nodes from selecting an intruder as their successors. This method also eliminates

intruders' chances of becoming a time source and disrupts the synchronization of 6TiSCH networks [22].

Altaf and Majeed proposed a specification-based intrusion detection model for OLSR. The specification-based approach analyzed the protocol specification of an ad hoc routing protocol to establish a finite-state-automata (FSA) model that captures the correct behavior of nodes supporting the protocol. Then, this method extracted constraints on the behavior of nodes from the FSA model. Thus, this approach reduced the intrusion detection problem to monitoring the individual nodes for violation of the constraints [23].

These studies have been conducted to counter zero-day attacks. Most research on zero-day attacks has been focused on detecting security threats in the past. However, the detection of a zero-day attack requires stored data during the zero-day attack period. There are no studies on how to store zero-day data.

These studies used forensic analysis on all packets stored during the zero-day period or perform for real-time detection. Consequently, as the zero-day attack period increases, the storage capacity and cost increase rapidly. In addition, the analysis time is increased. Also, these studies do not consider how to store and retain data during the zero-day period.

Therefore, we propose a ZDRS to cope with blind spots that can be solved. The ZDRS solves the problem of existing forensic storage by drastically reducing the storage cost through the first-N packet storage method.

3. Zero-Day Intrusion Detection and Response System (ZDRS)

In this section, we first classify and define the type of blind spot. Subsequently, the architecture of the ZDRS is introduced, which is based on a first-N packet storage method, RSC algorithm, and drill-down metadata searching algorithm. In addition, for metadata generation and packet storage, the method of session metadata generation, first-N packet storage method, RSC algorithm, and drill-down searching algorithm are described.

3.1. Blind Spot. Figure 1 shows the blind spots of the network security area. t_1, t_2, t_3, and t_x denote points of time of event related to intrusion. Particularly, t_1, t_2, and t_3 are points of time where detection information is provided. t_x is the point of time where a new intrusion occurred without it being recognized. This intrusion may not be a zero-day intrusion. However, we consider for case of t_x is a zero-day intrusion.

t_1 is the point of time that provided detection information for any intrusion recognized before t_1.

t_2 is the point of time when detection information is provided regarding the intrusion of t_x. It means that zero day ends for the threat at t_x.

t_3 means the situation that detection information is not provided yet for the intrusion of t_x. It means that zero day does not end for the threat at t_x.

We classify security blind spots into three types: operational blind spots, temporal blind spots, and unknown blind spots.

An operational blind spot is defined as a blind spot that can detect an intrusion due to the presence of detection information; however, it was not recognized by mistake.

In Figure 1, detection information was released at the point of time t_1 for any intrusion that occurred before t_1, but there is a delay for reflecting the new detection rule or an operator's mistake. In the operational blind spot, detection rules have already been provided, so confirming an intrusion for stored past data is necessary. At this point of time, the important thing is a time of stored past data. The longer time is saved, the more data needs to be saved.

A temporal blind spot is defined as a blind spot where the intrusion occurred in the past, but detectable information is not recognized because that information is released today.

In Figure 1, a new intrusion occurred in t_x, but it was not recognized. In t_2, the detection rule for the intrusion was provided. Therefore, performing inspection on past data using the detection rule provided in t_2 is necessary. As for the temporal blind spot, the number of packets that need to be inspected increases as the time increases when the detection rule is delayed. Therefore, it is important how quickly detection rules are provided and applied, or how long period packets are stored for a zero-day attack period.

Unknown blind spots are defined as blind spots that have not been recognized because intrusion has occurred in the past, and detectable information has not yet been identified. In Figure 1, it is a situation corresponding to t_3, indicating a situation wherein not only the release of detection rules but also the intrusion was not recognized. In other words, because they do not even know that there was an attack, they cannot respond to anything.

Extensive research is being conducted to solve blind spots. However, most research on blind spots has been conducted for unknown blind spots. The unknown blind spot is an area that cannot be solved in the current system, which has no choice but to reactively respond to unknown network attacks. Therefore, research on unknown blind spots is being used as an abnormal behavior-based method. To fundamentally solve the unknown blind spot, studying a different method from the current mechanism is necessary. Operational blind spots are not a technical research area related to network attacks but rather a management tool research area. A study on the temporal blind spot used forensic data analysis to check packets during the zero-day period.

Most studies involving the blind spot are studies on the detection method, as mentioned in Section 2. However, the common problem with solutions is the storage space.

After recognition of the intrusion, a detection check should be performed on the stored packet using a released detection rule. However, as the time for intrusion recognition and provision of detection rules increases, the number of stored packets increases, and it takes considerable time to perform a check on the increased amount of data.

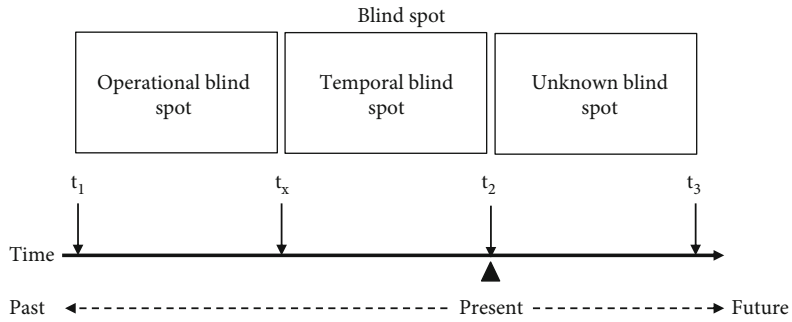

Figure 1: Classification of blind spot.

In general, the method of performing detection and checking for intrusion uses a forensic inspection method that stores all packets and checks all packets [24–27].

However, to solve the common storage problem of blind spots, we use the first-N packet storage method to reduce the packet storage capacity while maintaining a certain level of detection rate.

3.2. Architecture of ZDRS. The main purpose of the ZDRS is to reduce the amount of packet storage and to perform and verify fast RSC while maintaining a high detection rate in detecting and responding to zero-day attacks.

Figure 2 shows the architecture of the ZDRS to achieve the abovementioned purpose of the ZDRS. The ZDRS is generally located at the edge between the local area network (LAN) and the gateway.

The ZDRS is composed of a ZDRS network processing unit and a ZDRS security processing unit.

The ZDRS network processing unit performs packet capture, general packet processing, and flow and session metadata generation. This creates metadata to be used for RSC verification. In addition, it performs the operation of a general network processing unit.

The ZDRS network processing unit is composed of the following:

(i) *Packet Capture Interface Module*. It captures packets in the network.

(ii) *Packet-Processing Module*. It performs conventional packet processing.

(iii) *ZDRS Flow Processing Module*. It creates flow metadata of packets.

(iv) *ZDRS Session Processing Module*. It creates session metadata of packets.

The ZDRS network processing unit is a software module that operates in a host system in the case of a 1 Gbps bandwidth and operates in a separate hardware smart network interface card (NIC) if the bandwidth is above 10 Gbps.

For bandwidths of 10 Gbps, because a large amount of processor resources is required for packet capture, packet processing, and storage, additional hardware smart NIC is installed to satisfy the performance of the ZDRS network processing unit.

The ZDRS security processing unit performs intrusion detection and response to zero-day attacks that store packets, performs RSC, and verifies the results of the RSC.

The ZDRS security processing unit is composed of following:

(i) *Storage Management Module*. This module is used to set the configuration of the first-N packet storage method.

(ii) *Retroactive Security Check Module*. The RSC is performed, and the RSC result is verified using the drill-down session metadata searching algorithm.

(iii) *External Interface Module*. This module is used to set a retroactive security check module and to confirm results from the outside.

(iv) *Storage Device*. It is conventional storage I/O device.

The ZDRS captures incoming packets and performs general packet-processing procedures in the packet-processing module. In addition, according to the configuration of the first-N packet storage method set in the storage management module, the flow metadata and session metadata are created in the ZDRS flow processing module and the ZDRS session processing module. Then, the packets are stored in the storage device.

Thereafter, when a zero-day attack is recognized and a detection rule is generated, the corresponding rule is updated to the retroactive security check module by the external interface module. The retroactive security check module performs RSC for stored packets based on the updated detection rule. Then, the retroactive security check module verifies the RSC result using the drill-down session metadata search algorithm based on session metadata and flow metadata.

3.3. Metadata Generation and Packet Storage. In this section, we introduce the metadata generation method and the first-N packet storage method.

3.3.1. Metadata Generation. For verification of the RSC result, we used a drill-down searching algorithm. The drill-

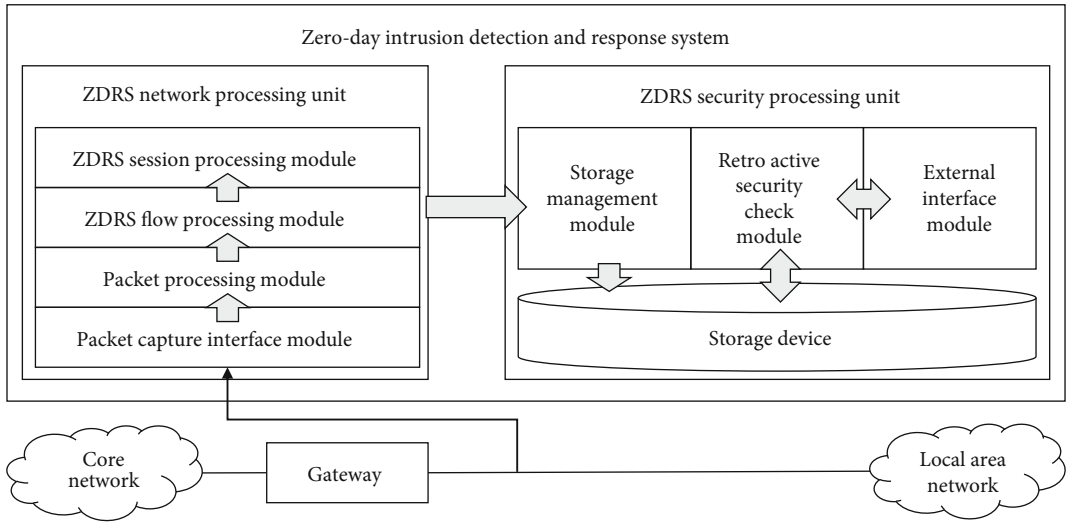

Figure 2: Architecture of the ZDRS.

```
struct flow _metadata
{
    uint64_t id; //Flow ID
    five_tuple tuple; //5-tuple information
    uint32_t packet_count; // The number of packet of the flow
    timestamp start_time; //Flow start time
    timestamp end_time; //Flow closed time
    tcpflag tcp_flag; //TCP flag information of the flow
    uint16_t flow_state; //Flag state information: active or closed
    uint64_t session_id; //Session id of the flow
    uint64_t bytes; //Total packet size(bytes) of the flow
}
```

Figure 3: Flow metadata structure.

```
struct session_metadata
{
    uint64_t id; //Session ID
    five_tuple tuple; //5-tuple information
    uint32_t flow_count; //The number of flow of the session
    uint32_t packet_count; //The number of packet of the session
    timestamp start_time; //Session start time
    timestamp end_time; //Session closed time
    tcpflag tcp_flag; //TCP flag information of the session
    uint16_t session_state; //Session state information: active or closed
    uint64_t bytes; //Total packet size (bytes) of the session
}
```

Figure 4: Session metadata structure.

down searching algorithm uses the flow metadata and session metadata for high-speed searching. We consider that the same flow is a set of packets having the same 5-tuple information within an active timeout and an inactive timeout. Therefore, to classify the flow, the active timeout and inactive timeout or TCP FIN flag of the packet is used. We consider that the same session is a set of packets with the same 5-tuple information until the application starts and ends communication. Therefore, to classify the session, we use the inactive timeout or TCP FIN flag of a packet. This criterion for classifying sessions and flows is used for packet storage in the first-N packet storage method and for generating the metadata. In general, the start and end of a session are recognized by analyzing the packet header information. Therefore, to reduce the burden of recognizing a session by analyzing all packets, we generate metadata and perform a metadata-based quick search.

First, we create the flow metadata from a packet. Figure 3 shows the structure of flow metadata. The flow metadata comprises an *id*, *tuple*, *packet_count*, *start_time*, *end_time*, *tcp_flag*, *flow_state*, *session_id*, and *bytes*.

The *id* is a flow identifier that is randomly allocated a value to not be duplicated for the classification of flows. The *tuple* is 5-tuple information comprising a source IP address, destination IP address, source port number, destination port number, and protocol. The *packet_count* denotes the number of packets in the flow. The *start_time* is the start time of the flow. The *end_time* is the closed time of the flow. The *tcp_flag* is the TCP flag of the flow. The *flow_state* is flag state information on whether the flag is in an active or a closed state. The *session_id* is the session id of the flow, and the *bytes* denote the total packet sizes of the flow.

Session metadata is created using flow metadata. Figure 4 shows the structure of session metadata. The session metadata comprises an *id*, *tuple*, *flow_count*, *packet_count*, *start_time*, *end_time*, *tcp_flag*, *session_state*, and *bytes*.

The *id* is a session identifier that is randomly allocated a value to not be duplicated for the classification of sessions. The *tuple* is 5-tuple information comprising a source IP address, destination IP address, source port number, destination port number, and protocol. The *flow_count* is the number of flows in the session. The *packet_count* is the number of packets in each session. The *start_time* is the start time of the session, and the *end_time* is the closed time of the session. The *tcp_flag* is the TCP flag information for the session. The *session_state* is a session state information on whether the session is in an active or closed state, and the *bytes* denote the total packet sizes of the session.

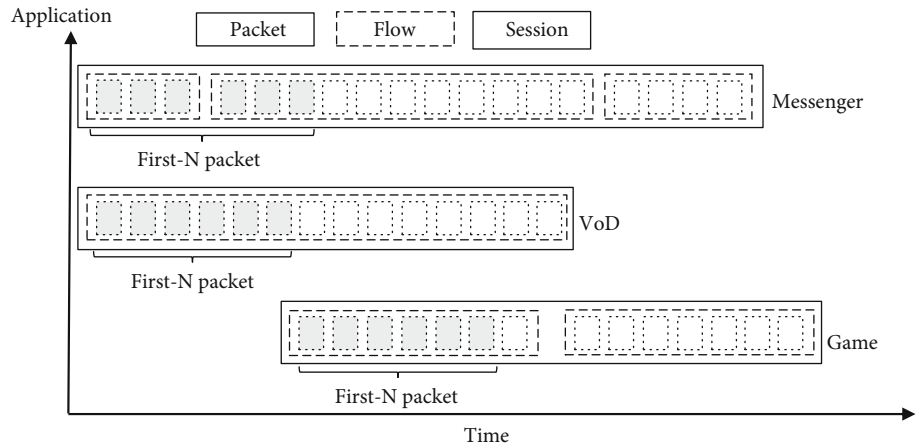

Figure 5: First-N packet storage method of ZDRS.

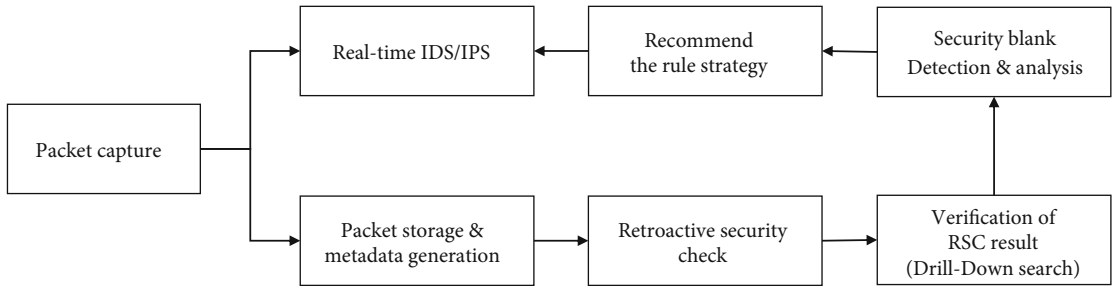

Figure 6: Operation of the RSC in the ZDRS.

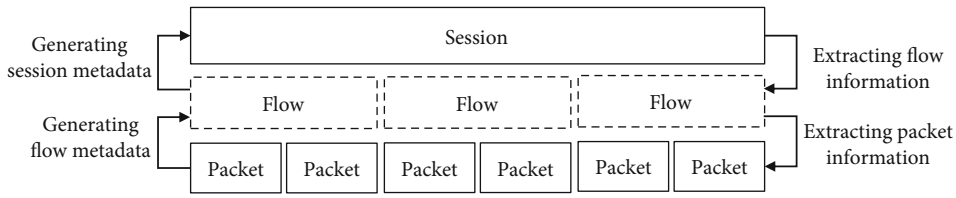

Figure 7: Drill-down search algorithm based on session and flow metadata.

The generated session metadata and flow metadata are used in the drill-down searching algorithm for fast packet search to verify the RSC result.

3.3.2. First-N Packet Storage Method. Figure 5 shows the first-N packet storage method of ZDRS, including examples of messengers, video on demand (VoD), and game applications [28]. The first-N packet storage method stores n-or n-sized packets for a session. Assuming that a session of a messenger application comprises 3 flows and a total of 18 packets, the initial 6 packets are stored to recognize the flow. In the case of VoD application, when a session is composed of 1 flow and a total of 14 packets, the initial 6 packets are stored. In the case of a game application, when a session comprises 2 flows and a total of 14 packets, the initial 6 packets are obtained. In other words, it stores the initial 6 packets for any session regardless of the application type and flow.

The length of the session differed depending on the application. In addition, the number of flows and packets included in each session is different. However, if the initial few packets can be checked for a session, it can confirm the network signature [29]. In Park et al.'s research, the optimal number of n packets was 5 [29]. However, the optimal number may vary depending on the network conditions and user requirements. The n unit can also be used as the number or length of packets to be stored. In particular, when used as the meaning of the length of a packet for a continuous syn attack, its characteristics can be better captured. Therefore, in this study, we implemented and tested its length.

When a detection rule for a zero-day attack is issued, the stored packets using the first-N packet storage method are checked by the RSC module in the ZDRS.

3.4. Retroactive Security Check. In this section, we introduce the RSC algorithm and the drill-down session metadata search algorithm in the RSC module. In the RSC module, the RSC operation and the drill-down session metadata

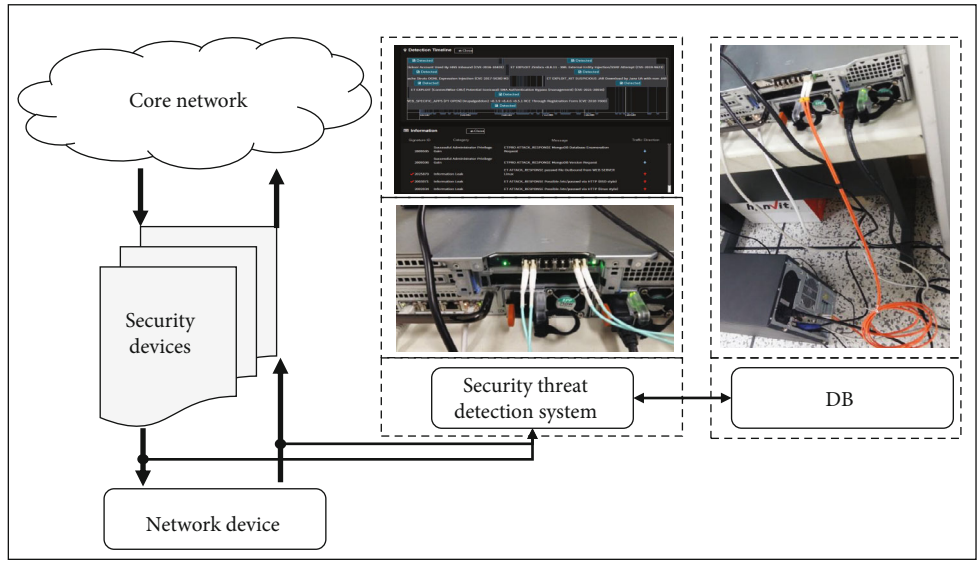

Figure 8: Implementation environment.

Table 2: Specifications of ZDRS security processing unit.

List	Specification
Processor	Intel Zeon Silver 4110 2.1 GHz (8 core; 16 thread)
Memory	16 GB (8 GB RDIMM ×2)
Operating system	Linux (Debian 8.9)
HDD	SAS 7.2 k RPM 48 TB (8 TB ×6)

Table 3: Specifications of ZDRS network processing unit.

List	Specification
Network processor	36-core 1.2 GHz Tile-Gx36 processor
Memory	DDR3 SODIMM type 8 GB
Interface	1/10 Gbps (SFP/SFP+) 4-port
PCIe type	PCIe Gen2 8-lane
Supported operating system	Linux: CentOS, Red Hat
Performance	Full 20 Gbps transfer to host

Table 4: Stored packets and storage rate of the ZDRS test.

First-N size	5 kB	10 kB	20 kB	30 kB	50 kB	100 kB	300 kB	1 MB	Full
Stored packets (GB)	62	85	128	159	209	291	445	662	3,400
Storage rate (%)	1.8	2.5	3.8	4.7	6.1	8.6	13.1	19.5	
Number of detections	13,882	13,885	13,890	13,895	13,895	13,900	13,900	13,902	13,944

searching algorithm were used to detect zero-day attacks in the past.

3.4.1. Operation of Retroactive Security Check. Figure 6 shows the operation of the RSC in ZDRS. The RSC conducts a check for false-negative packets for a zero-day period. When a packet is captured by the ZDRS in the network, the ZDRS conducts real-time IDS/IPS for all captured packets. Simultaneously, packets are stored according to the setting of the first-N storage method. Subsequently, the ZDRS flow and session metadata processing module create metadata before storing the packets.

In the ZDRS, the RSC module performs RSC to respond to operational blind spots and temporal blind spots. To respond to a temporal blind spot, the RSC module performs the RSC by updating a detection rule or confirming a zero-

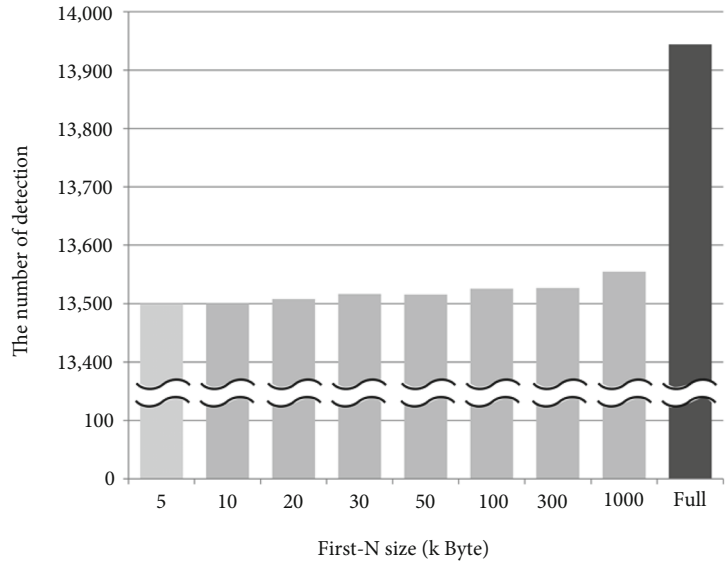

Figure 9: Number of detections of each first-N storage method versus the full-packet storage method.

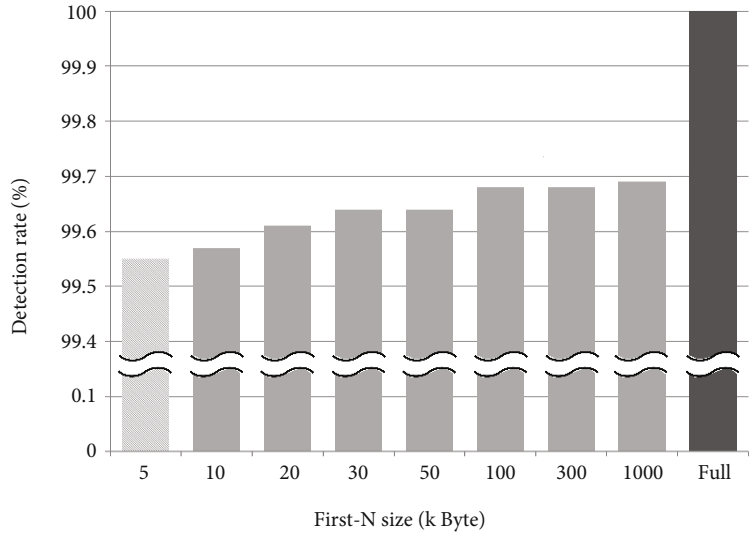

Figure 10: RSC detection rate for each first-N storage methods versus the full-packet storage method.

day attack. For an operational blind spot, the RSC module conducts periodic RSC according to the cycle set by the operator.

After the RSC is executed, the RSC execution result is verified through a drill-down search based on previously created metadata. Thereafter, a security check was performed through the continuous detection and analysis of the security blank. When a new rule strategy is created, it is reflected in the real-time IDS/IPS to remove the blind spots for a zero-day attack.

3.4.2. Drill-Down Session Metadata Searching Algorithm. Figure 7 shows the drill-down search algorithm based on metadata to quickly verify the RSC results. After the RSC module receives a search requirement owing to the predetermined detection rule or the updated detection rule, this module conducts a check for zero-day attacks in a blind spot. Subsequently, the drill-down search algorithm performs the following procedure to obtain false positives in the RSC module. The RSC module finds an objective session to compare false positive packets to session metadata by using the session metadata information. The RSC module then obtains an objective flow to compare the requested flow to the flow metadata. The RSC module subsequently obtains an objective packet to verify the RSC results to prevent false positive detection and then rechecks the objective packet.

The drill-down search algorithm does not search all packets or flows but instead searches for information in the sessions, flows, and packet order. Therefore, the effect of high-speed search was obtained.

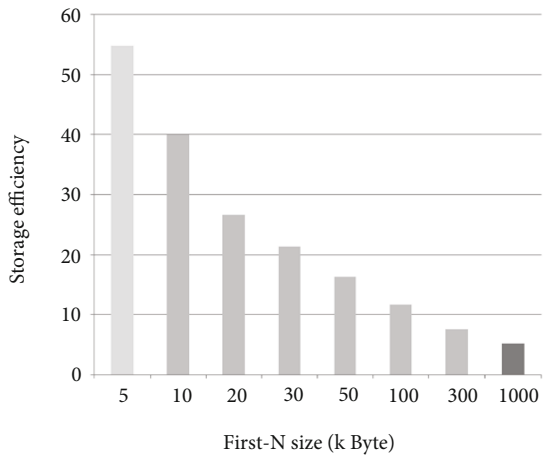

Figure 11: Storage efficiency for first-N storage methods versus the full-packet storage method.

4. Implementation

In this section, we introduce the implementation environment and the results of the proposed ZDRS. We analyzed the detection rate, storage efficiency, and RSC time in the ZDRS versus the full-packet storage method.

4.1. Implementation Environment. The implementation environment for the performance analysis of the ZDRS is described. Figure 8 shows the implementation environment at the campus level. The ZDRS was connected to the egress point of Chungbuk National University's local network to collect data. The collected data were approved by Chungbuk National University to be used only for this study.

In general, a campus-level network may consist of one or more LANs within the campus network and is connected to an external core network through a network device such as a router or gateway. Here, when connecting to the core network, it passes through a security device to block security threats. For test of the ZDRS, we mirror packets between the security device and the network device to the ZDRS. In addition, a database is configured to store rule sets, inspection history, and traffic information. We used the emerging threat Suricata open rule set for the detection rule. The flow inactive timeout, flow active timeout, and session inactive timeout were set as 15 s, 1800 s, and 1860 s, respectively.

Table 2 lists the specifications of the ZDRS security processing unit. The processor is an 8-core 16-thread Intel Zeon Silver 4110 with 16 gigabyte (GB) RDIMM memory. It operates on the Debian 8.9 Linux, and the HDD is a 48-terabyte (TB) SAS (7.4 k RPM).

The ZDRS security processing unit is a software module, but if it uses a 10 Gbps interface, it has to use an external hardware network processing unit; otherwise, if it uses a 1 Gbps interface, it can be implemented on the host system. In this implementation, we implement the ZDRS using an external hardware network processing unit to use a 10 Gbps interface.

Table 3 lists the hardware specifications of the ZDRS network processing unit. The network processor was a Tile-G36 processor.

Memory is a DDR3 type and is 8 GB. It has an SFP+4-port network interface. It uses 8-lane of the PCIe Gen2. It operates on CentOS or Red Hat. These hardware specifications make it transfer packets of 20 Gbps to the host. These hardware specifications allow packets of 20 Gbps to be transmitted to the host.

4.2. Implementation Result. We collected the raw 3.4 TB PCAP data during 9 hours and confirmed the storage and detection rates of the collected data in the ZDRS. And then, through replaying based on this full packet, we resaved the data as each first-N storage method. Using this sample data, the implementation results were compared with the results of the proposed first-N storage method and full-size storage method. In this implementation, the application classification method based on the port number was applied in all implementations to compare the results of the proposed first-N storage scheme with those of the full-size storage scheme.

In general, research on detection algorithms for zero-day attack suggests correct detection rate and false positive rate as experimental results. However, this paper does not propose a detection algorithm, but the ZDRS that solves the problems of the forensic storage method through the first-N packet storage method. Therefore, we used Suricata open rule set. That is, the purpose of this experiment is to confirm the detection rate of inspection for packets collected by the first-N method compared to the full-packet method. Therefore, true rate and false rate are not considered in this study.

Table 4 shows the stored packets, storage rate, and the number of detections of the ZDRS test. While the full-packet storage method stored and analyzed approximately 3.4 TB of data, the first 5-kilobyte (kB) size storage method stored and analyzed approximately 62 GB of data. The first 5 kB storage used approximately 1.8% storage space versus the full-packet storage method. Also, the number of detections increases as approaches the full packet.

Figure 9 shows the number of detections for each first-N storage method and the full-size storage method. The full-size storage scheme detected 13,944 cases, while the first 5 kB and 1 MB storage schemes detected 13,500 and 13,555 cases, respectively.

Figure 10 shows the detection rate of the first-N storage method and the full-packet storage method. We calculate the detection rate as follows using the number of detections in Table 4.

$$\text{Detection rate} = \frac{N_{df}}{N_{dn}} * 100. \quad (1)$$

N_{df} is the number of detections in full-packet storage method. N_{dn} is the number of detections in first-N storage methods. When the detection rate of the full-packet storage method was taken as 100%, the detection rates of the first 5 kB and 1 MB storage methods were 99.55% and 99.69%, respectively. The first-N storage methods are more than 99.55%, and there is little difference compared to the full-packet storage method. However, the difference between

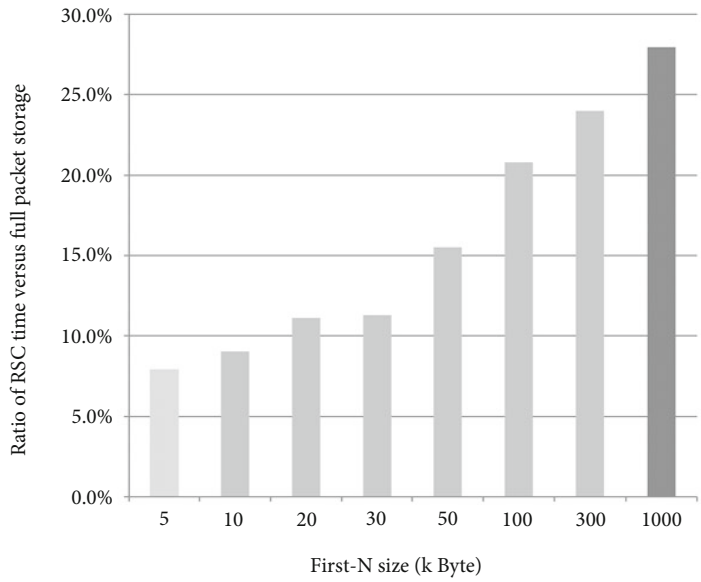

Figure 12: Ratio of RSC time for first-N storage methods versus full-packet storage method.

the first 5 kB size storage method and the first 1 MB size storage method was only 55 cases.

Figure 11 shows storage efficiency for the first-N size storage method versus the full-packet storage method. The result of the first 5 kB size showed that the data storage rate decreased approximately 55-fold compared with the full-packet method. Moreover, the result of the first 1 MB size showed that the data storage rate decreased approximately fivefold compared with the full-packet method.

Figure 12 shows the ratio of RSC time for first-N storage methods versus the full-packet storage method. The first 5 kB size storage method spent 7.9% RSC time versus the full-packet storage method. Moreover, the first 1 MB size storage method spent 28% RSC time versus the full-packet storage method. In other words, as the size of N decreases, the RSC time decreases.

To evaluate the performance of ZDRS, it is necessary to look at the all factors which are number of detections, detection rate, storage efficiency, and RSC time. For example, the first 5 kB storage method has 444 fewer detections than the full-packet method. However, the detection rate is only 0.45%, and the storage efficiency is 55 times higher. Moreover, the efficiency is only 7.59% of the time required to perform RSC in full-packet method. In other words, depending on the application target, it is better to take a small amount of first-N size in order to have the maximum storage efficiency, and it is better to take a large first-N size for a strict security check.

5. Conclusion

In this study, we designed and demonstrated a data storage and regression security check system for responding to network security blind spots. In detail, we propose a system structure for regression security inspection, a metadata generation algorithm, and a metadata search algorithm to provide a solution for operational blind spots and temporal blind spots. In addition, we used the first-N packet storage method to drastically reduce the storage cost. As a performance analysis result of implementation, the amount of data storage decreased from 3.4 TB to 62 GB compared to the full-packet storage method by 1.82%, and storage efficiency increased by 54.84 times. The detection rate of 99.55% based on the first 5 kB size compared to the full-packet storage method was confirmed. That is, depending on the application target, it is better to take a small amount of first-N size in order to have the maximum storage efficiency, and it is better to take a large first-N size for a strict security check. In conclusion, it was confirmed that even if the packet is saved with ZDRS's first-N technology, it can be sufficiently utilized for security check.

In this paper, we used the session-based first-N storage method. In a future study, we plan to apply the flow-based first-N storage method and compare the storage efficiency and accuracy with the session-based first-N storage method. This study used the session-based first-N method. However, the flow-based first-N may have different inspection volume and detection rate than the session-based first-N and may have different characteristics depending on the type of attack. Therefore, we plan to study the flow-based first-N storage method. In addition, we will collect and test more samples at various sites—research institutes and companies.

Data Availability

No data were used to support this study.

Conflicts of Interest

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

Authors' Contributions

Won-Seok Choi contributed to the conception of the main idea. Won-Seok Choi and Si-Young Lee contributed to the writing of manuscript and preparation of all figures. Won-Seok Choi and Si-Young Lee contributed to the implementation and test. Won-Seok Choi and Seong-Gon Choi contributed to the analysis of the results. Seong-Gon Choi contributed to the supervision and conceptualization.

Acknowledgments

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2020R1A6A1A12047945).

References

[1] V. Hassija, V. Chamola, V. Saxena, D. Jain, P. Goyal, and B. Sikdar, "A survey on IoT security: application areas, security threats, and solution architectures," *IEEE Access*, vol. 7, pp. 82721–82743, 2019.

[2] H. Tabrizchi and M. K. Rafsanjani, "A survey on security challenges in cloud computing: issues, threats, and solutions," *The Journal of Supercomputing*, vol. 76, no. 12, pp. 9493–9532, 2020.

[3] H. Tabrizchi and M. K. Rafsanjani, "Digital twin networks: a survey," *IEEE Internet Of Things Journal*, vol. 8, no. 18, pp. 13789–13804, 2021.

[4] K. B. Kelarestaghi, M. Foruhandeh, K. Heaslip, and R. Gerdes, "Intelligent transportation system security: impact-oriented risk assessment of in-vehicle networks," *IEEE Intelligent Transportation Systems Magazine*, vol. 13, no. 2, pp. 91–104, 2019.

[5] N. Sameera and M. Shashi, "Deep transductive transfer learning framework for zero-day attack detection," *ICT Express*, vol. 6, no. 4, pp. 361–367, 2020.

[6] A. Blaise, M. Bouet, V. Conan, and S. Secci, "Detection of zero-day attacks: an unsupervised port-based approach," *Computer Networks*, vol. 180, article 107391, 2020.

[7] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Unsupervised algorithms to detect zero-day attacks: strategy and application," *IEEE Access*, vol. 9, pp. 90603–90615, 2021.

[8] A. Ghosal and M. Conti, "Security issues and challenges in v2x: a survey," *Computer Networks*, vol. 169, article 107093, 2020.

[9] R. Singh, H. Kumar, R. K. Singla, and R. R. Ketti, "Internet attacks and intrusion detection system: a review of the literature," *Online Information Review*, vol. 41, no. 2, pp. 171–184, 2017.

[10] X. Sun, J. Dai, P. Liu, A. Singhal, and J. Yen, "Using Bayesian networks for probabilistic identification of zero-day attack paths," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2506–2521, 2018.

[11] H. Al-Rushdan, M. Shurman, S. H. Alnabelsi, and Q. Althebyan, "Zero-day attack detection and prevention in software-defined networks," in *2019 International Arab Conference on Information Technology (ACIT)*, pp. 278–282, Al Ain, UAE, 2019.

[12] H. Hindy, R. Atkinson, C. Tachtatzis, J.-N. Colin, E. Bayne, and X. Bellekens, "Utilising deep learning techniques for effective zero-day attack detection," *Electronics*, vol. 9, no. 10, p. 1684, 2020.

[13] G. P. Bherde and M. Pund, "Technique for detecting zero day attack by using signature based and knowledge based method, International Journal of Scientific Research in Science," *Engineering and Technology*, vol. 4, pp. 649–652, 2018.

[14] C. Patidar and H. Khandelwal, "Zero day attack detection using machine learning techniques," *IJRAR*, vol. 6, no. 1, pp. 1364–1367, 2018.

[15] Y. Mirsky, T. Doitshman, Y. Elovici, A. Shabtai, and A. Kitsune, "An ensemble of autoencoders for online network intrusion detection," 2018, https://arxiv.org/abs/1802.09089.

[16] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescape, "A hierarchical hybrid intrusion detection approach in IoT scenarios, in proceedings of the GLOBECOM 2020, Taipei," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, pp. 1–7, Taipei, Taiwan, 2020.

[17] N. Innab, E. Alomairy, and L. Alsheddi, "Hybrid system between anomaly based detection system and honeypot to detect zero day attack," in *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pp. 1–5, Riyadh, Saudi Arabia, 2018.

[18] P. Duessel, C. Gehl, U. Flegel, S. Dietrich, and M. Meier, "Detecting zero-day attacks using context-aware anomaly detection at the application-layer," *International Journal of Information Security*, vol. 16, no. 5, pp. 475–490, 2017.

[19] R. C. Aygun and A. G. Yavuz, "Network anomaly detection with stochastically improved autoencoder based models," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, pp. 193–198, New York, NY, USA, 2017.

[20] H. Bostani and M. Sheikhan, "Hybrid of anomaly-based and specification-based ids for Internet of Things using unsupervised OPF based on MapReduce approach," *Computer Communications*, vol. 98, pp. 52–71, 2017.

[21] J. Y. Siu and S. K. Panda, "A specification-based detection for attacks in the multi-area system," in *IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society*, pp. 1526–1526, Singapore, 2020.

[22] A. Althubaity, H. Ji, T. Gong, M. Nixon, R. Ammar, and S. Han, "Arm: A hybrid specification-based intrusion detection system for rank attacks in 6TiSCH networks," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–8, Limassol, Cyprus, 2017.

[23] I. Altaf and I. Majeed, "A specification-based intrusion detection model for OLSR, international journal of computer science," *Engineering Technology*, vol. 8, pp. 273–282, 2017.

[24] M. Ali, S. Shiaeles, N. Clarke, and D. Kontogeorgis, "A proactive malicious software identification approach for digital forensic examiners," *Journal of Information Security and Applications*, vol. 47, pp. 139–155, 2019.

[25] K. Demertzis, P. Kikiras, N. Tziritas, S. L. Sanchez, and L. Iliadis, "The next generation cognitive security operations center: network flow forensics using cybersecurity intelligence," *Big Data and Cognitive Computing*, vol. 2, no. 4, p. 35, 2018.

[26] P. Prasad, N. Sowmya, K. Reddy, and P. Bala, "Introduction to dynamic malware analysis for cyber intelligence and forensics," *International Journal of Mechanical Engineering and Technology*, vol. 9, no. 1, pp. 10–21, 2018.

[27] A. Nisioti, G. Loukas, A. Laszka, and E. Panaousis, "Data-driven decision support for optimizing cyber forensic investigations," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2397–2412, 2021.

[28] S. M. Ryu, S. Y. Lee, C. W. Ryu, and S. G. Choi, "A method of the first-n packets storage by using the advanced-PCA, Contemporary," *Engineering Sciences*, vol. 9, no. 14, pp. 647–653, 2016.

[29] J.-S. Park, S.-H. Yoon, and M.-S. Kim, "Performance improvement of signature-based traffic classification system by optimizing the search space," *Journal of Internet Computing and Services*, vol. 12, no. 3, pp. 89–99, 2011.