Future Generation Computer Systems 100 (2019) 811–825

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Unsupervised multi-stage attack detection framework without details on single-stage attacks

Jinmyeong Shin [a], Seok-Hwan Choi [a], Peng Liu [b], Yoon-Ho Choi [a,*]

[a] School of Computer Science and Engineering, Pusan National University, Busan, 26241, Republic of Korea
[b] Pennsylvania State University, University Park, PA 16802, United States

HIGHLIGHTS

- Single-stage attacks classification without pre-defined knowledge.
- Correlation of suspicious flows using context between flows.
- Automatic multi-stage attack detection rule generation.
- Evaluation with large volume data to show the feasibility.

ARTICLE INFO

Article history:
Received 21 November 2018
Received in revised form 10 February 2019
Accepted 11 May 2019
Available online 16 May 2019

Keywords:
Network intrusion detection
Multi-stage attack
Cyber kill chain
Detection rule generation

ABSTRACT

Majority of network attacks currently consist of sophisticated multi-stage attacks, which break down network attacks into several single-stage attacks. The early multi-stage attack detection methods focused on describing the detection rule based on the occurrence sequence of each single-stage attacks. That is, such works assumed that after details on single-stage attack behavior are obtained from attack knowledge, attack semantics or attack statistical analysis, the detection rules can be generated from their possible occurrence sequence. However, their practical usage is limited due to the high false negative ratio while detecting multi-stage attack that consists of diverse combinations of single-stage attacks during the long time period. In this paper, we propose a new multi-stage attack detection framework, which consists of multi-stage attack detection rule generation phase and multi-stage attack detection phase. After comparing the incoming traffics with the generated multi-stage attack detection rules, various multi-stage attack patterns are detected without pre-observed details on the single-stage attack behavior. From DARPA LLS DDoS dataset, we show that all the possible multi-stage attack patterns are correctly detected. Also, from datasets in CTU-13 including the large volume of multi-stage attack patterns, we observe F_1-measure of 0.938 at maximum.

1. Introduction

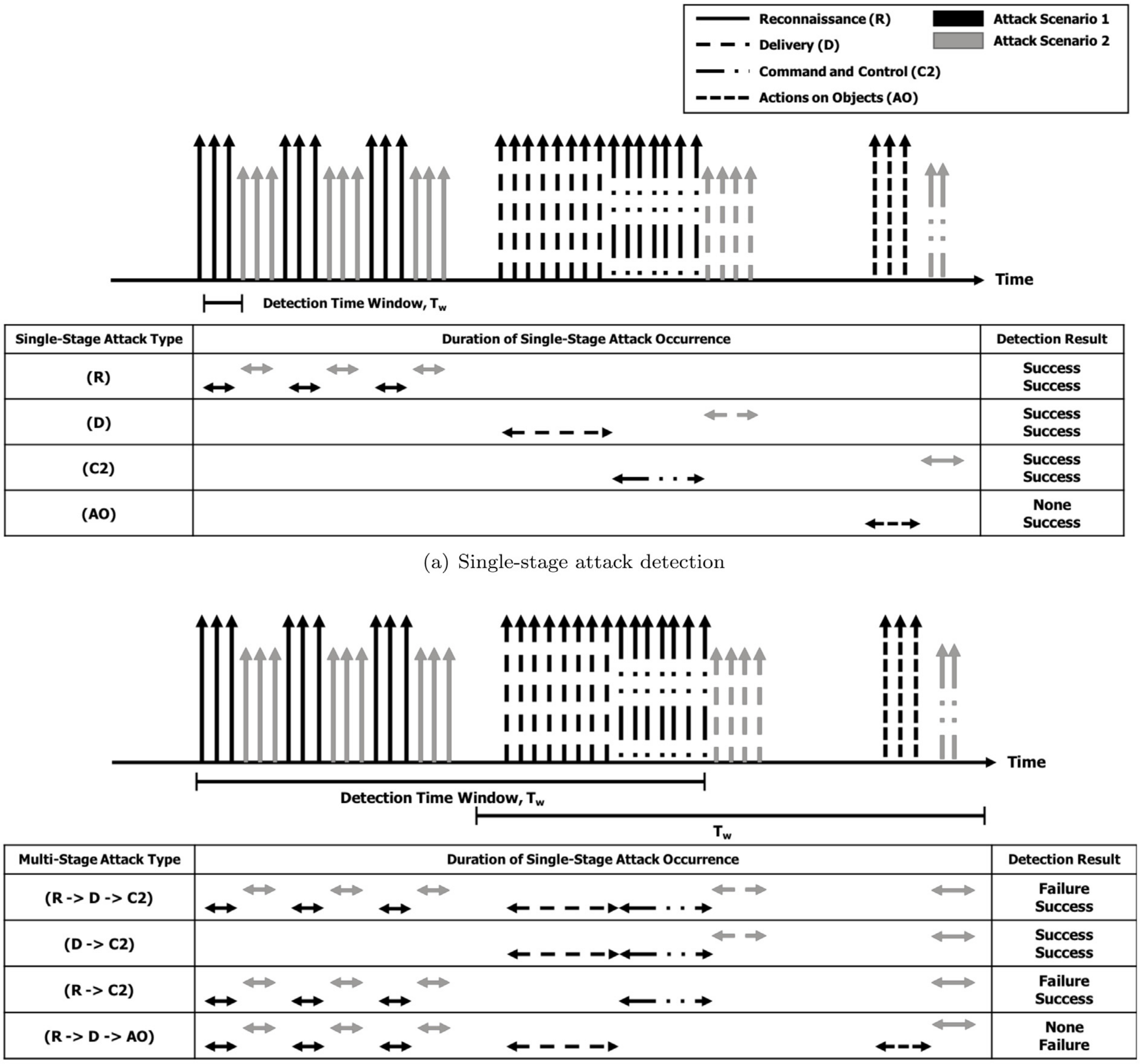
Majority of current malicious behavior in network consists of sophisticated multi-stage attacks. According to Lockheed Martin [1], these multi-stage attacks, also called cyber kill chain, break down into the following single-stage attack types: (1) reconnaissance; (2) Weaponization; (3) Delivery; (4) Exploitation; (5) Installation; (6) Command and Control (C2); and (7) Actions on Objectives. While most inline or endpoint protection products have the capability of detecting a single-stage attacks, their capability lacks of detecting a multi-stage attack.

The Fig. 1 shows the difference between single-stage attack and multi-stage attack detection. To exploit the target system, a single-stage attack conducts simple and indiscriminate attack trails in a short time period. However, because the single-stage attack shows indiscriminate similar behavior many times and leaves the corresponding attack evidences from a lot of trails in a short time period, most inline or endpoint protection products can easily identify it as shown in Fig. 1(a).

Different from the single-stage attack detection, the multi-stage conducts a sophisticated attack for a long period of time compared to the single-stage attack as shown in Fig. 1(b). For example, to circumvent the conventional security configuration, the time periods of the multi-stage attack range from a few minutes to several months. Thus, to detect and counter the multi-stage attack properly, the administrator need to keep track and correlate single-stage attack alerts from different machines and attack scenarios, even if detecting each single-stage attack is not a difficult task. Thus, it is very difficult to detect multi-stage attacks without knowing knowledges on multi-stage attack scenarios in

[*] Corresponding author.
E-mail addresses: sinryang@pusan.ac.kr (J. Shin), danialsh@pusan.ac.kr (S.-H. Choi), pliu@ist.psu.edu (P. Liu), yhchoi@pusan.ac.kr (Y.-H. Choi).

https://doi.org/10.1016/j.future.2019.05.032
0167-739X/© 2019 Elsevier B.V. All rights reserved.



Single-Stage Attack Type	Duration of Single-Stage Attack Occurrence	Detection Result
(R)		Success Success
(D)		Success Success
(C2)		Success Success
(AO)		None Success

(a) Single-stage attack detection

Multi-Stage Attack Type	Duration of Single-Stage Attack Occurrence	Detection Result
(R -> D -> C2)		Failure Success
(D -> C2)		Success Success
(R -> C2)		Failure Success
(R -> D -> AO)		None Failure

(b) Multi-stage attack detection

Fig. 1. Characteristic comparison between single-stage attack detection and multi-stage attack detection.

advance. Recently, the multi-stage attacks with their own attack scenarios are evolved while maximizing the attack success probability by hiding their activity evidences, propagating themselves in the slow speed, and so on.

During a multi-stage attack, consisting of seven single-stage attack types, the following four single-stage attack types are commonly identified from the network traffic [1,2]:

- **Reconnaissance:** Target selection process, collecting information of reachable systems that have potential exploitable vulnerabilities. This process make the attack stealthy by preventing it from being attempted blindly.
- **Delivery:** Transfer source code, program or payloads for exploit to target system.
- **Command and control (C2):** Construct communication channel to command and control bots or target system.
- **Actions on objectives:** Achieving final goal, such as destroying the target system, getting confidential information, infecting another system which reachable and etc.

Since these four single-stage attack types cannot be captured or can be captured one or more times according to the network environment [3–5] or multi-stage attack scenario [6–12], their occurrence sequence also frequently varies. As a result, diverse combinations of single-stage attacks can exist. Since the complexity of multi-stage attack scenario is large, it is difficult to identify the multi-stage attack correctly without identifying a clear pattern of a particular kind of multi-stage attack in advance. For example, improper correlation with alerts on unrelated attacks belonging to other attack scenarios causes false detection. Thus, the detection accuracy on multi-stage attacks is practically low under the existence of multi-stage attack variants.

To increase the low detection accuracy, various detection methods using knowledge-based model [8,9], attack semantics [10,13,14] and statistical model [6–9] are proposed. However, since these models analyze the correlation of the alerts generated from signature-based intrusion detection system(IDS), they require pre-observed details on the single-stage attack activity. Thus, their performance is limited while detecting diverse combinations of single-stage attacks. To overcome such a limitation, recent security solutions such as Security Information and Event Management(SIEM) used specific domain knowledge obtained from diverse logs, honeypots [11] and Software-Defined

Network(SDN) [12] as an extra input attribute. However, since the correlation process requires the clear pre-defined rules for identifying single-stage attack and multi-stage attack, the comprehensive understanding of adversary's behavior pattern is required. Thus, without pre-observed details on single-stage attacks, their performance is also limited when detecting variants of multi-stage attacks.

The contribution of this paper can be summarized as follows: (1) We proposed a new framework for multi-stage attack detection. Different from the previous works, multi-stage attack rules are automatically generated without knowing pre-defined details on single-stage attack activities. Since there is no dependency on pre-defined details, the proposed method can provide countermeasure strategies against a new multi-stage attack; (2) Under the large volume of a well-known public dataset, called CTU-13, we showed that the proposed method identifies different multi-stage attacks with a high accuracy. Under the CTU-13 dataset, F_1-measure was 93.80% at maximum.

The rest of the paper is organized as follows. In Section 2, we describe previous works for multi-stage attack detection. After describing the proposed framework in Section 3, we show evaluation results in Section 5. Finally, we summarize the paper in Section 6.

2. Related work

2.1. Network attack defenses

With the growth of the Internet and the emergence of various services [15,16], numerous cyber threats have created and studies to prevent threats, also, have conducted [17–20]. In this section, we introduce some backgrounds and researches related to the proposed method.

2.1.1. Firewall

The firewall is most common and widely deployed defense scheme against network attacks. It is usually located between the local network and the Internet. By filtering out unauthorized packets according to policy, the firewall prevents potential threats. Since operations of firewall are very simple, it is easy to implement and deploy the firewall in real network environment.

However, the simplicity of operations causes a big tradeoff between security and user convenience. The weak policy provides high level of convince by allowing potentially harmful actions. On the contrary, the strong policy can ensure high security level. However, since services demanded by users are diverse, the strong policy has issues such as considerable inconvenience in service use, requirement of frequent reconfiguration of policy, and performance degradation due to large amounts of computation.

2.1.2. Decoy-based authentication

In decoy-based authentication scheme, bogus data which appears to be important to adversaries, i.e., honeypot [21,22], honeytoken [23], honeyfiles [24,25] and etc., is placed and advertised. Since authorized users know the presence of decoys and location of real data, they avoid decoys and access to the service with real data. However, adversaries who do not recognize the presence of decoys access to the service with decoys. As a result, the adversary cannot obtain any important information and the system administrator can detect the unauthorized access of the adversary by matching the access information with decoys.

Since such a scheme misleads all accesses of unauthorized users to the unimportant part of the system, the important part of the system free from performance degradation caused by additional computation for detecting attacks and zero-day attacks can be, also, prevented. However, once the presence of the decoy is revealed, adversaries can avoid the decoy easily [26]. Such a scheme, also, has limitations that are vulnerable to insider's attacks.

2.1.3. Intrusion detection system

The IDS is a system which monitors all traffics between local network and the Internet and detects malicious actions. Since the IDS does not operate blocking actions such as filtering out traffic or preventing access suspicious users, it has advantages such as no performance degradation of network and low deployment cost. Additionally, different from decoy-based authentication, the exposure of IDS to adversaries is not critical and the IDS can detect attacks of insider easily.

The IDS is categorized into signature-based IDS and anomaly-based IDS according to detection approaches and characteristics. Signature-based IDSs recognize malicious actions by comparing patterns with signatures. By considering the pattern which exactly matching with the signature as an attack, the signature-based IDS shows relatively low false positive. However, since signatures should be defined before malicious actions occur, the signature-based IDS cannot respond to novel attacks.

In anomaly-based IDSs, All actions which are different from pre-defined normal actions are considered as attacks. The model to define normal actions is constructed using machine learning techniques mostly. Contrary to signature-based IDSs, anomaly-based IDSs have the advantage of being able to detect novel zero-day attacks. However, since all anomalies including unmalicious actions are considered as attacks, it shows relatively high false positive.

2.2. Multi-stage attack detection

After multi-stage attack incidents such as Solar Sunrise [29] and Nimda [30] are recognized as the most dangerous malwares, multi-stage attack detection methods have been studied actively by many researchers [31]. Studies on multi-stage attacks are commonly categorized into three groups: (1) modeling multi-stage attack; (2) multi-stage attack detection *without* operational domain knowledge; (3) multi-stage attack detection *with* operational domain knowledge.

To analyze multi-stage attack activities, early studies on multi-stage attacks focused on modeling multi-stage attacks. From the book, *Secrets & Lies*, schneier described the attack tree, which is one of the simplest representation for the multi-stage attack [32]. To show relation between single-stage attacks intuitively, the attack tree represents multi-stage attack activities in a goal-oriented form. However, due to the simplicity of the attack tree, it is not efficient while representing a comprehensive and complicated multi-stage attacks.

Daley et al. proposed a new structural framework for modeling multi-stage attacks [31]. Using three node types, i.e., event, state and top level, and two link types, i.e., implicit and explicit links, the framework generates Stratified Node Topology (SNT), which makes it possible to represent network context information on attack tree.

Dawkins et al. proposed a systematic model that analyzed the multi-stage attack as a network state transition [33]. A multi-stage attack is represented into an attack chain, which considers the change of actions observed from pre-defined network zones,

Wang et al. proposed a new multi-stage modeling method using Finite State Machine (FSM) [34]. Each single-stage attack type composing multi-stage attack is represented into a FSM, called an atom FSM (aFSM). By cascading multiple aFSMs, Multi-stage FSM (M-FSM) is constructed and each M-FSM maps into a multi-stage attack scenario.

Table 1
Comparison of multi-stage attack detection methods.

Name	Characteristics	Drawbacks
Applications of hidden Markov models to detecting multi-stage network attacks [6]	Applying HMM to multi-stage attack detection showing the best performance among C4.5, *k*-NN and HMM	Vulnerable to zero-day attack due to dependency on pre-defined details
Architecture for multi-stage network attack traceback [13]	Detect accurate source of multi-stage attack using stepping stone recognition	No detection on multi-stage attack itself
A novel probabilistic matching algorithm for multi-stage attack forecasts [7]	Predict next single-stage attacks during multi-stage attack using JEAN model	Vulnerable to zero-day attack due to dependency on pre-defined details
MARS: Multi-stage attack recognition system [8,9]	Multi-stage attack detection based on correlation of knowledge-based and statistical model Experimented with LLS DDoS dataset	Vulnerable to zero-day attack due to dependency on pre-defined details
Extracting attack scenarios using intrusion semantics [10]	High understandability on multi-stage attack behavior based on ontological relations of single-stage attacks Experimented with LLS DDoS dataset	Vulnerable to zero-day attack due to dependency on pre-defined details
Predicting multi-stage attacks based on IP information [14]	Pre-defined details on single-stage attack are not required	Geographic location of each IP address is required Experiment with private data only
Multistage attack detection system for network administrators using data mining [27]	Automatic multi-stage attack rule generation using data mining Visualization of multi-stage attacks	Pre-defined details on single-stage attacks are required No considering on order of attacks in a mining period
Frequent item set mining-based alert correlation for extracting multi-stage attack scenarios [28]	Automatic multi-stage attack rule generation using window-based data mining	Pre-defined details on single-stage attacks are required No considering on order of attacks in a window
Multi-stage attack detection and signature generation with ICS honeypots [11]	Available to respond to zero-day attacks due to honeypot based system	Inheriting the drawbacks of decoy-based authentication Experiment with private data only
A multi-stage attack mitigation mechanism for software-defined home networks [12]	Applying SDN/NFV environment data to multi-stage attack detection	Vulnerable to zero-day attack due to dependency on pre-defined details Experiment with private data only
Proposed framework	No pre-defined details on single-stage and multi-stage attacks are required Automatic multi-stage attack rule generation considering order of attacks Experimental results with large volume of datasets (LLS DDoS, CTU-13)	Low understandability on multi-stage attack behavior Increasing computational complexity according to time

Camtepe et al. proposed Enhanced Attack Tree (EAT) [35]. To improve the expressiveness of attack tree, they used time order of preconditions and occurrence probability.

Mathew et al. proposed a method for visualizing heterogeneous event traffics [36]. This method aggregated and correlated traffic events, log files and so on to extract attack activities. They also implemented Event Correlation for Cyber-Attack Recognition System (ECCARS) which enables the analyst to notice these attack tracks in real-time.

To detect multi-stage attack *without* operational domain knowledge, Ourston et al. proposed a method that applies Hidden Markov Model (HMM) [6]. By HMM, This method probabilistically reduced false positives caused by various multi-stage attack scenarios. From the experimental results using self-generated dataset, they showed that the HMM showed better performance than C4.5 and Nearest Neighborhood (NN).

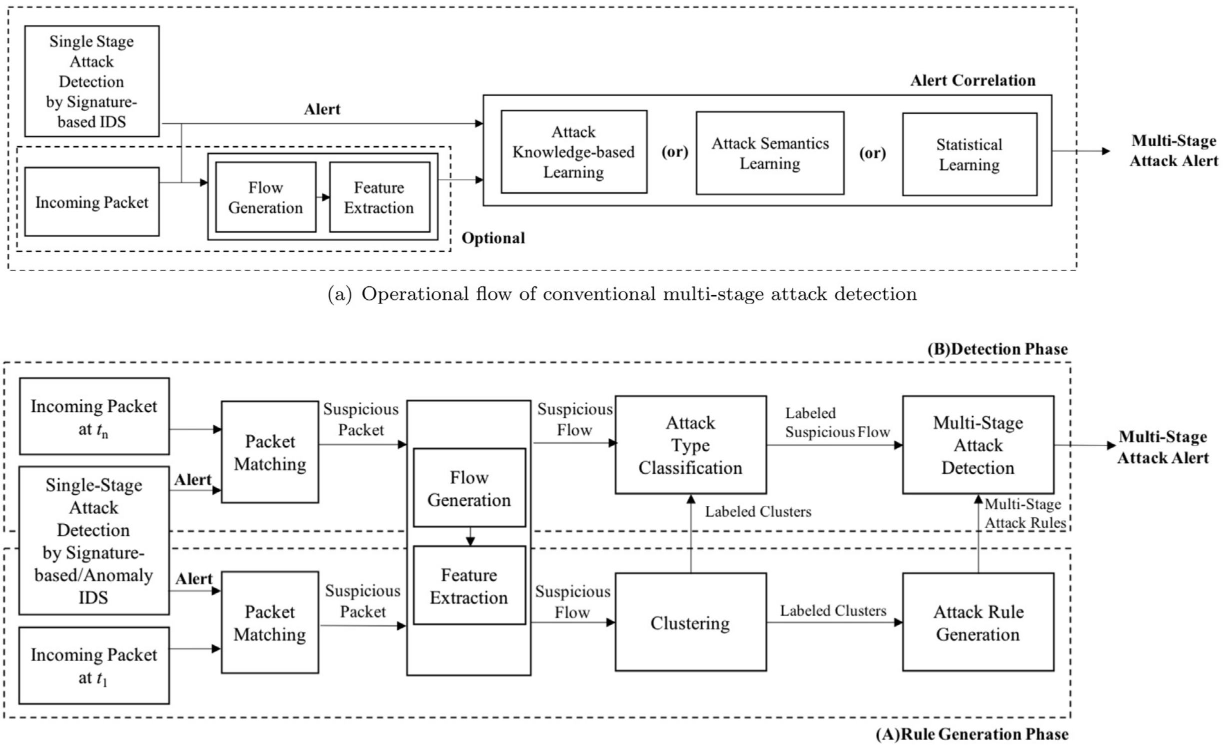
Strayer et al. proposed an architecture, called Stealthy Tracing Attackers Research Light TracE (STARLITE) which trace-backed and detected source IP of multi-stage attack using BBN technologies Source Path Isolation Engine (SPIE) [13]. To reduce false positives, STARLITE integrated stepping stone detection method with SPIE. Here, stepping stone means a laundering hosts that hides the information of the original attack source.

Cheng et al. proposed a probabilistic matching algorithm for multi-stage attack forecast [7]. This work focused on improving the low matching accuracy caused by complexity of multi-stage attack. For this purpose, the algorithm analyzed similarity between pre-defined alerts and currently generated alert by using J-Fusion [37]. As a result, the algorithm determined whether two attacks are the same attack or not.

Alserhani et al. proposed a framework for alert correlation that overcomes the shortcomings of methods based on knowledge-based and statistical models [8,9]. The framework consists of online and offline components. The online component received the alerts from IDS and applied multi-stage attack recognition in real time. The offline component generated the rule for multi-stage attack detection by using knowledge-based and statistical models.

Sadd et al. proposed a method that extracts attack scenarios by intrusion semantics, which were used to correlate alerts semantically [10]. They also proposed an algorithm that detected missing attack steps by analyzing attack impacts according to attack scenarios.

Almutairi et al. proposed multi-stage attack prediction method based on the reputation of IP addresses [14]. The reasoning module which consists of fuzzifier, rule-base and inference engine,



(a) Operational flow of conventional multi-stage attack detection

(b) Operational flow of proposed multi-stage attack detection

Fig. 2. Operation comparison between conventional multi-stage attack detection and proposed multi-stage attack detection.

predicted multi-stage attack using the reputation of IP information.

Katipally et al. [27] and Lagzian et al. [28] used a data mining technique to generate multi-stage attack rules. After aggregating alerts with the same IP address into the same group, they generated dominant multi-stage attack rules using frequent item set mining.

All these works assume that pre-observed details of single-stage attacks are available. Thus, if the pre-observed details are not given, these works show high false negatives when the multi-stage attack has diverse occurrence sequences of single-stage attacks.

To detect multi-stage attack *with* operational domain knowledge, vasilomanolakis et al. proposed multi-stage attack rule generation framework using honeypots in Industrial Control System (ICS) [11]. By modifying HosTaGe honeypot to adapt into ICS, they formulated the detection mechanism with Extended Finite State Machines (EFSMs). They tried to improve false negatives by generating Bro signature [38].

Luo et al. proposed a multi-state attack mitigation mechanism targeting on Software-defined Home Networks (SDHN) [12]. By assessing attack events on SDN and Network Function Virtualization (NFV), a multi-state attack mitigation mechanism tried to improve the false negative alerts under heterogeneous network environments.

Since these works require specific domain knowledges corresponding to the operational environments such as ICS, SDN/NFV, they are hardly used in general operational environments (see Table 1).

2.3. Limitations of existing detection methods

Previous researches on multi-stage attack detection have three main limitations as follows: (1) All introduced multi-stage attack detection methods except the method of vasilomanolakis et al. are based on pre-observed details which provided from signatures of single-stage attacks. Thus, these methods have the limitation of not detecting novel multi-stage attacks which including zero-day single-stage attacks; (2) In addition to the vulnerability to zero-day attacks, multi-stage attack detection rule generation scheme [27] has the limitation of high false positive caused by not considering the order of single-stage attacks; (3) Different from these methods, the method of vasilomanolakis et al. can handle novel multi-stage attacks with zero-day single-stage attacks. However, since the defense approach is based on honeypot, it inherits drawbacks of decoy-based authentication scheme such as exposure of the presence of decoys.

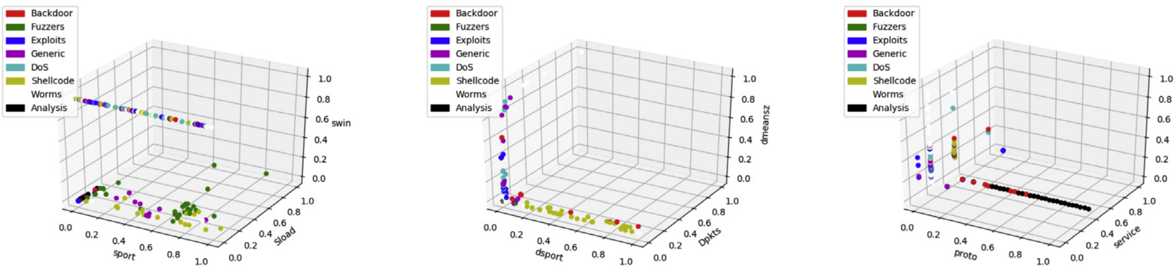
To handle these limitations while mitigating multi-stage attack rule generation and multi-stage attack detection, we propose a new multi-stage attack detection method. By extracting key features from flows directly, the proposed method solves vulnerability to the novel multi-stage attack which includes zero-day attacks. To handle the limitation of existing detection rule generation methods, we introduce an order-aware data mining technique that is modified version of the Apriori algorithm and new multi-stage attack detection rule generation method. Since the defense scheme is based on IDS, the proposed method is free from the drawbacks of decoy-based authentication scheme.

3. Proposed method

In this section, we describe the overall architecture and the operational procedure of each component in details.

3.1. Overall architecture

In Fig. 2, we show that the overall operational flow of conventional and proposed multi-stage attack detection systems. As



(a)

(b)

(c)

Fig. 3. Single-stage attack distributions according to features.

shown in Fig. 2(a), the conventional multi-stage attack detection system generally consists of signature-based IDS for single-stage attack detection, optional flow generation and feature extraction module, and alert correlation modules. After gathering single-stage attack alerts from the signature-based IDS, the conventional system extracts features of flow optionally. Next, the conventional multi-stage attack detection system correlates gathered alerts and features to detect multi-stage attacks using knowledge-based, attack semantics or statistical models. However, knowledge-based, attack semantics and statistical models require pre-observed details of single-stage attacks. Thus, the conventional multi-stage attack detection system shows high false negatives while detecting multi-stage attacks which contain modified or unobserved single-stage attacks.

In Fig. 2(b), we show the proposed multi-stage attack detection architecture. To generate multi-stage attack detection rules, the proposed architecture consists of signature-based or anomaly IDS for single-stage attack detection, packet matching, flow generation and feature extraction, clustering and attack rule generation modules. To detect the multi-stage attack, the proposed architecture consists of signature-based or anomaly IDS for single-stage attack detection, packet matching, flow generation and feature extraction, attack type classification, multi-stage attack detection modules. The operational flow of the proposed framework consists of two operational phases as follows:

(A) Rule generation phase: The multi-stage attack rule is generated using suspicious traffics. To generate single-stage attack type automatically, each suspicious flow is labeled based on their clustering results in the clustering module. By using data mining technique, the proposed multi-stage attack detection system generates multi-stage attack rules by extracting dominant activity patterns from correlated flows.

(B) Detection phase: Incoming traffics are inspected to detect multi-stage attack. To identify the single-stage attack type existing on an incoming flow, the classifier model trained from labeled clusters inspects the suspicious flow. In multi-stage detection module, suspicious flows are correlated with each other to generate multi-stage flow activity. Also, if the multi-stage flow activity matches with a multi-stage attack rule, the module generates the corresponding alerts the multi-stage attack.

We describe operational details of modules composing the proposed multi-stage attack detection framework in followings. In Table 2, we summarize the terms and notation used.

3.2. IDS & packet matching

The packet matching module filters incoming network packets whose headers match with IDS alerts. Since the proposed method creates information to classify single-stage attack itself based on clustering, the presence of alert and features of corresponding flow are needed only. Thus, different from the conventional multi-stage attack detection system, the proposed architecture can use signature-based IDS alerts or anomaly IDS alerts together.

Table 2
Terms and notation.

Terms	Notation
F_{total}	Total flow nodes for rule generation
TH_{sup}	Minimum support value for frequent pattern mining
TH_{len}	Minimum rule length for frequent pattern mining
R_{atk}	The list of multi-stage attack detection rules generated by proposed method
CT	Flow chain tree for flow correlation
f	An flow node that contains labeled flow information
CH_{train}	Flow chains for multi-stage attack rule generation which consist of correlated flow nodes
r_{CT}	Root node of flow chain tree CT
$F_{absence}$	Flag variable to check whether a flow element f exists in the leaf table LT_{CT}
LT_{CT}	A table to record leaf node of CT
l	An element of the leaf node table LT_{CT}
src_f	Source IP address of flow f
dst_f	Destination IP address of flow f
DT_{CT}	A hash table to record the most recent flow node f with the dst_f as a key
ts_f	Timestamp value of flow f
n_{CT}	A node of flow chain tree CT
CN_n	The list of child nodes of node n
PN_n	The list of parent nodes of node n
D	Input dataset for Apriori algorithm
k	The iteration number of Apriori algorithm
L_k	Frequent pattern list of length k
e	A single item that appears in dataset D
C_k	Frequent pattern candidates of length k

3.3. Flow generation & feature extraction

To cluster suspicious flows, the flow generation module and feature extraction module extract flow features using a flow analysis tool such as Bro [38]. The bro supports customizable script allowing users to determine specific feature information from flows.

The flow generation module aggregates packets using 4-tuple information, i.e., source/destination IP address and port, SYN and FIN flags of TCP packets and inter packet timeout. In case of TCP flow, the flow consists of a series of packets from SYN packet to the FIN packet with same 4 tuple information. If the inter packet arrival time exceeds 5 min without FIN packet, the flow is considered to be over. UDP flow, also, consists of a series of packets whose 4-tuple information is same. However, it does not contain flag information and the inter packet timeout of UDP flow is 1 min.

The proposed feature extraction module is designed by focusing on features, which are extracted from the Bro alone, in

Algorithm 1 Attack Rule Generation

```
Input: $F_{total}$, $TH_{sup}$, $TH_{len}$
Output: $R_{atk}$
    Initialization :
    $CT$ = new FlowChainTree()
    $R_{atk}$ = [ ]
1: for $f \in F_{total}$ do
2:     $CT$.addFlow($f$)
3: end for
4: $CH_{train}$ = toFlowChains($r_{CT}$)
5: $R_{atk}$ = modifiedApriori($CH_{train}$, $TH_{sup}$, $TH_{len}$)
6: return $R_{atk}$
```

UNSW-NB15 dataset [39]. A flow consists of 34 attributes which are grouped into five categories. Each feature contains flow identification, payload contents information, time related information and additional key information for specific single-stage attack detection like scanning attack of Nmap [40] and etc. Details on features are described in Appendix.

Each graph in Fig. 3 shows the distribution of single-stage attacks which are 50 sampled in UNSW-NB15 dataset according to features. In Fig. 3(a) whose x-axis is *s_port*, y-axis is *s_load* and z-axis is *s_win*, the single-stage attack type named `Fuzzers` are relatively distinguished well. In Fig. 3(b) whose x-axis is *d_port*, y-axis is *d_pkts* and z-axis is *d_meansz*, the type named `Shellcode` is distinguished well from other types. In Fig. 3(c) whose x-axis is *proto*, y-axis is *service* and z-axis is *tcprtt*, The `Shellcode` is concentrated on the middle of x–y plain and the `Analysis` is distinguished well from others. From these observations, we can notice that an attack type can be distinguished from others with some combinations of extracted features.

3.4. Clustering

The clustering module groups suspicious flows with extracted features by the feature extraction module. Since features of a flow contains nominal and boolean type data that cannot be used directly in clustering algorithm, nominal values are mapped into arbitrary numbers and boolean values are mapped into 0(**False**) and 1(**True**) except for flow identification information such as timestamp and IP addresses. To avoid poor clustering result caused by high dimensionality, Principal Component Analysis (PCA) is used before clustering.

Since the false positive alerts from IDS can degrade the quality of clustering result, outliers in clusters are eliminated by using Density-based Spatial Clustering of Applications with Noise (DBSCAN) [41]. After labeled clusters are given from clustering, they are used to generate multi-stage attack rules and to train attack type classifier.

3.5. Attack rule generation

The attack rule generation module generates multi-stage attack rules while analyzing labeled suspicious flows F_{total} with threshold support TH_{sup} and threshold rule length TH_{len}. The operational details are shown in Algorithm 1. To generate multi-stage attack detection rules, the attack rule generation module mainly executes two phases: (1) flow chain generation; (2) attack rule extraction.

Algorithm 2 Flow Chain Tree Generation

```
Input: $f$
    Initialization :
    $F_{absence}$ = true
1: for all $l \in LT_{CT}$ do
2:     if isSameFlow($l, f$) then
3:         $F_{absence}$ = false
4:         if not isSameType($l, f$) then
5:             $l$.addChild($f$)
6:             $LT_{CT}$.remove($l$)
7:             $LT_{CT}$.add($f$)
8:         end if
9:         break
10:    end if
11: end for
12: if $F_{absence}$ then
13:     if ($DT_{CT}[src_f] \in DT_{CT}$) and
        isNotCyclic($DT_{CT}[src_f], f$) then
14:         $p\_node = DT_{CT}[src_f]$
15:         $p\_node$.addChild($f$)
16:         if $p\_node \in LT_{CT}$ then
17:             $LT_{CT}$.remove($p\_node$)
18:         end if
19:     else
20:         $r_{CT}$.addChild($f$)
21:     end if
22:     $LT_{CT}$.add($f$)
23: end if
24: $DT_{CT}[dst_f] = f$
```

3.5.1. Flow chain generation

A flow node is represented into a flow chain tree CT, which is used while generating and managing correlated flow information. The flow chain tree CT is generated by reflecting the flow correlation based on labeled flow clusters and flow identification information (Lines 1 to 3). After flow chain tree generation, each successively connected flow nodes from root node of CT, r_{CT}, to leaf node becomes a flow chain (Line 4). The operational details are shown in Algorithms 2 to 4

3.5.2. Attack rule extraction

To determine dominant patterns of flow activities in flow chains, we use a frequent pattern mining algorithm, called Apriori algorithm [42]. Since the original Apriori algorithm does not consider sequence information, which is one of critical semantics in multi-stage attack scenario. we modified Apriori algorithm to consider sequence information (Line 5). The operational details are shown in algorithms 5 to 6

Algorithm 2 shows how to generate the flow chain tree, i.e., `addFlow(f)` in Algorithm 1. Searching every node in a tree, while finding relevant flow nodes for extending tree nodes under the large amount of network traffic, is very complex workload. To overcome this issue, the flow chain tree CT keeps two tables, LT_{CT} and DT_{CT}. LT_{CT} is a list of leaf nodes in CT and DT_{CT} is hash table whose key is the destination IP address of a flow. The operational details are as follows.

We find leaf node ls which belong to the same flow sequence with incoming flow node f (Lines 1 to 2). If such leaf nodes ls are found, we set $F_{absence}$ into **false** (Line 3). However, if types of l and f are different, we add f into CT as a child node of l (Lines 4 to 8). If the value of $F_{absence}$ is **true**, which means no leaf node belongs to the same flow sequence with f (Line 12), we check whether src_f, is the same as a key in DT_{CT}. If src_f is the same as a key in DT_{CT} and the insertion node f does not make cycle in flow chain, we

Algorithm 3 Flow Comparison

```
Input: $f_1, f_2$
Output: true or false
1: if $ts_{f_1} > ts_{f_2}$ then
2:     return false
3: end if
4: if ($src_{f_1} == src_{f_2}$ and $dst_{f_1} == dst_{f_2}$) or ($src_{f_1} == dst_{f_2}$ and
   $dst_{f_1} == src_{f_2}$) then
5:     return true
6: end if
7: return false
```

Algorithm 4 Flow Chain Generation

```
Input: $n_{CT}$
Output: $CH_f$
    Initialization :
    $CH_{train} = [\ ]$
1: if isEmpty($CN_{n_{CT}}$) then
2:     return [[$n_{CT}$]]
3: end if
4: for $n \in CN_{n_CT}$ do
5:     $tmp$ = toFlowChain($n$)
6:     for $c \in tmp$ do
7:         $c$.insertFront($n_{CT}$)
8:         $CH_{train}$.add($c$)
9:     end for
10: end for
11: return $CH_{train}$
```

insert flow node f into CT as a child node of $DT_{CT}[src_f]$ (Lines 13 to 18). Otherwise, we insert flow node f into CT as a child node of root node r_{CT}. Next, we updates DT_{CT} with the destination IP address of flow node f, i.e., dst_f, as key and f as value.

Algorithm 3 shows how to compare two flow nodes f_1 and f_2, i.e.,`isSameFlow`(f_1, f_2) to determine whether the two flows are in the same flow sequence. The algorithm compare timestamp ts_{f_1}, ts_{f_2}. If timestamp of f_1, i.e., ts_{f_1}, is faster than the timestamp of f_2, i.e., ts_{f_2}, the algorithm returns **false** and then, terminates the operation (Lines 1 to 3). As a result, we ignore the incoming flow node f_2 whose timestamp is lagged compared to the comparative flow f_1. Otherwise, flow's source and destination IP addresses, i.e., $src_{f_1}, dst_{f_1}, src_{f_2}$ are compared. If src_{f_1} and dst_{f_1} are equal to src_{f_2} and dst_{f_2}, respectively, or src_{f_1} and dst_{f_1} are equal to dst_{f_2} and src_{f_2}, respectively, the algorithm returns **true**. Otherwise, the algorithm returns **false**.

The generated flow chain tree CT is converted into flow chain, which is expressed into the data type of list. That is, the operational details of `toFlowChains`(r_{CT}) in Algorithm 1 is shown in Algorithm 4. If the input node n_{CT} has no child nodes CN_{n_CT}, it returns a table record corresponding to a flow chain containing the input node n_{CT} only (Lines 1 to 2). If CN_{n_CT} is not empty, `toFlowChains()` is called for all child nodes n in CN_{n_CT}. The result is stored recursively in tmp and generates flow chains c. The generated flow chains c are added to CH_{train} after n_{CT} is inserted as first element of c.

In Algorithm 5, we describe how to design modified version of Apriori algorithm for frequent flow pattern mining, i.e., `modifiedApriori`$(CH_{train}, TH_{sup}, TH_{len})$ in Algorithm 1. As being mentioned, the sequence information of flows is very important semantics for multi-stage attack detection. Thus, we modified Apriori algorithm to use order information of input data. By using list data structure, not set data structure which is used in original Apriori algorithm, the modified Apriori algorithm considers order information of input data. The operational flow is as follows.

Algorithm 5 Modified Apriori

```
Input: $D, TH_{sup}, TH_{len}$
Output: $R_{atk}$
    Initialization :
    $L_1 = \{[e] | e \in d \land d \in D\}$
    $k = 2$
    $R_{atk} = [\ ]$
1: while not isEmpty($L_{k-1}$) do
2:     $C_k$ = generateCandidates($L_{k-1}, k, L_1$)
3:     for $d \in D$ do
4:         for $c \in C_k$ do
5:             if isSubList($d, c$) then
6:                 $count[c] = count[c] + 1$
7:             end if
8:         end for
9:     end for
10:    $L_k = \{c | c \in C_k \land count[c]/N_D \geq TH_{sup}\}$
11:    if $k \geq TH_{len}$ then
12:        $R_{atk}$.add($L_k$)
13:    end if
14:    $k = k + 1$
15: end while
16: return $R_{atk}$
```

Algorithm 6 Candidates Generation

```
Input: $L_{k-1}, k, L_1$
Output: $C_k$
    Initialization :
    $C_k = \{\}$
    $L_{k-1}$ = list($L_{k-1}$)
    $L_1$ = list($L_1$)
1: for $i = 0$ to len($L_{k-1}$) do
2:     for $j = 0$ to len($L_1$) do
3:         $C_k$.add($L_{k-1}[i] + L_1[j]$)
4:     end for
5: end for
6: return $C_k$
```

The initial frequent pattern list L_1 is initialized as a set of lists, each of which contains an element e observed at least once in dataset D. The iteration number k is also initialized as 2 (Initialization). After generating candidate list C_k, the appearance number of each candidate c in D is counted (Lines 1 to 9). If the appearance number of candidates in D exceeds threshold support TH_{sup}, such candidates are added into L_k. If k is larger than threshold length TH_{len}, L_k is added into attack rule R_{atk} (Lines 10 to 13). After increasing k by 1, lines 1 to 14 in Algorithm 5 are repeated until the latest L_{k-1} is empty.

The difference from the original Apriori algorithm is found at the function `generateCandidates`(L_{k-1}, k, L_1) of Algorithm 5 described in Algorithm 6. L_{k-1} and L_1 are casted into list at initialization. After $L_1[j]$ is appended to each elements of L_{k-1} for all j, the appended list is added as candidates C_k (Lines 1 to 5). Since the operation between $L_{k-1}[i]$ and $L_1[j]$ is append, not union, the C_k contains all possible combination of sequences. As a result, the modified Apriori algorithm maintains order information of input data.

Algorithm 7 Multi-stage Attack Detection

```
Input: $f$
1: $CH_f = CT_d$.addFlow($f$)
2: $R_{match}$ = findMatchedRule($CH_f$, $R_{atk}$)
3: if not isEmpty($R_{match}$) then
4:     alert($R_{match}$)
5: end if
```

Algorithm 8 Flow Chain Tree Generation for Multi-stage Attack Detection

```
Input: $f$
Output: $CH_f$
    Initialization :
    $F_{absence}$ = true
1: for all $l \in LT_{CT}$ do
2:     if isSameFlow($l, f$) then
3:         $F_{absence}$ = false
4:         if not isSameType($l, f$) then
5:             $l$.addChild($f$)
6:             $LT_{CT}$.remove($l$)
7:             $LT_{CT}$.add($f$)
8:         end if
9:         break
10:    end if
11: end for
12: if $F_{absence}$ then
13:    if $DT_{CT}[src_f] \in DT_{CT}$ and
       isNotCyclic($DT_{CT}[src_f], f$) then
14:        $p\_node = DT_{CT}[src_f]$
15:        $p\_node$.addChild($f$)
16:        if $p\_node \in LT_{CT}$ then
17:            $LT_{CT}$.remove($p\_node$)
18:        end if
19:    else
20:        $r_{CT}$.addChild($f$)
21:    end if
22:    $LT_{CT}$.add($f$)
23: end if
24: $DT_{CT}[dst_f] = f$
25: $CH_f = CT_d$.toFlowChain($f$)
26: return $CH_f$
```

3.6. Attack type classification

In attack type classification module, suspicious flows are classified into different attack types. We uses k-Nearest Neighbors(k-NN) algorithm [43] for classification. Before detection phase, the classifier is trained using labeled clusters. In detection phase, the classifier inspects and then, labels suspicious flows.

3.7. Multi-stage attack detection

In algorithm 7, we show the operational flows of multi-stage attack detection module. To detect and alert the multi-stage attack, the multi-stage attack detection module performs the following three steps: (1) flow chain generation; (2) matched rule detection; (3) multi-stage attack alert.

In flow chain generation step, the labeled flow f is added into chain tree CT_d for detection. For the labeled flow f, CT_d returns a chain CH_f (Line 1). In matched rule detection step, the module finds matched rules R_{match} by comparing CH_f with R_{atk} (Line 2). In multi-stage attack alert step, if matched rules R_{match} are found, the corresponding alerts are generated (Lines 3 to 5).

Algorithm 9 Flow Chain Generation for Multi-stage Attack Detection

```
Input: $f$
Output: $CH_f$
    Initialization :
    $CH_f$ = [ ]
1: if $PN_f == r_{CT_d}$ then
2:     return [$f$]
3: end if
4: $CH_f$ = toFlowChain($PN_f$)+[$f$]
5: return $CH_f$
```

Algorithm 10 Attack Rule Matching

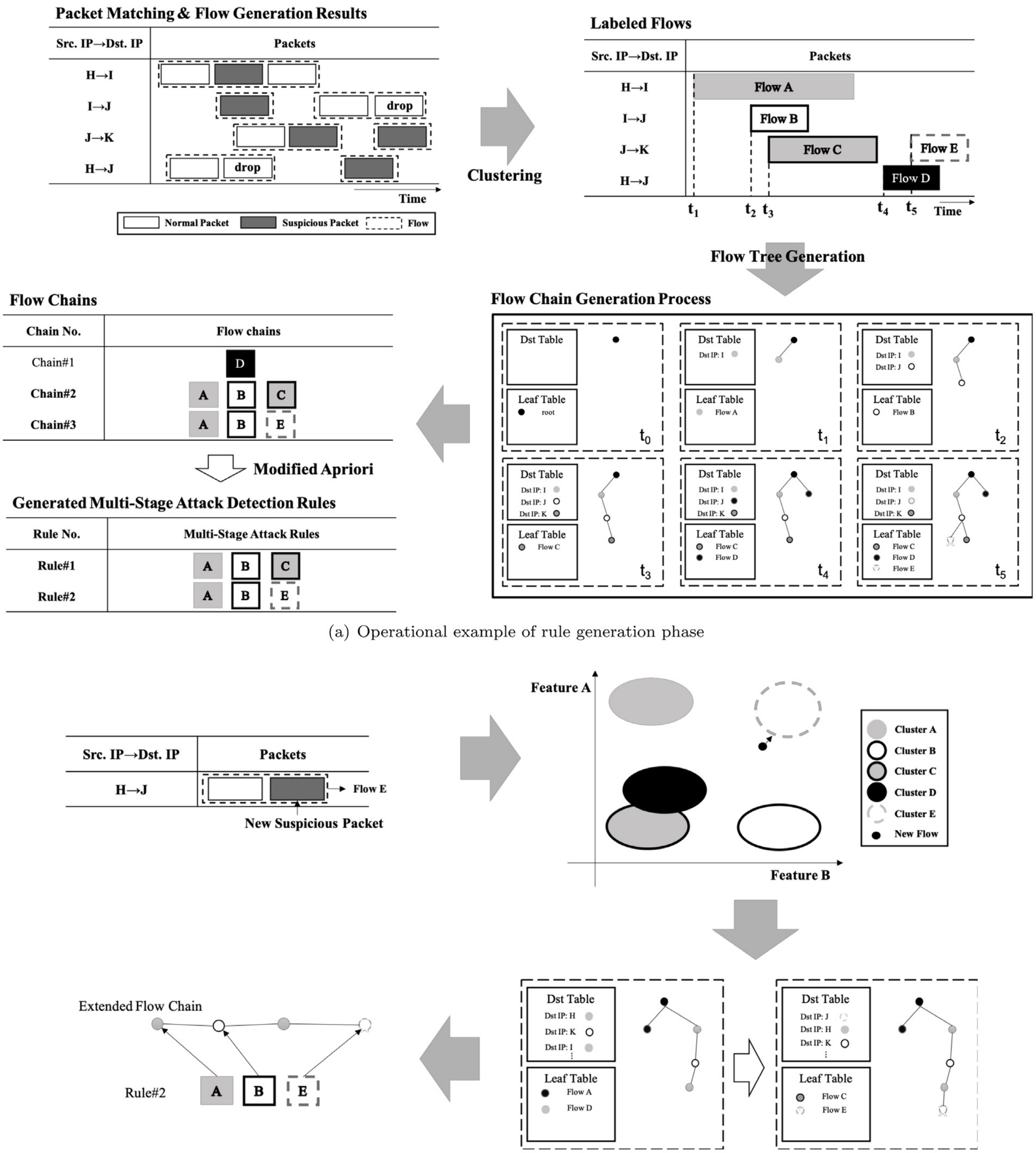
```
Input: $CH_f$, $R_{atk}$
Output: $R_{match}$
    Initialization :
    $R_{match}$ = [ ]
1: for all $r \in R_{atk}$ do
2:     $i = 0$
3:     $j = 0$
4:     while $i <$ len($CH_f$) do
5:         if $CH_f[i] == r[j]$ then
6:             $j = j + 1$
7:         end if
8:         if $j \geq$ len($r$) then
9:             $R_{match}$.add($r$)
10:            break
11:        end if
12:        $i = i + 1$
13:    end while
14: end for
15: return $R_{match}$
```

Algorithm 8 shows the detailed operation of the function `addFlow(`f`)` in Algorithm 7. The overall operation of this function is the same as Algorithm 2 (Lines 1 to 24). Only the difference from Algorithm 2 is found at line 25. That is, to inspect incoming flow node f online, flow chain CH_f including flow node f is immediately returned. Algorithm 9 shows the detailed operation of function `toFlowChain(`f`)` in Algorithm 8. If current input flow node f is the root node of CT_d, i.e., r_{CT_d}, the algorithm returns list of f (Lines 1 to 3). Otherwise, the algorithm returns the flow chain CH_f that the result of `toFlowChain(`PN_f`)` after f is appended (Lines 4 to 5).

To determine whether the flow chain CH_f is detected into multi-stage attack or not, all attack detection rules R_{atk} generated from multi-stage attack rule generation module are compared with the flow chain as shown in Algorithm 10. The flow chain node index i and detection rule node index j are initialized into 0s, respectively, and each flow chain node is compared with detection rule nodes by increasing the index i until i reaches the length of the flow chain. (Lines 4 to 13). If the ith node of flow chain $CH_f[i]$ is same as jth node of the rule r, j is increased by 1 (Lines 5 to 7). If the value of j exceeds the length of detection rule r, r is considered into the rule matched with CH_f and added into R_{match} (Lines 8 to 11). These operations are repeated for all rule rs in R_{atk}.

3.8. Operational example

In this section, we describe brief examples of detection rule generation and multi-stage attack detection phase. The Fig. 4 shows operational example of proposed method.



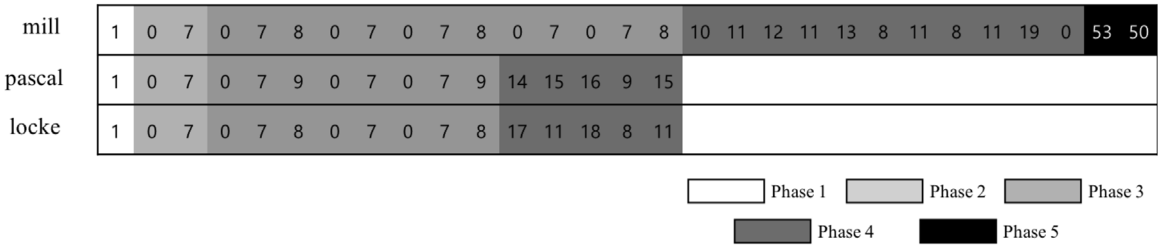
(a) Operational example of rule generation phase

(b) Operational flow of proposed detection phase

Fig. 4. Operational example of proposed system.

In rule generation phase which is shown in Fig. 4(a), packets whose header is matching with IDS's alert are classified as suspicious packet. After collected packets are grouped into flows, flows not including suspicious packets are filtered and features of the unfiltered suspicious flow are extracted in flow generation and feature extraction module. Then, suspicious flows are clustered in clustering module. For the convenience of explanation, we assume that each flow forms a cluster and the label of a flow is equal to the name of cluster.

To generate flow chain from labeled suspicious flow, Algorithm 2 generates flow tree. First, flow A is appended as a child node of root node and added into leaf table while removing root node. Then, flow B is added to the chain tree. Since source IP(H) and destination IP(I) of flow B are not match with that of any node in leaf table and source IP of flow B(I) which is equal to destination IP of flow A is in dst table, flow B is appended as a child node of flow A and added to leaf table while removing flow A. At t_3, flow C is appended into the flow chain tree in the same way as flow B. At t_4, since source IP(H) and destination IP(J) of flow D are not match with that of any node in leaf table and source IP of flow D(H) is not in dst table, flow D is appended into the chain tree as a child node of root node and added to leaf table. Then, repeat above steps until all flows are appended into the chain tree. After flow chains are generated from the flow chain tree according to Algorithm 4, multi-stage attack detection rules are extracted using modified Apriori algorithm described



mill	1	0	7	0	7	8	0	7	0	7	8	0	7	0	7	8	10	11	12	11	13	8	11	8	11	19	0	53	50
pascal	1	0	7	0	7	9	0	7	0	7	9	14	15	16	9	15													
locke	1	0	7	0	7	8	0	7	0	7	8	17	11	18	8	11													

Fig. 5. Rules generated from LLS DDoS 1.0 dataset.

in Algorithm 5. Since we assume that the minimum length of detection is 2, two rules are extracted from flow chains.

Fig. 4(b) shows the operational example of Detection phase. When a new suspicious flow is generated with the arrival of a new suspicious packet, the flow is arbitrarily named(Flow E) and features of suspicious flow are extracted from packets of the flow. After the flow is labeled with the name of the closest cluster(Cluster E), the flow is added into current flow chain tree. Since source IP of flow E(H) is in dst table, the flow is appended as a child node of flow D and added to leaf node while removing flow D in the table. After a flow chain is generated from extended part of the flow chain tree according to Algorithm 9, the flow chain compared with detection rules generated in rule generation phase. In example, since the extended flow chain contains all components of rule#2 with right order, the flow matches with the rule#2 and the alert is generated.

4. Computational complexity analysis

Since network flow generation technique based on source IP & port and destination IP & port is very common and there are many number of researches for efficient flow analysis [44,45], we address the computational complexity of modules that work after the suspicious flow is generated.

In rule generation phase, the clustering module performs clustering operation using DBSCAN whose average complexity is $O(N_{F_{total}} \log N_{F_{total}})$ when the number of F_{total} is $N_{F_{total}}$.

The operation of attack rule generation module is divided into three processes, *i.e.*, flow chain tree generation, flow chain generation and rule generation. In flow chain tree generation process, the computational complexity of worst case that the hit rate of leaf table is 1 is $O(N^2_{F_{total}})$ and the complexity of the best case that the hit rate of leaf table is 0 is $O(N_{F_{total}})$. Thus, the average complexity of flow chain tree generation process is $O(N_{F_{total}} \log N_{F_{total}})$. Let, d_i is depth of ith node of leaf table and n_{LT} is number of node in leaf table. Since $\sum_{i=1}^{n_{LT}} d_i = N_{F_{total}}$, the complexity of flow chain generation process is $O(N_{F_{total}})$. Since the rule generation process is same with the modified Apriori algorithm, it's complexity is also equal to the complexity of Apriori algorithm which is $O(N_{F_{total}} l_{max}) + (1 - (\log N_{F_{total}})^{l_{max}} / (1 - \log N_{F_{total}}))$ when l_{max} is the length of the longest flow chain.

Therefore, the complexity of rule generation phase is

$$O(N_{F_{total}}(\log N_{F_{total}} + 1 + l_{max}) + \frac{(1-(\log N_{F_{total}})^{l_{max}})}{(1-\log N_{F_{total}})}) \quad (1)$$

The detection phase consists of two modules, *i.e.*, attack type classification and multi-stage attack detection module. Since k-NN algorithm is used to classify attacks, the complexity of attack type classification is $O(kN_{F_{total}})$.

The multi-stage attack detection module is divided into three processes, *i.e.*, flow chain tree generation, flow chain generation and multi-stage attack detection. Although the operation of flow chain tree generation process is same with that of rule generation phase, the incoming flow is processed one by one. Thus, the complexity is $O(N_{F_c})$ when N_{F_c} means the number of suspicious flows collected since the start of detection phase. Also, the complexity of flow chain generation process is $O(l_{max_c})$ when l_{max_c} means the longest flow chain since the start of detection phase. Let, $N_{R_{atk}}$ is the number of detection rule and $l_{max_{R_{atk}}}$ is length of the longest detection rule. The complexity of multi-stage attack detection process is $O(N_{R_{atk}} l_{max_{R_{atk}}})$. Therefore the complexity of detection phase is

$$O(kN_{F_{total}} + N_{F_c} + l_{max_c} + N_{R_{atk}} l_{max_{R_{atk}}}) \quad (2)$$

As shown in Eq. (1), the complexity of rule generation phase is pretty big. However, considering the rule generation phase is not time critical and the complexity is almost equal to the complexity of previous detection rule generation methods, the complexity of rule generation is affordable.

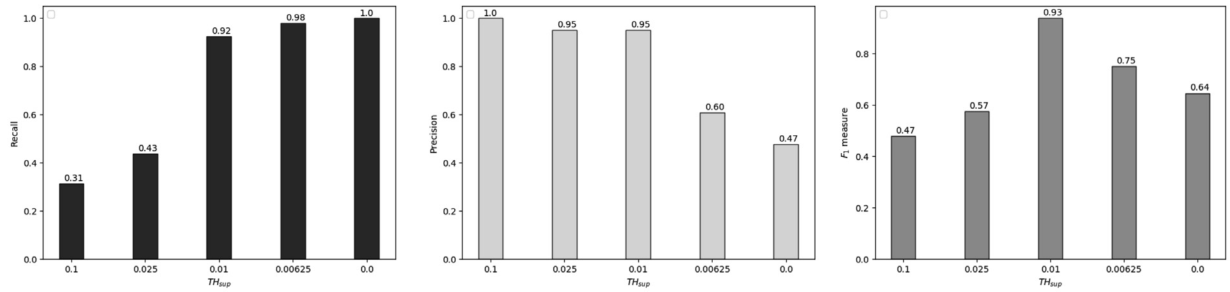
Different from the rule generation phase, the detection phase is time critical. However, considering values of $N_{F_{totola}}$ and $N_{R_{atk}}$ are constant in detection phase and the complexity of the classification can be decrease by applying other efficient machine learning algorithms, the complexity of detection phase is approximate to $O(N_{F_c})$. Therefore, the proposed method seems to be realistic in practice.

5. Evaluation results

In this section, we show the evaluation results of the proposed multi-stage attack detection method. To concretely show the accuracy of the generated attack detection rule, we compared the generated multi-stage attack detection rules with multi-stage attack scenarios in DARPA LLS DDoS 1.0 dataset [46]. It is observed that the attack scenario in DARPA LLS DDoS 1.0 dataset is designed by combining three single-stage attack types. To show that the proposed method is efficient even when many attack scenarios exist, we measured detection accuracy on a large volume of dataset, called CTU-13 [47]. As a botnet traffics captured in the Czech Technical University in Prague(CTU), The CTU-13 dataset consists of 13 number of scenarios observed from various botnets. The dataset consists of raw packet data, labeled netflow [48] data, network environment information and malware files used while generating the dataset. After implementing the proposed method using python 2.7 and scikit-learn, which is a python package for machine learning [49], the performance is evaluated on Ubuntu 16.04.3 LTS with kernel version 4.13.0-43-generic, Intel Xeon E5-2630 v3 CPU, and 64 GB RAM.

5.1. Experimental results from DARPA LLS DDoS 1.0

The operational details of the multi-stage attack in DARPA LLS DDoS 1.0 dataset are as follows. The adversary(202.77.162.213) scans IP addresses from xxx.xxx.xxx.1 to xxx.xxx.xxx.254 in four networks: 172.16.115.0/24, 172. 16.114.0/24, 172.16.113.0/24, 172.16.112.0/24 (: phase 1). To determine which host is running the `sadmind` remote administration tool of Solaris OS, the adversary sends probes to observed hosts via telnet(port number 23) and sunrpc(port number 111) (: phase 2). Next, the attacker



(a) Recall (b) Precision (c) F_1 measure

Fig. 6. Recall, precision, and F_1-measure for TH_{sup} from `CTU-Malware-Capture-Botnet-43`.

exploits three hosts which are named mill(172.16.115.20), pascal(172.16.112.50) and locke(172.16.112.10) using vulnerability of `sadmind` (: phase 3). After exploiting hosts, the adversary creates a directory on hosts and installs a backdoor program named `.rhost`. Different from hosts named into pascal and locke, the `mstream` master program, called `master-sol`, is also installed on the host named into mill(: phase 4). Finally, the attacker launches DDoS attack to a target(131.84.1.31) (: phase 5).

In Fig. 5, we show the multi-stage attack rules generated from three attack scenarios. Each row represents the multi-stage attack detection rule observed for each host, whose name is shown at the leftmost column. Each number shown in column is labeled into a flow type. The flow type number is assigned arbitrary from the result of clustering. Also, a background color represents an attack phase.

From LLS DDoS 1.0 dataset, we observe that multi-stage attack rules are correctly generated with the accuracy by as much as 100%. As shown shown in the first column of all rules, flows for IPsweep phase(: phase 1) are clustered into flow type 1. Two flow types, which are marked into 0 and 7, are probing attacks via telnet and sunrpc, respectively (: phase 2). Exploit and verification flow types via telnet, sunrpc and custom ports are marked into 0, 7 and 8(9), respectively (: phase 3). The dominant flow type for phase 4 is not found from hosts pascal and locke. Since the `mstream` master program, called `master-sol`, is installed on the host named into mill, flow types for phase 4 are observed from host mill (: phase 4). Finally, the DDoS attack from mill is marked into 53 and 50 (: phase 5).

5.2. Experimental results from CTU-13

To show the feasibility of the proposed multi-stage attack detection method, we used two datasets in the CTU-13 dataset: `CTU-Malware-Capture-Botnet-42` and `CTU-Malware-Capture-Botnet-43`. The number of flows and the number of multi-stage attack scenarios for each dataset are shown in Table 3. Here, the number of flows was measured by Argus [50] and the number of multi-stage attacks was measured manually. Since these two datasets are generated from the same botnet, we used `CTU-Malware-Cap ture-Botnet-42` as a training dataset to generate the multi-stage attack detection rules and used `CTU-Malware-Captur e-Botnet-43` as a test dataset to detect the multi-stage attacks using the generated multi-stage attack detection rules. The performance of the proposed multi-stage attack detection method was measured in terms of recall ($Recall = \frac{TP}{TP+FN}$), precision ($Precision = \frac{TP}{TP+FP}$) and F_1-measure ($F_1 - measure = \frac{2}{\frac{1}{Recall}+\frac{1}{Precision}}$) while varying the value of TH_{sup}. TH_{sup} means how many flows chains are generated into multi-stage attack detection rule. For example, TH_{sup} values of 0.1 and 0.025 means that a multi-stage attack detection rule is generated using flow patterns belonging to more than 10% and 2.5% of attack scenarios, respectively.

Table 3
Number of flows and attack scenarios of two datasets in the CTU-13 dataset.

Dataset	# of flows	# of scenario
`CTU-Malware-Capture-Botnet-42`	40 961	4195
`CTU-Malware-Capture-Botnet-43`	20 941	1671

Table 4
Flow identifier.

#	Name	Data type	Notation
1	*ts*	I	Flow measurements start time
2	*uid*	N	Identifier of the flow
3	*s_ip*	N	Source IP address
4	*d_ip*	N	Destination IP address

In Fig. 6, TH_{sup} varies from 0.1 to 0.0 to show the influence ofTH_{sup} on recall, precision and F_1-measure. For recall, 31.53%, 43.92%, 92.46%, 98.03%, and 100% were observed for 0.1, 0.025, 0.01, 0.00625, and 0.0 of TH_{sup}s, respectively. For precision, 100%, 95.08%, 95.18%, 60.91%, and 48.71% were observed for 0.1, 0.025, 0.01, 0.00625, and 0 of TH_{sup}, respectively. Also, for F_1-measure, 47.95%, 57.59%, 93.80%, 75.13%, and 64.66% were observed for 0.1, 0.025, 0.01, 0.00625 and 0.0 of TH_{sup}, respectively. From these results, we observed that as the value of TH_{sup} decreased, the recall increased and the precision decreased. We also observed the maximum F_1-measure of 0.938 when the value of TH_{sup} is equal to 0.01.

6. Conclusion

Network attacks based on single-stage attacks or multi-stage attacks have become elaborate and complicated. Especially, it is observed that the current majority of network attacks consist of sophisticated multi-stage attacks under different time periods. Since the multi-stage attacks break down network attacks into several single-stage attacks, it is necessary to identify various and specific behavior patterns. To identify various and specific behavior patterns, three groups of multi-stage attack methods were proposed: (1) modeling multi-stage attack; (2) multi-stage attack detection without operational domain knowledge; (3) multi-stage attack detection with operational domain knowledge. However, since these works require specific domain knowledges corresponding to the operational environments, they are hardly used in general operational environments. However, the performance was limited because they requires pre-observed details on single-stage attack behavior. In this paper, to overcome such a limitation, we proposed a new multi-stage attack detection framework,

Table 5
Basic flow information.

#	Name	Type	Description
5	*s_port*	I	Source port number
6	*d_port*	I	Destination port number
7	*proto*	N	Transport layer protocol
8	*duration*	F	Total duration of flow
9	*s_byte*	I	Total source to destination bytes
10	*d_byte*	I	Total destination to source bytes
11	*s_ttl*	I	TTL value of packets from src. to dst.
12	*d_ttl*	I	TTL value of packets from src. to dst.
13	*s_loss*	I	Dropped packet bytes from src. to dst.
14	*d_loss*	I	Dropped packet bytes from dst. to src.
15	*service*	N	Application protocol *ex. http, dns*
16	*s_load*	F	Transmitted bytes per second from src.
17	*d_load*	F	Transmitted bytes per second from dst.
18	*s_pkts*	I	Number of source to dst. packets
19	*d_pkts*	I	Number of destination to src. packets

Table 6
Payload contents information.

#	Name	Type	Description
20	*s_win*	I	TCP window advertisement of source
21	*d_win*	I	TCP window advertisement of destination
22	*s_tcpb*	I	TCP sequence number of source
23	*d_tcpb*	I	TCP sequence number of destination
24	*s_meansz*	F	Average size of packets from source
25	*d_meansz*	F	Average size of packets from destination
26	*trans_depth*	I	Pipelined depth into the connection of this request (or response) transaction
27	*res_bdy_len*	I	Uncompressed content size of the data transferred from the server

Table 7
Time-related information.

#	Name	Type	Description
28	*synack*	F	Time interval between the SYN and the SYN_ACK of the TCP connection
29	*ackdat*	F	Time interval between the SYN_ACK and the ACK of the TCP connection
30	*tcprtt*	F	Sum of *synack* and *ackdat* of the TCP connection

which consists of multi-stage attack detection rule generation and multi-stage attack detection. By matching matching, generating flow and extracting feature, and clustering suspicious traffics, multi-stage attack detection rule is generated. After comparing the incoming traffics with the generated multi-stage attack detection rules, various multi-stage attack patterns are detected without pre-observed details on the single-stage attack behavior. The evaluation results even under the large volume of multi-stage attack trace, the proposed framework showed the good-enough performance. Specifically, we showed that all the possible multi-stage attack patterns in DARPA LLS DDoS dataset were correctly detected. Also, from two datasets in CTU-13 [47] including the large volume of multi-stage attack scenarios, F_1-measure of 0.9380 at maximum was observed.

Acknowledgments

This work was supported by basic science research program through national research foundation Korea (NRF) funded by the ministry of science, ICT and future planning, Republic of Korea (NRF-2018R1D1A3B07043392) and the Information Technology Research Center(ITRC) support program (2014-1-00743) supervised by the Institute for Information communications Technology Promotion(IITP), Republic of Korea. Peng Liu was supported by NSF CNS-1814679, ARO W911NF-13-1-0421 (MURI), and ARO W911NF-15-1-0576.

Table 8
Additional features.

#	Name	Type	Description
31	*is_sm_ips_ports*	B	If *s_ip* equals to *d_ip* and *s_port* equals to *d_port*, then this variable has **True** else **False**
32	*ct_flw_http_mthd*	I	The number of packets which have method such as Get and Post in http service
33	*is_ftp_login*	B	If a user accesses to ftp session with password, then **True** else **False**
34	*ct_ftp_cmd*	I	The number of packets which have a command of ftp session

Conflict of interest statement

None.

Declaration of competing interest

The authors declared that they had no conflicts of interest with respect to their authorship or the publication of this article.

Appendix. Flow features

In this section, we describe all features used in feature extraction module. Features are represented into one of four data types, i.e., nominal data type (N), integer data type (I), floating point data type (F), and Bool data type (B). Flow features are grouped into five different categories: basic flow information; flow identification; payload contents information; time-related information; and additional features. In Tables 4 to 8, we show the name and notation of features belonging to each category.

In Table 5, we show features belonging to basic flow information, which is used to identify the behavior of devices in communication. Thus, these features are used to detect anomalies from exploit attempts. Features shown in Table 4 are grouped into flow identifier. Multi-stage attack rule generation and multi-stage attack detection modules use these features to correlate flows. In Table 6, features observed from the TCP, UDP or HTTP payload are shown. Since the contents of malicious flows includes a particular pattern, these features may represent monotonic communication pattern between adversaries and victims. Also, in Table 7, time-related features are shown. These features are used as key features while detecting attacks such as DDoS(Distributed Denial of Service). Additional features shown in Table 8 are special features, each of which is used to detect a specific attack such as user to root(U2R).

References

[1] Cyber kill chain. URL https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html.

[2] P. Bhatt, E.T. Yano, P. Gustavsson, Towards a framework to detect multi-stage advanced persistent threats attacks, in: 2014 IEEE 8th International Symposium on Service Oriented System Engineering, 2014, pp. 390–395, http://dx.doi.org/10.1109/SOSE.2014.53.

[3] S.R. Snapp, J. Brentano, G.V. Dias, T.L. Goan, L.T. Heberlein, C. lin Ho, K.N. Levitt, B. Mukherjee, S.E. Smaha, T. Grance, D.M. Teal, D. Mansur, Dids (distributed intrusion detection system) - motivation, architecture, and an early prototype, in: In Proceedings of the 14th National Computer Security Conference, 1991, pp. 167–176.

[4] M.-Y. Huang, R.J. Jasper, T.M. Wicks, A large scale distributed intrusion detection framework based on attack strategy analysis, Comput. Netw. 31 (23) (1999) 2465–2475, http://dx.doi.org/10.1016/S1389-1286(99)00114-0, URL http://www.sciencedirect.com/science/article/pii/S1389128699001140.

[5] H. Sallay, K.A. Alshalfan, O.B.F. J, K. Words, A scalable distributed ids architecture for high speed networks, in: IJCSNS International Journal of Computer Science and Network Security, VOL. 9, No. 8, 2009.

[6] D. Ourston, S. Matzner, W. Stump, B. Hopkins, Applications of hidden markov models to detecting multi-stage network attacks, in: 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, 2003, p. 10, http://dx.doi.org/10.1109/HICSS.2003.1174909.

[7] B.C. Cheng, G.T. Liao, C.C. Huang, M.T. Yu, A novel probabilistic matching algorithm for multi-stage attack forecasts, IEEE J. Sel. Areas Commun. 29 (7) (2011) 1438–1448, http://dx.doi.org/10.1109/JSAC.2011.110809.

[8] F. Alserhani, M. Akhlaq, I.U. Awan, A.J. Cullen, P. Mirchandani, Mars: Multi-stage attack recognition system, in: 2010 24th IEEE International Conference on Advanced Information Networking and Applications, 2010, pp. 753–759, http://dx.doi.org/10.1109/AINA.2010.57,

[9] F. Alserhani, A framework for multi-stage attack detection, in: 2013 Saudi International Electronics, Communications and Photonics Conference, 2013, pp. 1–6, http://dx.doi.org/10.1109/SIECPC.2013.6550973.

[10] S. Saad, I. Traore, Extracting attack scenarios using intrusion semantics, in: Proceedings of the 5th International Conference on Foundations and Practice of Security, in: FPS'12, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 278–292, http://dx.doi.org/10.1007/978-3-642-37119-6_18.

[11] E. Vasilomanolakis, S. Srinivasa, C.G. Cordero, M. Mühlhäuser, Multi-stage attack detection and signature generation with ics honeypots, in: NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium, 2016, pp. 1227–1232, http://dx.doi.org/10.1109/NOMS.2016.7502992.

[12] S. Luo, J. Wu, J. Li, L. Guo, A multi-stage attack mitigation mechanism for software-defined home networks, IEEE Trans. Consum. Electron. 62 (2) (2016) 200–207, http://dx.doi.org/10.1109/TCE.2016.7514720.

[13] W.T. Strayer, C.E. Jones, B.I. Schwartz, J. Mikkelson, C. Livadas, Architecture for multi-stage network attack traceback, in: The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)L, 2005, pp. 8 pp.–785, http://dx.doi.org/10.1109/LCN.2005.33.

[14] A. Almutairi, D. Parish, J. Flint, Predicting multi-stage attacks based on ip information, in: 2015 10th International Conference for Internet Technology and Secured Transactions (ICITST), 2015, pp. 384–390, http://dx.doi.org/10.1109/ICITST.2015.7412127.

[15] A.P. Plageras, K.E. Psannis, C. Stergiou, H. Wang, B.B. Gupta, Efficient iot-based sensor big data collection–processing and analysis in smart buildings, Future Gener. Comput. Syst. 82 (2018) 349–357.

[16] M.S. Hossain, G. Muhammad, W. Abdul, B. Song, B. Gupta, Cloud-assisted secure video transmission and sharing framework for smart cities, Future Gener. Comput. Syst. 83 (2018) 596–606.

[17] B. Gupta, D.P. Agrawal, S. Yamaguchi, Handbook of Research on Modern Cryptographic Solutions for Computer and Cyber Security, first ed., IGI Global, Hershey, PA, USA, 2016.

[18] C. Stergiou, K.E. Psannis, B.-G. Kim, B. Gupta, Secure integration of iot and cloud computing, Future Gener. Comput. Syst. 78 (2018) 964–975, http://dx.doi.org/10.1016/j.future.2016.11.031, URL http://www.sciencedirect.com/science/article/pii/S0167739X1630694X.

[19] S. Gupta, B.B. Gupta, Detection, avoidance, and attack pattern mechanisms in modern web application vulnerabilities: present and future challenges, Int. J. Cloud Appl. Comput. (IJCAC) 7 (3) (2017) 1–43.

[20] P. Negi, A. Mishra, B. Gupta, Enhanced cbf packet filtering method to detect ddos attack in cloud computing environment, IJCSI Int. J. Comput. Sci. Issues (2013).

[21] I. Kuwatly, M. Sraj, Z.A. Masri, H. Artail, A dynamic honeypot design for intrusion detection, in: The IEEE/ACS International Conference OnPervasive Services, 2004. ICPS 2004. Proceedings., 2004, pp. 95–104, http://dx.doi.org/10.1109/PERSER.2004.1356776.

[22] J. Nazario, Phoneyc: A virtual client honeypot, in: Proceedings of the 2Nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and more, in: LEET'09, USENIX Association, Berkeley, CA, USA, 2009, p. 6, URL http://dl.acm.org/citation.cfm?id=1855676.1855682.

[23] L. Spitzner, Honeytokens: The other honeypot (2003). URL https://www.symantec.com/connect/articles/honeytokens-other-honeypot.

[24] J. Yuill, M. Zappe, D. Denning, F. Feer, Honeyfiles: deceptive files for intrusion detection, in: Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004., 2004, pp. 116–122.

[25] J. Voris, Y. Song, M.B. Salem, S. Hershkop, S. Stolfo, Active authentication using file system decoys and user behavior modeling: results of a large scale study, Comput. Secur. (2018) http://dx.doi.org/10.1016/j.cose.2018.07.021, URL http://www.sciencedirect.com/science/article/pii/S0167404818311258.

[26] S. Innes, C. Valli, Honeypots: How do you know when you are inside one? in: Australian Digital Forensics Conference, 2006, p. 28.

[27] R. Katipally, W. Gasior, X. Cui, L. Yang, Multistage attack detection system for network administrators using data mining, in: Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research, in: CSIIRW '10, ACM, New York, NY, USA, 2010, pp. 51:1–51:4, http://dx.doi.org/10.1145/1852666.1852722, URL http://doi.acm.org/10.1145/1852666.1852722.

[28] S. Lagzian, F. Amiri, A. Enayati, H. Gharaee, Frequent item set mining-based alert correlation for extracting multi-stage attack scenarios, in: 6th International Symposium on Telecommunications (IST), 2012, pp. 1010–1014, http://dx.doi.org/10.1109/ISTEL.2012.6483134.

[29] Solar sunrise. URL http://malware.wikia.com/Solar_Sunrise.

[30] Nimda. URL http://malware.wikia.com/Nimda.

[31] K. Daley, R. Larson, J. Dawkins, A structural framework for modeling multi-stage network attacks, in: Proceedings. International Conference on Parallel Processing Workshop, 2002, pp. 5–10, http://dx.doi.org/10.1109/ICPPW.2002.1039705.

[32] B. Schneier, Secrets & Lies: Digital Security in a Networked World, first ed., John Wiley & Sons, Inc., New York, NY, USA, 2000.

[33] J. Dawkins, J. Hale, A systematic approach to multi-stage network attack analysis, in: Second IEEE International Information Assurance Workshop, 2004. Proceedings., 2004, pp. 48–56, http://dx.doi.org/10.1109/IWIA.2004.1288037.

[34] Y.-M. Wang, Z.-L. Liu, X.-Y. Cheng, K.-J. Zhang, An analysis approach for multi-stage network attacks, in: 2005 International Conference on Machine Learning and Cybernetics, Vol. 7, 2005, pp. 3949–3954, http://dx.doi.org/10.1109/ICMLC.2005.1527628.

[35] S.A. Camtepe, B. Yener, Modeling and detection of complex attacks, in: 2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops - SecureComm 2007, 2007, pp. 234–243, http://dx.doi.org/10.1109/SECCOM.2007.4550338.

[36] S. Mathew, R. Giomundo, S. Upadhyaya, M. Sudit, A. Stotz, Understanding multistage attacks by attack-track based visualization of heterogeneous event streams, in: Proceedings of the 3rd International Workshop on Visualization for Computer Security, in: VizSEC '06, ACM, New York, NY, USA, 2006, pp. 1–6, http://dx.doi.org/10.1145/1179576.1179578, URL http://doi.acm.org/10.1145/1179576.1179578.

[37] S. Xiao, Y. Zhang, X. Liu, J. Gao, Alert fusion based on cluster and correlation analysis, in: 2008 International Conference on Convergence and Hybrid Information Technology, 2008, pp. 163–168, http://dx.doi.org/10.1109/ICHIT.2008.197.

[38] V. Paxson, Bro: a system for detecting network intruders in real-time, Comput. Netw. 31 (23–24) (1999) 2435–2463, URL http://www.icir.org/vern/papers/bro-CN99.pdf.

[39] N. Moustafa, J. Slay, Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set), in: 2015 Military Communications and Information Systems Conference (MilCIS), 2015, pp. 1–6, http://dx.doi.org/10.1109/MilCIS.2015.7348942.

[40] G.F. Lyon, Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning, Insecure, USA, 2009.

[41] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, in: KDD'96, AAAI Press, 1996, pp. 226–231, URL http://dl.acm.org/citation.cfm?id=3001460.3001507.

[42] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, in: VLDB '94, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1994, pp. 487–499, URL http://dl.acm.org/citation.cfm?id=645920.672836.

[43] N.S. Altman, An introduction to kernel and nearest-neighbor nonparametric regression, Amer. Statist. 46 (3) (1992) 175–185, http://dx.doi.org/10.1080/00031305.1992.10475879, arXiv:https://www.tandfonline.com/doi/pdf/10.1080/00031305.1992.10475879. URL https://www.tandfonline.com/doi/abs/10.1080/00031305.1992.10475879.

[44] Y.-H. Choi, W. Park, S.-H. Choi, S.-W. Seo, Deep packet inspection time-aware load balancer on many-core processors for fast intrusion detection, IEIE Trans. Smart Process. Comput. 5 (2016) 169–177, URL http://www.dbpia.co.kr/Article/NODE06717038.

[45] Y. Choi, W. Park, S. Choi, S. Seo, Steal: Service time-aware load balancer on many-core processors for fast intrusion detection, in: 2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2016, pp. 65–70, http://dx.doi.org/10.1109/INFCOMW.2016.7562047.

[46] 2000 darpa intrusion detection scenario specific data sets. URL https://www.ll.mit.edu/r-d/datasets/2000-darpa-intrusion-detection-scenario-specific-data-sets.

[47] S. Garca, M. Grill, J. Stiborek, A. Zunino, An empirical comparison of botnet detection methods, Comput. Secur. 45 (2014) 100–123, http://dx.doi.org/10.1016/j.cose.2014.05.011, URL http://www.sciencedirect.com/science/article/pii/S0167404814000923.

[48] R. Sommer, A. Feldmann, Netflow: Information loss or win? in: Proceedings of the 2Nd ACM SIGCOMM Workshop on Internet Measurment, in: IMW '02, ACM, New York, NY, USA, 2002, pp. 173–174, http://dx.doi.org/10.1145/637201.637226, URL http://doi.acm.org/10.1145/637201.637226.

[49] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[50] Argus-the all seeing. URL http://argus.tcp4me.com/links.html.

Jinmyeong Shin was born in Busan, South Korea, in 1994. He received the B.E. degree in computer science and engineering from the Pusan National University, Busan, South Korea, in 2017. In 2017, he joined the Department of Electrical Engineering, Pusan National University, as a master's student.

Seok-Hwan Choi was born in Busan, Korea, in 1992. He received the B.E. degree from Pusan National University, Busan, Korea, in 2016. He is currently working on his M.E. in Computer Science and Engineering at Pusan National University, Busan, Republic of Korea. His research interests include intrusion detection, network security and adversarial deep learning

Peng Liu received the B.S. and M.S. degrees from the University of Science and Technology of China and the Ph.D. degree from George Mason University, in 1999. He is a professor of information sciences and technology, founding director of the Center for Cyber-Security, Information Privacy, and Trust, and founding director of the Cyber Security Lab, Penn State University. His research interests include all areas of computer and network security. He has published a monograph and more than 260 refereed technical papers. His research has been sponsored by US National Science Foundation, ARO, AFOSR, DARPA, DHS, DOE, AFRL, NSA, TTC, CISCO, and HP. He has served on more than 100 program committees and reviewed papers for numerous journals. He received the DOE Early Career Principle Investigator Award. He has co-led the effort to make Penn State an NSA-certified National Center of Excellence in Information Assurance Education and Research. He has advised or co-advised more than 30 Ph.D. dissertations to completion. He is a member of the IEEE

Yoon-Ho Choi received the M.S. and Ph.D. degrees from Seoul National University, Seoul, Korea, in 2004 and 2008, respectively. From September 2008 to December 2008, he was a Post-doctoral Scholar with Seoul National University. From January 2009 to December 2009, he was a Postdoctoral Scholar with Pennsylvania State University, University Park, PA, USA. While working as a Senior Engineer with Samsung Electronics from May 2010 to February 2012, he had been deeply involved in the development of a commercial Long-Term Evolution cloud communication center system. Also, he was an assistant professor at Kyonggi University, Suwon, Korea from May 2012 to August 2014. He is currently an associate professor with the School of Computer Science and Engineering, Pusan National University, Busan Republic of Korea. His research interests include deep packet inspection, anomaly detection algorithms, privacy preserving machine learning, algorithms, vehicular network security, embedded system security and so on. Dr. Choi has served as a member of several technical program committees in various international conferences and journals .