JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 223-242 (2019)
DOI: 10.6688/JISE.201901_35(1).0012

Dealing with Interleaved Event Inputs for Intrusion Detection

HSING-KUO PAO, FONG-RUEI LEE AND YUH-JYE LEE+
Department of Computer Science and Information Engineering
National Taiwan University of Science and Technology
Taipei, 106 Taiwan
E-mail: pao@mail.ntust.edu.tw; ssshfray@gmail.com
+Department of Applied Mathematics
National Chiao Tung University
Hsinchu, 300 Taiwan
Email: yuhjye@math.nctu.edu.tw

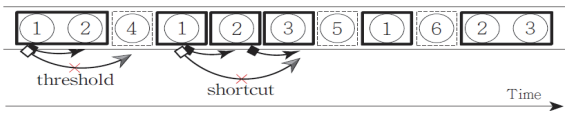
We propose an intrusion detection method that can deal with interleaved event inputs. The event sequences may be alert sequences in a network or running processes on a host which are both considered to contain mixed behaviors with unpredictable orders in the temporal domain. To detect intrusions with interleaved event sequences, one of the major difficulties is to separate the interleaved events that are produced by different users or for different intentions. We propose a novel ATM algorithm to extract subsequences that characterize different behaviors; afterwards, a method that is based on graph representation is used to detect intrusions. In a network, there could be intruders who plan a DDoS attack on an environment that has mostly benign users. The proposed method can distinguish between different pieces of network data that represent different behaviors and locate where the intrusion is. On a host, users without enough privilege may inappropriately gain access to data that they are not supposed to see. The proposed method can detect the event subsequence that is associated with the unauthorized activity given a usage sequence from users such as process, command or log sequences. Given the network or host-based data, the experiment results show that the proposed method can reach high precision and recall rates at the same time in the intrusion detection task. Moreover, the graphs produced by the proposed ATM method are also compared to the graphs generated from other methods to confirm that the ATM-based graph representation indeed describes meaningful transitions between events.

Keywords: event sequence, host-based intrusion, interleaved event, intrusion detection, network-based intrusion

1. INTRODUCTION

We propose a unified intrusion detection method that can deal with both of the network data and host-based data under the same detection framework. One of the key issues to successfully detect intrusions given an event sequence collected either in a network-based or host-based environment is to separate events that correspond to different behaviors or users where the various kinds of events are interleaved with each other in the sequence. Given an event sequence such as the one in Fig. 1 (a), separating the interleaved events and retrieving event subsequences that belong to single behaviors or single users are essential because the event transitions after the separation show more evidences of causality or possible multi-step attack scenarios than the transitions in the original interleaved sequence. In a sequence of alert events, we may observe an intruder whose activities trigger alerts scattering on many other false alerts produced by legitimate users. As another case, multiple users may share the same computing resources at

Received January 5, 2017; revised December 31, 2017; accepted February 12, 2018.
Communicated by Meng Chang Chen.



(a) An interleaved event sequence.

(b) The event transition graphs for threads in the sequence in (a).

(c1) The graph from ATM ($W_c = 2$). (c2) The graph from ATM ($W_c = 4$).

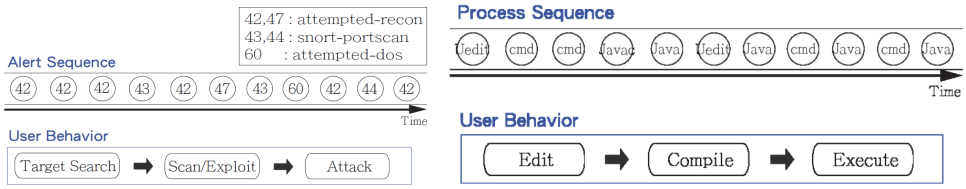
Fig. 1. The challenges of extracting threads from an interleaved event sequence. (a) shows an interleaved event sequence where all events are numbered to indicate various event types. There are two subsequences (called threads) interleaved with each other in (a): a squared one (1-2-1-2-3-1-2-3) and another (4-5-6). Given the original sequence in (a), we can use the proposed method to find the graphs in (c1) and (c2) (via different parameters) which are almost identical to one of the true graphs in (b).

a period of time under a cloud service. If we consider the sequence of running processes recorded in a time slice on the cloud, we find the processes, possibly for different computing purposes likely interleaved with each other. It is clear that extracting alerts or processes that belong to single behaviors or single users is critical for intrusion detection or cloud environment monitoring.

The event separation for interleaved sequence may not be trivial. The event features may provide clues for us to know how the events are generated, or generated by whom and we can surely use that information to separate events that belong to different behaviors or different users. In a network environment, we can use source or destination IP addresses to separate alert events that belong to different users or for different purposes. On a host, we can use usernames to discover who own different sets of processes. However, the above information such as IP address or identity may not always be available and reliable, especially when the network or host is under attack. To effectively detect intrusions robustly, we have to find an alternative approach.

We propose an intrusion detection method that can extract sequential events, called *threads* where each belongs to a single activity from an interleaved event sequence and detect intrusions in the set of extracted event subsequences (threads). In Fig. 2, we discuss two examples that we can apply our method for intrusion detection. The first example, which is shown in Fig. 2 (a) focuses on network intrusions. To avoid network attacks, we usually adopt a network-based Intrusion Detection System (IDS) such as Snort [1] to issue alerts constantly for subsequent experts' confirmation. Given the alert sequence, we can extract threads or subsets of alerts from the sequence that are associated with single users or behaviors. In Fig. 2 (a), we have an extracted alert sequence (or a thread) with the alert types shown by numbers. We may find a match between the alert thread and a multi-step attack scenario: searching a target, scanning, and attacking. Note that such scenario is not obvious to be observed in the original interleaved alert sequence.

The second example, as in Fig. 2 (b) is about host-based intrusions. Given a set of sequential process data collected on a host or a virtual machine with multiple users, the proposed method can extract the process patterns that belong to different users and separate intrusive users from authorized users. In cloud service environment, user identification is especially important if the regular identification routines in the cloud are not reli-



(a) An attack behavior extracted from an alert sequence.

(b) A software development behavior extracted from a process sequence.

Fig. 2. The user behaviors in two kinds of event sequence.

able and there is more than a single user who shares the same resources in the environment. As long as we can separate event threads that belong to different users, we can build a model for each user to describe his/her unique behavior and detect intrusions or unauthorized use afterwards. In (b), given an extracted process sequence from a cloud, we may find a user who runs processes that correspond to the loop of editing, compilation, and execution in a software development task. At the same time, we may also find another attacker in the same cloud who plans to block the service as much as he/she can to maximize the computation power; or attempts to cause a fault in the operating system and gain control of the authorized users' applications, systems, or networks. We can use the proposed method to separate the above two behaviors from the same cloud environment even their process data are highly interleaved from each other.

In this work, we focus on intrusion detection given two types of event sequences: the alert sequence from network environment, and the process sequence monitored on a host or a cloud. We state the focused problem as follows.

The Focused Problem and Proposed Method

Given an event sequence such as an alert sequence or a process sequence $\mathbf{s} = (s_1, s_2, \ldots, s_n, \ldots, s_N)$, $s_n \in \mathcal{S}$, where $\mathcal{S}$ is the set of all alert types or process types, and a set of known behavior patterns $\mathcal{B}$, we aim to solve the following problems:

1. Extract threads/subsequences **t**'s where each **t** belongs to a single behavior; and
2. For each found **t**, classify **t** as a type of behavior $b \in \mathcal{B}$.

Given an alert sequence or a process sequence, behavior patterns may consist of a normal one and a malicious one, and the problem simply becomes a binary classification problem. Before the classification, we have to extract threads that correspond to different behaviors from the interleaved sequence. For instance, Fig. 1 (a) shows an event sequence which includes a thread (1,2,1,2,3,1,2,3) and another thread (4,5,6), each could be from a unique user or form a unique behavior. To understand patterns of the event sequence, we first need to recover the "pre-mixed" threads, one thread for each individual user or behavior. As soon as the threads are extracted, the behavior patterns (instead of mixed, complicated, noisy and summarized patterns) can be relatively easy to be observed and described for each single user/behavior.

In this work, the key features as well as our major contributions include:

- We detect intrusions with two different types of inputs, namely the network data and host-based data under a unified view.
- To deal with interleaved event sequences, we propose a novel algorithm called ATM, to extract meaningful individual patterns from the sequences.

- We propose a graph-based representation to describe behavior patterns and a dissimilarity measure to compare between different patterns.

In this work, a graph (or we call it a *hyper-event*) rather than a single event is a high-level representation for each individual pattern. The graph structure provides rich information such as the causal correlations between events for domain experts to confirm the final pattern classification (*e.g.*, to be malicious or not).

The rest of the article is organized as follows. In Section 2, we review the previous work about intrusion detection, sequence analysis and other related topics. After that, in Section 3, we introduce the method for thread extraction and intrusion detection given the alert sequences and process sequences. In Section 4, we describe the data sets that we use in this study and their statistics. It is followed by Section 5, where we show the performance of the proposed method and compare the result with that from other approaches. At last, we conclude our work in Section 6.

2. RELATED WORK

In this section, we discuss previous work that is related to the proposed method. For the application point of view, we survey some past work that is related to *intrusion detection* and *alert correlation*. From the methodology viewpoint, we discuss some previous work about *sequential data* analysis. Some graph-based methods are also mentioned.

2.1 Intrusion Detection

There are many intrusion detection techniques that can be used to recover attack scenarios. We focus on the techniques that take alert sequences as the input to find attacks or malicious behaviors. Given an alert sequence issued by an IDS on a network, considering several alerts within a period rather than one alert at a time is usually the approach to detect intrusions and to understand the true intention from the network data. By doing so, we not only enhance the possibility and confidence to classify alerts to be benign or malicious ones; at the same time, we also reduce the number of alerts for classification because we classify alerts in a group-based fashion.

Several alert correlation approaches [2-5] have been proposed to solve the problem. Basically, the approaches are categorized into three kinds of techniques. The first category correlates the alerts by the similarity measurement on the alert attributes. Both of Staniford *et al.* [6], and Valdes *et al.* [7] proposed the alert correlation methods based on the information such as the source, target IP addresses and the port numbers. They can correlate these alerts aggregately with the same IP addresses or port numbers. However, the drawback is that they cannot easily discover the whole causal relation for those related alerts not from the same address. Julisch [8] clustered the related alerts by the root causes, because the related alerts usually share the same root causes. In principle, the approach can significantly reduce the number of alerts for analysis compared to previous approaches, however, with the trade-off of longer computing time.

The second category is based on the attack scenarios defined by human experts or learned from data. Cuppens *et al.* [9] and Dain *et al.* [10] fall into this category. Cuppens and Ortalo created a declarative language called LAMBDA to specify attacks. Dain and Cunningham combined data from heterogeneous sources to build scenarios. Both of their methods can deal with the data containing forged IPs, therefore, more powerful than the first category methods in general.

The third category is based on the specification of individual attacks. Templeton and Levitt [11], and Ning *et al.* [12, 13] proposed the alert correlation methods by matching the preconditions and consequences of alerts. Their methods correlate alerts if the preconditions of some later alerts are satisfied by the consequences of some previous alerts. These methods can discover the potential causal relationship between alerts. There are also some other techniques that used precondition and consequence of attacks; afterwards, the techniques produced the scenario graph [14, 15] which were applied to network security. Other than that, the model checking technique determines whether or not a formal model of a system satisfies a given property. Ammann [16] used model checking for vulnerability analysis of networks. But they only can obtain one counterexample. Wing [17] modified the model to produce the attack graphs [18-20] for possible attack representations. The path in an attack graph shows the way of compromising the system. Zhu and Ghorbani [21] built attack graphs to represent the transition probabilities between different alert types. A more recent work, Idika and Bhargava [22] proposed a suite of graph-based security metrics and an algorithm to decide vulnerabilities. Also, Ramaki *et al.* attempted to correlate alerts for early attack warning [23]. There are some previous efforts that focused on the intrusion detection applied to one of the most popular KDD'99 dataset [24-27]. The classifiers for intrusion detection were built based on a set of expert-selected features. Apparently, significant human efforts are necessary for these approaches.

2.2 Sequence Analysis

One of our objectives is to understand behavior patterns from event sequences. To extract patterns from event sequences, Lin *et al.* [28] proposed a symbolic representation, called SAX for time series or streaming data, but mainly for continuous data. Toivonen *et al.* [29] processed events of Nokia routers, and reported frequent subsequences. In mining of frequent patterns with gapped constraint, several works focus on the reduction of subsequence candidates based on Apriori property [30, 31]. Ji *et al.* [32] studied the minimal distinguishing sequences (MDSs) with a gapped constraint which occurs frequently in one class but infrequently in another class, and they developed an efficient algorithm named ConSGapMiner to prune the generated candidate patterns. Mining MDSs among sequences could also be regarded as the enhancement of measuring the dissimilarity between sequences. Mining the frequent closed sequences is also a way to provide more compact result with better efficiency. Ding *et al.* [31] proposed an efficient approach for finding closed repetitive gapped subsequences which generates much smaller candidates of sub-sequences. Zaki *et al.* [33] proposed a combined technique for modeling complex pattern in sequence data. Wang *et al.* [34] proposed an efficient algorithm for mining closed sequences without maintaining any candidates. All the above approaches intend to discover specific patterns in efficient way. To speak of our case of distinguishing behaviors from difference categories of event sequences with interleaving or high variance of patterns, it is unlikely that those methods can be helpful. In the next section, we discuss the proposed method.

3. PROPOSED METHOD

In this section, we introduce the proposed intrusion method for network-based and host-based intrusions. Given an event sequence, to detect intrusions of many kinds, the first goal is to "understand" the behavioral patterns hidden in the sequence and then clas-

sify the behaviors into a benign or a malicious type of some sorts. To solve the problems, we propose a system which consists of three components: *sequence partition*, *behavior thread extraction and behavior dissimilarity measurement*, further described below. Before doing so, we introduce some definitions and notations that are used in this work.

3.1 Definitions and Notations

- *Event sequence*: An event sequence $\mathbf{s} = (s_1, s_2, \ldots, s_n, \ldots, s_N)$ where $s_n \in \mathscr{S}$ and $\mathscr{S} = \{\tau_1, \tau_2, \ldots, \tau_K\}$ is the set of all K event types. Examples include an alert sequence or a process sequence.
- *Thread*: A thread $\mathbf{t} = (t_1, t_2, \ldots) = (s_{t(1)}, s_{t(2)}, \ldots.)$ is a subsequence of an event sequence that contains a specific normal or malicious behavior. For instance, an FTP task consists of at least three steps: typing password, going to a particular directory and uploading/downloading files. When the task sequence is transmitted through a network, we expect the sequence interleaved with packets of other behaviors in the network. In this situation, we would like to recover the original sequence such as the packets corresponding to the FTP task for the later behavior recognition.
- *Major/minor thread*: An event sequence may include more than one thread in the sequence, which corresponds to different user behaviors. Among those threads, one thread could have more observations (higher *saliency*) than others. In math, given a thread $\mathbf{t} = (t_1, \ldots, t_m, \ldots, t_M)$, we define the number of occurrences for a pair of event types (τ_i, τ_j) to be $\#(\tau_i, \tau_j) = |\{(t_m, t_{m+1}): \tau_i = t_m, \tau_j = t_{m+1}, 1 \le m \le M - 1\}|$. Given a sequence s, a thread t in s is a major thread if all the numbers of event pair occurrences in t are larger than or equal to all the numbers of event pair occurrences in any other threads $\mathbf{t}'$ in **s**. All threads that are not major threads are called minor threads. It is simply an ideal case to discuss between major threads and minor threads.
- *Correlation window*: The size of correlation window ($W_c \ge 1$) defines the maximum range of directly correlated event pairs. That is, given an event sequence **s**, we call two events s_n, s_m in **s** to be possibly (directly) correlated, if $|m - n| \le W_c - 1$.
- *Scenario window*: Given an event sequence, we define the scenario window (W_s) to be the size of the window that is likely to include all the events, from the beginning to the end for a group of similar behaviors. Typically, we set W_s to be around a few hours to a day in this work.
- *Thread transition graph*: Given a thread, we build a *directed* transition graph $\mathscr{G} = (V, E, M)$ with vertex set V, edge set E, and transition matrix M. The vertex set V includes all possible event types such as alert types or processes in a thread. The edge set E denotes the transitions between vertices and M is the transition matrix where M_{ij} records the transition probability/counts going from vertex (event type) i to vertex (event type) j.
- *Shortcut*: Given two vertices $u, v \in V$ on a directed graph $\mathscr{G} = (V, E)$, if there exists two paths going from u to v such as $\pi_1(u, v) = (u = u_1^1, u_2^1, \ldots, u_{\ell_1+1}^1 = v)$ and $\pi_2(u, v) = (u = u_1^2, u_2^2, \ldots, u_{\ell_2+1}^2 = v)$, and the length of π_1 is less than or equal to π_2, or $\ell_1 \le \ell_2$, we call π_1 a shortcut from u to v if compared to π_2. When $\ell_1 = 1$, we call it a *strong shortcut*, otherwise, a *weak shortcut*. We focus only the strong shortcut in this work.

3.2 Sequence Partition

Given an event sequence, we first apply a preprocessing step to partition the sequence into several different subsequences so that it is easier to analyze the events and the event correlations in each individual subsequence. We discuss two approaches: the

partition based on a temporal consideration and the partition based on a spatial consideration, *i.e.*, the partition based on IP address or user information, further described below.

3.2.1 Temporal based partition

Given an event sequence such as from NIDS alert records or user process data, we assume that each thread terminates in a limited window size. For instance, we can consider a partition of sequence $\mathbf{s} = (s_1, s_2, \ldots)$ into W_s-length subsequences $\mathbf{s}_1 = (s_1, s_2, \ldots, s_{W_s})$, $\mathbf{s}_2 = (s_{W_s+1}, s_{W_s+2}, \ldots, s_{2W_s})$, *etc*., given a pre-defined *scenario window*.

3.2.2 Spatial/IP based partition

In network data, the IP address is usually informative to separate packets from different network behaviors. In a cloud-based environment with unknown users connected for resource sharing, the account information can be used to separate the tasks from different users. As discussed, the above information, either the IP address or username is not necessarily trustful due to the possibility of, for instance, dynamic IP assignment, user camouflage, *etc*. Still, we can use such information, called spatial information on sequence partition and help us to find subsequences where event correlations can be easily observed in each subsequence. Because the spatial information is not 100% accurate, we shall apply a thread extraction algorithm (discussed in the next sub-section) to further improve the result.

3.3 Behavior Thread Extraction

Given the sequence partition result, from either the temporal or spatial consideration, the goal of behavior thread extraction is to extract threads from the partitioned sequences and use a graph to describe the correlations between events for each thread. Ideally, we expect each thread (or its corresponding graph) represents an intention or a behavior from a specific user (a normal one or an intruder) such as someone performing an FTP download, or trying to break an account. We propose an approach, called *ATM*, which includes three steps: All-pair correlation computation, Thresholding low frequency counts for noise removal, and building Maximum directed spanning tree to find thread graphs from the partitioned sequence, further described below.

3.3.1 All-pairs

Given a partitioned subsequence obtained from sequence partition, we want to extract threads from it so that a thread contains only events for a single behavior or from a single user. For a partitioned event sequence (or subsequence) $\mathbf{s} = (s_1, s_2, \ldots, s_n, \ldots)$, we consider an event pair to be the possible correlated events if they are within a *correlation window* distance, denoted by W_c. The rationale behind it is that two *directly* correlated events should not be very far away from each other (within W_c). On the other hand, two events with distance within W_c may not be true correlated events and counting all within W_c pairs to be correlated events should provide an *over-estimated* result. We further refine the all-pair result by thresholding and maximum spanning tree computation to remove some unwanted event pairs.

In math, given a partitioned sequence $\mathbf{s} = (s_1, s_2, \ldots)$, we count 1 for each occur-

rence of event pair (s_n, s_m) from **s** if $m - n \leq W_c - 1$ and $1 \leq n < m \leq |\mathbf{s}|$, where $|\mathbf{s}|$ is the sequence length of **s**. The result is stored in a matrix M where M_{ij} denotes the number of counts that we observe in the sequence for a pair of (ordered) event types (τ_i, τ_j). We can normalize the counts to probabilities by

$$P_{ij} = P(\tau_j \mid \tau_i) = M_{ij} / \left(\sum_{k=1}^{|\mathcal{S}|} M_{ik} \right), \tag{1}$$

where $|\mathcal{S}| = K$ denotes the number of all possible K event types. As mentioned before, the frequency counts and the probabilities are expected to be over-estimated by the all-pair consideration.

Using either the frequency counts or probabilities for event type pairs, we construct a *directed* graph called *thread transition graph* to reveal the (directed) event correlations. The thread transition graph $\mathcal{G} = (V, E, M)$ is composed of a vertex set V for event types, an edge set E, and a transition matrix M that serves as the weights on the graph. A directed edge from i to j indicates that there exists significant amount of observations of the pair ($s_n = \tau_i$, $s_m = \tau_j$) in the sequence. We build a graph for each partitioned sequence. For instance, if we use the temporal partition with scenario window size W_s, we shall have one graph for each of the sequences $\mathbf{s}_1 = (s_1, s_2, \ldots, s_{W_s})$, $\mathbf{s}_2 = (s_{W_s+1}, s_{W_s+2}, \ldots, s_{2W_s})$, *etc*.

3.3.2 Thresholding and maximum spanning tree

For the sequence in Fig. 1 (a), the thread transition graphs in Fig. 1 (b) is the ideal set of graphs that we want to find. In our approach, based on the all-pair result, we first discard the edges that have counts lower than a threshold and second, select a maximum directed spanning tree to approximate the true graphs. We believe that even the sequence is highly interleaved, those false correlations can be removed after the two additional procedures.

There are two kinds of event transitions that were over-estimated by the all-pair consideration. For the sequence in Fig. 1 (a), given $W_c \geq 3$, we may count the pair (1, 4), or the pair (3, 5), called *random link* which connects two events from different threads. In general, we expect the random links to have small counts because there is no reason to produce a certain event pair with large counts if there is no causal relationship between the pair of events. We set a constant threshold and consider edges with counts below the threshold to be random links and discard them.

The second kind of over-estimation comes from the so-called shortcut. In Fig. 1 (a), the pair (1, 3) links two events that are not directly related, but in the same thread. To make the transition graph easier to visualize and analyze, we intend to remove those shortcuts. We propose a *maximum directed spanning tree* (MDST) algorithm to find the subset graph that does not have such shortcuts. The rationale why the MDST can select edges without shortcuts is because *the shortcuts always contribute less than or equal to the nearby directly correlated pairs*. For instance, the counts corresponding to (1, 3) should be smaller than the counts of (1, 2) and the counts of (2, 3). Because whenever we observe a pair (1, 3), we observe the pair (1, 2) and the pair (2, 3) as well. On the other hand, sometime we only observe the pair (1, 2) or the pair (2, 3) but not the pair (1, 3) because (1, 3) may occupy in a window larger than the correlation window, if W_c is small.

Algorithm 1: ATM (All-pair, Threshold, MDST)

Input: $\mathbf{s} = (s_1, \ldots, s_n, \ldots, s_N)$, an event type set $\mathcal{S}$, a constant integer $W_c \geq 2$: the correlation window size, and a constant θ : random link threshold.

Output: A thread transition graph $\mathcal{G} = (V, E, M)$ with vertex set V, edge set E, and transition matrix M.

```
begin
    create G = (V(S), E ← ∅, M(|S| × |S|));
    for w = 2: W_c do
        for n = 1: N − w + 1 do
            M_{i,j} ← M_{i,j} + 1 (assuming s_n = τ_i & s_{n+w-1} = τ_j);
        end
    end
    let E = {(i, j): M_{i,j} > 0};
    foreach (i, j) ∈ E do
        if M_{i,j} ≤ θ then
            M_{i,j} ← 0;
            delete (i, j) from E;
        end end
    return MDST(G);
end
```

We show the complete ATM algorithm, consisting of All-pair counting, Thresholding, and maximum directed spanning tree computation, in Algorithm 1 and Algorithm 2. In Fig. 3, for the graph in (a), we can apply the thresholding to produce the graph in (b), and to produce the graph in (c) by the MDST algorithm. To combine both the thresholding and MDST procedures, we have the graph in (d). A slightly different version of the algorithms can work on probabilities (normalized counts) instead of counts.

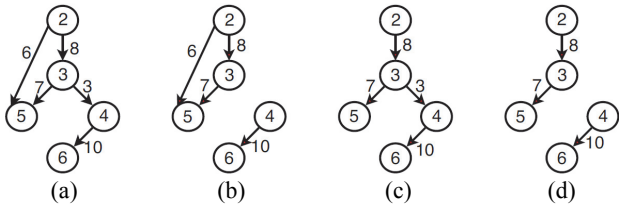


Fig. 3. Given (a) an original graph; selecting the top four edges produces the graph in (b), if the threshold θ is chosen to be 5; and based on (a), the MDST generated by Algorithm 2 is shown in (c). If we apply the thresholding followed by MDST, we produce the graph in (d).

3.4 Behavior Dissimilarity Measurement

Given a pair of thread transition graphs representing two user behaviors, we compute their dissimilarity for subsequent behavior classification. We consider a graph-based dissimilarity measurement. For two graphs $\mathcal{G}_1 = (V_1, E_1, M_1)$ and $\mathcal{G}_2 = (V_2, E_2, M_2)$ where M_1 and M_2 are non-negative, we define their dissimilarity by

$$D(\mathcal{G}_1, \mathcal{G}_2) = 1 - \frac{\sum_{(u,v) \in E_1 \cup E_2} I(u,v)}{\sum_{(u,v) \in E_1 \cup E_2} U(u,v)}, \tag{2}$$

Algorithm 2: MDST (Maximum Directed Spanning Tree)

Input: A graph $\mathcal{G}(V, E, M)$
Output: A refined graph $\mathcal{G}'(V, E', M')$ without shortcuts
begin
  $E' \leftarrow \varnothing$;
  sort the edge E into non-increasing order according to M;
  foreach $(i, j) \in E$ **do**
    if there exists no (directed) path from i to j in E' **then**
      $E' \leftarrow E' \cup \{(i, j)\}$;
    end
  end
  $M'_{i,j} \leftarrow M_{i,j}$ if $(i, j) \in E'$ or $M'_{u,v} \leftarrow 0$ if $(i, j) \notin E'$
  return $\mathcal{G}' = (V, E', M')$
end

where

$$I(u, v) = \min\{M_1(u, v), M_2(u, v)\} \text{ and} \tag{3}$$

$$U(u, v) = \max\{M_1(u, v), M_2(u, v)\}, \tag{4}$$

record the minimum and maximum from two corresponding edges in two graphs, respectively. We define the weight $M_i(u, v) = 0$ if (u, v) does not exist in graph $\mathcal{G}_i$. That means when an edge exists in one graph but not the other, the edge weight will contribute none to the numerator. Clearly, we have $0 \le D(\mathcal{G}_1, \mathcal{G}_2) \le 1$. The dissimilarity measure describes the dissimilarity between two behaviors. When two behaviors have the same transition graph, their dissimilarity (or distance) is equal to 0. On the contrary, if two behaviors are different, their distance is larger than 0, but no more than 1.

3.5 Graph Embedding and Classification

Given an event sequence, we partition the sequence into subsequences and extract threads from the subsequences. For each thread, we build a transition graph to describe the behavior. Afterwards, we compute the pairwise behavior dissimilarities. Based on the dissimilarities, in principle we can plug in any classifiers to classify an unlabeled behavior graph into one with certain intention, or any clustering methods for behavior grouping. For instance, given an alert sequence, we can extract threads from the sequence and classify the corresponding thread graph as either normal or malicious one; and given user process data, we can classify the corresponding thread graph to be with a specific behavior or one owned by a particular user.

We adopt a *manifold learning* method called Isomap [35] for thread transition graph visualization and representation. The map produced by Isomap provides the interpretable insight for further investigation for domain experts; also, it may enhance the performance of graph classification. As a purpose of evaluation, we conduct a SVM [36], or more specifically smooth SVM [37] as the classification method to demonstrate how effective the proposed method is given the result from behavior dissimilarity computation followed by Isomap.

4. THE DATA SET

We use two types of data sets for the evaluation: a *network-based* data set and a *host-based* data set to demonstrate how effective the proposed method is. The network data includes alert sequences generated by an IDS. Basically we tend to use as little payload information as possible to detect intrusions due to the privacy issue. The design is to allow easy deployment of the proposed system to clients who do want a high standard of privacy protection. The network-based data set is a private, real-world data set collected from one of the sensors at Acer[1] eDC (Acer e-Enabling Data Center). The host-based data set is the one for monitoring processes executed on desktop computers, a self-collected data set. Table 1 summarizes the data statistics of the focused data sets.

Table 1. Table 1. Data statistics. The labels in the Acer07 dataset show whether each alert event is a true alert or false one; and the labels of *User Process* denote who executes the processes.

Dataset	Duration	Event No.	Event Type No.	Labels
Acer07	Aug. 30~Sep. 7, '07	302433	15	2
Process10	Mar. 9~10, '10	1265	195	4

4.1 Network-based Data

The first data set used to evaluate the proposed method is the Acer eDC 2007 dataset (Acer07). It is a real-word data set, comprises of alerts generated for inside packets collected from August 31 to September 7, 2007, from Acer eDC (Acer e-Enabling Data Center). Other than the alert information, we obtained the ground truth of attack labels (attack or non-attack/benign) that were identified during the data collection period by the system administrators at Acer following a complete analysis process and the issue of anomaly tickets to the monitored organizations.

The data format for the network-based data includes the record ID, when the attack (or normal behavior) occurred, the source and destination IP addresses of the attack (or normal behavior) and its alert type. We use only the alert type as the input for our system. In some occasions, we adopt the IP addresses, as the spatial information for the sequence partition procedure. It is also compared to the procedure when no IP addresses have been used in the partition.

4.2 Host-based Data

The real large-scale data for host-based intrusion detection with interleaved events such as the event data for host-based intrusion detection in a resource-sharing PaaS environment is not trivial to acquire. Moreover, we hardly know what the real identities behind the scene for the PaaS users even the raw event data can be acquired. Instead, we collected a lab-scale data called Process10 that consists of user processes from several lab members and assume the user processes can represent certain user behaviors in a period of time. We simulate the process data from real PaaS environment by collecting process data from several hosts, mixing them according to time stamps, and removing the original host information afterwards for analysis. In this case, we have a highly interleaved sequence for analysis without the users' identity and hopefully, we can still pick up the patterns from different user behaviors.

We use a tool to monitor user processes on hosts. According to the *Microsoft De-*

[1] http://www.acer-group.com/public/The Group/overview.htm

veloper Network (MSDN) [38], the .NET framework provides us the Process class to access the local and remote processes and we can start or stop the system process recording when necessary. We use some class properties to collect the process data, listed as follows,

- The StartTime: The time when the process is started.
- The HasExited: Indicating whether or not a specific process has been terminated.
- The ProcessName: The process identity.

In the .NET framework, the program checks the executed processes and records the start time of the processes. Only the process type is used as the input for our analysis. The rest data are used for various comparisons in our evaluation.

5. EXPERIMENT

In this section we evaluate the proposed method. We test how effective the proposed method can help to detect intrusions given network alert data or host-based process data. We also check carefully how well the thread transition graphs can summarize the correlations between each pair of event types. Based on the evaluation, for network-based data, we can use the thread transition graphs to describe malicious or normal behaviors and the graph-based labeling is effective to differentiate the true alerts (associated with attack behaviors) from false alerts (associated with normal behaviors). For host-based data, the graphs can describe users' intentions when they use computers or cloud-based services.

5.1 Experimental Setting

The correlation window size W_c is used to decide how far the (direct) event correlations still exist in a sequence. In principle, a larger W_c can catch more correlations, but at the same time give us more unwanted pairwise event counts. On the other hand, setting a large random link threshold θ may be able to remove those unwanted counts. In the experiments, for each fixed choice of W_c, we set q by searching the best choice from $\theta =$ 0.1 to $\theta = 0.9$ in the ATM algorithm to see which combination can give us the best performance. We set the scenario window size W_s to be 0.5 to 1 hour. The number of neighbors used for Isomap is 4 for all the experiments. The *intrinsic dimensionality* of Isomap, *i.e.*, the dimensionality after Isomap is set to be 15 at all cases for simplicity. We also give the result in 2-D for the purpose of visualization. All the parameters used in this work are in shown in Table 2.

To evaluate the proposed method, we separate the data into the training part and test part, as shown in Table 2. Note that all the event threads are transformed into graphs, and the prediction to be a behavior of certain type is all graph-based in this work. We construct the thread transition graphs and label them as true or false based on the individual event labels in the original thread. For the network data, as a conservative way, a graph is called malicious if any of the alerts in the thread is labeled as attack (positive); otherwise, we label the graph as normal (negative). For the process data, we call a graph to be with a certain label according to the majority of identities in the thread. Some graph statistics are shown in Table 3.

Table 2. Parameter settings and the size of training/test sets.

	W_s	W_c	K_{Iso}	Dim.	Training	Test
Acer07	0.5 hr	2-11	4	15	Aug. 30~Sep. 6	Sep. 7
Process10	1 hr	4	4	15	Mar. 9	Mar. 10

Table 3. Thread transition graph statistics. (a) the number of graph vertices for Acer07, and (b) the size of the training and test sets for both of the Acer07 & Process10 datasets. The size of training and test sets is not fully controlled by us but decided by the training and test periods (Table 2).

(a)

	Label	Max	Min	Avg.	Std.
Acer07	Normal	13	3	7.20	2.50
	Attack	3	1	1.68	0.60

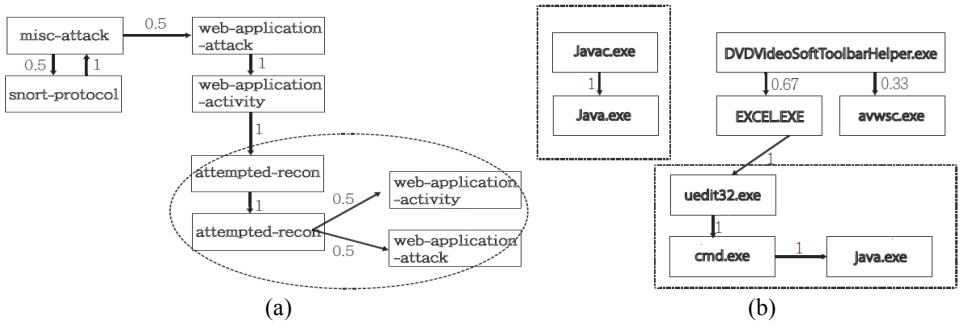
(b)

	#of training graphs	#of test graphs
Acer07	127	23
Process10	16	27

After the classification by SSVM, we provide several metrics to evaluate the proposed method. We compute the precision $P = TP/(TP+FP)$ and the recall $R = TP/(TP+FN)$, where TP is the number of true positives which represents the number of true attacks that are detected, FP is the number of false positives, and FN is the number of false negatives. Also, we compute F-score by $F = 2PR/(P+R)$, which should give a fair comparison even the data is unbalanced, *i.e.*, having a large portion data of one kind or so (such as the Acer07 dataset). For host-based data, we want to verify which person owns the behavior, given some pre-assigned identity. In other words, if there are n users in the dataset, we do $\binom{n}{2}$ trials and compute the average result for evaluation. That is, we have one as the genuine user and the other as the unauthorized user in each trial.

5.2 User Behavior Extraction on Real Data

We study the real-world data sets and demonstrate the thread transition graphs that are produced by the ATM algorithm for the data sets. First, we show a network-based user behavior graph to confirm a case of multi-step attack. It is followed by an example of host-based graph to confirm a Java programming behavior. The results are shown in Fig. 4 (a), for the network data; and in (b), for the host-based process data. In (a), we capture various multi-step attacks from the DARPA99 dataset, while in (b), we observe that a user focuses on Java programming, editing, compiling and running the code.



(a) (b)

Fig. 4. (a) Given the DARPA99 dataset, the multi-step attack captured by the ATM algorithm, and (b) given the Process10 dataset, the user behavior captured by the ATM algorithm.

5.3 Quantitative Analysis

To further confirm how effective the proposed method is, we design some experiments for quantitative analysis. For network data, we demonstrate how the proposed method can help for intrusion detection; and for host-based data, we show the proposed method can detect account misuse, such as stealing user privilege from one with normal authority to with root authority.

5.3.1 Sensitivity analysis

First we run some sensitivity analysis on the proposed method and discuss how different choices of the parameters can influence the performance. Based on the ATM algorithm, we use the correlation window size W_c to adjust how far between two events that we still consider them as possible correlated events. Basically, a larger W_c includes more true correlations, but at the same time some random correlations are also covered. In real cases, we should tune an appropriate window size according to different environments. In Table 4, a real-world dataset Acer07 is considered to have highly interleaved data (a large number of threads), than the lab-produced dataset DARPA99. Therefore, we suppose to use a larger W_c for the Acer07 dataset than for the DARPA99 dataset. The table shows exactly the result. For the Acer07 dataset, the best result is when we choose W_c = 9 to 10; however, for the DARPA99 dataset, we should choose W_c = 4 to obtain the best result.

Table 4. The classification result based on different choices of correlation window size (W_c = 2 to 11) for the Acer07 and DARPA99 datasets. In terms of F-score, we can obtain the best result when W_c = 9 or 10 for the Acer07 dataset and W_c = 4 for the DARPA99 dataset. A larger correlation window is necessary for the Acer07 dataset implies that Acer07 contains more highly interleaved events than the DARPA99 dataset.

	Acer07					DARPA99				
W_c	F-score	P	R	Training Err.	Test Err.	F-score	P	R	Training Err.	Test Err.
2	0.36	1.00	0.22	0.14	0.31	0.67	0.80	0.58	0.78	0.10
3	0.29	0.53	0.21	0.15	0.44	0.66	0.68	0.64	0.75	0.10
4	0.44	0.66	0.33	0.18	0.34	**0.73**	0.66	0.82	0.15	0.23
5	0.38	0.66	0.26	0.08	0.38	0.63	0.78	0.52	0.10	0.24
6	0.40	1.00	0.25	0.11	0.34	0.61	0.80	0.50	0.16	0.24
7	0.35	0.59	0.25	0.13	0.40	0.63	0.73	0.55	0.09	0.25
8	0.50	0.75	0.37	0.17	0.34	0.65	0.79	0.55	0.05	0.23
9	**0.76**	0.71	0.83	0.16	0.18	0.57	0.72	0.47	0.18	0.27
10	**0.76**	0.71	0.83	0.10	0.19	0.51	0.70	0.41	0.06	0.30
11	0.57	0.66	0.50	0.13	0.31	0.41	0.91	0.26	0.10	0.30

5.3.2 Effectiveness of ATM and graph-based dissimilarity measure

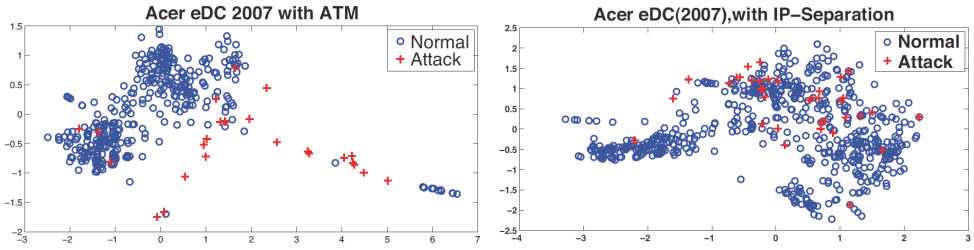
In network-based data, such as an alert sequence generated by IDS, our goal is to separate true alerts from false alerts. In our case, we extract threads from the sequence, and transform each thread to a transition graph. Based on the transition graphs and the

proposed graph-based dissimilarity measure (Eq. 2), we can distinguish the graphs of normal behavior (called *normal graphs*) from the graphs with malicious intention involved (called *attack graphs*). We adopt SSVM [37] to classify graphs into normal ones, or ones with attacks involved. As a result, the proposed ATM method works better than the previous approach that has been applied to the same dataset. In Table 4, we observe that the proposed ATM achieves the F-measure up to 0.76 (W_c = 9 or 10), which is superior to that from the IGS (Intrinsic Graphical Signature) approach proposed by Pao *et al.* [39]. Note that the comparison is hard to be 100% fair because the evaluation for IGS is not from the graph-based but sequence-based metrics.

As a comparison to another approach to directly dealing with interleaved events, we compare the proposed method to a naïve benchmark approach from IP filtering, *i.e.*, using the IP addresses (the spatial information, see Subsub-Section 3.2.2) to extract threads from a sequence, then build transition graphs for classification. As a result, the ATM algorithm gives better results than those from the IP filtering (Table 5). Our explanation is that IP information is informative to catch the network activities; however, it is not enough because some IP information can be faked or dynamic, therefore introduces difficulty for us to spot the intruders based only on IP information. To take a look of the visualized result, Fig. 5 shows the 2-D plots after Isomap. Each instance in the plots represents either a normal graph (circle sign), or attack graph (cross sign). Again, based on the ATM method, we can obtain better separation between two groups of instances than the IP filtering method. Note that the classification is not necessarily done in this 2-D space, but in the space with intrinsic dimensionality (equal to 15 in this case, see Table 2).

Table 5. Given the Acer07 dataset, The evaluation of the proposed ATM algorithm and the IP filtering method based on various evaluation metrics: F-score, precision, recall, and error rates. The bold-face numbers indicate the best result in the category. The ATM algorithm gives better result than the IP filtering method at all times.

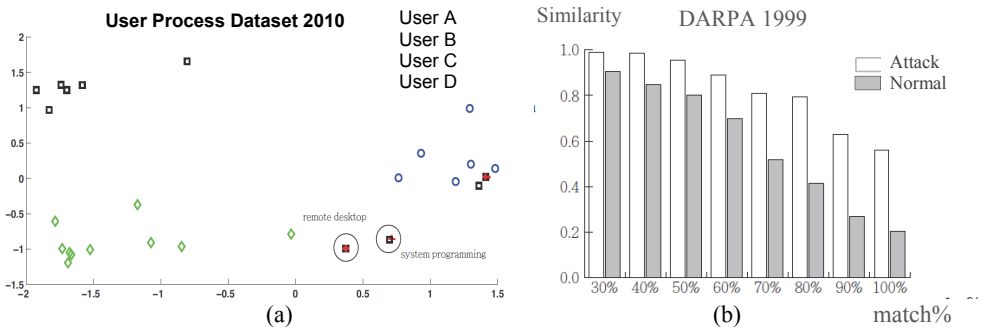
	ATM								IP filtering
Threshold (θ)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	–
F-score	0.75	**0.76**	0.69	0.54	0.39	0.38	0.32	0.35	0.48
Precision	**0.82**	0.75	0.80	0.72	0.44	0.34	0.27	0.33	0.64
Recall	0.70	**0.78**	0.61	0.43	0.36	0.44	0.40	0.41	0.38
Training Err.	0.03	**0.02**	0.03	0.05	0.13	0.16	0.23	0.28	0.04
Test Err.	0.04	**0.03**	0.04	0.06	0.14	0.17	0.24	0.29	0.05



(a) ATM (θ = 0.2) (b) IP filtering

Fig. 5. Given the Acer07 dataset, the 2-D graph-based plots based on Isomap for attack and normal graphs. In (a), we use the ATM algorithm to build the graphs, while in (b), we use IP information to build the graphs. The plot produced by the ATM algorithm clearly gives better separation between the positive (attack) and negative (normal) instances/graphs.

For the host-based data which consists of process sequences, the proposed method can help us to verify the user identity according to the transition graphs. In the experiment, we run the test for each pair of users to see if the behavior extracted from the transition graph can confirm the given identity. That is, we assume different users may have different behavior patterns. Fig. 6 (a) shows the 2-D Isomap plot of all user behaviors. Each point denotes a graph which describes user behavior of one whole hour. The clear separation between different users' behavior implies that different users indeed have different behavioral patterns. The only overlapped instances exist on the bottom-right corner, where different users all involve in similar activities such as system programming or in a remote desktop. Table 6 shows the authentication result by SSVM. It gives 6% average test error in the user authentication/verification task. That means that 6% of the true account users can be mis-judged as intruders, or intruders mis-judged as the true account owners.



(a) (b)

Fig. 6. (a) Given the Process10 dataset, the 2-D Isomap plot for four user behaviors. Each instance represents a transition graph (or a thread) which describes the process transitions and therefore may reveal user behavior. We observe clear separation between different users. The instances on the bottom-right area belong to the remote desktop or system programming activities. That is the only overlapping between those users. There are some moments that we can not easily tell the difference between different users. For instance, when a host is just started, only system processes are running and we have no hint to tell who uses the host resources. (b) The comparison between the graphs produced by the ATM algorithm and the graphs produced by the IP filtering method, given the DARPA99 dataset. The x-axis shows different values of α, as the lower bound to decide two graphs are similar to each other.

Table 6. The summary of verification result based on the ATM algorithm and the proposed dissimilarity measure. The correlation window size is set to 4.

	ATM	
Data Set	Training Error	Test Error
User Process 2010 (4 labels)	0.11	0.06

5.3.3 The similarity between the graphs generated by the ATM and IP filtering methods

In the last part of our experiments, we would like to measure the similarity between the graphs generated by the ATM algorithm and the graphs generated by the IP filtering method[2]. We use a similarity function defined below to measure the similarity between a graph generated by ATM $\mathcal{G}_{\text{ATM}}(V, E_{\text{ATM}})$ and a graph generated by IP filtering $\mathcal{G}_{\text{IP}}(V, E_{\text{IP}})$

[2] We conduct the experiment on the DARPA99 dataset because the IP information seems reliable in the set.

as $\mathrm{Sim}(\mathscr{G}_{\mathrm{ATM}}, \mathscr{G}_{\mathrm{IP}}) = \frac{|E_{\mathrm{ATM}} \cap E_{\mathrm{IP}}|}{|E_{\mathrm{ATM}} \cup E_{\mathrm{IP}}|}$, and we define the similarity of the result produced by the ATM method and the result produced by the IP filtering as $\textit{Similarity} = \frac{\#(\mathrm{Sim}(\mathscr{G}_{\mathrm{ATM}}, \mathscr{G}_{\mathrm{IP}}) > \alpha)}{\#\mathscr{G}_{\mathrm{IP}}}$, where the parameter a is to determine whether two graphs are similar. In Fig. 6 (b), we show the histogram of similarity with different a for the DARPA99 dataset. Based on the result, two sets of graphs have higher similarity in attack graphs than in normal graphs. About 80% of the attack graphs from the proposed ATM approach and the IP filtering approach have close to 80% shared edges. On the other hand, the dissimilarity between two versions of normal graphs looks not so important because the normal graphs just describe the transitions between false alerts. In summary, we believe that the ATM algorithm can produce reliable attack graphs given their similarity to the attack graphs obtained from IP filtering by using the DARPA99 dataset, a dataset seems to be with reliable IP information.

6. CONCLUSION

In this work, we proposed a unified approach for intrusion detection given network-based and host-based data which contain highly interleaved events. The proposed ATM algorithm can be used to extract threads of individual behavior from an interleaved event sequence. We then combined the ATM algorithm and a graph-based dissimilarity measure to separate different behaviors. In the evaluation, we have shown that the proposed method performs well in terms of F-scores on the intrusion detection given network-based or host-based data. Especially, the proposed ATM approach outperforms the existing methods such as Intrinsic Graphical Signature (IGS) based approach and IP filtering. The Isomap-projected visualization also shows that the ATM algorithm well separates graphs representing different behaviors, for both network-based data and host-based data. To focus on the graphs that are generated by the ATM algorithm, we have about 80% similarity in about 80% of all attack graphs, to the ones that are generated by IP addresses. Overall, we believe that the proposed approach can provide a new direction to the intrusion detection, user authentication and other related research topics.

ACKNOWLEDGMENT

This research was supported in part by the Ministry of Science and Technology of Taiwan (MOST 106-2221-E-011-159, 106-2633-E-002-001, 107-2221-E-011-123, 107-2633-E-002-001), National Taiwan University (NTU-106R104045, 107L104039), and Intel Corporation.

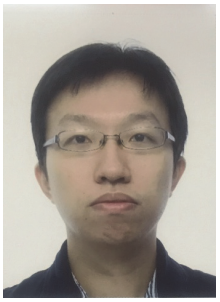
REFERENCES

1. M. Roesch, "Snort, the open source network intrusion system," http://www.snort.org.
2. H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Recent Advances in Intrusion Detection*, ser. LNCS 2212, 2001, pp. 85-103.
3. F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of IEEE Symposium on Security and Privacy*, 2002, p. 202.
4. F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "A comprehensive approach

to intrusion detection alert correlation," *IEEE Transactions on Dependable Security and Computing*, Vol. 1, 2004, pp. 146-169.
5. R. Sadoddin and A. A. Ghorbani, "Alert correlation survey: framework and techniques." in *PST*, *ser. ACM International Conference Proceeding Series*, Vol. 380, 2006, p. 37.
6. S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, Vol. 10, 2002, pp. 105-136.
7. A. Valdes and K. Skinner, "Probabilistic alert correlation," *Lecture Notes in Computer Science*, 2001, pp. 54-68.
8. K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Transactions on Information and System Security*, Vol. 6, 2003, p. 471.
9. F. Cuppens and R. Ortalo, "LAMBDA: A language to model a database for detection of attacks," in *Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection*, 2000, p. 197.
10. O. Dain and R. Cunningham, "Fusing a heterogeneous alert stream into scenarios," in *Proceedings of ACM Workshop on Data Mining for Security Applications*, 2001, pp. 1-13.
11. S. Templeton and K. Levitt, "A requires/provides model for computer attacks," in *Proceedings of ACM Workshop on New Security Paradigms*, 2001, pp. 31-38.
12. P. Ning, Y. Cui, D. Reeves, and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Transactions on Information and System Security*, Vol. 7, 2004, p. 318.
13. P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 245-254.
14. T. Hughes and O. Sheyner, "Attack scenario graphs for computer network threat analysis and prediction," *Complexity*, Vol. 9, 2003, pp. 15-18.
15. S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, 2002, pp. 49-63.
16. R. Ritchey, P. Ammann, A. Booz, H. Inc, and F. Church, "Using model checking to analyze network vulnerabilities," in *Proceedings of IEEE Symposium on Security and Privacy*, 2000, pp. 156-165.
17. O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing, "Automated generation and analysis of attack graphs," in *Proceedings of IEEE Symposium on Security and Privacy*, 2002.
18. L. Wang, T. Islam, T. Long, A. Singhal, and S. Jajodia, "An attack graph-based probabilistic security metric," *Data and Applications Security XXII*, 2008, pp. 283-296.
19. S. Noel and S. Jajodia, "Optimal IDS sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, Vol. 16, 2008, pp. 259-275.
20. D. Saha, "Extending logical attack graphs for efficient vulnerability analysis," in *Proceedings of the 15th ACM Conference on Computer and Communications Security*, 2008, pp. 63-74.
21. B. Zhu and A. Ghorbani, "Alert correlation for extracting attack strategies," *International Journal of Network Security*, Vol. 3, 2006, pp. 244-258.
22. N. Idika and B. Bhargava, "Extending attack graph-based security metrics and aggregating their application," *IEEE Transactions on Dependable and Secure Computing*, Vol. 9, 2012, pp. 75-85.

23. A. A. Ramaki, M. Amini, and R. E. Atani, "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection," *Computers & Security*, Vol. 49, 2015, pp. 206-219.
24. KDD'99, "KDD cup 1999 data," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.
25. C. Elkan, "Results of the KDD'99 classifier learning," *ACM SIGKDD Explorations Newsletter*, Vol. 1, 2000, pp. 63-64.
26. M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proceedings of the 2nd IEEE Symposium on Computational Intelligence for Security and Defense Applications*, 2009, pp. 53-58.
27. B. Kavitha, D. S. Karthikeyan, and P. S. Maybell, "An ensemble design of intrusion detection system for handling uncertainty using neutrosophic logic classifier," *Knowledge-Based Systems*, Vol. 28, 2012, pp. 88-96.
28. J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003, pp. 2-11.
29. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, 1997, pp. 259-289.
30. M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip, "Mining periodic patterns with gap requirement from sequences," *ACM Transactions on Knowledge Discovery from Data*, Vol. 1, 2007, p. 7.
31. B. Ding, D. Lo, J. Han, and S.-C. Khoo, "Efficient mining of closed repetitive gapped subsequences from a sequence database," in *Proceedings of IEEE 25th International Conference on Data Engineering*, 2009, pp. 1024-1035.
32. X. Ji, J. Bailey, and G. Dong, "Mining minimal distinguishing subsequence patterns with gap constraints," in *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005, pp. 194-201.
33. M. J. Zaki, C. D. Carothers, and B. K. Szymanski, "VOGUE: A variable order hidden Markov model with duration based on frequent sequence mining," *ACM Transactions on Knowledge Discovery from Data*, 2009, pp. 5:1-5:31.
34. J. Wang, J. Han, and C. Li, "Frequent closed sequence mining without candidate maintenance," *IEEE Transactions on Knowledge & Data Engineering*, Vol. 19, 2007, pp. 1042-1056.
35. J. Tenenbaum, V. Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, Vol. 290, 2000, p. 2319.
36. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, Vol. 2, 1998, pp. 121-167.
37. Y.-J. Lee and O. L. Mangasarian, "SSVM: A smooth support vector machine for classification," *Computational Optimization and Applications*, Vol. 20, 2001, pp. 5-22.
38. Microsoft, "MSDN library," http://msdn.microsoft.com/en-us/default.aspx.
39. H.-K. Pao, C.-H. Mao, H.-M. Lee, C.-D. Chen, and C. Faloutsos, "An intrinsic graphical signature based on alert correlation analysis for intrusion detection." *Journal of Information Science and Engineering*, Vol. 28, 2012, pp. 243-262.

Hsing-Kuo Pao (Kenneth) is a Professor in the Department of Computer Science and Information Engineering and Vice Dean of College of Electrical Engineering and Computer Science in National Taiwan University of Science and Technology. He received the bachelor degree in Mathematics from National Taiwan University, and M.S. and Ph.D. degrees in Computer Science from New York University. From 2001 to 2003, he was a Post-Doctorate Research Fellow in the University of Delaware, and later he joined Vita Genomics as a research scientist. Since 2003, he joined the Department of Computer Science and Information Engineering in National Taiwan University of Science and Technology. His current research interests include machine learning, information security and computer vision.



Fong-Ruei Lee received the master degree in Computer Science and Information Engineering from National Taiwan University of Science and Technology in 2010. He is currently working on network security in industry.

Yuh-Jye Lee received the Ph.D. degree in Computer Science from the University of Wisconsin-Madison in 2001. Now, he is Chief Executive Officer of Taiwan Information Security Center at Academia Sinica. He is an Adjunct Professor of Department of Applied Mathematics at National Chiao-Tung University. He also serves as a SIG Chair at the NTU IoX Center. His research is primarily rooted in optimization theory and spans a range of areas including network and information security, machine learning, data mining, big data, numerical optimization and operations research. During the last decade, Dr. Lee has developed many learning algorithms in supervised learning, semi-supervised learning and unsupervised learning as well as linear/nonlinear dimension reduction. His recent major research is applying machine learning to information security problems such as network intrusion detection, anomaly detection, malicious URLs detection and legitimate user identification. Currently, he focusses on online learning algorithms for dealing with large scale datasets, time series data and behavior based anomaly detection for the needs of big data and IoT security problems.