Dealing with Interleaved Event Inputs for Intrusion Detection*

HSING-KUO PAO, FONG-FUEI LEE AND YUH-JYE LEE
Dept. of Computer Science & Information Engineering
National Taiwan University of Science & Technology
No. 43, Keelung Rd., Taipei 106, Taiwan
E-mail: {pao; m9715057}@mail.ntust.edu.tw; yuhjye@math.nctu.edu.tw

We propose a method to detect intrusions given interleaved event sequences. The event sequences include alert sequences in a network and running processes on a host which are both considered to contain mixed items with unpredictable orders in the temporal domain. To detect intrusions with the 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 characterizing 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 owning different activities and locate where the intrusive behavior is. On a host, users without enough privilege may 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 and host-based data, the experiment results show that the proposed method can reach high detection accuracy while maintaining low false positives 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 graph representation indeed describes the high confidence 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, for instance, the event sequence in Figure 1(a). To separate the interleaved events and retrieve event subsequences that belong to single behaviors or single users is essential because the event transitions after the separation show more evidences of causalities 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 CPU resources at a period of time on a cloud. If

*

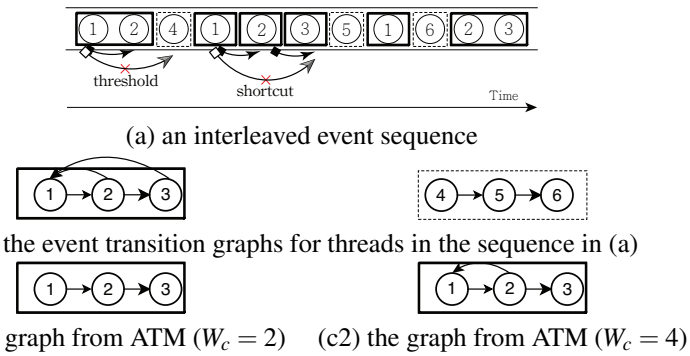


Fig. 1. The challenges of extracting threads from an interleaved event sequence. (a) shows an interleaved event sequence where all events are numbered to indicate various event types. There are two subsequences (called threads) interleaved to each other in (a): a squared one (1-2-1-2-3-1-2-3) and another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

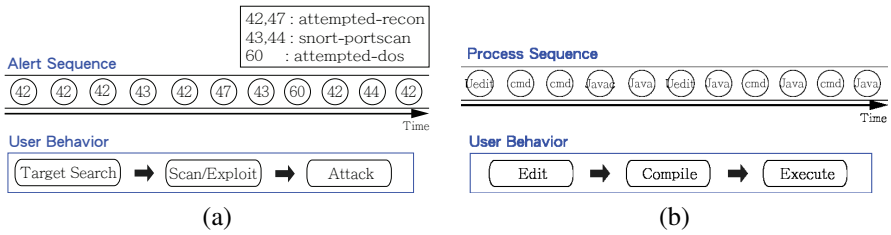


Fig. 2. The user behaviors in two kinds of event sequence: (a) an attack behavior extracted from an alert sequence; (b) a software development behavior extracted from a process sequence.

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

The event separation for interleaved sequences 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 clues to separate events that belong to different behaviors or different users. For network environment, we can use source or destination IP addresses to separate alert events that belong to different users or for different purposes. On hosts, we can use usernames to discover who own different sets of processes. However, the above information such as IP addresses or identities 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 *thread*s where each belongs to a single activity from an interleaved event sequence and detect intrusions in the set of extracted event subsequences. In Figure 2, we discuss two examples that we can apply our method for intrusion detection. The first example, which is shown in Figure 2(a) focuses on network intrusions. In cloud environment, we often see attacks that can compromise not only traditional but also virtual networks. 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 events from the sequence that are associated with different users or behaviors. In (a), we have an extracted alert sequence (or a thread) with the alert types shown by numbers. In this sequence, we may find a match between the sequential alerts (or a thread) and a multi-step attack scenario: searching a target, scanning, and attacking if the alert thread forms an attack sequence.

The second example, as in Figure 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 reliable 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, 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 an attacker in the 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.

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 server. Given an event sequence, we can extract events that belong to different users from the sequence based on a ATM algorithm[1], represent the events of a single user in a pattern graph, and classify the events as a normal or malicious behavior. We state the focused problem and introduce the proposed method below.

Focused Problem and Proposed Method

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

1. *Extract threads/subsequences* $\mathbf{t}$*'s where each* $\mathbf{t}$ *belongs to a single behavior; and*

2. *For each found* $\mathbf{t}$*, classify* $\mathbf{t}$ *as a type of behavior* $b \in \mathfrak{B}$*.*

Given an alert sequence or a process sequence, behavior patterns may include 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, Figure 1(a) shows an event sequence which consists of 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. 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:

[1] We further discuss the details in Section 3.

- 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 for behavior patterns and a dissimilarity measure for the comparison of two 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 and 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) and therefore should be useful for intrusion detection.

The rest of the article is organized as follows. In Section 2, we review the previous works about intrusion detection, alert correlation, 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

From the methodology viewpoint, we deal with event sequences and discuss some previous works related to *sequential data* analysis. For the application point of view, we survey some past works on the problems of *intrusion detection* and *alert correlation*. Some graph-based methods are also mentioned.

2.1 Sequence Analysis

One of our objectives is to detect behavior patterns from event sequences. In this part, we discuss some previous works on event mining, including a rich set of works on sequence alignment. To understand the patterns in event sequences, it is helpful if we can define the relationship between two event sequences. One of the most widely used methods to describe the relationship of two sequences is *edit distance*. In the case of heavy interleaving in a sequence, the edit distance is not appropriate to describe the relationship between two sequences. Likewise, various *sequence alignment* methods, e.g., finding the *longest common subsequences* (LCS) [2] do not work either as they are considerably sensitive to the orders of one-by-one individual events.

To extract patterns from event sequences, Lin et al. [3] proposed a symbolic representation, called SAX for time series or streaming data, but mainly for continuous data. Toivonen et al. [4] processed events of Nokia routers, and reported frequent subsequences, by extending the *Apriori* algorithm [5]. In mining of frequent patterns with gapped constraint, several works focus on the reduction of subsequence candidates based on Apriori property [6, 7]. Ji et al. [8] 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. [7] proposed an efficient approach for finding closed repetitive gapped subsequences which generates much smaller candidates of sub-sequences. Zaki et al. [9] proposed a combined technique for modeling complex pattern in sequence data. Wang et al. [10] 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 propose our system.

2.2 Intrusion Detection

There are many intrusion detection techniques to recover attack scenarios[2]. To consider the techniques that are the most related to this work, we focus on the the IDS that takes alert sequence 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 of 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 that are necessary for classification because we classify alerts in a group-based fashion.

Several alert correlation approaches [12, 13, 14, 15] 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. [16], and Valdes et al. [17] proposed the alert correlation methods based on the information such as the source, target IP addresses and the port number. 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 [18] clustered the related alerts by the root causes, because the related alerts usually share the same root causes. In principle, the approach is believed to highly 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. [19] and Dain et al. [20] 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 previous methods in general.

The third category is based on the specification of individual attacks. Templeton and Levitt [21], and Ning et al. [22, 23] 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 [24, 25, 26] 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 [27] used model checking for vulnerability analysis of networks. But they only can obtain one counterexample. Wing [28] modified the model to produce the attack graphs [29, 30, 31] for possible attack representations. The path in an attack graph shows the way

[2]See [11] for a list of most recent IDS related systems.

of compromising the system. Zhu and Ghorbani [32] built attack graphs to represent the transition probabilities between different alert types. A more recent work, Idika and Bhargava [33] proposed a suite of graph-based security metrics and an algorithm to decide vulnerabilities. There are some previous efforts that focused on the intrusion detection applied to one of the most popular KDD'99 dataset [34], [35], [36] [37]. The classifiers for intrusion detection were built based on a set of expert-selected features. Apparently, significant human efforts are necessary for these approaches.

3. Proposed Method

In this section, we introduce the proposed method for intrusion detection for network-based and host-based intrusions. Given an event sequence, to detect intrusions of many kinds, the first goal is to "understand" the behavior patterns hidden in the sequence, and then classify the behaviors into a benign or a malicious type of some sorts, or the behavior that belongs to a particular account owner. In reality, there are problems that prohibit the classification from reaching the perfection. For instance, in a network setting, packet streams come in an *interleaved* manner and various behaviors are mixed together. To find out what the true intention of the network packets, we need to separate subsequences (or *threads*) that belong to different sources/users. When different users share a powerful server for computation at the same time, the server must need to identify if there is any account misuse, such as stolen accounts or one account with multiple login-on sessions from different users, if given system logs for forensics. It is again the case that we need to work on interleaved data streams because we can not simply use account usernames to sort data into several types with each belongs to a single individual and, to decide who the true user is or what kind of user behavior it belongs to. To solve the above problems, we propose a system with novel algorithms for the tasks.

The main components of the system include three parts: *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*: A sequence of events $\mathbf{s} = (s_1, s_2, \ldots, s_n, \ldots, s_N)$ 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 actions: typing password, going to a particular directory and uploading/downloading files. When the task is transmitted through a network, we expect the action sequence interleaved with packets of other actions in the network. In this situation, we would like to recover the original action sequence such as to extract the packets only corresponding to the FTP actions in this case.

- *Major/minor thread*: An event sequence may include more than one thread in the sequence, which correspond 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 $\#(\tau_i, \tau_j) = |\{(t_m, t_{m+1}) : \tau_i = t_m, \tau_j = t_{m+1}, 1 \leq m \leq M-1\}|$. Given a sequence $\mathbf{s}$, a thread $\mathbf{t}$ in $\mathbf{s}$ is a major thread if all the numbers of

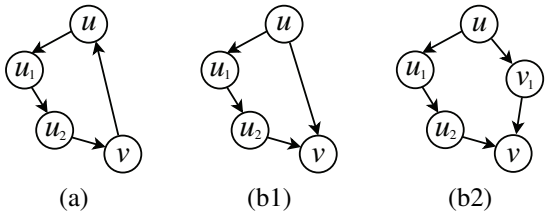


Fig. 3. Directed graphs contain: (a) a cycle, (b1) a strong shortcut and (b2) a weak shortcut. In (b1), the shortcut path $\pi'(u,v) = (u,v)$ has length 1, shorter than the path $\pi(u,v) = (u,u_1,u_2,v)$ which has length 3. The path π' is also a strong shortcut. On the other hand, in (b2), $\pi''(u,v) = (u,v_1,v)$, a shortcut of length 2 is a weak shortcut. We aim to remove the strong shortcut path π' by our proposed algorithm.

event pair occurrences in $\mathbf{t}$ are larger than or equal to all the numbers of event pair occurrences in any other threads $\mathbf{t}'$ in $\mathbf{s}$. 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. In reality, even the the assumption is too strong to be true for finding major threads, we can still obtain more than acceptable outcomes from the studied datasets most of the time.

- *Correlation window*: The size of correlation window (W_c) defines the range of direct correlation between a pair of events. That is, given an event sequence $\mathbf{s}$, we call two events s_n, s_m in $\mathbf{s}$ (directly) correlated, or event s_n *triggers* event s_m if $n < m$ and $|m-n+1| \leq W_c$.

- *Scenario window*: Given an event sequence, we define the scenario window (W_s) as the size of the window that is likely to include all the events, from the beginning to the end for a group of 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 different possible event types, such as all types of alerts, or processes; 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 $\ell_1 \leq \ell_2$, 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*. One example is shown in Figure 3 (b1) & (b2). Basically, we tend to remove strong shortcuts, while weak shortcuts may remain in our thread transition graph construction.

3.2 Sequence Partition

Given an event sequence, first of all, we apply a preprocessing step to chop 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:

a) the partition based on a temporal consideration; and b) the partition based on a spatial consideration, i.e., the partition based on IP address or host information, further described below.

3.2.1 Temporal Based Partition

Given an event sequence such as 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.

3.2.2 Spatial/IP Based Partition

In network data, the IP address is usually informative to separate packets from different network users. In a cloud-based environment with unknown users connected for resource sharing, the account information is usually 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 performance.

3.3 Behavior Thread Extraction

Given the sequence partition result, 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* (All-pairs, Threshold, directed Maximum spanning tree), including three steps: All-pair correlation computation, Thresholding low frequency counts for noise removal, and building directed Maximum spanning tree to find thread graphs from the partitioned sequence, further described below.

- All-pair computation: In a partitioned sequence, we consider all event pairs to be candidates of correlated events if the event pair is within a pre-specified window, called *correlation window*. The design should give an *over-estimated* result. We further refine the result by the following two steps to remove some unwanted event pairs for correlated event consideration.

- Thresholding: We discard the event pairs with low frequency count to remove noisy or erroneous relations. In the corresponding thread graph, we discard the edge that has weight (or counts) smaller than a pre-specified threshold (θ).

- Finding directed maximum spanning tree: In the *directed* thread graphs, we discard the shortcut edges by finding the maximum spanning tree in the graphs.

We discuss each of the steps in more details in the next three subsub-sections.

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 we expect each pair of events to be the possible candidates of *directly* correlated events if they are not very far away from each other (within W_c). Clearly, some of the candidates are not true correlated events, the next two steps can help us to find the "true" correlations among those candidates.

In a sequence $\mathbf{s}$, we count 1 for each occurrence of event pair (s_n, s_{n+w-1}) if $n \geq 1$, $n+w-1 \leq |\mathbf{s}|$, and $2 \leq w \leq W_c$, where $|\mathbf{s}|$ is the sequence length after sequence partition. The result is stored in a matrix M, with M_{ij} denotes the number of counts that we observe in the sequence for a pair of (ordered) event types (τ_i, τ_j). In case that we need to compute probabilities, we can normalize the counts to probabilities by

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

where $|\mathscr{S}| = K$ 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 $\mathscr{G} = (V, E, M)$ is composed of a vertex set V for all event types, an edge set E, and a transition matrix M that serves as the weights on the graph. A directed edge from u to v indicates that there exists significant amount of observations of the pair (s_u, s_v) in the sequence. We build a graph for each partitioned sequence. For instance, if we use the temporal information with scenario window size W_s for sequence partition, we have one graph for each of the sequences[3] $\mathbf{s}_1 = (s_1, s_2, \ldots, s_{W_s})$, $\mathbf{s}_2 = (s_{W_s+1}, s_{W_s+2}, \ldots, s_{2W_s})$, etc.

To give a concrete example, we consider a partitioned sequence of length 11 in Figure 1(a) which consists of two interleaved threads $\mathbf{t}_1 = (1,2,1,2,3,1,2,3)$ (shown in solid squares) and $\mathbf{t}_2 = (4,5,6)$ (shown in dash squares), we want to extract the threads, with corresponding graphs in (b) by our method. The all-pair counts for $W_c = 2$, $W_c = 4$ are

$$M_{W_c=2} = \begin{array}{c} \\ \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{array} \begin{array}{c} \begin{array}{cccccc} \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 \end{array} \\ \begin{bmatrix} & 2 & & & & 1 \\ & & 2 & 1 & & \\ & & & & 1 & \\ 1 & & & & & \\ 1 & & & & & \\ & 1 & & & & \end{bmatrix} \end{array}, \text{ and } M_{W_c=4} = \begin{array}{c} \\ \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{array} \begin{array}{c} \begin{array}{cccccc} \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 \end{array} \\ \begin{bmatrix} 1 & 3 & 2 & 1 & 1 & 1 \\ 2 & 1 & 2 & 1 & 1 & \\ 1 & & & & 1 & 1 \\ 1 & 1 & 1 & & & \\ 1 & 1 & & & & 1 \\ & 1 & 1 & & & \end{bmatrix} \end{array},$$

respectively. The final result from the proposed ATM approach are in (c1) and (c2), for the cases of $W_c = 2$ and $W_c = 4$, respectively. (c2) basically catches the major thread in (b), further discussed below.

[3]Using sliding windows can also be a choice.

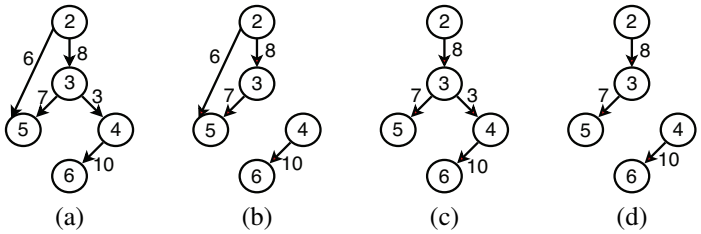


Fig. 4. Given (a) an original graph; selecting the top four edges produces the graph in (b), if the threshold θ is chosen to be 5; and based on (a), the DMST generated by Algorithm 2 is shown in (c). If we apply the thresholding followed by DMST, we produce the graph in (d). A variant procedure can work on probabilities instead of counts.

3.3.2 Threshold and Maximum spanning tree

For the sequence in Figure 1(a), the thread transition graphs in (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 directed maximum 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 Figure 1(a), given $W_c \geq 3$, we may count the pair $(1,4)$, or the pair $(3,5)$, called *random link* which is a pair connecting two events in 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 Figure 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 *directed maximum spanning tree* (DMST) algorithm to find the subset graph that does not have such shortcuts. The rationale why the DMST can select edges without shortcuts is because *the shortcuts always contribute less than or equal to the nearby 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.

We show the complete ATM algorithm, consisting of All-pair counting, Thresholding, and directed Maximum spanning tree computation, in Algorithm 1 and Algorithm 2. In Figure 4, 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 DMST algorithm. To combine both the thresholding and DMST procedures, we produce the graph in (d). A slightly different version of the algorithms can work on probabilities (normalized counts) instead of counts.

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 $\mathscr{G}_1 = (V_1, E_1, M_1)$ and $\mathscr{G}_2 = (V_2, E_2, M_2)$ where

Input: $\mathbf{s} = (s_1, \ldots, s_n, \ldots, s_N)$, an event type set $\mathscr{S}$, a constant integer $W_c \geq 2$: the correlation window size, and a constant θ: random link threshold.
Output: A thread transition graph $\mathscr{G} = (V, E, M)$ 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_{s_n,s_{n+w−1}} ← M_{s_n,s_{n+w−1}} + 1 ;
        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 DMST(𝒢) ;
end
```

Algorithm 1: ATM (All-pair, Threshold, directed MST)

Input: A graph $\mathscr{G}(V, E, M)$
Output: A refined graph $\mathscr{G}'(V, E', M')$ without shortcuts

```
begin
    E' ← ∅ ;
    sort the edge E into non-increasing order according to M ;
    foreach (i, j) ∈ E do
        if there exists no (directed) path from i to j in E' then
            E' ← E' ⋃ {(i, j)} ;
        end
    end
    M'_{i,j} ← M_{i,j} if (i, j) ∈ E' or M'_{u,v} ← 0 if (i, j) ∉ E' ;
    return 𝒢' = (V, E', M') ;
end
```

Algorithm 2: DMST (Directed Maximum Spanning Tree)

M_1 and M_2 are non-negative, we define their dissimilarity by

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

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. Here we define the weight $M_i(u,v) = 0$ if (u,v) does not exist in graph $\mathscr{G}_i$. 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 $0 \leq D(\mathscr{G}_1, \mathscr{G}_2) \leq 1$. The dissimilarity measure describes the dissimilarity between two behaviors. When two behaviors are exactly the same or 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 network data like an alert sequence, we can extract threads from the sequence and classify the corresponding graph as either normal or malicious one for each extracted behavior thread; and given user process data, we can classify the corresponding graph for a thread to be with a specific behavior or one owned by a particular user.

We adopt a *manifold learning* method called Isomap [38] for transition graph visualization and representation. The map produced by Isomap provides the interpretable insight for further investigation from domain experts; also, it may further enhance the performance of graph classification. As a purpose for evaluation, we conduct some classification tasks to demonstrate how effective the proposed method is. In this study, we adopt SVM [39] for the classification tasks. There are several SVM variants that have been proposed. We select smooth SVM (SSVM), which solves an unconstrained minimization problem [40] for our evaluation.

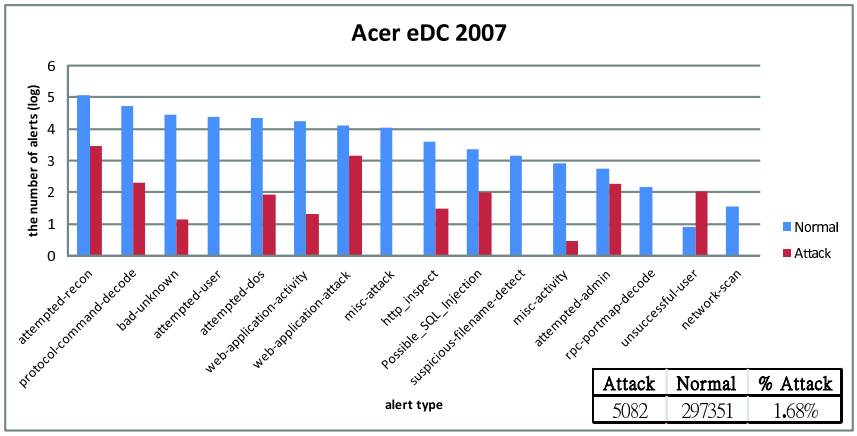
4. Data Set Description

We use two types of data sets for our 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[4] eDC (Acer e-Enabling Data Center). The host-based data set is the one for monitoring processes executed on desktop computers, also a self-collected data set. Table 1 summarizes the data statistics of our data sets.

Table 1. Data statistics. The labels in the Acer07 dataset indicate whether each alert event is a true alert or false alert; and the label of *User Process* denotes 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]http://www.acer-group.com/public/The_Group/overview.htm



Attack	Normal	% Attack
5082	297351	1.68%

Alert Type ID	Alert Type	% Attack
13	attempted-admin	25.67%
15	unsuccessful-user	93.33%

Fig. 5. The distributions (in *log*-scale) of alert types for normal behaviors and attacks in the Acer07 dataset.

4.1 Network-based Data

The first data set used to evaluate the proposed method is the Acer eDC 2007 dataset (Acer07)[5] [41]. 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. Two kinds of network intrusion detection systems (NIDSs) are used to identify suspicious network behavior in internal DMZ.

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.

To give a quick look of the Acer07 dataset, Figure 5 shows the distribution of all alert types. Compared to the well-known the DARPA99 dataset [42, 43], the Acer07 dataset does not have as many *true* alerts of types `attempted-recon` and `snort-portscan`. That may due to that these two types of "attacks" are no longer common, or no longer effective nowadays. In fact, most `attempted-recon` and `snort-portscan` alerts in Acer07 are false alerts or considered harmless. One explanation is that the Peer-to-Peer (P2P) becomes popular in recent years. If we just want to do port scanning to search the resources shared in the network, these actions may be considered legal. The network evolves, so as the intrusion detectors if we want them to remain effective.

[5] http://linker3.csie.ntust.edu.tw/∼pao/research/acer07.htm

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 Developer Network* (MSDN) [44], the .NET framework provides us the `Process` class to access the local and remote processes and we can enable the start and the stop the system process recording when necessary. We use some class properties to collect the process data, listed as follows,

- The `StartTime property`: Get the time when the process is started.
- The `HasExited property`: Get a value indicating whether or not a specific process has been terminated.
- The `ProcessName property`: Get the process name.

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 us for behavior analysis and use the analyzed result 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 event type pairs. Based on our 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. In the next sub-sections, we list all experimental parameters, followed by showing series of experiments for intrusion detections on network-based data and host-based data.

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 θ 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.

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, for (a) graph node statistics for overall sets, and (b) the size of the training and test sets.

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

(a)

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

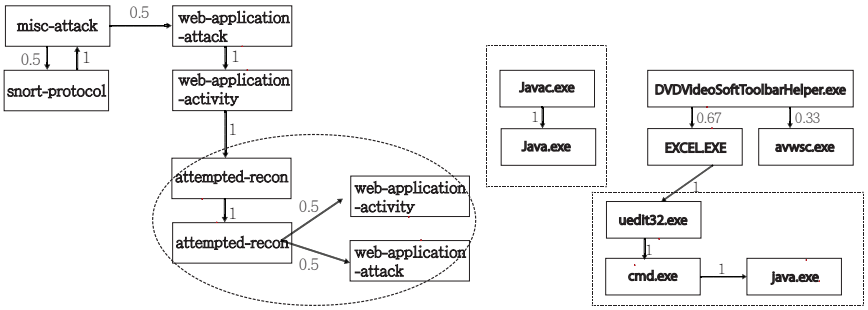
(b)

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

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 Figure 6(a), for the network data; and in (b), for the host-based process data. In (a) ,



(a) (b)

Fig. 6. (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.

we capture various multi-step attacks from the the DARPA99 dataset, while in (b), we observe that a user is trying Java programming, editing, compiling and running the code.

5.3 Quantitative Analysis

In the previous sub-section, we show the qualitative analysis about how the transition graph can represent user behavior. In real applications, the graph is shown to domain experts where the graph can help experts to build intuition and make decision to either accept the legal use or reject the malicious use from remote or host users. 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 `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 correlation window (of size W_c) to adjust how far we should still consider the two events correlated. 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 have the best result.

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 [40] to classify graphs into normal ones, or ones

Table 4. The classification result based on different choices of correlation window size ($W_c = 2$ to 11) and the Intrinsic Graphical Signature (IGS) method proposed by Pao et al. [41] (combined with SVM, Naïve Bayes (NB) and Logistic Regression (LR)), for the Acer07 dataset. To compare with the performance on the DARPA99 dataset (listed on the right), we choose the best $W_c = 9$ or 10 for the Acer07 dataset which is believed to contain highly interleaved events than that in the DARPA99 dataset (with the best $W_c = 4$). Also, the proposed ATM approach works better, in terms of F-measure than the IGS method [41] for the Acer07 dataset.

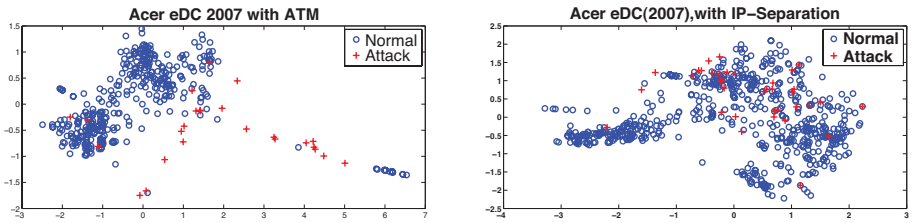
	Acer07					DARPA99				
W_c	F-score	P	R	Training Err.	Test Err.	F-score	P	R	Training Err.	Test Err.
2	0.36	1.00	0.22	0.14	0.31	0.67	0.80	0.58	0.78	0.10
3	0.29	0.53	0.21	0.15	0.44	0.66	0.68	0.64	0.75	0.10
4	0.44	0.66	0.33	0.18	0.34	**0.73**	0.66	0.82	0.15	0.23
5	0.38	0.66	0.26	0.08	0.38	0.63	0.78	0.52	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
IGS+SVM	0.46	0.36	0.64			—				
IGS+NB	0.50	0.39	0.71							
IGS+LR	0.50	0.50	0.50							

Table 5. Given the Acer07 dataset, F-score, precision, recall, and error ratios, if we use the proposed ATM algorithm, and the IP filtering method respectively. 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

with attacks involved. As a result, the proposed ATM method works better than the previous approach that has been applied for the same dataset. In Table 4, we observe that the proposed ATM achieve the F-measure equal to 0.76, which is superior to that from the IGS approach proposed by Pao et al. [41], when $W = 9$ or 10. Note that the comparison is hard to be 100% fair because the computation on detection accuracy, precision or recall can base on one alert, a group of alerts [41] or can be graph-based, which is the case for the proposed ATM method.

As a comparison with another approach to directly dealing with interleaved events, we compare the proposed method with a naïve benchmark approach from IP filtering, i.e., using the IP addresses (the spatial information, see Subsub-Section 3.2.2) to extract



(a) ATM ($\theta = 0.2$) (b) IP filtering

Fig. 7. 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.

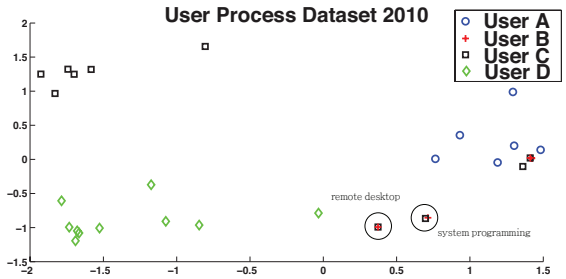


Fig. 8. Given the Process10 dataset, the 2-D Isomap plot for four user behaviors. Each instance represents a transition graph (or a thread) which describes the process transitions and therefore may reveal user behavior. We observe clear separation between different users. The instances on the bottom-right 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.

threads from sequences, then build transition graphs for classification, the ATM algorithm gives better results. 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 discuss the visualized result, Figure 7 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).

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. Figure 8 shows the 2-D Isomap plot of all user behaviors. Each

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

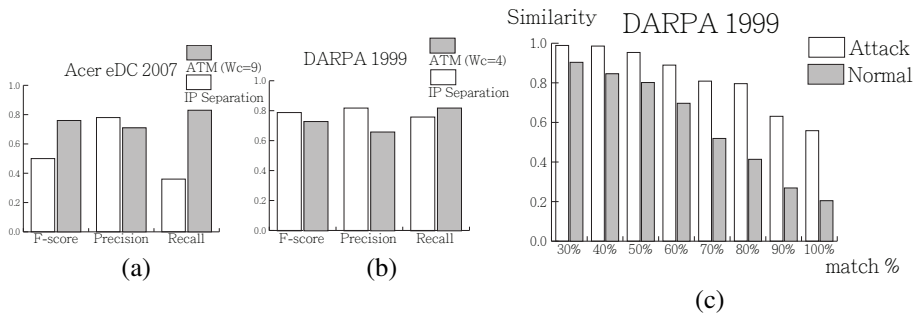


Fig. 9. The comparison between the ATM method and the IP filtering method for the performance of intrusion detection on (a) the Acer07 dataset and (b) the DARPA99 dataset. Both use the same graph dissimilarity measure. In (a), we have better performance from the ATM method, while in (b), the IP method gives better performance. It is because that the IP information in the DARPA99 dataset, a lab-produced dataset is likely to be reliable and a complex method like ATM looks redundant in this case. However, we suggest to use the ATM algorithm for real-world applications, as shown from the performance of the Acer07 dataset. (c) is 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.

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 behavior 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. Of course, we can apply any model fusion methods (e.g., [45]) and combine with other biometrics measures to further enhance the accuracy.

5.3.3 The Performance Comparison of ATM and IP Filtering Methods

We would like to compare the ATM method with the IP filtering method for the performance of intrusion detection. Figure 9 shows the comparison results on both of the Acer07 and DARPA99 datasets, using the same graph dissimilarity measure proposed in this work. For the Acer07 dataset, our proposed method performs better than the IP filtering method; on the other hand, for the DARPA99 dataset, our system is not as effective as the IP filtering method. How DARPA99 is generated may explain why this happens. DARPA99 is a lab-produced dataset, and most IP information is considered highly reliable; therefore, using only IP information for thread separation should perform well and

the ATM procedures such as all-pair computation, thresholding and MSDT computation look redundant. In the next paragraph, we show the IP filtering is no longer effective if we have some portion of fake IP existed in the DARPA99 dataset.

5.3.4 The Influence of Using Fake IP

As aforementioned, the IP address in the Internet environment may be fabricated. This subsub-section discusses about the performance of using forged IP addresses. In the experiment, we randomly replace some source IP addresses with wrong IP addresses. The wrong IP addresses are from an IP list that we randomly generate. In the experiment, we set a parameter to decide how many fake IP addresses in our data set. Table 7 shows the experiment result. If using the IP filtering approach in the dataset with some fake IPs, we will not have acceptable result. The ATM method can keep the same performance with or without the faked IPs because the IP information is not used in the method.

Table 7. The classification performance with fake IPs by the IP filtering method for the DARPA99 dataset.

Fake IP %	5%	15%	25%	35%
F-score	0.70	0.57	0.61	0.40
Precision	0.76	0.83	0.63	0.92
Recall	0.67	0.43	0.60	0.26
Training Error	0.21	0.23	0.22	0.23
Test Error	0.13	0.10	0.14	0.09

5.3.5 The Similarity of 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[6]. We use a similarity function defined below to measure the similarity between a graph generated by ATM $\mathscr{G}_{\text{ATM}}(V, E_{\text{ATM}})$ and a graph generated by IP filtering $\mathscr{G}_{\text{IP}}(V, E_{\text{IP}})$ as $\text{Sim}(\mathscr{G}_{\text{ATM}}, \mathscr{G}_{\text{IP}}) = \frac{|E_{\text{ATM}} \cap E_{\text{IP}}|}{|E_{\text{ATM}} \cup E_{\text{IP}}|}$, and we define the similarity of the result produced by the ATM method and the result produced by the IP filtering as $Similarity = \frac{\#(\text{Sim}(\mathscr{G}_{\text{ATM}}, \mathscr{G}_{IP}) > \alpha)}{\#\mathscr{G}_{\text{IP}}}$, where the parameter α is to determine whether two graphs are similar. In Figure 9(c), we show the histogram of similarity with different α 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 look not so important because the normal graphs just describe the transitions between false alerts. We believe that we can obtain reliable attack graphs based on the alert sequences derived from the DARPA99 dataset. That implies that our proposed ATM algorithm can produce reliable attack graphs.

[6]We design the experiment on the DARPA99 dataset because the IP information is likely to be reliable on this set.

6. Conclusion

In this work, we proposed a unified approach for intrusion detection given network-based and host-based event data which contain highly interleaved data. The proposed ATM algorithm can be used to extract threads 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 IGS [41] and IP filtering. The Isomap-projected visualization also shows that the ATM algorithm well separates points representing different behaviors, for both network-based data and host-based data. To focus on the graphs that we build based on the ATM algorithm, the attack transition graphs produced by the proposed ATM algorithm have about 80% similarity in about 80% of all graphs to the ones that are generated by the spatial information, i.e., IP addresses. A limit for the proposed approach is that the parameter setting, especially the choice of W_c could be hard to decide for different environment when we have no information about the number of different behaviors. Moreover, the information could be very dynamic and a timely adjustment is necessary. Overall, we believe that the proposed approach can provide a new direction to the intrusion detection, user authentication and other related research topics.

ACKNOWLEDGMENT

Research partially supported by the Ministry of Science and Technology Grant MOST 104-2221-E-011-165 and also supported in part by the Ministry of Science and Technology, National Taiwan University, and Intel Corporation under Grants MOST 105-2633-E-002-001 and NTU-ICRP-105R104045.

REFERENCES

1. M. Roesch, "Snort, the open source network intrusion system," in *http://www.snort.org*.
2. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. The MIT Press, 2009.
3. J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. ACM Press, 2003, pp. 2–11.
4. H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD-95)*, U. M. Fayyad and R. Uthurusamy, (eds.). Montreal, Canada: AAAI Press, 1995. [Online]. Available: citeseer.ist.psu.edu/mannila95discovering.html
5. R. Agrawal, T. Imielinski, and A. Swami, "Database mining: a performance perspective," *IEEE Trans. on Knowledge and Data Engineering*, Vol. 5, no. 6, 1993, pp. 914–925.

6. M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip, "Mining periodic patterns with gap requirement from sequences," *ACM Trans. Knowl. Discov. Data*, Vol. 1, no. 2, 2007, p. 7.
7. B. Ding, D. Lo, J. Han, and S.-C. Khoo, "Efficient mining of closed repetitive gapped subsequences from a sequence database," in *ICDE'09*, March 2009.
8. X. Ji, J. Bailey, and G. Dong, "Mining minimal distinguishing subsequence patterns with gap constraints," in *ICDM'05*, 2005.
9. 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 TKDD*, Vol. to appear, 2009.
10. J. Wang, J. Han, and C. Li, "Frequent closed sequence mining without candidate maintenance," *IEEE TKDE*, Vol. 19, no. 8, 2007, pp. 1042–1056.
11. DMOZ, "DMOZ: Open Directory Project," http://www.dmoz.org/Computers/Security/Intrusion_Detection_Systems/.
12. 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.
13. F. Cuppens and A. Miege, "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.
14. F. Valeur, G. Vigna, C. Kruegel, and R. A. Kemmerer, "A comprehensive approach to intrusion detection alert correlation," *IEEE Trans. Dependable Secur. Comput.*, Vol. 1, no. 3, Jul. 2004, pp. 146–169. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2004.21
15. R. Sadoddin and A. A. Ghorbani, "Alert correlation survey: framework and techniques." in *PST*, ser. ACM International Conference Proceeding Series, Vol. 380. ACM, 2006, p. 37. [Online]. Available: http://dblp.uni-trier.de/db/conf/pst/pst2006.html#SadoddinG06
16. S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, Vol. 10, no. 1, 2002, pp. 105–136.
17. A. Valdes and K. Skinner, "Probabilistic alert correlation," *Lecture notes in computer science*, 2001, pp. 54–68.
18. K. Julisch, "Clustering intrusion detection alarms to support root cause analysis," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 6, no. 4, 2003, p. 471.
19. F. Cuppens and R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," in *Recent advances in intrusion detection: third international workshop, RAID 2000, Toulouse, France, October 2-4, 2000: proceedings*. Springer Verlag, 2000, p. 197.
20. O. Dain and R. Cunningham, "Fusing a heterogeneous alert stream into scenarios," in *Proceedings of the 2001 ACM Workshop on Data Mining for Security Applications*, November 2001, pp. 1–13.
21. S. Templeton and K. Levitt, "A requires/provides model for computer attacks," in *Proceedings of the 2000 workshop on New security paradigms*. ACM, 2001, pp. 31–38.
22. P. Ning, Y. Cui, D. Reeves, and D. Xu, "Techniques and tools for analyzing intrusion alerts," *ACM Transactions on Information and System Security (TISSEC)*, Vol. 7, no. 2, 2004, p. 318.

23. P. Ning, Y. Cui, and D. S. Reeves, "Constructing attack scenarios through correlation of intrusion alerts," in *In Proceedings of the 9th ACM conference on Computer and communications security*, Washington, D.C., November 2002, pp. 245–254.
24. T. Hughes and O. Sheyner, "Attack scenario graphs for computer network threat analysis and prediction," *Complexity*, Vol. 9, no. 2, 2003, pp. 15–18.
25. S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, 2002 2002, pp. 49–63.
26. O. M. Sheyner, "Scenario graphs and attack graphs," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.
27. R. Ritchey, P. Ammann, A. Booz, H. Inc, and F. Church, "Using model checking to analyze network vulnerabilities," in *2000 IEEE Symposium on Security and Privacy, 2000. S&P 2000. Proceedings*, 2000, pp. 156–165.
28. 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*, May 2002.
29. 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.
30. S. Noel and S. Jajodia, "Optimal IDS sensor placement and alert prioritization using attack graphs," *Journal of Network and Systems Management*, Vol. 16, no. 3, 2008, pp. 259–275.
31. D. Saha, "Extending logical attack graphs for efficient vulnerability analysis," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 63–74.
32. B. Zhu and A. Ghorbani, "Alert correlation for extracting attack strategies," *International Journal of Network Security*, Vol. 3, no. 3, 2006, pp. 244–258.
33. N. Idika and B. Bhargava, "Extending attack graph-based security metrics and aggregating their application," *Dependable and Secure Computing, IEEE Transactions on*, Vol. 9, no. 1, Jan 2012, pp. 75–85.
34. KDD'99, "KDD cup 1999 data," http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html, 1999.
35. C. Elkan, "Results of the KDD'99 classifier learning," *SIGKDD Explor. Newsl.*, Vol. 1, no. 2, Jan. 2000, pp. 63–64. [Online]. Available: http://doi.acm.org/10.1145/846183.846199
36. M. Tavallaee, E. Bagheri, W. Lu, and A.-A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *Proceedings of the Second IEEE Symposium on Computational Intelligence for Security and Defence Applications 2009*, 2009.
37. 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705111002681
38. J. Tenenbaum, V. Silva, and J. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, Vol. 290, no. 5500, 2000, p. 2319.
39. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, Vol. 2, no. 2, 1998, pp. 121–167.
40. Y.-J. Lee and O. L. Mangasarian, "SSVM: A smooth support vector machine for classification," *Comput. Optim. Appl.*, Vol. 20, no. 1, 2001, pp. 5–22.
41. 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."

J. Inf. Sci. Eng., Vol. 28, no. 2, 2012, pp. 243–262. [Online]. Available: http://dblp.uni-trier.de/db/journals/jise/jise28.html#PaoMLCF12
42. M. L. Lab, "1999 DARPA intrusion detection evaluation data set," http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html, 1999.
43. J. McHugh, "The lincoln laboratory intrusion detection evaluation: A critique," in *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition*, 2000.
44. Microsoft, "MSDN library," http://msdn.microsoft.com/en-us/default.aspx.
45. T. Sim, S. Zhang, R. Janakiraman, and S. Kumar, "Continuous verification using multimodal biometrics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29, 2007, pp. 687–700.

Hsing-Kuo Pao Hsing-Kuo Pao (Kenneth) 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 in Vita Genomics as a research scientist. In 2003, he joined the department of computer science and information engineering in National Taiwan University of Science and Technology, and now he is an associate professor. His current research interests include machine learning, information security and computer vision.

PLACE
PHOTO
HERE

Fong-Fuei Lee Fong-Fuei Lee received the master degree in computer science and information engineering from from National Taiwan University of Science and Technology in 2010. He is currently working on network security in industry.

Yuh-Jye Lee Yuh-Jye Lee received the PhD degree in Computer Science from the University of Wisconsin-Madison in 2001. He is currently a Professor of Department of Computer Science and Information Engineering at National Taiwan University of Science and Technol- ogy. He also serves as a principal investigator at the Intel-NTU Connected Context Computing Center. His research is primarily rooted in optimization theory and spans a range of areas including network and information security, machine learning, big data, data mining, 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 focus on online learning algorithms for dealing with large scale datasets, stream data mining and behavior based anomaly detection for the needs of big data, Internet of Things data analytics and machine to machine communication security problems.