The ISC Int'l Journal of Information Security

ISeCure

July 2011, Volume 3, Number 2 (pp. 77–101)

http://www.isecure-journal.org

Alert Correlation and Prediction Using Data Mining and HMM

Hamid Farhadi [1], Maryam AmirHaeri [1], and Mohammad Khansari [2]

[1] *Data and Network Security Lab, Department of Computer Engineering, Sharif University of Technology, Tehran, Iran*

[2] *Faculty of New Sciences and Technologies, University of Tehran*

ARTICLE INFO.

Article history:

Received: 12 July 2010

Revised: 13 September 2011

Accepted: 28 September 2011

Published Online: 28 October 2011

Keywords:

Alert Correlation, Multistep Attack Scenario, Plan Recognition, Hidden Markov Model, Intrusion Detection, Stream Mining.

ABSTRACT

Intrusion Detection Systems (IDSs) are security tools widely used in computer networks. While they seem to be promising technologies, they pose some serious drawbacks: When utilized in large and high traffic networks, IDSs generate high volumes of low-level alerts which are hardly manageable. Accordingly, there emerged a recent track of security research, focused on alert correlation, which extracts useful and high-level alerts, and helps to make timely decisions when a security breach occurs.

In this paper, we propose an alert correlation system consisting of two major components; first, we introduce an Attack Scenario Extraction Algorithm (ASEA), which mines the stream of alerts for attack scenarios. The ASEA has a relatively good performance, both in speed and memory consumption. Contrary to previous approaches, the ASEA combines both prior knowledge as well as statistical relationships. Second, we propose a Hidden Markov Model (HMM)-based correlation method of intrusion alerts, fired from different IDS sensors across an enterprise. We use HMM to predict the next attack class of the intruder, also known as *plan recognition*. This component has two advantages: Firstly, it does not require any usage or modeling of network topology, system vulnerabilities, and system configurations; Secondly, as we perform high-level prediction, the model is more robust against over-fitting. In contrast, other published plan-recognition methods try to predict exactly the next attacker action. We applied our system to DARPA 2000 intrusion detection scenario dataset. The ASEA experiment shows that it can extract attack strategies efficiently. We evaluated our plan-recognition component both with supervised and unsupervised learning techniques using DARPA 2000 dataset. To the best of our knowledge, this is the first unsupervised method in attack-plan recognition.

1 Introduction

Nowadays, intrusions into computer systems have grown enormously. Attacks are increased both in quantity and sophistication. Intrusion Detection System (IDS) is a tool that can detect attacks with practically reasonable accuracy.

We can classify IDSs based on their detection mechanism into two classes: misuse detection and anomaly detection. An IDS can take advantage of either one of them or can be a hybrid of both methods. Misuse detection method uses a predefined database of attack signatures. Any matching traffic with a signature fires an attack alarm. In anomaly detection, we try to make some profiles from benign traffic in the learning phase. Then, in the testing phase, any traffic that deviates from the benign traffic profile is announced as anomalous traffic. The former method

Email addresses: farhadi@alum.sharif.edu (H. Farhadi), haeri@ce.sharif.edu (M. AmirHaeri), m.khansari@ut.ac.ir (M. Khansari).

ISSN: 2008-2045

suffers from high false negative detection rate caused by inadaptability of the method to new attacks. The latter may suffer from high false positive rates due to the inaccuracy of profiling methods, used in the IDS.

When we are dealing with large networks with many sensors, we cope with too many alerts fired from IDS sensors each day. Managing and analyzing such amount of information is a difficult task. There may be many redundant or false positive alerts that need to be discarded. Therefore, in order to extract useful information from these alerts we use an *alert correlation* algorithm, which is the process of producing a more abstract and high-level view of intrusion occurrences in the network from low-level IDS alerts.

Alert correlation is also used to detect sophisticated multistep attacks. A *multistep attack* is defined as a sequence of simple attacks that are performed successively to reach some goal. During each step of the attack some vulnerability, which is the prerequisite of the next step, is exploited. In other words, the attacker chains a couple of vulnerabilities and their exploits to break through computer systems and escalate his privilege. In such circumstances, IDSs generate alerts for each step of the attack. In fact, they are not able to follow the chain of attacks and extract the whole scenario. Getting advantage of alert correlation, it is possible to detect complex attack scenarios out of alert sequences. In short, alert correlation can help the security administrator to reduce the number of alerts, decrease the false-positive rate, group alerts based on alert similarities, extract attack strategies, and predict the next step(s) of the attacks.

In this paper, we propose an alert correlation system consisting of two major and two minor components. Minor components are the Normalization and the Preprocessing components that convert heterogeneous alerts to a unified format and then remove redundant alerts. Major components are the ASEA and the Plan Recognition components that extract current attack scenario and predict the next attacker action.

In the ASEA component, we used data mining to correlate IDS alerts. The stream of attacks is received as input, and attack scenarios are extracted using stream mining. While reducing the problem of discovering attack strategies to a stream-mining problem has already been studied in the literature, current data-mining approaches seem insufficient for this purpose. We still need more efficient algorithms as there are a plethora of alerts and we need real-time responses to intrusions. This issue is discussed in more details subsequently.

Intrusion detection methods require human knowledge for supervised gathering and acquisition of data. Using Machine Learning (ML) methods, one can omit such a manual necessity and automate the process. It also makes the method independent of various intrusion types. However, ML techniques face some challenges. There are only a few examples of complex Internet breaches publicly available. Thus, it is difficult to apply ML on them, since most of ML algorithms require many examples to be trained. Besides, this problem affects the testing phase and performance measurement. As a new approach, we move to probabilistic approaches like Hidden Markov Models (HMMs). This tool can model both sequential and probabilistic nature of sophisticated intrusions. The insight behind this idea is that ordering property of intrusions provides some advantages over other properties such as frequency properties in detection.

In the Plan Recognition component, we used HMM to predict the next attack class of the intruder that is also known as *plan recognition*. The main objective of the attack plan recognition is to arm the management with information supporting timely decision making and incident responding. This helps to block the attack before it happens and provides appropriate timing for organizing suitable actions.

Consider a four-step attack in which the first three steps are occurred and detected by IDSs. If the fourth step is a zero-day attack, it is unlikely to be detected by the IDSs so that they fail to alert the administrator. On the other hand, if we try to predict the future attack at a high granularity level, we may determine (at least) the next class of attack that is going to happen. Given the dynamic nature of attacks, it is beneficial to perform prediction at coarser granularity level. Since particular attack methods usually evolve during time, modeling and prediction in fine-grained levels can be error prone. Thus, prediction at coarser levels gives an overall sight and analysis of the target network rather than specific but not necessarily precise prediction at small granularities.

1.1 The Reference Architecture

Figure 1 represents the integrated correlation process in which our solution plugs. “Normalization” and “Pre-Processing” take place on each incoming alert. The former converts heterogeneous events from varying sensors into a single standardized format which is accepted by the other correlation components. This unit is very helpful when we are dealing with different encodings from different sensors. The latter component ensures that all required attributes in alerts such as source, destination, and start time of the attack have meaningful values.

The “Alert Fusion” component combines alerts issued from different sensors, but related to the same

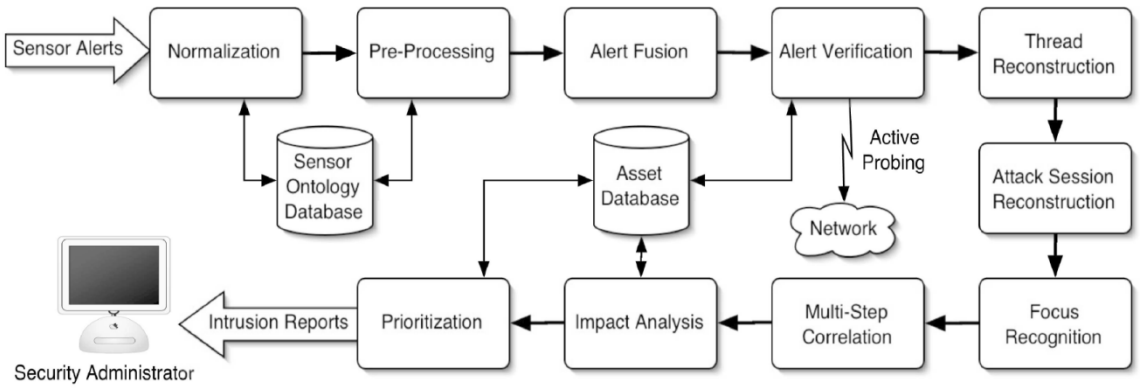


Figure 1. Correlation Process Overview [1]

activity. The "Alert Verification" unit takes an alert as an input and determines if the suspicious corresponding attack is successfully performed. Failed attacks are then labeled so that their effectiveness will be decreased in upcoming correlation phases. The "Thread Reconstruction" component combines and series the attacks having the same source and target addresses. In the "Attack Session Reconstruction" component, both network-based alerts and host-based alerts that are related to the same attacks are gathered and associated.

All four previous components (i.e. Alert Fusion, Alert Verification, Thread Reconstruction, and Attack Session Reconstruction) operate on similar alerts and try to simplify later processing through some sort of unification and reducing redundancy. The next two units (i.e. "Focus Recognition" and "Multi-Step Correlation") deal with attacks that are potentially targeted at wide range of hosts in the enterprise. The "Focus Recognition" component identifies those hosts to which a considerable number of attacks are targeted or originated from. This component hopefully detects port scanning attempts as well as Denial of Service (DoS) attacks. The "Multi-Step Correlation" component identifies common attack patterns, which are composed of different individual attacks possibly occurring at different zones in the network. Finally, the "Impact Analysis" component calculates the impact factors of current attack(s) on the target network and assets. In the last two components (i.e. "Impact Analysis" and "Proioritization"), based on the target network, alerts will be contextualized as the final stage of the correlation process. The "Proioritization" component ends the process with classifying events in different importance groups providing faster ability to find relevant information about a specific host or site.

In this research we focus on the "Multi-Step Correlation" component to extract attack scenarios using sequence mining and to predict future attacker behavior based on a learned Hidden Markov Model (HMM).

The remainder of this paper is organized as follows: The next section discusses the related work. Section 3 describes our system architecture. Our correlation model and attack prediction algorithm is discussed in Section 4. Section 5 explains the ASEA components. Section 6 briefly reviews HMM. Section 7 details the Plan Recognition component. In Section 8 and 9, we present the experimental results. The last section concludes the paper.

2 Related Work

The related works fall into two categories. First, works related to intrusion correlation for detecting complex attack scenarios, and second, works related to predicting the attacker's next action(s).

2.1 Intrusion Detection

Valeur *et al.* [1] proposed a comprehensive approach to alert correlation. The paper defines a five step process to correlate alert sequences: preprocessing, reconstructing attack session, attack prioritization, effect analysis and reporting intrusion.

However, most proposed alert correlation methods do not consider all of these processes. Pietraszek [2], Smith *et al.* [3], and Morin *et al.* [4] tried to reduce the amount of alerts as well as decreasing the false positive rate. Some concentrate on clustering similar alerts (as in Cuppens and Miège [5], and Peng *et al.* [6]) and others focus on discovering attack strategies (as in Li *et al.* [7], Zhu and Ghorbani [8], and Al-Mamory and Zhang [9]).

The community proposed variant approaches based on attacks' prerequisite relations. Among them JIGSAW (Templeton and Levitt [10]) introduced a language to describe attacks in terms of their prerequisites and consequences and made a database of such attacks to detect complicated multistage attack scenarios.

Morin *et al.* [4] proposed M2D2 data model and used formal tools for intrusion alert correlation. They put together some motivating and relevant concepts to construct a model. The model tries to raise the total detection accuracy while keeping the false positive as low as possible.

Al-Mamory and Zhang [9] proposed a different approach to discovering attack scenarios based on context free grammars. They express the similarity between the "construction of attack graph from alerts" and "compiling a program and generating its machine code". The method describes attack scenarios using an attribute context-free grammar. Attributes denote prerequisites and consequences of attacks. For each alert stream, an SLR parser generates the parse tree and then the attack graphs are constructed similar to generating executable code.

Zhu and Ghorbani [8] utilized neural networks for extracting attack scenarios without using prior knowledge. They used a multilayer perceptron is used to extract the causal correlation between a pair of alerts. In this method, the causal correlations all newly arrived alerts are calculated to determine which previous alert should be correlated to the new alert

In addition to previous methods, some data mining based approaches are presented to the community. Shin and Jeong [11] presented a data mining framework for alert correlation. Their framework consists of three components: (1) association-rule miner, (2) frequent-episode miner and (3) cluster miner. This method incorporates some data mining methods which are inefficient for this application such as *Apriori* algorithm to extract frequent episodes. This algorithm suffers from being too time consuming. Li *et al.* [7] proposed a statistical filtering approach which is based on sequence-mining. The filter discards irrelevant and scattered alerts to reduce the amount of alerts. The algorithm is incapable of discovering infrequent attacks while it can mine for frequent attack sequences.

2.2 Attack Plan Recognition

Bayesian networks and HMMs were used for alert correlation. In De Vel *et al.* [12], the input data is divided into T intervals, each of which forms a hidden state of an HMM. Observations are organized as a Bayesian network including case specific evidences of criminal activities. In computer forensics, this structure is used to detect evolution over time of a criminal activity. Even though, the idea of embedding a Bayesian network in an HMM is quite novel in the community. The processing is performed in the off-line mode and after crime commitment. Therefore, the prediction capability of the system is not effective. In Ourston *et al.* [13], HMM is used to detect the current intrusion through its prerequisites. To the best of our knowledge, this is the first attempt to use HMM for alert correlation.

In addition to Bayesian networks and HMMs, signature-based correlation engines have also been used as in Lee and Qin [14] to detect multi-stage attacks. It uses a six-stage predefined HMM for each attack (as the attack signature) and puts them all in a signature database. Then, using the HMM sequence decoding algorithm and alerts as observations, it can compute the most probable signature in the database to detect current multi-stage attack. Of course, detection capability of the system is limited to the predefined signature database. Similar to the other signature based detection systems, it suffers from poor adaptiveness to newly introduced attack patterns.

Detecting the current attack using its prerequisites and/or consequences is proposed in Zhai *et al.* [15] where the concept of state-based evidences (e.g. open ports) is defined. It uses this kind of extra information in addition to raw IDS alerts as the input of a Bayesian network based inference engine to cope with the detection of current attacks.

All the aforementioned methods use the sequential nature of multi-step cyber attacks, getting advantage of sequence modelers such as Bayesian networks and HMMs. Nevertheless, unlike previously explained methods, in plan recognition component, we address the problem of predicting the attacker's next possible action employing ordering properties of correlated IDS alerts. In other words, the component is not an intrusion detection system; it settles on top of a series of IDS sensors to obtain attack patterns and predict their behaviors. Some old approaches to sequence prediction exist (such as Ehrenfeucht and Mycielski [16]) that look for the longest repeated suffix of a sequence. More complicated approaches were introduced later, including Qin and Lee [17] and Lee and Qin [18]. These series of complementary articles use adaptive Bayesian-based correlation engine for attack steps that are directly related to each other (i.e. pre/post conditions) similar to a signature based detection engine. Moreover, they exploit Granger Causality Test (GCT) to recognize unknown attacks without any prior knowledge close to an anomaly detection engine. Limitation of this method is its non-automatic construction of the initial lattice by a domain expert. The technique also suffers from performance related issues because of its computational complexity. Furthermore, the GCT target needs to be specified manually; otherwise, the pairwise GCT operation can be quite time consuming.

In Ning and Gong [19], another method was proposed named Max-1-Connected Bayesian Network (M1CBN), which improved the computation performance (in particular with polynomial complexity) of

prior polytree-based approaches (Lee and Qin [18]). The polytree is defined as the library of attack plans. The paper has also provided more expressive power to illustrate attack plans. Although it tries to improve the computational performance of polytree, it still has the problem of detecting newly introduced attacks. It uses a prepared and relied attack plan library which results in the same bottleneck as static signature based approaches.

Finite-state models (i.e. Markov based models) are also center of concentration in attack behavior prediction. In Fava *et al.* [20], Variable Length Markov Model is exploited for attacker plan prediction. The idea is a sequence modeling technique ported from other fields like text compression and is adaptive to newly observed attack sequences. Farhadi *et al.* [21] uses Markov Model as a supervised tool to predict the next attacker action.

In the following, after some explanations about our system architecture, we first discuss our Alert Scenario Extraction Algorithm (ASEA) that detects complex attack scenarios, and then discuss our HMM-based attack plan recognition method.

3 System Architecture

Figure 2 depicts main components of the proposed system. In the following, we first describe Normalization and preprocessing components, and then we discuss ASEA and Plan recognition components in the next sections.

3.1 Alert Normalization

Correlation engines, especially in large networks, usually receive a vast number of alerts from many heterogeneous detection sensors each of which may encode alerts in a different format. In alert normalization, these inconsistencies are converted into a single common unified format so that later processing units can operate transparently from detection sensors. In such a conversion, the alert format of each type of sensor must be syntactically and semantically defined in advance. The conversion can be implemented either in a centralized form or in a decentralized form. In the centralized implementation, sensors send alerts in their vendor defined format and then all alerts are translated into the appropriate format in the correlation engine side. In contrast, in decentralized implementation the load is distributed among normalization modules on sensors. The former has simpler implementation while the latter has better performance. We use the Intrusion Detection Message Exchange Format (IDMEF) standardized by the Internet Engineering Task Force (IETF) as alert input data model.

3.2 Alert Pre-Processing

Due to the possible existence of many security devices in our target environment, an abundant number of alerts might be generated per second. This fact makes it difficult to perform any kind of processing on the data in real time or at least in a timely fashion. Accordingly, we need to reduce redundancies via clustering, aggregation, and fusion to decrease the volume of alerts. Our preprocessing approach in this unit is based on Lee and Qin [18]. The goal of this unit is alert volume reduction, keeping important and effective alert attributes such as timestamp, source and destination IP addresses, port number, user name, process name, attack class, and sensor ID. These attributes are all defined in the IDMEF standard.

In alert aggregation, we gather alerts which have slightly different timestamps but contain the same attributes to form a *super alert*.

Assume the collection of intrusion alerts to be $\Lambda = \{\delta_i, i = 1, 2, \ldots\}$, in which δ_i is a super alert. δ_i is an aggregated set of alerts $\{a_{(i,1)}, a_{(i,2)}, a_{(i,3)}, \ldots\}$ holding the same attributes (e.g. same port number and source IP address) but with slightly different timestamps that falls into the predefined window size of T. Let $a_{(i,j)}$ be an intrusion alert composed of several attributes $\langle u_1, u_2, u_3, \ldots\rangle$. For instance, u_1 can be the source IP address and u_2 can represent the destination IP address and so on. Now, we denote a super alert as follows:

$$\delta_i = \left\{ \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{pmatrix}_{(i,1)}, \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{pmatrix}_{(i,2)}, \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \end{pmatrix}_{(i,3)}, \ldots \right\}$$

4 ASEA

In this section we introduce ASEA for extracting attack scenarios from the stream of IDS alerts. Subsection 4.1 describes several data mining terms we use later. Subsection 4.2 is dedicated to formalization of our model. In Subsection 4.3, we outline the proposed algorithm.

4.1 Data Mining Terminology

In this section we proposed the definition of terms and expressions that are used throughout the paper. Even though the data mining community is familiar with them, they may be alien to security experts.

Itemset: A non-empty set of items.
Sequential Pattern: An ordered list of items in a

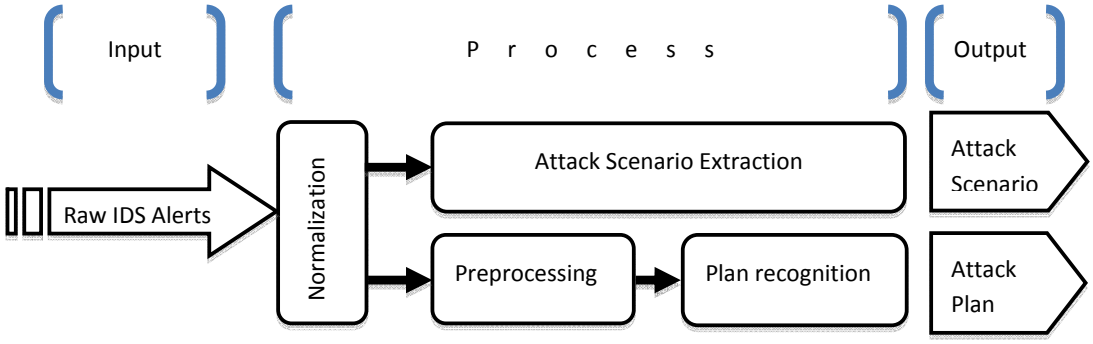


Figure 2. System Architecture

data stream.

Support Count: The frequency of an itemset in a data set.

Minimum Support Count (`min_sup`): The minimum number of an itemset in a data set in order to be considered as frequent.

Frequent Itemset: An itemset with support count of at least `min_sup`.

Frequent Sequential Pattern: A pattern $s = s_1, s_2, \ldots, s_n$ which, for all $1 \leq i < j \leq n$, we have $s_i \neq s_{i+1}$, and item s_i has occurred at least `min_sup` times before item s_j in the data stream.

Super-pattern: A pattern which has all items of another pattern in the same order. Formally, pattern $s = s_1, s_2, \ldots, s_n$ is a super-pattern of pattern $t = t_1, t_2, \ldots, t_m$ if there exists m indices $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ such that for every $1 \leq j \leq m$, we have $s_{i_j} = t_j$

Proper Super-pattern: A pattern which is super-pattern of another pattern, but is not equal to it.

Maximal Sequential Pattern (`MSP`): A frequent sequential pattern which none of its proper super-patterns are frequent.

4.2 A Formal Description of the Model

We assume a network in which several IDSs are deployed. IDSs generate a stream of alerts, which is collected at a central place for further investigation (i.e. alert correlation). This stream should be sorted based on the timestamp of the alerts.

We formally describe an alert as a binary string. In other words, the set of possible alerts is $\mathcal{A} \subset \{0,1\}^*$. Each alert is assigned an ID, which distinguishes its "type". For instance, all alerts of type `Sadmind_Ping` are assigned the same alert ID. Let $\mathcal{ID} \subseteq \mathbb{N}$ be the set of possible alert types. Define a function $@\colon \mathcal{A} \to \mathcal{ID}$, which takes an alert and returns its type. (We use the symbol "@"—pronounced AT—as a mnemonic for "Alert Type"). As a convention, to distinguish alerts from alert types, we hereafter use lowercase Greek letters for alerts, and lowercase Latin letters for alert types.

We need the correlation of any two alert types to construct attack scenarios. Some approaches use *statistical analysis* (as in Li *et al.* [7] and Zhu and Ghorbani [8]) while others use *prior knowledge* to determine the correlation (as in Al-Mamory and Zhang [9], and Bahreini *et al.* [22]). The correlation should reflect the causal relationships of alert types. For instance, if alert type $a \in \mathcal{ID}$ is a prerequisite to alert type $b \in \mathcal{ID}$ but the inverse is not true, the correlation of a and b should be higher than the correlation of b and a. To emphasis this property, we hereafter refer to correlation as *causal correlation*, and denote it by function $\Psi\colon \mathcal{ID} \to [0,1]$. The above description is then formally translated into $\Psi(a,b) \gg \Psi(b,a)$. We use prior knowledge to compute causal correlation. Note that unlike correlation coefficient, Ψ cannot be negative.

Similar to Li *et al.* [7], Zhu and Ghorbani [8], and Valdes *et al.* [23], we use a matrix to store causal correlations of alert types. A *Causal Correlation Matrix* (`CCM`) for alert types $a_1, a_2, \ldots, a_n \in \mathcal{ID}$ is defined as an $n \times n$ matrix whose (i,j)-element is $\Psi(a_i, a_j)$, where $1 \leq i, j \leq n$. This matrix is usually asymmetric.

Two alert types are *correlated* if their causal correlation is not less than a certain threshold. Formally, let τ be the causal correlation threshold, adjusted by network administrator. Alert types (in the specified order) are correlated if and only if . Two alerts are correlated if their corresponding alert types are correlated. That is, alerts (in the specified order) are correlated if and only if . We define a Correlated Frequent Sequential Pattern (`CFSP`) as a sequential pattern in which every two adjacent alerts are correlated.

Our approach to find `CFSP`s is simple: An alert is deemed important if it is frequent. Thus, we first find `MSP`s and add them to `MSP_Table`. Based on the definition of `MSP`, the `MSP`s of each window contain all frequent sequential patterns (`FSP`s) of it. So, keeping track of `MSP`s instead of `FSP`s helps in saving memory.

Then we check whether each sub-pattern of an MSP is CFSP, by verifying the correlation condition described above. If an FSP is found to be a CFSP, we add it to CFSP_Table.

Importance caused by high frequency can sometimes be inaccurate. There are frequent alerts, which are unimportant such as "stop words" in the case of search engines. Moreover, we may face some infrequent alerts which are important. In both cases we need to take care of such alerts and we should recognize them beforehand, through trial-and-error. As a simple solution, it is possible to assign an *importance factor* to each alert type, based on which its *minimum support count* is raised (in the former case) or lowered (in the latter case). We define the *importance factor* function $\mathcal{I}\colon \mathcal{ID} \to (0, \infty)$. The function takes and alert type, and returns its importance. The more important an alert, the lesser its associated "minimum support count". In the initialization step, the algorithm computes $\mathcal{I}(a)$ for each alert type $a \in \mathcal{ID}$, and computes $\texttt{min\_sup}_a$ based on $\mathcal{I}(a)$ and a system-wide parameter min_sup.

Finding FSPs in a stream of alerts is a resource-intensive task. Usually we are dealing with long sequences that it is infeasible to calculate FSPs for them. To overcome this limitation, algorithms split the stream into windows, and find FSPs in each window separately. The window size (L) can be either a time interval or the alert count. In the former case, the window contains a number of alerts raised in fixed time intervals. In the latter case, the window contains a fixed number of alerts. We adopt the second approach.

Suppose that $\alpha_1, \alpha_2, \ldots$ is a stream of alerts. We can divide the window-based algorithms into three broad categories:

Sliding window: In this approach, once the algorithm is run with a window, the window "slides" Δ alerts in the stream ($1 \leq \Delta \leq L$). That is, if $[\alpha_i, \alpha_{i+1}, \ldots, \alpha_{i+L-1}]$ is a window, the next window will be $[\alpha_{i+\Delta}, \alpha_{i+\Delta+1}, \ldots, \alpha_{i+\Delta+L-1}]$. Any two adjacent windows share $L - \Delta$ alerts.

Landmark window: In this approach, a "landmark" alert is selected, and every window contains alerts from the landmark up until now. That is, if α_i is the landmark alert, and α_t is the current alert, the windows will be $[\alpha_i]$, $[\alpha_i, \alpha_i + 1]$, $\ldots$, $[\alpha_i, \alpha_{i+1}, \ldots, \alpha_t]$. Note that if the landmark alert is the first one, the last landmark window is the whole alert stream.

Damped window: This approach is similar to the sliding-window model, but it gives "weights" to each window in computations. In general, the more recent a window, the higher the weight.

Our approach is similar to the sliding-window model, but it borrows ideas from the other two models. We describe the similarities after justifying our choice of parameters.

The sliding window deals with two parameters: Δ and L. At one extreme, $\Delta = 1$. At the other extreme, $\Delta = L$. For a stream with N alerts, the choices correspond to $N - L + 1$ and $\lceil \frac{N}{L} \rceil$ windows, respectively. It is obvious that in large data streams, the latter choice outperforms the former by a factor of L, but it is less accurate. In order not to sacrifice accuracy in favor of performance, we introduced the *retrospect factor* (ρ). This parameter determines how many prior windows should be looked back in order to find longer correlated patterns. Subsection 4.3 gives a more detailed description.

Experimental results show that, by picking ρ and L wisely, the loss of accuracy due to the adoption of $\Delta = L$ is negligible.

4.3 Algorithm Outline

Algorithm 1 is an outline of ASEA. We tried to make the outline as readable as possible, yet there are some points that need to be explained. We will go through the algorithm step-by-step:

Steps i–ii (Initialization): In these steps, the algorithm computes $\texttt{min\_sup}_a$ for each alert type $a \in \mathcal{ID}$. The computation can be tweaked based on the network environment, but a naïve suggestion is to let $\texttt{min\_sup}_a \leftarrow \lceil \frac{\texttt{min\_sup}}{\mathcal{I}(a)} \rceil$.

Step 1–2: Step 1 sets the current window number ($i \leftarrow 1$). Step 2 sets the sequence—in which we temporarily store alerts—to empty ($Q \leftarrow \varnothing$). The sequence holds alerts until its size equals L.

Steps 3–4: The algorithm blocks at this step, until an alert receives. The alert triggers steps 3–21 (We used event-based model to simplify the algorithm. It can be easily translated into poll-based model). In step 4, we concatenate the newly-received alert to Q ($Q \leftarrow Q \parallel \alpha$, where the binary operator $\parallel$ denotes concatenation).

Step 5: Checks whether the size of the temporary sequence is L. If so, there are enough alerts to form a window, and we proceed to steps 6–21.

Steps 6–8: The alerts are sorted based on their timestamps, copied to a window, and the temporary sequence is freed.

Step 9: MSPs of current window are mined and added to MSP_Table. This table stores MSPs as well as the window number in which they are found.

Step 10: Each FSPs of current window is checked whether it is CFSP according to the CCM. Add the current CFSPs to the CFSP_Table.

Step 11–18: After Finding the current CFSPs CFSP_Table should be examined that if the current CFSPs

Algorithm 1 ASEA

Input:

- Alert stream (alerts are collected from all sensors or IDSs in the network and sorted by their occurrence time).
- Window size (L)
- Minimum Support Count (`min_sup`)
- Importance Factor (Function $\mathcal{I}$: $\mathcal{ID} \rightarrow (0, \infty)$)
- Retrospect Factor (ρ)
- Causal correlation Matrix (`CCM`)
- Correlation Threshold (τ)

Output:

- Attack Scenarios

Algorithm:

Initialization (Required once):

i. For each alert type $a \in \mathcal{ID}$ in the stream, compute $\mathcal{I}(a)$.

ii. Calculate $\texttt{min\_sup}_a$ according to `min_sup` and $\mathcal{I}(a)$.

1: **let** $i \leftarrow 1$.
2: **let** $Q \leftarrow \varnothing$.
3: On receipt of each new alert α:
4: **let** $Q \leftarrow Q \mid\mid \alpha$.
5: **if** $|Q| = L$ **then**
6: Sort alerts in Q based on their timestamps.
7: **let** $window_i \leftarrow Q$.
8: **let** $Q \leftarrow \varnothing$.
9: Find `MSPs` in $window_i$, and add them to `MSP_Table`.
10: Compute `CFSPs` from `MSPs`, and add them to `CFSP_Table`.
11: **for** $j = 1$ to $\min(i-1, \rho)$ **do**
12: **let** $\texttt{CFSP}_i$ be any `CFSP` in `CFSP_Table` belonging to $window_i$.
13: **let** $\texttt{CFSP}_j$ be any `CFSP` in `CFSP_Table` belonging to $window_{i-j}$.
14: **let** a and b be the first and the last alert type of $\texttt{CFSP}_i$ and $\texttt{CFSP}_j$, respectively.
15: **if** $\Psi(a, b) \geq \tau$ **then**
16: Add $\texttt{CFSP}_i \mid\mid \texttt{CFSP}_j$ to `CFSP_Table`.
17: **end if**
18: **end for**
19: **let** $i \leftarrow i + 1$
20: **end if**

can be correlated to recent `CFSPs`. If the last alert of each recent `CFSP` is correlated to the first alert of current `CFSP`, then current `CFSP` is concatenated to recent `CFSP`. A recent `CFSP` is a `CFSP` that the difference between its window number and current window number is less than the *Retrospect Factor* (ρ). For each CFSP the window number of the last alert is stored as its window number. This step helps us to aggregate the previous alert to the current alert and construct multistep attack scenarios.

Step 19: The current window marker (i) is incremented by 1.

4.4 Exemplifying the Algorithm

We illustrate the operation of the algorithm by a simple example:

Let $L = 10$, `min_sup` $= 3$, $\tau = 0.5$, and suppose `CCM` is as shown in Table 1. Also, suppose that the importance factor of all alerts is equal. Consider the following stream, where each letter represents an alert type:

$$AKAKACDAKK \mid BCBBCCCDCF \mid FDDAFDAFAD$$

Once the first window is received, its `MSPs` are calculated, as shown in Table 2.

Then we should extract `CFSPs` from `MSPs`. In the first window, AK is a frequent pattern, but for extracting attack scenarios, being frequent is not sufficient, the alerts should also be correlated. So if alert A is followed by alert type K in a `CFSP`, alert type K should be correlated to alert A according to the `CCM`. All `CFSPs` are extracted from `MSPs` in this stage. `CFSPs` of this

Table 1. The causal correlation matrix (CCM).

Alert	A	B	C	D	K	F
A	0.2	0.7	0.1	0.3	0.6	0.2
B	0.2	0.3	0.1	0.8	0.1	0.7
C	0.4	0.1	0.2	0.7	0.3	0.4
D	0.3	0.4	0.1	0.3	0.8	0.2
K	0.3	0.2	0.9	0.1	0.4	0.3
F	0.1	0.9	0.2	0.5	0.1	0.3

Table 2. MSPs (a).

Window number	Maximal Sequential Pattern
1	AK

phase are shown in Table 3.

Table 3. CFSPs (a).

Correlated Frequent Sequential Pattern	Window number
A	1
K	1
AK	1

After receiving the next window, MSPs of this window are calculated first (see Table 4), and then CFSPs are generated. In this window BC is a frequent pattern but according to the correlation matrix, C is not correlated to B. Thus BC is not added to CFSP_Table. Now all longer CFSPs should be generated from CFSP_Table. For example, AK is a CFSP that occurred in the first window, and C is a CFSP, which occurred in the second window. So AK is raised before C. Also, according to the correlation matrix, C is correlated to K so AKC is a CFSP. All of CFSPs are generated and added to CFSP_Table in this manner, as shown in Table 5.

Table 4. MSPs (b).

Window number	Maximal Sequential Pattern
1	AK
2	BC

The third window is processed as described above. MSPs and CFSPs after receiving the third window are demonstrated in Tables 6 and 7.

5 Updating CCM

In this section we propose an algorithm to obtain causal correlation between each pair of alerts and propose an algorithm for updating alert correlation

Table 5. CFSPs (b).

Correlated Frequent Sequential Pattern	Window number
A	1
K	1
AK	1
B	2
C	2
AB	2
KC	2
AKC	2

Table 6. MSPs (c).

Window number	Maximal Sequential Pattern
1	AK
2	BC
3	FD, A

matrix. This algorithm belongs to ASEA component in the system.

One of the helpful data mining methods is association rules. They demonstrate the interesting relations between data in large databases. Association rules can help us to determine correlation between alerts, the confidence of association rules can be considered as causal correlation.

For example the following rule denotes that the Sadmind_Amslverify_Overflow alert is raised after the Sadmind_Ping alert after the time lag elapses:

$$\texttt{Sadmind\_Ping} \xrightarrow[\text{time lag}]{} \texttt{Sadmind\_Amslverify\_Overflow}$$

The confidence of an association rule $A \rightarrow B$ is defined as $\frac{\sup(AB)}{\sup(A)}$. The average delay between the occurrence of each rule antecedent and consequent is considered using a time lag for each rule to denote, as in Harms and Deogun's algorithm in [24].

Since IDSs generate many alerts, we need an efficient processing to respond to all alerts. The association rule algorithm has to satisfy three requirements:

(1) Data stream support. The majority of association rule algorithms reflect relations among attributes. This feature is usually beneficial in case of processing database transactions. Besides, we are coping with alert sequences and need to look at the whole sequence to extract the relation between its elements.
(2) Low overhead. As the application is performing in a real time manner it should be light both in

Table 7. CFSPs (c).

Correlated Frequent Sequential Pattern	Window number
A	1
K	1
AK	1
B	2
C	2
AB	2
KC	2
AKC	2
F	3
D	3
FD	3
BF	3
ABF	3
BD	3
BFD	3
$ABFD$	3
CD	3
KCD	3
$AKCD$	3

terms of computational complexity as well as memory consumption. In this application only helpful association rule should be mined.

(3) Temporal relationships support. Typical association rules for transactional dataset do not care about temporal relations between antecedent and consequent of the rules.

In our case, we need only two items association rules. A sufficient way to do so is to find one and two-items sequential patterns. Therefore, once we receive each window, first, we find all one and two frequent sequential patterns of the window. Next, we generate possible association rules. An association rule $a_i \rightarrow a_j$ is generated in two cases:

- If alert $a_i a_j$ is a frequent sequential pattern in the current window.
- If alert a_i is frequent in this window and alert a_j is frequent in the recent windows.

The time lag of each rule is stored in an $n \times n$ matrix called matrix T. T_{ij} is the average time lag of $a_i a_j$. At the beginning of the algorithm all elements of T are set to "–".

In the correlation measure of two alerts, the similarity between alerts and weighted confidence of a rule is calculated as fallow:

$$\text{Weightedconf}(A \Rightarrow B) = \frac{\sup(AB)}{\sup(A)} \times f(\text{timelag}) \times sim(A, B)$$

where f is a function of time lag and sim is a function of similarity between antecedent and consequent alerts. If we increase the time lag of a rule, the weighted confidence of the rule should decrease. As the result, function f is defined as fallows.

$$f(\text{timelag}) = \alpha^{-\text{timelag}} \qquad \alpha \in (1, 2]$$

$$sim(a_1, a_2) = \tfrac{1}{2} \cdot \text{destip\_sim}(ip_1, ip_2) + \tfrac{1}{2} \cdot \text{destport\_match}(port_1, port_2)$$

$$\text{destport\_match}(port_1, port_2) = \begin{cases} 1 & \text{if } port_1 \text{ and } port_2 \text{ match} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{destip\_sim}(ip_1, ip_2) = n/32$$

where n is the maximum number of number high order bits that these two IP addresses match.

Attackers may use IP spoofing techniques to hide their IP addresses. Hence, we only use destination IP address and port number to calculate the similarity in a pair of alerts.

The pseudocode for updating CCM is listed in Algorithm 2.

5.1 Exemplifying the Algorithm

In this section we consider an example usage of the discussed algorithm to clarify the subject. Consider the stream of example in subsection 4.4: once we received the first window, we compute all frequent sequential patterns of the window, as in Table 8.

Table 8. FSP_Table (a).

FSP	Window number	Frequency
A	1	4
K	1	4
AK	1	4

We store the window number and frequency (i.e. support count) for each frequent sequential pattern. Then, we generate association rules. In the first window AK is a frequent sequential pattern, so $A \rightarrow K$ is an association rule. The next step is to calculate time lag. That is, the difference between the window number and its antecedent and consequent. In the other words, the time lag indicates the delay between the occurrence of its antecedent and consequent.

Algorithm 2 Update CCM.

Input:

- `FSP_Table` extracted from `MSP_Table`.
- Retrospect Factor (ρ).
- Causal correlation Matrix (`CCM`).

Output:

- Updated `CCM`.

Algorithm:

1: **let** $i \leftarrow$ `current window number`.
2: **let** $\texttt{T\_FSP}_i$ be any `FSP` of size 2 in `FSP_Table` belonging to $window_i$.
3: **let** a and b be the first and second alert types of $\texttt{T\_FSP}_i$.
4: Generate the rule $a \rightarrow b$.
5: **let** (time lag$(a \rightarrow b)) \leftarrow 0$.
6: Compute weighted confidence $(a \rightarrow b)$.
7: Update $\Psi(a, b)$.
8: **for** $j = 1$ to $\min(i-1, \rho)$ **do**
9: **let** $\texttt{O\_FSP}_i$ be any `FSP` of size 1 in `FSP_Table` belonging to $window_i$.
10: **let** $\texttt{O\_FSP}_j$ be any `FSP` in `FSP_Table` belonging to $window_{i-j}$.
11: **let** a and b be the alert type of $\texttt{O\_FSP}_i$ and $\texttt{O\_FSP}_j$, respectively.
12: Generate the rule $a \rightarrow b$.
13: **let** (time lag$(a \rightarrow b)) \leftarrow j$.
14: Compute weighted confidence $(a \rightarrow b)$.
15: Update $\Psi(a, b)$.
16: **end for**

Table 9. Association rules table (a).

Association rule	time lag	Weighted Confidence
$A \rightarrow K$	0	$1 \times f(0) \times sim(A, K)$

Association rule set of this step is shown in Table 9.

In the next stage, using the next window, we can calculate the frequent sequential patterns. These patterns are shown in Table 10. Then association rules are generated from frequent sequential pattern of this window and previous window. BC is frequent in this window so $B \rightarrow C$ is an association rule with time lag 0. Moreover, B is frequent in the second window and A is frequent in the first window so $A \rightarrow B$ is an association rule with time lag 1. The rest of the rules can be generated in a similar manner. The output association rule set of step is illustrated in Table 11.

6 Hidden Markov Model

In this section we briefly review the Hidden Markov Model, one of the famous members of the Markov family of models. This model has been used widely in a verity of applications.

Table 10. `FSP_Table` (b).

FSP	Window number	Frequency
A	1	4
K	1	4
AK	1	4
B	2	5
C	2	3

Table 11. Association rules table (b).

Association rule	time lag	Weighted confidence
$A \rightarrow K$	0	$1 \times f(0) \times sim(A, K)$
$A \rightarrow B$	1	$1 \times f(1) \times sim(A, B)$
$K \rightarrow B$	1	$1 \times f(1) \times sim(K, B)$
$B \rightarrow C$	0	$0.6 \times f(0) \times sim(B, C)$
$A \rightarrow C$	1	$0.75 \times f(1) \times sim(A, C)$
$K \rightarrow C$	1	$0.75 \times f(1) \times sim(C, K)$

Assume we have a system that can be presented at any time as being in one state from a set of N distinct states $S_1, S_2, S_3, \ldots, S_N$. In a discrete space time, the system will change its state back and forth based on the probabilistic nature of transitions among the states. State changes are associated with time instants of $t = 1, 2, 3, \ldots$. Each state at time t will be indicated as q_t. Here we are discussing discrete first-order Markov Chains; so when we want to describe the probabilistic behavior of the whole system, we need to only denote specification of the current state and its preceding state. Accordingly, we have:

$$\begin{aligned} &\Pr\left[q_t = S_j \mid q_{t-1} = S_i, q_{t-2} = S_k, \ldots\right] \\ &= \Pr\left[q_t = S_j \mid q_{t-1} = S_i\right] \end{aligned} \quad (1)$$

Now if we consider processes at the right hand side of (1) as independent processes, we can compute state transition probabilities of the form a_{ij} as:

$$a_{ij} = \Pr\left[q_t = S_j \mid q_{t-1} = S_i\right] \qquad 1 \leq i, j \leq N, \quad (2)$$

where state transitions have these properties:

$$\begin{aligned} a_{ij} &\geq 0 \\ \sum_{j=1}^{N} a_{ij} &= 1 \end{aligned} \quad (3)$$

These properties hold because they obey standard stochastic constraints (Rabiner [25]). At each instant of time, if each state corresponds to a physical (observable) event, the model can be named *Observable*

Markov Model because these states can be imagined as outputs of the process (Rabiner [25]). It is possible to have states that are not pertaining to an observable event. In such cases, we try to relate states to observations using the probabilistic method. Hidden Markov Model is an extended version of Markov Model that replaces physical Markov states with hidden states that can produce some observable (physical) events. We now formally define the following notations for an HMM:

(1) N: The number of states in the model.
(2) M: The number of distinct observation symbols per state. We denote the individual symbols as $V = v_1, v_2, v_3, \ldots, v_M$.
(3) $A = a_{ij}$: The state transition probability distribution where

$$a_{ij} = \Pr\left[q_{t+1} = S_j \mid q_t = S_i\right] \qquad 1 \leq i, j \leq N.$$

(4) $B = b_j(k)$: The observation symbol probability distribution in state j where

$$b_j(k) = \Pr\left[O_t = v_k \mid q_t = S_j\right] \qquad 1 \leq j \leq N, 1 \leq k \leq M$$

(5) $\pi = \pi_i$: The initial state distribution where

$$\pi_i = \Pr[q_1 = S_i] \qquad 1 \leq i \leq N.$$

Using the values of the above five symbols, we can construct an HMM as the generator of the observation sequence $O = O_1, O_2, \ldots, O_T$ where and T is the number of observations in the sequence.

For convenience, we use the compact notation $\lambda = (A, B, \pi)$ to indicate the complete parameter set of the model (Rabiner [25]). The parameter λ can illustrate if the model represents an observation sequence as well as the optimal state sequence $S = S_1, S_2, \ldots, S_T$ where $S_t \in S$.

7 Attack Plan Recognition

In this section, we introduce our attack prediction method based on the probabilistic reasoning technique. In particular, we use the so called *Hidden Markov Model*, to predict future intruder activities based on current observable facts and information gathered from sensors. Such heterogeneous detection sensors are installed throughout the enterprise network.

7.1 Basic Principles

In practice, when an intruder breaks into a system, either she continues attacking on the same target to escalate her privilege on the victim or uses it as a zombie to launch attacks against other systems. We can use an attack consequence to investigate whether it is the prerequisite of another attack. This kind of reasoning requires a database of preconditions (prerequisites) and postconditions (consequences) of each attack. Due to the infeasibility of gathering all such conditions for every possible attack, we apply probabilistic reasoning into attacker plan recognition.

Plan recognition has been an active research area in artificial intelligence for years, where observations are received from an agent and then the object's goals will be inferred from those activities (observations). There are two types of plan recognition: *keyhole recognition* and *intended recognition* (Cohen *et al.* [26]). The main difference between these two is the role of the target agent. In keyhole recognition, the agent is not informed whether it is under observation and so it is unable to affect the recognition process. Contrary to keyhole recognition, in intended recognition, the agent intentionally tries to help the recognition process. Unfortunately, in our case, none of the above plan recognition schemes are applicable. The reason is that the intruder is neither unaware of the existence of monitoring systems nor she is trying to help intrusion detection systems to predict her future activities. Instead, she tries to deceive the monitoring agents, and hide her activities from being sensed. Therefore, this sort of plan recognition can be categorized as *adversary recognition* where there are more uncertainty and complexity (Qin and Lee [17]).

Assumptions of traditional plan recognition methods do not hold anymore in intrusion plan recognition. For instance, those methods assume that any reachable target by the agents has a plan record in the database. This is impractical in the area of network security, as human intruders are creative and even unpredictable in many ways. In addition, in plan recognition, a plan is carried out when a complete ordered sequence of tasks is performed; whereas, in network security there may exist different incomplete sequences of tasks in which some observations are made invisible by the attacker.

7.2 Sequence Modeling

The next step is to build a model that captures the ordered properties of sequences of super alerts. We use finite-state models (Bell [27]), also known as Hidden Markov Models, which form one of two major sequence characterization techniques [1] . HMM is composed of some observables or events plus some hidden states. The Markov Model used for HMM is a first order model. That is, the state transition probabilities only depend on the previous state. HMM can specify the likelihood of an attack based on a given observation

[1] The other group of models is called finite-context models. For more information please refer to Bell [27].

sequence. In order to specify an HMM λ, we need to determine the following parameters:

(1) N: The number of states in the model;
(2) M: The number of distinct observation symbols per state;
(3) A: The state transition probability distribution;
(4) B: The observation symbol probability distribution;
(5) π: The initial state distribution.

Alerts can be categorized in a way that each group represents a single-stage of a multi-stage attack scenario. The name of each class shows the general category of that class. The number of states in our model is the number of attack classes. This classification is based on Snort IDS (Roesch [28] and RealSecure IDS [27]) signature description manuals. These classes are derived in Al-Mamory and Zhang [9] and listed in Table 12.

Table 12. Alert Classes Indicating Stages of Attack Scenarios

No.	Attack Class
1	Enumeration
2	Host Probe
3	Service Probe
4	Service Compromise
5	User Access
6	Root or Administrator Access
7	System Compromise
8	Sensitive Data Gathering
9	Active Communication Remote
10	Trojan Activity
11	Availability Violation
12	Integrity Violation
13	Confidentiality Violation

Alert classes as our hidden nodes determine the current state of the target system using super alerts as its input. In our model, observations are distinct alerts that are fired from IDS sensors installed on the target network. Given a sequence of super alerts $O = O_1, O_2, O_3, \ldots, O_T$, it should be decoded to help determining the most probable matching state sequence $O = S_1, S_2, S_3, \ldots, S_T$. Thus, S would be the most probable sequence of states that has generated O. The last state in the sequence S shows the current state of the system which we use as the base in the plan recognition component. By applying the Viterbi algorithm, we define the highest probability along a single path at time t which corresponds to the first $t \leq T$ observations ending in state S_i as:

Algorithm 3 Attack Class Prediction

Predict (int `current_node`)

1: **initialize** `prediction` $\leftarrow 0$, $i \leftarrow 1$.
2: **repeat**
3: **if** $a_{(\texttt{current\_node},i)} >$ `prediction` **then**
4: **let** `prediction` $\leftarrow 0$.
5: **end if**
6: **until** $i = \#$ of states
7: **if** $a_{(\texttt{current\_node},i)} >$ `threshold` **then**
8: **return** `prediction`.
9: **else**
10: **return** false.
11: **end if**

$$\delta_t(i) = \underset{S_1,\ldots,S_{t-1}}{\arg\max} \Pr\left[S_1, \ldots, S_t, O_1, \ldots, O_t \mid \lambda\right]$$

Now, using induction we can determine the state sequence. Note that we need to keep track of the values of $\delta_t(i)$: $\delta_{t+1}(j) = \underset{1 \leq i \leq N}{\arg\max} \{\delta_t(i) a_{ij}\} \cdot b_i(O_{t+1})$.

7.3 Sequence Prediction (Plan Recognition)

Let $B = \{\beta_1, \beta_2, \beta_3, \ldots, \beta_n\}$ where β_i is an attack class, be a sequence of attack actions that is decoded by the correlation unit. In sequence prediction, our goal is to recognize the next attacker activity β_{n+1} assuming we have the sequence of previous actions B and a learned HMM λ. Let $\Pr[\beta']$ denote the probability of the next symbol in B being β' according to the HMM. The intuition is to find the most likely path from the current state node. Using (4) we can find the most probable path from the current state which is identical to the next attacker action.

$$\Pr[\beta'] = \underset{1 \leq i \leq N}{\arg\max} \{\Pr\left[q_t = \beta' \mid q_{t-1} = \beta_n\right]\} \quad (4)$$

In order to reduce false positive rate, we may set a probability threshold such that lower probabilities are discarded in the plan recognition process. Algorithm 3 illustrates the pseudo code of our attack class prediction. Note that in the algorithm $a_{(\texttt{current\_node},i)}$ is an element of the state transition probability matrix $A = a_{ij}$ where $a_{ij} = \Pr\left[q_{t+1} = S_j \mid q_t = S_i\right]$, $1 \leq i, j \leq N$.

In the following section we discuss evaluation results of the proposed system. First, we explain evaluation results of the ASEA components and then we describe experimental results of the attack plan recognition component which uses HMM as the base.

8 Attack Scenario Extraction Experiments

In this section we discuss our experimental results. We used the famous DARPA 2000 dataset to evaluate our algorithm [29]. The dataset consists of two sub datasets (i.e. LLDDOS 1.0 and LLDDoS 2.0). We used both sets to test our algorithm.

8.1 LLDDOS1.0

LLDDOS1.0 is a packet dump corresponding to a multistep attack scenario with five phases:

(1) Using ping sweeping to detect live IP addresses in the target network.
(2) Detecting `sadmind` demon running on live IPs obtained in the previous step.
(3) Using the `sadmind` vulnerability to get access to the vulnerable systems.
(4) DDoS software via `telnet`, `rcp` or `rsh`.
(5) Launching a DDoS attack.

DARPA dataset is a large set of raw network packets dumps. But, our system receives IDS alerts as input. So, we should feed these packets into IDS, and dump the output alerts. Instead, we used RealSecure IDS alert log [30]. This alert log was produced by playing back the "Inside-tcpdump" of LLDDOS1.0.

The Input parameter of our algorithm is set as follows:

Window Size: $L = 20$.
There is a trade off in determining the L. If we choose a large L, we then need a considerable time and memory to computer all frequent patterns. Also we should wait until L alerts are generated. Therefore, the real time characteristics of the algorithm do not hold anymore. In contrast, a small L causes the algorithm to be inaccurate.
We use some heuristics to determine the window size. L is selected such that there is no more than some number (e.g. 6) of alert types in each window.
Causal Correlation Matrix: The `CCM` is filled with values determined by prior knowledge, as shown in Table 13.
Correlation Threshold: $\tau = 0.5$.
Retrospect Factor: $\rho = 3$.
Importance Factor: $\mathcal{I}(a) = 3$ for each critical alert types, $\mathcal{I}(a) = 2$ for typical alert types and $\mathcal{I}(a) = 1$ for non-trivial alerts.
Minimum Support Count: `min_sup` $= 4$.
$\texttt{min\_sup}_a$ is calculated as follows:

$$\texttt{min\_sup}_a = \begin{cases} 0.25 \times \texttt{min\_sup} & \text{if } \mathcal{I}(a) = 3 \\ \texttt{min\_sup} & \text{if } \mathcal{I}(a) = 2 \\ 3 \times \texttt{min\_sup} & \text{if } \mathcal{I}(a) = 1 \end{cases}$$

Some alerts, such as `Email_Ehlo`, are unimportant; so the importance factor ($\mathcal{I}$) of these alert are low and they are almost ignored because of their high `min_sup` threshold. Ignoring `Email_Ehlo` alerts, the number of alerts is reduces by 56 percent. Thus, considering different importance factors leads to a more efficient algorithm. Also, the `Stream_DoS` alert is an important alert that may not be frequent so we set the importance factor ($\mathcal{I}$) of this alert to 3.

After setting parameters we ran the algorithm on dataset. The following sequences are detected as a multistep attack scenario:

`Sadmind_Ping`, `Sadmind_Amslverify_Overflow`, `Admind`, `Rsh`, `MStream_Zombie`, `Stream_DoS`.

`Sadmind_Ping`, `Admind`, `Sadmind_Amslverify_Overflow`, `Rsh`, `MStream_Zombie`, `Stream_DoS`.

The RealSecure generates the fallowing alerts for each phase of this multistage attack scenario [8, 17].

(1) **IP sweep:** Since RealSecure ignores ICMP ping, no alert is generated for this phase.
(2) **Probing `sadmind` service:** For this phase RealSecure raises a number of `Sadmind_Ping` alerts.
(3) **Exploiting `sadmind` vulnerability:** In this phase two alert types are raised:
   (a) `Sadmind_Amslverify_Overflow`: This alert is raised because of exploiting sadmind vulnerability and trying to break into the `sadmind` service.
   (b) `Admind`: It is raised for illegal access to Solaris administration.
   Since alert types `Sadmind_Amslverify_Overflow` and `Admind` are usually raised by a single attack step, each of which can be considered as a prerequisite to the other. This fact is reflected in the Causal Correlation Matrix: This is why both of the extracted attack scenarios are valid.
(4) **Installing DDos Trojan:** `Rsh` and `MStream_Zombie` are alerts raised in this step.
(5) **Launching DDos attack:** Finally a `Stream_DoS` alert is raised. Some approaches do not detect this alert since it occurs only once. We are able to detect it because we value this alert type by increasing its *importance factor*.

The details of detecting each step of the attack is as follows: The `Sadmind_Ping` was detected in window 10, `Sadmind_Amslverify_Overflow` is frequent in window 11, the pattern `Rsh`→ `MStream_Zombie` is

Table 13. CCM.

Alert	Sadmind_Ping	TelnetTerminaltype	Email_Almail_Overflow	Email_Ehlo	FTP_User	FTP_Pass	FTP_Syst	HTTP_Java	HTTP_Shells	Admind	Sadmind_Amslverify_Overflow	Rsh	MStream_Zombie	HTTP_Cisco	SSH_Detected	Email_Debug	TelnetXdisplay	TelnetEnvAll	Stream_DoS
Sadmind_Ping	0	0	0	0	0	0	0	0	0	0.5	0.5	0.25	0.25	0	0	0.25	0.25	0	0
TelnetTerminaltype	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0.25	0.25	0.25	0.25	0	0.25	0.25	0	0.25	0
Email_Almail_Overflow	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0	0	0.25	0	0	0	0.25	0	0	0
Email_Ehlo	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0.25	0.25	0.25	0	0.25	0.5	0.5	0	0	0
FTP_User	0	0	0.5	0.25	0.5	0.5	0.5	0	0	0	0	0	0	0.25	0	0.25	0	0	0
FTP_Pass	0	0.5	0	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0.25	0	0	0
FTP_Syst	0	0.5	0	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0.25	0.25	0	0	0	0
HTTP_Java	0	0.5	0	0.5	0.25	0.25	0.25	0.5	0.25	0	0	0	0	0	0	0	0	0	0
HTTP_Shells	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	0	0	0
Admind	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0	0	0	0.25	0.25	0
Sadmind_Amslverify_Overflow	0	0	0	0	0	0	0	0	0	0.5	0.5	0.5	0.5	0	0	0	0.25	0.25	0
Rsh	0	0	0	0	0	0	0	0	0	0	0	0.5	0.25	0	0	0	0.25	0.25	0
MStream_Zombie	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0.5
HTTP_Cisco	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0
SSH_Detected	0	0	0	0.25	0	0	0	0	0	0	0	0	0	0	0.25	0.25	0	0	0
Email_Debug	0	0	0	0.25	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
TelnetXdisplay	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0.25	0
TelnetEnvAll	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0
Stream_DoS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

detected in windows 13 and 14, and `Stream_DoS` is raised in window 15.

Our scenario is comparable with the attack graph extracted by algorithm proposed in Zhu and Ghorbani [8], and Ning *et al.* [31]. Figure 3 shows the attack graph of Zhu and Ghorbani [8], and Figure 4 demonstrates the hyper-alert correlation graph of Ning *et al.* [31].

8.2 LLDDOS2.0

LLDDOS2.0 is a packet dump corresponding to a multistep attack scenario including five phases:

(1) Probe of public DNS server, via the HINFO query;
(2) Exploiting sadmind vulnerability and breaking into a vulnerable host;
(3) Installing DDoS software and attack scripts via FTP;
(4) Trying to compromise two other hosts and achieving one successful exploitation;
(5) Telnet to the compromised host and initiate mstream DDoS attack.

To test our alert correlation algorithm, again we used IDS alert logs from [30]. This alert log includes 494 alerts from 18 distinct types. The input parameter setup was the same as Section 5.1 and the Causal Correlation Matrix is depicted in Table 15.

After initial parameter setup we ran the algorithm on the dataset. The algorithm extracted following attack sequences illustrating two multistep attack scenarios:

`Sadmind_Amslverify_Overflow`, `Admind`, `FTP_Put`, `MStream_Zombie`, `Stream_DoS`.

`Admind`, `Sadmind_Amslverify_Overflow`, `FTP_Put`, `MStream_Zombie`, `Stream_DoS`.

Each phase of the above attacks causes the RealSecure IDS to generate the following alerts.

Table 14. Updated CCM.

Alert	Sadmind_Ping	TelnetTerminaltype	Email_Almail_Overflow	Email_Ehlo	FTP_User	FTP_Pass	FTP_Syst	HTTP_Java	HTTP_Shells	Admind	Sadmind_Amslverify_Overflow	Rsh	MStream_Zombie	HTTP_Cisco	SSH_Detected	Email_Debug	TelnetXdisplay	TelnetEnvAll	Stream_DoS
Sadmind_Ping	0	0	0	0	0	0.15	0	0	0	0.5	0.63	0.25	0.25	0	0	0.25	0.25	0	0
TelnetTerminaltype	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0	0.25	0.25	0.25	0.25	0	0.4	0.25	0	0.25	0
Email_Almail_Overflow	0	0.5	0.5	0.5	0.5	0.5	0.5	0.19	0	0	0	0.25	0	0	0	0.25	0	0	0
Email_Ehlo	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0.25	0.25	0.25	0	0.25	0.5	0.5	0	0	0
FTP_User	0	0.17	0.5	0.25	0.5	0.81	0.5	0	0.37	0	0.19	0	0	0.25	0	0.25	0	0	0
FTP_Pass	0	0.61	0	0.5	0.5	0.5	0.65	0	0.31	0	0.15	0	0	0	0	0.25	0	0	0
FTP_Syst	0	0.63	0	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0.25	0.25	0	0	0	0
HTTP_Java	0	0.5	0	0.5	0.25	0.25	0.25	0.5	0.25	0	0.14	0	0	0	0	0	0	0	0
HTTP_Shells	0	0	0	0	0	0	0	0.32	0	0.18	0	0	0	0	0	0	0	0	0
Admind	0	0	0	0	0	0	0	0	0	0.5	0.82	0.65	0.5	0	0	0	0.25	0.25	0
Sadmind_Amslverify_Overflow	0	0	0.19	0	0	0.19	0	0	0	0.85	0.5	0.64	0.5	0	0	0	0.25	0.25	0
Rsh	0	0	0	0	0	0	0	0	0	0	0	0.5	0.25	0	0	0	0.25	0.25	0
MStream_Zombie	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0.5
HTTP_Cisco	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0
SSH_Detected	0	0	0	0.25	0	0	0	0	0	0	0	0	0	0	0.25	0.25	0	0	0
Email_Debug	0	0	0	0.25	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0	0
TelnetXdisplay	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0.25	0
TelnetEnvAll	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0	0	0
Stream_DoS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(1) **IP sweep:** Since RealSecure ignores ICMP ping packets, no alert is generated for this phase.
(2) **Exploiting sadmind vulnerability:** In this phase, two alert types are raised:
   (a) `Sadmind_Amslverify_Overflow`: This alert is fired after successful exploitation of sadmind vulnerability.
   (b) `Admind`: It is generated because of illegal access to Solaris administration.

   Since `Sadmind_Amslverify_Overflow` and `Admind` alert types are usually raised throughout a single attack step, each of them can be considered as a prerequisite to the other. This fact is reflected in the Causal Correlation Matrix and that is why both extracted attack scenarios are valid.
(3) **Installing DDoS Software:** The attacker tries to install the DDoS software on a host via FTP. So, the IDS raises `FTP_Put` alert.
(4) **Distributing DDoS Software:** `Rsh` and `MStream_Zombie` alerts are raised while the intruder tries to install the DDoS software on two other hosts.
(5) **Launching DDoS attack:** Finally a `Stream_DoS` alert is raised. Some approaches do not detect this alert since it occurs only once. We are able to detect it because we value this alert type by increasing its importance factor.

The extracted scenario is comparable with the attack graph constructed by the algorithm proposed in Zhu and Ghorbani [8]. Figure 5 shows their attack graph, where we can see the fifth stage of the attack scenario is not detected by their algorithm. The rest of the phases are detected similar to ASEA. Table 16 illustrates the updated CCM. Note that gray cells are changed by the algorithm.

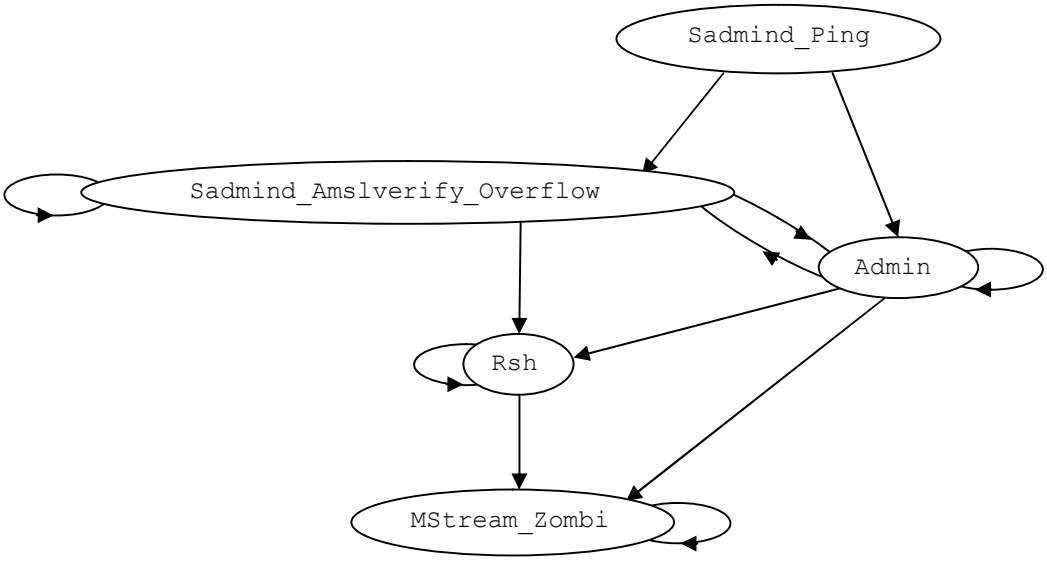


Figure 3. Attack graph for LLDOS1.0 extracted by Zhu and Ghorbani [8].

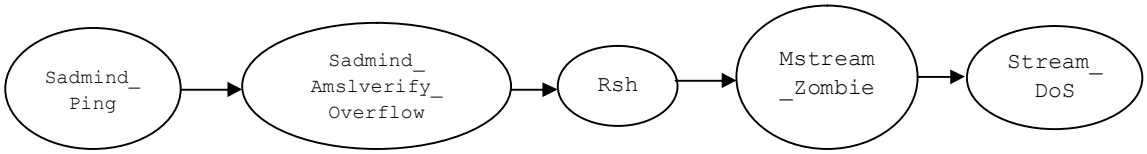


Figure 4. A hyper-alert correlation graph discovered in LLDOS1.0 by Ning *et al.* [31].

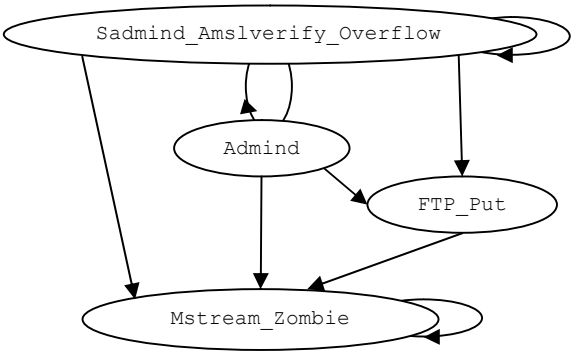


Figure 5. Attack graph for LLDOS2.0 extracted by Zhu and Ghorbani [8].

9 Attack Plan Prediction Results

This section describes our experiments to evaluate the effectiveness of the plan recognition component in the real world.

Let $B_k = \{\beta_{(k,1)}, \beta_{(k,2)}, \beta_{(k,3)}, \ldots, \beta_{(k,n)}\}$ be the k^{th} new observation sequence of attacks captured by IDSs where $\beta_{(k,i)}$ is a super alert of the k^{th} sequence. A prediction set S includes likely future classes of attacks $S_k = \{\alpha_{(k,1)}, \alpha_{(k,2)}, \alpha_{(k,3)}, \ldots, \alpha_{(k,m)}\}$, where $\alpha_{(k,j)}$ is an attack class, calculated for each B_k. Now we can define our prediction performance rate R as:

$$R = \frac{\text{number of correctly predicted classes}}{\text{total number of predictions}}$$

To make our results comparable with the research community, we use the same naming scheme of one of the recent similar works (Fava *et al.* [20]). The performance rates are given in terms of *top-n* where $n = |S_k|$ is the prediction set size. Here we use $n = \{1, 2\}$ meaning that we calculate top-1 and top-2 sets for each $\beta_{(k,i)}$. Using top-1, a correct prediction means that the model has predicted *one* future symbol (i.e. the most probable symbol) and the attacker performed the same action. Similarly, in case of correct prediction of the type top-2, we mean that the attacker performed the action that is included in *two* most likely symbols determined by the model. In Fava *et al.* [20], there is also top-3 set defined and used while we do not include it here.

Note that the false positive rate is not applicable here, because false positive rate is only meaningful when we perform detection; instead we are performing prediction in which we predict a fact that can be either

Table 15. CCM.

Alert	RIPAdd	TelnetTerminaltype	FTP_Put	Email_Almail_Overflow	Email_Ehlo	FTP_User	FTP_Pass	FTP_Syst	HTTP_Java	HTTP_Shells	Admind	Sadmind_Amslverify_Overflow	Email_Turn	MStream_Zombie	HTTP_Cisco	Port_Scan	EventCollector_Info	Stream_DoS
RIPAdd	0	0	0	0	0	0	0	0	0.25	0	0	0	0	0	0	0	0	0
TelnetTerminaltype	0	0	0.25	0.5	0.5	0.5	0.5	0.5	0	0	0.25	0.25	0.25	0.25	0	0	0	0
FTP_Put	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0
Email_Almail_Overflow	0	0.5	0	0.5	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0	0
Email_Ehlo	0	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0.25	0.25	0	0	0.25	0	0	0
FTP_User	0	0	0	0.5	0.25	0.5	0.5	0.5	0	0	0	0	0	0	0.25	0	0	0
FTP_Pass	0	0.5	0	0	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0	0
FTP_Syst	0	0.5	0	0	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0.25	0	0	0
HTTP_Java	0	0.5	0	0	0.5	0.25	0.25	0.25	0.5	0.25	0	0	0	0	0	0	0	0
HTTP_Shells	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Admind	0	0	0.5	0	0	0	0	0	0	0	0.5	0.5	0	0.5	0	0	0	0
Sadmind_Amslverify_Overflow	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	0.5	0	0	0	0
Email_Turn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MStream_Zombie	0	0	0.5	0	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0.5
HTTP_Cisco	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0
Port_Scan	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EventCollector_Info	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Stream_DoS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

true or false. So, the false positive rate is meaningless in case of predication. Also, the true positive, true negative and false negative metrics are not applicable in our case. These metrics are usable in intrusion detection, while we do intrusion prediction. In general, when we predict something, it can be either true or false. In other words, it is not possible to receive a false positive from any type of predictor.

Similar to ASEA experiments, for these experiments, we also use the sensor alert report using RealSecure network intrusion detection system version 6.0 on DARPA 2000 dataset publicly available by the Secure Open Systems Initiative Laboratory at North Carolina State University as a part of TIAA project [30].

9.1 Preprocessing

In the aggregation process, we first fused redundant alerts having the same attributes. Next, using the results of the first step, we aggregated alerts with the same attributes but varied on timestamp if they are close enough to fall into a predened time window (e.g. 100 seconds). As a result, we had our super alert collection available for further processing. To increase the timing precision, we used the timestamp of the latest alert. This is due to the fact that in practice, attackers usually stop attacking after the last successful attack. Thus, using the last timestamp means choosing the time that is closer to the successful attack in the real world.

After retrieving super alerts from raw alerts, we grouped super alerts by their destination IP address and then sorted each group in time to make a sequence. Table 17 shows statistics of aggregation and sequence generation on both LLDDoS versions. Note that for each dataset, the numbers from the *DMZ* and the *Inside* zone are added together.

We omitted 5 and 10 alerts fired from the "event_collector_1" sensor with the type of "event_collector_1" from LLDDoS 1.0 and LLDDoS 2.0.2, respectively. This is because many fields of these alerts (in log traces) were nullified by the IDS, and they were not

Table 16. Updated CCM.

Alert	RIPAdd	TelnetTerminaltype	FTP_Put	Email_Almail_Overflow	Email_Ehlo	FTP_User	FTP_Pass	FTP_Syst	HTTP_Java	HTTP_Shells	Admind	Sadmind_Amslverify_Overflow	Email_Turn	MStream_Zombie	HTTP_Cisco	Port_Scan	EventCollector_Info	Stream_DoS
RIPAdd	0	0	0	0	0	0	0	0	0.14	0	0	0	0	0	0	0	0	0
TelnetTerminaltype	0	0	0.25	0.5	0.5	0.5	0.5	0.5	0	0	0.25	0.25	0.25	0.25	0	0	0	0
FTP_Put	0	0.25	0	0	0	0	0	0	0	0	0	0	0	0.5	0	0	0	0
Email_Almail_Overflow	0	0.5	0	0.5	0.5	0.5	0.5	0.5	0	0	0	0	0	0	0	0	0	0
Email_Ehlo	0	0	0	0.5	0.5	0.5	0.5	0.5	0.5	0	0.25	0.25	0	0	0.25	0	0	0
FTP_User	0	0	0	0.74	0.25	0.5	0.87	0.5	0.37	0	0	0	0	0	0.25	0	0	0
FTP_Pass	0	0.67	0	0.26	0.5	0.71	0.5	0.5	0.16	0	0	0	0	0	0	0	0	0
FTP_Syst	0	0.66	0	0.26	0.5	0.5	0.72	0.5	0.33	0	0	0	0	0	0.25	0	0	0
HTTP_Java	0	0.5	0	0	0.5	0.25	0.25	0.37	0.5	0.25	0	0	0	0	0	0	0	0
HTTP_Shells	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Admind	0	0	0.5	0	0	0	0	0	0	0	0.5	0.5	0	0.5	0	0	0	0
Sadmind_Amslverify_Overflow	0	0	0.5	0	0	0	0	0	0	0	0.5	0.5	0	0.5	0	0	0	0
Email_Turn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MStream_Zombie	0	0	0.5	0	0	0	0	0.5	0	0	0	0	0	0.5	0	0	0	0.5
HTTP_Cisco	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.25	0	0	0
Port_Scan	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
EventCollector_Info	0	0.15	0	0.26	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Stream_DoS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 17. Aggregation Statistics

Dataset	LLDDoS 1.0	LLDDoS 2.0.2
# Alerts	1813	924
# Super Alerts	1630	830
# Super Alert Sequences	66	56

useful for processing. To add more details, we have not omitted alerts targeted to 255.255.255.255, which will be filtered by today's firewalls, though it introduces some errors in results.

9.2 Training

In our experiments we compared the results of both supervised and unsupervised trainings. In the following we describe our supervised and unsupervised trainings of the HMM.

9.2.1 Supervised Training

In order to train our model, we had to classify and label super alerts with related categories. By labeled super alerts, we refer to an instance annotated by its related class. Labeling process needed a domain expert as well as a considerable amount of time. We implemented a simple semi-automatic method with the help of subject-matter experts to classify examples. Particularly, we should prepare a mapping scheme between our attack types in DARPA 2000 dataset and the 13 attack classes defined in Table 12. DARPA 2000 dataset contains 28 attack types in theRealSecure terminology. For each type, we defined a simple rule to map each alert type to an attack class using RealSecure signature description manual.

One may argue the correctness of our classification. The output of the system is in terms of attack classes, and if we do not classify alerts correctly, the output of the predictor will not be correct. So, wrong attack classes as the output may mislead the system admin-

istrator(s). Indeed, in our experiments, we evaluated the ability of the algorithm to capture only the *sequential patterns* of attack classes. In order to measure the effectiveness of the algorithm we evaluated the results *relatively*, based on our own mapping (classification). Hence, it is not important to exactly classify super alerts and discuss the reason of classifying each type. Even though it is very important in operational environment, here, we only want to compute the proposed algorithm performance. Thus, we can avoid further discussion on the topic.

After labeling the super alerts, we randomly chose 50% of each LLDDoS sets for training, and the rest for testing. We used Jahmm (François [32]), a Java implementation of HMM for experiments. Since we knew the corresponding sequence of states for each learning sequence, we could estimate HMM parameters using (5):

$$a_{ij} = \frac{A_{ij}}{\sum_{l=1}^{N} A_{il}}, \quad b_i(k) = \frac{B_i(k)}{\sum_{l=1}^{N} B_i(l)} \; ; \\ 1 \leq i, j \leq N, 1 \leq k \leq M \quad (5)$$

where A_{ij} is the counter for transitions from state i to state j and $B_i(k)$ is the number of emissions of the k^{th} super alert in the state i.

9.2.2 Unsupervised Training

In supervised learning, we classified attacks semantically and manually. Then, we labeled each super alert with the related class, and after that, we fed sequences of such super alerts into the model for learning. In unsupervised learning, we will run the model to classify unlabeled super alerts on their own, in order to measure their effectiveness in classification as well as prediction. Therefore, after the aggregation of the data, we fed raw (unlabeled) super alert types into the model for training.

One of the inputs of HMM learning algorithm is the number of states. In supervised learning we determined this number using expert knowledge to classify attack types in an appropriate manner. Unlike supervised learning, in unsupervised training there is no predefined number of states for the algorithm. We used the same number of states as in supervised mode for unsupervised mode to ease the comparison of the outputs. We use the same number of distinct observable symbols in both modes. We followed two objectives in unsupervised mode: (1) To evaluate how powerful the HMM is in classifying alerts in comparison with manual semantic oriented classification; and (2) To determine how effective the HMM is in predicting attack classes in comparison with the supervised mode.

9.3 Plan Recognition

As mentioned earlier, the relationship between the next attacker action and her recent behavior can be inferred from the correlation of adjacent actions within the same sequence. Here, we used the first order HMM meaning that the next symbol is predicted only based on the one immediate previous symbol.

In the following, we report our investigations on the effectiveness of our model in plan recognition using a learned HMM, both in supervised and unsupervised modes.

9.3.1 Supervised Mode

We did not have enough sequences in our test data sets. Therefore, in order to increase the accuracy of our testing we performed several tests per sequence. Specifically, in the case of LLDDoS 1.0, the average length of each sequence in our test set was 23.6 super alerts. Therefore, we divided them into sets of four super alerts so that we have about six subsets of testing in each sequence. Subsets having less than four members were discarded. Then, given the first three members of each subset to the model, it first found the most probable matching states and then predicted the last symbol. In the case of LLDDoS 2.0.2, we kept the same approach but we chose three subsets per super alert observation sequence because we had shorter (i.e. an average of 15.2 super alerts) sequences in this dataset.

For each subset, we first computed the most probable state sequence and prediction set using Viterbi algorithm and Algorithm 3, respectively. Results presented for top-2 prediction sets retrieved using a modified version of Algorithm 3 that outputs a prediction set with two members (regarding top-2 results) rather than one. Note that, we set the probability threshold to zero in Algorithm 3. This is because (1) It strongly depends on the environmental variables such as the criticality of the target systems, the required sensitivity to attacks, and the average number of attacks received by the targets; (2) Setting this threshold to zero allows predictions with low probability, resulting in a decrease in the true prediction rates. That is, to show the effectiveness of the system even with low thresholds, we accepted this error and included it in all results presented here: and (3) If we used some thresholds, we would need to change our performance metric to a three-variable metric like *true prediction*, *false prediction*, and *no prediction* rather than the current binary true and false rates. This change would make comparing our results to those of the research community more difficult; since some people may believe that *no prediction* means *false prediction*, and some may reject any similarity among them.

Table 18 shows the prediction accuracy for both datasets in the cases of top-1 and top-2 prediction sets. The model achieved the rate of above 90% in predicting the next attack class. The result shows that, keeping off the low level (i.e. alert type level prediction) granularity in prediction, keeps off the model from over-fitting and performs accurately.

Table 18. Supervised Mode Prediction Rates

Dataset	LLDDoS 1.0	LLDDoS 2.0.2
Top-1 Rate	81.33%	82.6%
Top-2 Rate	98.1%	93.3%

To the best of our knowledge, this is the first unsupervised method that aims to predict the attacker's next action. Thus, it is difficult to compare our results with related works. For instance, we cannot compare our results with Qin and Lee [17] since its prediction rate is highly dependent on the attack library, while in our case, there is no such library; making the two methods incomparable. Another example is Yu and Frincke [33], where the authors used prerequisites and consequences of an attack for predication, and achieved 100% of correct prediction; while in our method we have not used any prior knowledge to predict future attacks. Moreover, there are some works that only used supervised methods to predict the next attacker action.

Re-aggregation. We can prune the HMM graph in different ways to increase the performance of the prediction. As an example, we can set a threshold in the learning phase and neglect edges whose labels are below some threshold probability. This can reduce the number of comparisons in Algorithm 3. Another way is to prune loop edges from the model. For instance, consider the attack sequence targeted to 172.016.113.204 in DMZ section of LLDDoS 1.0 that contains the following sequence of attack classes:

$$\{10, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4\}.$$

The sequence means, an attack from *Class 10* has been launched following by 17 (distinct) attacks that all belong to *Class 4*. If the predictor works correctly, it will alarm the administrator repeatedly stating the same class is going to happen, which may be overwhelming and useless; because the main objective of attack plan recognition is to arm the management with information supporting timely decision making and incident responding. Therefore, restating the same fact by the predictor will not solve any problem. We assume that, the administrator has made appropriate response to the predicted class; hence, redundant alarming does not support decision-making. In conclusion, redundant classes destined to the same IP address can be re-aggregated. That is, all 17 class numbers will be fused together reducing the sequence to $\{10, 4\}$. In this case, we changed the length of the sequence from 18 to 2, which is equivalent to 88% of reduction in length.

Re-aggregation avoids forming loop edges in the HMM graph in the learning phase. Moreover, it provides a finer granularity of prediction (i.e. in testing phase). The finer grained models are subject to underfitting phenomena. But our results indicate almost the same rate as nonre-aggregated data, particularly in top-2 mode of prediction. However, we have some natural growth in top-1 false prediction rate. Table 19 contains average prediction rates using re-aggregation technique on LLDDoS 1.0.

Table 19. Prediction Rates Using Re-aggregation

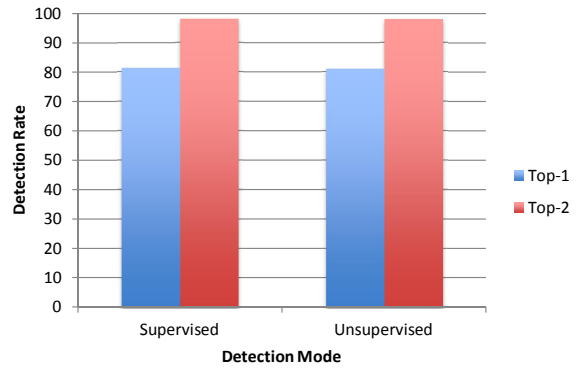
Dataset	LLDDoS 1.0
Top-1 Rate (%)	77
Top-2 Rate (%)	98.3

Figure 6(b) compares results from models learned using re-aggregated and aggregated (i.e. nonre-aggregated) sequences. We can see a decrease in top-1 prediction and a roughly equal rate for top-2 prediction. This indicates the potential ability of the re-aggregation process. As mentioned, the main aim of re-aggregation is performance enhancement, while keeping the same level of accuracy in prediction.

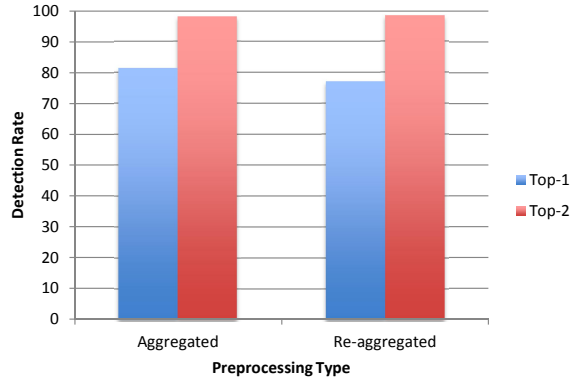
Figure 6(c) shows the number of symbols per sequence of one sample of our testing data. It shows that the re-aggregation method caused considerable alleviation in the lengths of the sequences. The average length of original sequences was 24.78, while the re-aggregation drops two third of the average length to achieve 8.06 symbols per sequence. In other words, we can reduce the processing time using re-aggregation technique. First, because it removes redundant symbols from the input for the purpose of removing loop edges from the model in the learning phase. Second, in testing phase, it shortens the sequences in size, resulting in less processing.

9.3.2 Unsupervised Mode

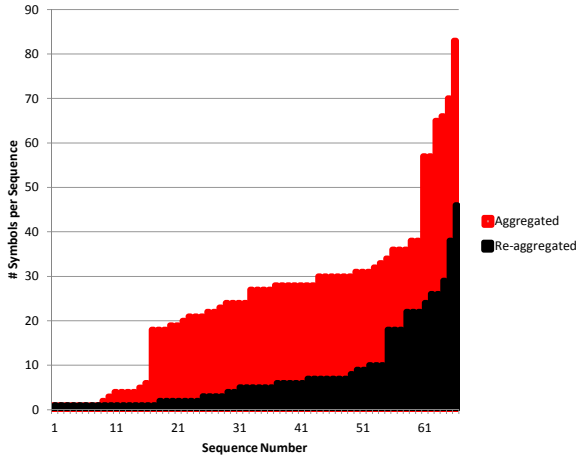
Table 20 shows the prediction accuracy for LLDDoS 1.0 dataset in the cases of top-1 and top-2 prediction sets. The results are impressive; unsupervised learning of the model reached almost the same prediction rate as supervised mode, even though we ran the model to classify alert types itself, without any labeling aid. Figure 6(a) compares the results from models learned in supervised versus unsupervised modes. The classification made by the HMM in unsupervised mode was



(a)



(b)



(c)

Figure 6. (a) Prediction Rates Using Supervised Mode vs. Unsupervised Mode, (b) Prediction Rates Using Aggregation vs. Re-aggregation, (c) Number of Symbols per Sequence.

slightly different from the one made manually. But, the feature that made it capable of accurate prediction is the ability to capturing frequent symbols and putting them in different classes. In the LLDDoS 1.0 dataset, we have a minority of alert types that happened with high frequency in time. The HMM captured these effective elements and separated them in different classes. For example, `FTP_Syst` alert type is one of the frequent alert types which semantically belongs to Service Probe attack class, and `TelnetTerminaltype` is another frequent alert type which is a part of *Active Communication Remote* class. The model correctly separated these two kinds of effective alert types in different classes. Similarly, two alert types of `Email_Almail_Overflow` and `Email_Ehlo`, both originally (semantically) members of Service Compromise class, grouped correctly under the same class during unsupervised learning. Consequently, unsupervised learning is able to capture effective symbols in prediction and understand underlying semantic relationships based on sequential patterns. Although not all decisions made by the model are completely correct, the classification was not erroneous. That is why we had near equal prediction rates in both modes.

Table 20. Unsupervised Prediction Rates

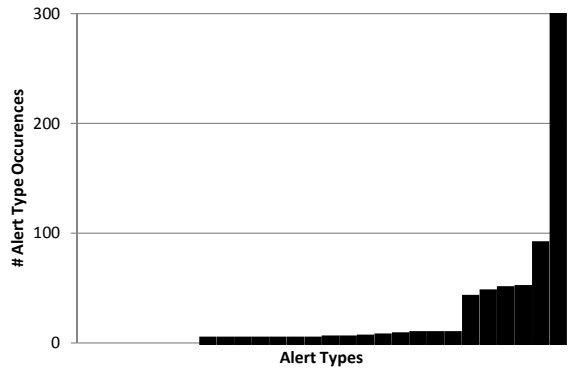
Dataset	**LLDDoS 1.0**
Top-1 Rate (%)	81.16
Top-2 Rate (%)	98

Figure 7(a) shows alert type occurrences in one of our testing sets from LLDDoS 1.0 (i.e. 50% of the alerts included in LLDDoS 1.0). We can see very different occurrence statistics for various alert types.

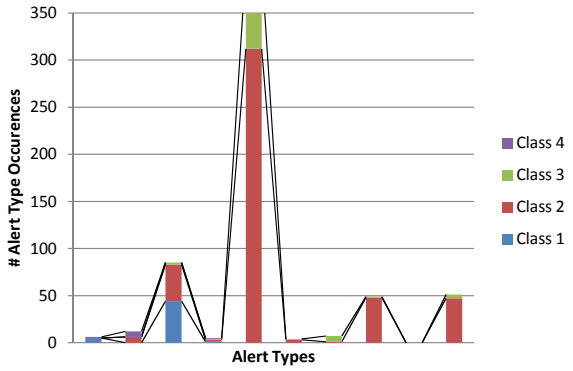
Interestingly, as illustrated, alert type occurrence statistics follow *the principle of factor sparsity* (also known as the 80–20 or Pareto Rule). Put another way, about 90% of the total number of alert type occurrences caused by only 21% of alert types. These alert types have high efficacy in unsupervised alert type classification and thus in prediction.

Figure 7(b) shows the number of alert type occurrences within each class. We chose four sample classes to put in the figure, since more than four curves are not easily distinguishable in a single chart. We can see large variances of alert type frequencies within a class. Effective alert types in prediction are local maximum points in each curve. These alert types are the most effective ones within each class that affect prediction more than other types. Note that each point in the horizontal axis represents more than one alert type (particularly four types in this case). As instance, a single point in horizontal axis shared between curves related to *Class 3* and *Class 4* may represent *alert type 2 in class 3* and *alert type 2 in class 4*, respectively; where the former type is *Active Communication Remote* and the latter type is *Root or Administrator Access*. All in all, Figure 7(b) demonstrates the varying alert type occurrences within each class determined by unsupervised training.

In practice, our objective is to alarm system administrator(s) about possible future attack classes. In the



(a)



(b)

Figure 7. (a) Number of Alert Types Occurrences in a Training Sample, (b) Number of Alert Type Occurrences within each Class.

learning phase of unsupervised mode, the HMM classified the alerts on its own. We assigned names to those classes based on the most frequent alert types of each class (i.e. effective alert types). As explained, highly frequent (and thus effective in prediction) elements in semantic classes (i.e. classes retrieved via manual classification) grouped almost correctly using unsupervised learning. Hence, we see similar prediction rates in supervised and unsupervised modes. We needed be aware of these elements and then name classes based on them. In this manner, we had almost the same performance in prediction as supervised mode. That is how we calculated prediction rates in unsupervised mode.

However, the proposed solution has its own limitations. Underlying IDSs play an important role in the system. Attacks missed by sensors would have some negative effects on prediction system.

Moreover, the model does not adapt to new patterns of attacks; but as it is operating in high levels of abstraction, we do not face the change of attack patterns in this level in an on-going manner. In contrast, if we consider attack type patterns, we can see ever-changing patterns of newly introduced attacks in time.

10 Conclusion

In this paper, we presented a system to correlate intrusion alerts and extract attack scenarios as well as to predict the next attacker action. We reduced the problem of finding multistage attacks to sequence mining and the problem of finding next attacker action to sequence modeling and prediction. We used DARPA 2000 to evaluate system performance and accuracy. The results show that the system can efficiently extract the attack scenarios and predict the attacker's next action. Particularly, the proposed system has the following advantages:

(1) The ASEA is able to operate in real-time environments,
(2) The simplicity of ASEA results in low memory consumption and computational overhead,
(3) In contrast to previous approaches, the ASEA combines both prior knowledge as well as statistical relationships to detect causal relationships,
(4) The prediction component proposes an unsupervised method to predict the next attacker action,
(5) The prediction component does not require any knowledge of the network topology, system vulnerabilities, and system configurations. Unlike Bayesian based methods that usually rely on a predefined attack plan library, HMM can perform in the absence of such information
(6) The prediction component performs high-level prediction; hence the model is more robust against over-fitting. In contrast, other plan recognition methods try to predict exactly the attacker's next action.

As the next step, we can exploit data mining to correlate alert types rather than using prior knowledge. We can even combine both to increase the proficiency of the method. In addition, preparing a public dataset appropriate to evaluate alert correlation algorithms is advantageous.

References

[1] F. Valeur, G. Vigna, C. Kruegel, and R.A. Kemmerer. A Comprehensive Approach to Intrusion Detection Alert Correlation. *IEEE Transactions on Dependable and Secure Computing*, 1(3):146–169, 2004.

[2] T. Pietraszek. Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection. In *Recent Advances in Intrusion Detection*, pages 102–124, 2004.

[3] R. Smith, N. Japkowicz, M. Dondo, and P. Mason. Using Unsupervised Learning for Network Alert Correlation. In *Advances in Artificial Intelligence*, pages 308–319, 2008.

[4] B. Morin, L. Mé, H. Debar, and M. Ducassé. M2D2: A Formal Data Model for IDS Alert Correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, RAID '02, pages 115–137, 2002.

[5] F. Cuppens and A. Miège. Alert Correlation in a Cooperative Intrusion Detection Framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, 2002.

[6] X. Peng, Y. Zhang, S. Xiao, Z. Wu, J. Cui, L. Chen, and D. Xiao. An Alert Correlation Method Based on Improved Cluster Algorithm. In *Proceedings of Computational Intelligence and Industrial Application*, PACIIA '08, pages 342–347, 2008.

[7] W. Li, L. Zhi-tang, L. Jie, and L. Yao. A Novel Algorithm SF for Mining Attack Scenarios Model. In *Proceedings of IEEE International Conference on e-Business Engineering*, ICEBE '06, pages 55–61, 2006.

[8] B. Zhu and A.A. Ghorbani. Alert Correlation for Extracting Attack Strategies. *International Journal of Network Security*, 3(3):244258, 2006.

[9] S.O. Al-Mamory and H. Zhang. IDS Alerts Correlation Using Grammar-based Approach. *Journal in Computer Virology*, 2008.

[10] S.J. Templeton and K. Levitt. A Requires/Provides Model for Computer Attacks. In *Proceedings of New Security Paradigms Workshop*, 2000.

[11] M.S. Shin and K.J. Jeong. An Alert Data Mining Framework for Network-Based Intrusion Detection System. In *Proceedings of the 6th International Workshop Information Security Applications*, pages 38–53, 2006.

[12] O. De Vel, N. Liu, T. Caelli, and T.S. Caetano. An Embedded Bayesian Network Hidden Markov Model for Digital Forensics. In *Proceedings of the International Conference on Intelligence and Security Informatics*, ISI '06, pages 459–465, 2006.

[13] D. Ourston, S. Matzner, W. Stump, and B. Hopkins. Applications of Hidden Markov Models to Detecting Multi-Stage Network Attacks. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences*, HICSS '03, 2003.

[14] D. Lee, D. Kim, and J. Jung. Multi-Stage Intrusion Detection System Using Hidden Markov Model Algorithm. In *Proceedings of the International Conference on Information Science and Security*, ICISS '08, pages 72–77, 2008.

[15] Y. Zhai, P. Ning, P. Iyer, and D.S. Reeves. Reasoning About Complementary Intrusion Evidence. In *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC '04, pages 39–48, 2004.

[16] A. Ehrenfeucht and J. Mycielski. A Pseudorandom Sequence—How Random Is It? *The American Mathematical Monthly*, 99:373–375, 1992.

[17] X. Qin and W. Lee. Attack Plan Recognition and Prediction Using Causal Networks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, ACSAC '04, pages 370–379, 2004.

[18] W. Lee and X. Qin. Statistical Causality Analysis of Infosec Alert Data. In *Proceedings of the 6th International Symposium on Recent Advances in Intrusion Detection*, RAID '03, pages 73–93, 2003.

[19] Z. Ning and J. Gong. An Intrusion Plan Recognition Algorithm Based on Max-1-Connected Causal Networks. In *Proceedings of the 7th International Conference Computational Science*, ICCS '07, 2007.

[20] D.S. Fava, S.R. Byers, and S.J. Yang. Projecting Cyberattacks Through Variable-Length Markov Models. *IEEE Transactions on Information Forensics and Security*, 3:359–369, 2008.

[21] H. Farhady, R. Jalili, and M. Khansari. Attack Plan Recognition Using Markov Model. In *Proceedings of the 7th International ISC Conference on Information Security and Cryptology*, 2010.

[22] P. Bahreini, M. AmirHaeri, and R. Jalili. A Probabilistic Approach to Intrusion Alert Correlation. In *Proceedings of 5th International ISC Conference on Information Security & Cryptology*, 2008.

[23] A. Valdes and K. Skinner. Probabilistic Alert Correlation. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, 2001.

[24] S. K. Harms and J. S. Deogun. Sequential Association Rule Mining with Time Lags. *Journal of Intelligent Information Systems*, 2004.

[25] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Readings in Speech Recognition*, 53:267–296, 1990.

[26] P.R. Cohen, C.R. Perrault, and J.F. Allen. *Beyond Question-Answering*. Bolt Branek and Newman Inc., 1981.

[27] T.C. Bell. *Text Compression*. Prentice Hall PTR, 1990.

[28] M. Roesch. Snort-Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th USENIX Conference on System Administration*, 1999.

[29] MIT Lincoln Laboratory. 2000 DARPA Intrusion Detection Scenario Specific Data Sets, 2000.

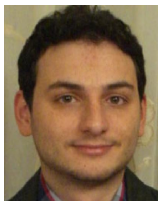
[30] North Carolina State University Cyber Defense

Laboratory. TIAA: A Toolkit for Intrusion Alert Analysis, Accessed May 24, 2009. Available from: http://discovery.csc.ncsu.edu/software/correlator/ver1.0/.

[31] P. Ning, Y. Cui, and D. Reeves. Analyzing Intensive Intrusion Alerts Via Correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection*, RAID '02, pages 74–94, 2002.

[32] J.M. François. Jahmm v0. 6.1, 2006. http://jahmm.googlecode.com.

[33] D. Yu and D. Frincke. Improving the Quality of Alerts and Predicting Intruder's Next Goal with Hidden Colored Petri-Net. *Computer Networks*, 51:632–654, 2007.



Hamid Farhadi received his BSc and MSc degrees from Shahid Beheshti University and Sharif University of Technology, International Campus, in 2007 and 2010 respectively. He is now student at Graduate School of Interdisciplinary Information Studies, Department of Applied Computer Science, The University of Tokyo, Japan. His research interests are cloud computing, infrastructure as a service, network virtualization, and network security.

Maryam AmirHaeri received her BSc and MSc degrees from Sharif University of Technology, Iran, in 2007 and 2009 respectively. She is now PhD student at Amirkabir University of Technology, Iran. Her research focuses on artificial intelligence, evolutionary computation, data mining, and machine learning.

Mohammad Khansari received his B.S, M.S and PhD degrees in Computer Engineering all from Sharif University of Technology, Tehran, Iran, in 1996, 1998 and 2008 respectively. He was the former faculty member at School of Science and Engineering, Sharif International Campus located in Kish Island for two years. Moreover, he was the head of Information Technology faculty and faculty member of Iran Telecommunication Research Center. Currently, he is the faculty member and assistant professor at faculty of new sciences and technologies, university of Tehran. He had a short-time research fellowship from DAAD, Germany and has given more than fifty invited talks on Free/Open Source Software (FOSS) in Iran and International conferences and summits. He is the co-author of four books in Free/Open Source Software topics and has more than thirty-five papers in international conferences and journals. His main research interests are pattern recognition, multimedia sensor networks, multimedia delivery over peer-to-peer networks, and free/open source software.