RESEARCH ARTICLE

A realistic graph-based alert correlation system

Ouissem Ben Fredj *
Taif University, Saudi Arabia

ABSTRACT

This paper introduces a graph-based attack description that comes with different analysis methods for alert correlation. The system encompasses an attack scenario detection method, an alert correlation method that recognizes multistep attacks, and graph-based classification method to extract different types of alerts. The performance analysis shows that the system can correlate a huge number of alerts (more than 442,000 alerts) into a dozens of attack graphs. The attack graph has permitted us to extract several attack properties with high precision. Copyright © 2015 John Wiley & Sons, Ltd.

KEYWORDS

security; correlation; attack graph; Markov chain

*Correspondence
Ouissem Ben Fredj, Taif University, Saudi Arabia.
E-mail: ouissem.benfredj@gmail.com

1. INTRODUCTION

Many works have been done in order to define a common distributed intrusion detection system (DIDS) framework [1–3]. The architecture model of the state-of-the-art DIDS shares almost the same global architecture. Figure 1 shows an example of such an architecture called (φ | π) [2]. The architecture encompasses a set of sensor elements that monitor the target system. A database is necessary to store alerts generated by the sensors. The system administrator uses correlation software in order to study the incoming alerts and react against intrusions by creating new detection rules. The new rules are used for subsequent detections by the sensors.

In order to have a global view of the system, the DIDS must gather events from sensors and Network IDS (NIDS). The events should be correlated to provide a simple view for the administrator with deep analysis of attacks.

The correlation system is a generic name that may include several tasks such as normalization, correlation, aggregation, and fusion. The alert normalization attempts to write the collected alert in a unique exchange format usually based on extensible markup language such as intrusion detection message exchange format [4]. The alert aggregation aggregates overlapped alerts according to one or several alert features. The alert correlation focuses on discovering the relationships between raised alerts.

As detailed in Sections 2 and 6, existing alert correlation systems fail to answer one or more correlation problems.

This paper considers as input the alerts generated by several IDS sensors and processes them across a global alert correlation system based on graphs and absorbing Markov chains. The main contributions of this paper are the following:

- a design of global alert correlation system that deals with heterogeneous alerts, unifies them, removes false alerts, aggregates the alerts, correlates them, creates attack scenarios, predicts future behavior of the intruders, classifies alerts, and visualizes the result to the administrator;
- a design of a context-based alert management component used to store the organization architecture, the firewall policies of the main hosts and the existing subnets. This component is used to drop incoming alerts that could not have any affect in the network based on the firewall policies (reduction of false positive). It could be used also to filter and to aggregate alerts according to a given host or a given subnet;
- a graph-based attack scenario model that ensures intuitive representation of correlated alerts and provides an elegant and robust attack analysis framework;
- an absorbing Markov chain-based model for attack description that enables adaptive and precise attack recognition, analysis, and prediction;
- using the absorbing Markov chain model, various properties of the attack scenarios and the intruders including the estimated number of steps required to reach the attack objective, the estimated number of each alert during an attack, the intruder gravity, critical alerts, near-critical alerts, and so on; and
a performance analysis using a new state-of-the-art dataset (CIF’17 data set).

In contrast to the IDS, the goal of the current method is not the accuracy and the precision of the alert labeling. However, the purpose of the correlation system is to correlate the incoming alerts in real-time, in scalability with the number of alerts and the network speed, and in a realistic way (gives a real picture of the system without losing any incoming alert).

The correlation system is faced to several challenges that include the following:

Real-time: The correlation system must handle all the events generated by the distributed IDS sensors. This is a major requirement. It becomes hard to establish when a complex attack is detected. The correlation system may take much time processing the attack and its underlying actions. Thus, it could become a bottleneck and may delay the administrator actions. The system must guarantee a real-time processing during all the involved steps.

Scalability: The system must be scalable. It must allow easily sensor addition and must handle the traffic as it grows. However, the scalable correlation system must keep the same accuracy and real-time properties.

Portability: The system should be able to communicate a wide variety of event formats and different sensor architectures. It should support different host hardware component types and different network interconnects. The system should deal with new attacks, new hardware, new software, new protocols, new networks, and new operating systems.

Graphical user interface: One of the most important challenges of the correlation system is to provide an intuitive and simple graphical user interface. The simplicity should not affect the accuracy and the quality of the correlation results. The system must provide a trade-off between the stability and the real-time properties of the results.

The remaining of the paper is organized as follows. Section 2 provides a state of the art existing correlation systems and methods. Section 3 introduces the different components of the correlation system. Section 4 details the context-based alert management system. Section 5 presents the correlation system. Section 6 shows the performance analysis of the system.

2. RELATED WORKS

There are a number of correlation method classes: association rule learning (dependency modeling), clustering, classification, and summarization. Each correlation system may include one or several methods from the following classes.

The clustering methods are unsupervised learning that generates possible clusters that encompass similar alerts. The similarity algorithm is often based on a subset of the alert features such as IP destination, IP source, and timestamp [5]. The authors of [5,6] state that each alert occurs for a reason called root cause. If two alerts have the same root causes, they are correlated. [7] aggregates alerts using triggering events. There are also methods that employ machine learning techniques for alert clustering. [8] combines the alerts into scenarios; each is composed of a sequence of alerts produced by a single actor or organization. It uses CART data mining method in order to estimate
the probability that a new alert belongs to a given scenario. The main drawback of the systems based on clustering is that they usually group totally disjoint alerts based on non-realistic similarity distances. For example, the systems allow grouping two alerts that have close alert names but does not belong to the same attack scenario and/or not have the same intruder source. Moreover, clustering similar alerts does not give much information about the attacks. In contrast, our method groups the alerts belonging to the same attack. The method begins by determining the intruder and then follows his behavior in order to build the attack scenario. The attack scenarios would be used as the classes of the classification and prediction processes.

The classification methods are supervised learning that direct a new alert to one class among a set of predefined classes. The main difficulties of these methods are the definition of the classes and the rule used to perform the classification task, thus leading to ignoring new attack classes. The difficulties come from the scarcity of the alert features. The methods give a good result while grouping alerts based on one feature or hierarchical features. Otherwise, classification methods will present the same problem as the clustering methods. In fact, the state-of-the-art classification methods combine similarity between different features. Similarity scores, which lead to non-realistic correlation. [9] proposes a mechanism that filters false alerts after a classification step. Our method does not use predefined classes but creates new attack scenarios at runtime depending on the incoming traffic. The construction of scenarios does not require any prior knowledge, recognizes new attacks, and gives a real picture of the current traffic.

The association rule learning combines data mining algorithms with artificial intelligence concepts in order to create hyperalerts and its eventual associated graphs. Hyperalerts are described in terms of predicates based on the prerequisite-consequence paradigm. This method, in addition to the correlation task that it ensures, allows to predict future alerts and the intruder’s objectives. Moreover, the association rule learning requires a prior knowledge of the prerequisites and their consequences. Moreover, systems in this class do not show unknown attacks and modified attacks or classify them as unrecognized. [10] proposes an attack graph-based method to correlate alerts raised for known attacks and hypothesize missed alerts [11,12]. Our method is based on graphs used to represent extracted attack scenarios from real network traffics and does not require prior knowledge such as prerequisites and consequences. The unknown attacks are recognized and corresponding attack graphs are created.

Summarization approaches just aggregate similar alerts and/or filter the alerts according to specific features and report them. They do not include advanced intelligence. These approaches are more adapted to small and vulnerable organizations. Most IDSs provide summarization mechanisms [13,3,14]. Our method aggregates alerts coming from the same intruder using network prefixes while dealing with stealthy attacks using forged IP addresses.

Our method is based on two fundamental models: the absorbing Markov chain and graphs. The absorbing Markov chain model is used to define the probabilities to transit from one attack action to another according to a given scenario before reaching one of some attack objectives. The transition probabilities are deduced from the data set, which allows a precise and adaptive real-time view of the network. Most of existing Markov chain-based alert correlation methods use the HMM (hidden Markov chain) in which the sequence of states is not observable. In [15], the authors claimed that HMMs are suited to recognize multistep attacks through its prerequisites. Hidden Markov Model for Attack Intention Prediction (HMM-AIP) [16] uses HMMs to design an offline training module and an online tracking and prediction module. Farhadi et al. [17] combined both prior knowledge and statistical relationships to build attack scenarios and used an HMM method to predict the next attack class of the intrusion. The Markov-based correlation systems share the same advantage to be well founded but also the same drawbacks, in contrast to our method, of requiring to set several probabilistic parameters and thresholds manually and to require a big and reliable training data set. They usually fail to detect unknown attacks with the same quality of known attacks. Our proposed method uses the absorbing Markov chain model to define the initial states and the objectives of each multistep attack and the probabilities to transit between states. The probabilities are deduced automatically in real time and do not require any prior knowledge.

Our method is also based on graphs to represent and analyze attack graphs and scenarios. Other methods are based on the network attack graph, which represents the possible ways an intruder can violate a security policy according to the network architecture of the organization. Network attack graphs are different from alert attack graphs; the latter, which are used in our work, are the result of the alert correlation process, but the network attack graph depicts how an intruder could leverage dependencies among vulnerabilities to violate a security policy. A network attack graph can be generated from network configuration details and known vulnerabilities within the network using scanner software like TVA, Nessus, or MuhlVAL. In [18] for example, the correlation method uses the Floyd–Warshall algorithm to map incoming IDS alerts to specific network attack graph nodes and edges in order to generate an alert dependency graph (attack scenarios). In the same way, Gruschke [19] uses the breadth–first search to find an eventual correlation between alerts using the network dependency graph. Wang et al. [20] transform a network attack graph into queue graphs. Each queue graph keeps in memory the latest alert matching each of the known exploits, thus ensuring that the complexity is independent from the incoming alerts. The queue graphs also guarantee a linear time complexity of the algorithm because its corresponding algorithms search in trees rather than graphs. An inconsistency between the network attack graph and the incoming alerts orders detects missed alerts (not reported by IDSs) and predicts future alerts.
In contrast to our method, all presented correlation methods require the network attack graph as input. However, for real networks, current network attack graphs are very large and dense. Figure 2 is an example of an attack graph for a subnet of only 14 machines, with less than 10 vulnerabilities per machine. The tools that generate network attack graphs do not give good results, and they should be continuously updated. The graphs usually require manual fine tunings and updates. The complexity of underlying correlation methods is very high and does not recognize new attack scenarios especially when the system does not find any correspondence between alerts and vulnerabilities described in the network attack graph. Our method does not require any prior knowledge of exploits and its consequences and the complexity of our method is time linear.

3. THE ARCHITECTURE OF THE ALERT CORRELATION SYSTEM

This section describes the global architecture of the alert correlation system. As shown in Figure 3, the system is composed of four layers; the first layer, from left to right, encompasses the IDS sensors. The system allows multiple IDS sensors of the same or different IDS types. The sensors store alerts in a common database.

The second layer is a preprocessing layer; it is split into two sub-layers. The first sub-layer takes advantage from a unified alert name database in order to link each generated alert to a unified alert name. The unified alert is a meta-alert that solves the heterogeneity between the alerts of different IDS sensors. In Fact, each IDS sensor has a predefined alert database that is different from other sensors. A same effective alert could have different names and descriptions in different alert databases. The unified alert name database creates for each set of similar alerts from different IDS sensors one meta-alert. The meta-alert will be used instead of its corresponding similar alerts thus decreasing the number of alerts considered by the system without losing any alert information.

The second sub-layer of the preprocessing layer is the context-based control (see section 4). This sub-layer checks the generated alerts with the real security policies of the organization. Depending on the policy rules of the different firewalls, it detects the false positive alerts that were generated by the IDS sensors and have not any effect in the organization because the corresponding traffic will be dropped or denied by the destination host or the future firewall. The result of the preprocessing layer is stored in the database of unified and cleaned alerts.

The generated unified alerts will then pass through the correlation layer, which is the core module of the system. The correlation layer analyzes the alerts in order to aggregate similar alerts, to correlate between them in order to deduce current attack scenarios, to predict scenarios, to classify true/false positive/negative alerts and to visualize the result to the administrator.

The administrator, upon receiving a summary about the security incidents, might perform actions to address current attacks and prevent future ones. These actions are considered as expertise of the system and will be used by the correlation system to improve the accuracy of the results and the inspection of the alert classes (true/false...
positive/negative). The actions will be also considered by the context-based control layer because the actions may alter the firewall rules.

This paper focuses on the context-based alert management and the correlation system.

4. THE CONTEXT-BASED ALERT MANAGEMENT

Many false positive alerts are caused by the unavailability of the target resource or the target port. For example, the IDS can raise a map alert for the host 191.10.11.12, port 43 while the host’s firewall denies the use of the port 43. Hence, we propose to add a context-based alert management layer in the correlation system. The layer includes a database and has two major functions: dropping and filtering.

The database encompasses the architecture of the organization and the firewall policies of the main hosts and the existing subnets. Figure 4 shows a snapshot of the database schema:

Each subnet is characterized by the subnet mask, its name, and the type of the subnet, which may be security-center, servers, or regular. The type of the subnet gives an idea about the vulnerability and the importance of the subnet. The subnets of the type of security center and servers should provide the firewall policies of the hosts that belong to it. The policies table contains a part of the firewall policies used by each host. The correlation system does not require all the policies to be listed but only the denied hosts and ports. Note that this database should be updated periodically or each time the policies change. This could be performed automatically by using a topology discovery and firewall rule parser tools.

The first main function of the context-based alert management layer is to drop all the alerts reported by the IDS that could not have any effect according to the saved security policies, thus reducing false positive alerts (called inactive alerts). This would decrease dramatically the number of generated alerts. These alerts should not be deleted from the system and they should be considered in the attack graph construction. This is because inactive alerts would be transformed to active alerts when some firewall rules are changed.

The firewall rules could be represented as binary decision diagrams (BDDs). A BDD [21,22] is suited for

Figure 3. The global architecture of the correlation system.

Figure 4. The organization context: the database schema of the organization architecture and the firewall policies of the main hosts.
A realistic graph-based alert correlation system

O. B. Fredj

5. THE CORRELATION SYSTEM

5.1. Notations

Let $D = (V, A)$ be a directed graph composed of a set of vertices $V$ and a set of edges $A$. For a vertex $v$ in $D$, we use the following notation:

- $N_D(v) = \{w \in V \mid (v, w) \in A\}$, $N_D^+(v) = \{w \in V \mid (w, v) \in A\}$

The sets $N_D(v)$, $N_D^+(v)$, and $N_D(v) = N_D^+(v) \cup N_D^-(v)$ are called the out-neighborhood, in-neighborhood, and neighborhood of $v$ respectively. We call the vertices in $N_D(v)$, $N_D^+(v)$, and $N_D^-(v)$ the out-neighbors, in-neighbors, and neighbors of $v$.

5.2. Graph-based correlation system

Our approach falls under the graph-based correlation class, which tries to depict attack scenarios but does not require prior knowledge. Once the attack graphs are built, the system will be able to correlate alerts within these attack graphs, predict future alerts, improve existing graphs depending on new alerts, and give the administrator an intuitive (easy to use) picture of the network.

Generally, an intruder performs several actions in a well-defined order called attack scenario. In an attack scenario, early actions alter a system and/or provide knowledge to an intruder in order to perform later actions. An attack scenario is modeled as a process for actions that transforms a system from one state to another, until reaching some targets that we call intrusion objective.

The definition of the intruder is explained in Section 5.6.

**Definition 1.** An attack graph is a graph $(V, E, \alpha, \beta)$ where

- $V$ is the finite set of nodes,
- $E \subseteq V \times V$ is the set of edges,
- $\alpha: V \to N$ is the node labeling function, and
- $\beta: E \to N$ is the edge labeling function.

The attack graph includes a finite node set $V$ that forms the alerts that could be generated by the IDS sensors. The node labeling function assigns to each node a unique label, which is the set $\mathcal{A}$ of alerts. The edge labeling function assigns to each edge a weight that corresponds to the number of repetitions of the transition from an alert to another. If we consider the example of Figure 5, the weight of the edge $(4, 14)$ is 2 means that the intruder has caused the generation of the alert $4$ then the alert $14$ two times.

5.3. Scenario and alert classification system

The attack graph could be transformed into a more realistic graph by changing the alert transition number to probabilities in two steps:

1. Given a vertex $v \in V$, let $\mathcal{W}(v) = \Sigma_{w \in N_D^+(v)} \beta(wv)$, the sum of the weights of the out-neighbors of $v$. Replace the weight of a given edge $wv$ by $\frac{\beta(wv)}{\mathcal{W}(v)}$, which is equal to $\frac{\beta(wv)}{\sum_{w \in N_D^+(v)} \beta(wv)}$.

Because $\beta(wv) = \Sigma_{v \in N_D^+(v)} \beta(vz)$, then the new weight of any edge $wv$ is in the range of $[0, 1]$. The new weight $(wv)$ could be interpreted as the probability to transit from the state $v$ to the state $w$. That is, the probability that the intruder does generates the alert $w$ after generating the alert $v$.

2. If a vertex $v$ does not have any outbound edge, create an edge $(v, v)$ and assign 1 as its weight.

The vertex $v$ correspond to the last alert generated by the intruder is considered as the attack objective. That is, the probability to leave $v$ is 0, and the probability to go to $v$ is 1. That is, the attack graph is being absorbed in the alert $v$.

**Proposition 1.** Let $g = (V, E, \alpha, \beta)$ an attack graph. If we suppose that the alert numbers are states and the future

DOI: 10.1002/sec
state depends only on the previous state, the graph produced by the previous changes (1 and 2) is a Markov chain.

Definition 2. A **Markov chain attack graph** is a graph \((V, E, \alpha, \beta)\) where 
- \(V\) is the finite set of nodes,
- \(E \subseteq V \times V\) is the set of edges (alert transition) is the state space,
- \(\alpha : V \rightarrow N\) is the node labeling function (alert numbers),
- \(\beta : E \rightarrow \{0,1\}\) is the edge labeling function (the probability to go from a state to another). Given an edge \((v,w)\), the probability \(\beta(v,w)\) that the alert \(w\) will be generated given that the current generated alert is \(v\), depends only on the state \(v\).

Definition 3. A **complete attack graph** is a Markov chain attack graph that includes at least one absorbing state.

Proposition 2. Let \(g = (V, E, \alpha, \beta)\) a Markov chain attack graph. If \(g\) contains at least one absorbing state then \(g\) is an absorbing Markov chain.

**Proof.** In order to prove that \(g\) is an absorbing Markov chain ID \(g\) contains at least one absorbing state, we must prove that it is possible to go from any state to at least one absorbing state in a finite number of steps. This is true because the intruder has passed through all the alerts and has reached the absorbing state. Thus, from every state, the absorbing state could be reached in a finite number of steps.

Proposition 3. A **Complete Attack Graph** is an absorbing Markov chain.

Definition 4. An **attack objective** is an absorbing state; that is, if \(d^*(v) = 0\) and \(d^*(v) > 0\), then the alert \(v\) is attack objective.

Definition 5. A **near-critical alert node** is a node whose removal of its corresponding alert (if raised) decreases the probability to reach the objective by at least \(a\).

Definition 6. A **critical alert node** is a node whose removal of its corresponding alert (if raised) disables reaching the objective.

Note that a critical alert is a 1-near-critical alert node.

Definition 7. An **active alert node** is a node whose corresponding alert is a true positive or false negative.

For example, in a context-based detection, an alert of secure shell port probing is active if the port 22 is open. Hence, the intruder may use this open port for the following attack.

Definition 8. An **inactive alert node** is a node whose corresponding alert is a false positive or true negative.

If the firewall closes the port 22, then the alert generated by the IDS about a secure shell attack is inactive and has no effect in the network. However, the alert generated would be useful when analyzing the attack scenario.

The above properties could be deduced from a complete attack graph (the absorbing Markov chain) as follows.

Let an absorbing Markov chain with transition matrix \(P\) that have \(t\) transient (regular alert) states and \(r\) absorbing states (attack objective). Then

\[
P = \begin{pmatrix}
Q & R \\
0 & I_r
\end{pmatrix}
\]

where \(Q\) is a \(t\)-by-\(t\) matrix, \(R\) is a nonzero \(t\)-by-\(r\) matrix, 0 is an \(r\)-by-\(t\) zero matrix, and \(I_r\) is the \(r\)-by-\(r\) identity matrix.
Thus, $Q$ describes the probability of transitioning from some transient state to another while $R$ describes the probability of transitioning from some transient state to some absorbing state.

If we suppose that a given complete attack graph includes only one absorbing state, which is an attack objective, the transition matrix would be as the following:

$$P = \begin{pmatrix} 0 & R \\ Q & 0 \end{pmatrix}$$

where $Q$ is a $t$-by-$t$ matrix, $R$ is a one column vector, $0$ is a one-zero row, and $1$ is the element $P_{r+t,s+1}$.

The following is the transition matrix associated to the example in Figure 6:

$$
\begin{pmatrix}
10 & 1/2 & 1/2 & 0 & 0 & 0 & 0 & 0 & 0 \\
8 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 1/2 & 1/4 & 0 & 1/4 & 0 & 0 \\
5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
9 & 0 & 0 & 0 & 0 & 0 & 2/3 & 0 & 1/3 \\
17 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2/3 \\
14 & 0 & 0 & 1/2 & 0 & 0 & 0 & 1/2 & 0 \\
12 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix}
$$

Let $u$ be the probability vector that represents the starting distribution. Then, the probability that the chain is in state $s_i$ after $n$ steps is the $i$th entry in the vector $u^{(n)} = uP^n$.

The limit $\lim_{n \to \infty} P^n$ gives us an idea about the convergence of the Markov chain process. Because the current chain is an absorbing Markov chain, the probability of reaching an absorbing state is nonzero, let it be $s$. Then, initially, the probability that the process will not be absorbed is $(1-s)$. After $n$ steps, this probability is equal to $(1-s)^n$. We know that as $n \to \infty$, $(1-s)^n \to 0$. Thus, for every transient state/alert $i$, the probability that the intruder remains in a transient state is 0. In other words, $\lim_{n \to \infty} Q^n = 0$.

Then

$$
\lim_{n \to \infty} P^n = \begin{pmatrix} 0 & R \sum_{i=0}^{\infty} Q^i \\ 0 & I \end{pmatrix}
$$

Let $N = \sum_{k=0}^{\infty} Q^k$ is called the fundamental matrix of $P$. $N_{ij}$ gives the expected number of steps that the process is in the transient state $j$ if it is started in the transient state $i$. $N_{ij}$ gives the expected number of each alert of the current scenario depending on the initial alerts raised by the intruder.

$$
\lim_{n \to \infty} P^n = \begin{pmatrix} 0 & RN \\ 0 & I \end{pmatrix}
$$

The column $i$ of $RN$ gives the probabilities of ending up in each of the absorbing states, given that the process started in the $i$th transient states.

$$
N = (I-Q)^{-1} \quad \text{because} \quad QN = Q \sum_{k=0}^{\infty} Q^k = Q(I + Q + Q^2 + \ldots) = N - I. \quad \text{Note that} \quad (I-Q)^{-1} \quad \text{exists because} \quad Q \quad \text{is a square matrix and} \lim_{n \to \infty} Q^n = 0 \quad [24].
$$

In the following is the fundamental matrix associated to the example in Figure 6:

Figure 6. The Markov chain attack graph associated to the attack graph of Figure 5.
Let \( I \) be the set of possible initial alerts that could be generated in the attack graph. It could be created by saving the initial alerts generated by each intruder that follows a given attack scenario.

The relevant rows of the matrix \( N \) are those corresponding to the initial alerts that might be generated by the intruder while following the current attack scenario.

If there is only one initial state \( i \), then only the row \( i \) of \( N \) is relevant. In our example, the intruder started by raising the alert 10, thus the row corresponding to the alert 10

\[
\begin{bmatrix} 2 & 1 & 4.04 & 1 & 3.05 & 1 & 3.03 & 2.03 \end{bmatrix}
\]

indicates the estimated number of each alert during the current attack.

If we add all the entries in the \( i \)th row of \( N \), we will have the expected number of steps in any of the transient states for a given starting state \( s_i \), that is, the expected steps required before being absorbed. \( t_i \) is the sum of the entries in the \( i \)th row of \( N \). Let \( t \) be the column vector whose \( i \)th entry is \( t_i \). Then

\[
t = Nc
\]

where \( c \) is a column vector; all of its entries are 1.

The value \( t_i \) gives us the expected steps required to reach the attack objective if the intruder has just raised the alert \( i \).

The value of \( t \) associated to the example is

\[
(17.19, 15.21, 17.16, 18.19, 14.21, 12.18, 11.18, 15.17)^T
\]

The vector \( t \) could be used also while drawing attack graphs. For a given attack objective, the farther the alert from the objective, the more time might be taken to reach it; the expected time given in \( t \) will be transformed to distance while drawing attack graphs.

Let \( \min_{\text{absorb}} = \min_{i \in \mathcal{A}_t} t_i \). \( \min_{\text{absorb}} \) is the minimum time required by an intruder to reach the attack objective of the current scenario.

Similarly, let \( \max_{\text{absorb}} = \max_{i \in \mathcal{A}_t} t_i \). \( \max_{\text{absorb}} \) is the maximum time required by the intruder to reach the objective of the current scenario.

If an intruder \( i \) has reached the objective of a given scenario in a time \( AT \), the intruder \( i \) could have assigned an **intruder gravity ratio** \( V_i \) computed as the following:

\[
V_i = \frac{AT - \min_{\text{absorb}}}{\max_{\text{absorb}} - \min_{\text{absorb}}}
\]

The intruder gravity ratio \( V_i \) is a real value that belongs to the range \([0,1]\).

### 5.4. Determining critical and \(\alpha\)-near-critical alerts

The detection of the critical and \(\alpha\)-near-critical alert nodes for each attack graph is very important to the security administrator of the organization. They allow to know the alerts that have high impact in reaching the attack objective. Knowing these alerts, the administrator tries to disable raising them by changing the firewall policies, for example.

As introduced in the definition of a critical alert, a critical alert node is a near-critical alert node. So that, only one algorithm is needed to determine critical and \(\alpha\)-near-critical alert nodes.

The idea of the algorithm is original and is based on the shortest path algorithm. Note that the detection of (near-) critical alert nodes is done only once for each attack graph. The algorithm would be replayed when the structure of the attack graph changes.

The algorithm is as the following: given an attack graph that encompasses \( n \) nodes, a source node \( s \), and an attack objective node \( o \), the algorithm finds the shortest paths between \( s \) and \( o \) that passes through each node.

Note that the shortest path from a node \( s \) to a node \( o \) that goes through a node \( p \) is the concatenation between the shortest path between \( s \) and \( p \) and the shortest path from \( p \) to \( o \).

The result of this step is a binary-valued matrix \( SP \) of size \((n-2)\times(n-2)\). \( SP_{ij}=1 \) if the path from \( s \) to \( o \) that passes through \( i \) goes also through \( j \), 0 otherwise. The \( SP \) matrix associated to the example in Figure 7 is as follows:

- Using the absorbing Markov chain probability as cost:

\[
\begin{bmatrix} 0.68 & 1.34 & 2.71 & 0.68 & 4.06 & 0.68 & 3.03 & 2.03 \\
0.68 & 0.34 & 2.71 & 0.68 & 4.06 & 0.68 & 3.03 & 2.03 \\
1.34 & 0.67 & 5.37 & 1.34 & 2.04 & 1.34 & 3.03 & 2.03 \\
2 & 1 & 4.04 & 2 & 3.05 & 1 & 3.03 & 2.03 \\
0.68 & 0.34 & 2.71 & 0.68 & 1.03 & 1.68 & 3.03 & 2.03 \\
0.68 & 0.34 & 2.71 & 0.68 & 1.03 & 1.68 & 3.03 & 2.03 \\
1 & 0.5 & 4.04 & 1 & 1.53 & 1 & 3.03 & 3.03 \\
\end{bmatrix}
\]
(SPₐₖ = 1 for all i ∈ V \{10,12\}), which means that the alert number 4 is critical.

A vector CP (critical probabilities) could be deduced from the previous matrix that summarizes how much the probability to reach the objective will be decreased after the removal of each node:

\[ CP_i = \sum_{a \in V / \text{source}, \text{objective}} SP_{a \mid i} \]

Each element of CP is the sum of rows of SP in the same column of the element divided by the total number of nodes of the attack graph except the source and the destination.

Note that the source and the attack objective are critical nodes.

Considering the example above,

\[ CP = \begin{pmatrix} 2 & 2 & 5 & 2 & 5 & 1 & 1 \end{pmatrix}^T \]
\[ = \begin{pmatrix} 8 & 3 & 5 & 9 & 17 & 4 & 14 \end{pmatrix}^T \]
\[ = \begin{pmatrix} 0.28 & 0.71 & 0.28 & 0.28 & 0.71 & 1 & 0.14 \end{pmatrix} \]

Thus, for example, the 0.7-near-critical alert nodes are \{10 (source), 3, 17, 4, 12 (target)\}. The critical alert nodes are \{10, 4, 12\}.

- Without using edge costs, SP will be as follows:

<table>
<thead>
<tr>
<th></th>
<th>8</th>
<th>3</th>
<th>5</th>
<th>9</th>
<th>17</th>
<th>4</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

5.5 A classification-based alert correlation system

A training data set is a data set of a real traffic that includes real security attacks from different intruders. A good data set should include recent traffic, a big amount of traffic, a corresponding organization architecture close to the organization architecture where the correlation system runs, and labeled alerts that show for each whether it is a true positive alert or a false positive alert.

A training data set has two major benefits. First, it allows building a predefined model including the correlation system parameters and rules (scan time window, attack graphs, critical alerts, etc.). The predefined model guarantees a productive correlation system when launching the system at first time. In fact, the system would maintain a good accuracy of the classification and correlation system from the beginning. The second advantage of the training data set is that it allows the detection of true negative and false negative alerts in the current systems starting from the earlier stage of the execution of the system. Note that our system can be used without training data sets. However, using training data sets would be beneficial.

The false negative alarm is an important alert that is not reported by the IDS. It could be detected when the correlation system maps a current attack scenario to an existing attack graph, and that the existing attack graph includes a critical alert or a near-critical alert that has not been reported by the current IDS. The unreported critical and α-near-critical alerts are the false negative alerts.

The true negative alert is the alert that was not reported by the IDS but it exists in the existing attack graph that the correlation system has mapped the current scenario to it.
The **true positive** alert is the alert that is reported by the IDS and it is labeled as true positive in the predicted attack graph.

The **false positive** alert is the alert that is reported by the IDS, and it is labeled as false positive in the training or past data set. The alerts that are reported by the IDS but are reported as ineffective by the context-based alert management layer are also false positive.

### 5.6. IP-based alert aggregation

Considering only IP addresses of physical hosts in a network as source and destination addresses may miss considerable attack paths. One of the major issues in alert correlation is to locate intruders, in particular when considering network address translation (NAT), IP spoofing, and the fact that the intruder could create an attack scenario from several hosts located in different organizations.

We suggest a set of aggregation rules that clusters the alerts into groups. Each group is assumed to collect the alert of a different intruder:

- Aggregate alerts from the same IP source.
- Aggregate alerts generated from inbound traffic that have the IP destination same as the intruder source address.
- Aggregate alerts that have the same class C network address.

The administrator could define specific network prefixes to be considered for alert aggregation. This could be defined for a specific organization, network, or autonomous system.

### 6. EVALUATION OF THE SYSTEM

This section aims to evaluate our correlation system. However, despite the huge effort done in the alert correlation field, there is not a clear formulation of the performance criteria that determine their effectiveness. To our best of knowledge, there is no comprehensive comparative study of the correlation systems for many reasons; first, there are several correlation methods; each one has its own architecture, its own philosophy, its own technique, its own capabilities, its own performance criteria, and its own data set. Second, there is no standard benchmark for the evaluation. Moreover, there is no standard labeled data set compliant with specific performance criteria. However, some survey studies provide simple comparative studies usually limited to a number of papers and to a number of capabilities without performance analysis. In the following, we list some of these works and the position of our method according to studied criteria.

Yusof *et al.* [25] classified alert correlation techniques into four groups: similarity-based, predefined attack scenarios, prerequisites and consequences, and statistical causality. Then, they proposed six capability criteria for evaluating alert correlation techniques: alert reduction, alert clustering, identification of multistep attacks, reduction of false alerts, detection of known attacks, and detection of unknown attacks. Their study results are shown in Table I.

Our proposed method performs alert reduction in the attack graph by showing only one vertex for each alert ID even if the alert is raised thousands of times during a given attack. Alert reduction is also a result of the filtering and aggregation operations in the correlation views (Section 4). The alert clustering is done by clustering alerts coming from the same the intruder. Identification of multistep attacks is ensured by linking alerts within attack graphs according to the actions performed by each intruder. The reduction of the false alerts is guaranteed by the context-based alert management component that drops all reported alerts that have no effect in the network according to the firewall and other security policies. The detection of known attacks is maintained by the classification of created scenarios according to existing attack graphs. If a new scenario is not similar to any of the existing attack graphs, it will be considered as a new unknown attack.

In the same context, the paper [26] studied some representative work in the alert correlation area. They categorized the correlation methods according to the underlying techniques into six categories: rule-based, model-based, case-based, neural networks, hybrid soft-computing, and model checker. Then, they compared some representative works from each category. The comparison covered only functional design and was limited to five operations:

| Table I. Alert correlation technique versus proposed capability criteria according to [25]. |
|-------------------------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|
| Alert reduction                                | Alert clustering                | Identification of multistep attacks | Reduction of false alerts      | Detection of known attacks      | Detection of unknown attacks   |
| Similarity-based                               | ✓                               |                                  | ✓                               | ✓                               |                                  |
| Predefined attack scenarios                    |                                  |                                  | ✓                               |                                  | ✓                               |
| Prerequisites and consequences                 |                                  |                                  |                                  | ✓                               |                                  |
| Statistical causality                          |                                  |                                  |                                  |                                  | ✓                               |
| Our method                                     | ✓                               |                                  |                                  | ✓                               | ✓                               |
normalization, verification, aggregation, correlation, and attack scenario analysis. The normalization operation represents the alerts in a standard form like the intrusion detection message exchange format (IDMEF), the verification operation reduces the false positive alerts, the aggregation operation groups alerts constituting a single attack, the correlation discovers the relationship among alerts or attacks to construct the multi-stage attack scenarios, the attack scenario analysis allows for example ranking various attack paths. Their conclusions are summarized in Table II. Note that the work of Qin et al. used an external tool to reduce false alerts; for this reason, its verification capability has been removed from the original table.

Our method provides all the six operations; it includes the alert unification phase that normalizes the alerts that comes from heterogeneous IDS sensors. The verification capability is the combination of the two capabilities proposed by the previous benchmark: the alert reduction and the reduction of false alerts. The aggregation operation is similar to the alert clustering operation proposed by the previous benchmark. The correlation capability is the main contribution of the current paper. Moreover, our method provides several ways to analyze the scenarios, rank them, identify the critical alerts, and severe intruders.

In the remaining, a deep performance analysis based on the Defcon’s CTF’17 data sets will be presented. Defcon is the largest Internet security community in the world. Defcon provides a “Capture the Flag” (CTF) contest, which is a contest of computer security attack and defense skills. CTF attracts several expert intruders in a legal opportunity to demonstrate their skills in a public forum. Each team has to defend its own flag, while trying to corrupt the flags of as many of the other teams. A flag is a data file on the team’s server. During the game, intruders seek to replace the flag on someone else’s server with their own flag, while defenders try to preserve their own flag on their own server. Figure 8 shows a score server periodically polls the player servers to

| Alert correlation technique versus proposed capability criteria according to [26]. |
|-----------------------------------------------|--------|--------|--------|--------|------------------|
| Rule-based (case 1)                          | ✓      | ✓      | ✓      |                    |
| (Cuppen et al [27])                          |        |        |        | use LAMDA language to specify known attack scenarios |
| Rule-based (case 2)                          | ✓      | ✓      | ✓      | ✓      | manually defines the prerequisites and consequences of each alerts |
| (Ning et al [11,12])                         |        |        |        | hard-coded rules |
| Rule-based (case 3)                          | ✓      | ✓      | ✓      | ✓      | rules specified manually |
| (Valeur et al [28])                          |        |        |        | known attack scenarios defined by rules |
| Rule-based (case 4)                          | ✓      | ✓      | ✓      |                    |
| (Debar et al [29])                           |        |        |        | need to define prerequisites of attacks |
| Rule-based (case 5)                          | ✓      | ✓      | ✓      |                    |
| (Kabir et al [30])                           |        |        |        | use attack sequence pattern mining, correlate alerts based on feature similarity with a weight for each feature, use dozens of parameter whose values are not specified |
| Model-based                                  | ✓      | ✓      | ✓      |                    |
| (Morin et al [31])                           |        |        |        | correlate alerts based on feature similarity with a weight for each feature, cannot reveal causal relationship between alerts |
| Statistical-based                            | ✓      | ✓      | ✓      | ✓      | Time-series based causal analysis problem (Granger Causality Test), no predefined rules, manual validation, external tool for false alert reduction (verification) |
| (Li et al [32])                              |        |        |        | No prior knowledge required, no training data set required, correlation and prediction of multi-stage attacks (graph) with precise transition probabilities, linear complexity |
| Probabilistic-based                          | ✓      | ✓      | ✓      | ✓      |                    |
| (Valdes et al [33])                          |        |        |        |                    |
| Statistical and probability-based            | ✓      | ✓      | ✓      | ✓      |                    |
| (Qin et al [34])                             |        |        |        |                    |
| Our method                                   | ✓      | ✓      | ✓      | ✓      |                    |
detect the identity of the flag on each, and score the
game accordingly.

Defcon has recently published the archive of CtF’17,
which includes all the traffic generated during the game.
The number of intruders is 97 (alert’s IP source) using
more than 32,000 distinct source ports and targeting more
than 24,000 ports within a network that encompasses 29
distinct hosts.

The CtF’17 traffic has been fed to the well-known IDS
called Snort for offline analysis. The traffic generated a
lot of alerts (more than 424,000 alerts), most of them using
Transmission Control Protocol (TCP) connections.

Using the current correlation system, the huge number
of alerts recorded by Snort is correlated into less than 23 attack
graphs when not considering network prefixes aggrega-
tion, and 10 attack graphs with 24 bits as network
prefixes (Figure 9).

Figure 9 also shows that the evolution of the number of
attack graphs according to the network prefixes is roughly
linear (we used the linear interpolation method). This a
good conjunction that should be considered by security
administrators in order to predict the number of attack
graphs for given network prefixes. Note that the parameters
of the deduced linear equation ($y = 11x - 260$) is specific
to CtF. However, the curve expressing the number of at-
tack graphs given a network prefixes should remain linear
independently from the traffic.

Figure 10 shows the number of edges (curve (a)) and
the number of nodes (curve (b)) of the largest attack graph
(green line) and the smallest attack graph (red line) while
changing the network prefixes used for aggregating IP
sources. The curves show that the minimum number of edges
is just one. The curves show also that the maximum number
of edges grows when the network prefixes is low. However,
the maximum number of edges is still small. The node distribu-
tion among attack graphs is similar to the edge distribu-
tion. The maximum number of nodes for the largest attack
graph is about 20, which gives the security administrator a
comfortable way to control the alert generation and react
against the critical intruder, scenario, or alerts.

The graph density describes the general level of connec-
tedness in a graph. It is a good criterion for the evalua-
tion of the readiness of attack graphs. The graph density for
a directed graph is defined as follows:

$$d = \frac{|E|}{|V|(|V| - 1)}$$

In our case, the graph density could be more than one
because loops (edge that starts and ends on the same node)
could exist in the graph.

Figure 11 shows the graph density evolution according
to the number of nodes in each attack graph. For a given
number of nodes, we have chosen the maximum value of
the graph density among all the attack graphs. The curve
shows that the density is always low (less than 0.5) when
the number of nodes exceeds six, which is a good result,
proving that the system scales well; when the number of
nodes increases, the number of edges is still low, which
allows the administrator to follow the intruder’s scenario
easily.
Note that the complexity to determine $\alpha$-near-critical alert nodes and near-critical alert nodes is not high. Most of the running time is spent in the shortest path algorithm. The running time of the Dijkstra's shortest path algorithm is dominated by $O(|V|^2)$.

The interpretation of four examples of an attack graph generated from CtF'17 data set is as follows.

The attack graph in Figure 12(a) is a simple scenario where the intruder (IP: 10.31.10.8) starts by pinging the host with IP address 10.31.1.2 using two Internet Control Message Protocol (ICMP) info request (BSD type: 2100368 and *NIX: 2100366) at the same time. Then, the intruder performs a brute-force attack to log into the MySQL server on port 3306 (alert 2010937). Note that 48 alerts are summarized in a graph with three nodes and three edges. The transition matrix according to the Markov chain attack graph is

$\begin{bmatrix}
2100366 & 2100368 & 48 \\
2100368 & 2100366 & 24 \\
2010937 & 2100368 & 24
\end{bmatrix}$

And the fundamental matrix $N$ is

$\begin{bmatrix}
2100366 & 24 & 24 \\
2100368 & 23 & 24 \\
2010937 & 2100368 & 48
\end{bmatrix}$

Because the initial state of the attack is the alert 2100366, the number of steps of the scenario (from the start to the end) is 47. Note that the alert 2100368 is a critical node.

Another example is shown in Figure 12(b); this scenario summarizes more than 1200 alerts into a graph with three nodes and four edges. The intruder (IP: 10.31.4.123) starts by sending a shell code (alert 17335, x86 OS agnostic FNSTENV get extended instruction pointer (EIP) byte XOR decoder) to the service on port 57005 using an XOR function to encrypt his shell code. Because XOR is reversible, the intruder can simply decode the shell code using the same universal function at the destination machine before executing it. The shell code uses The FNSTENV instruction, which writes a 32-byte floating-point unit (FPU) environment record to the memory address specified by the operand. The 12th byte of the FPU environment record contains the EIP of the last FPU instruction called. The FNSTENV is followed by a No Operation (NOP) instruction (alert 648), which is an innocuous instruction. The NOP instruction is used to perform a NOP sled attack; the FNSTENV will extract the EIP of NOP and place it on the top of the stack. Once the location of the NOP instruction is known, the attacked will adjust it to point to the XOR decoded code (call branch). The consecution of the two instructions is repeated, until the attack is succeeded and the call branch is executed. The decoded code usually creates a shell on a given port used as a back door. The intruder then checks the status of the remote host using a PING (alert 2100366). The transition matrix according to the Markov chain attack graph is

$\begin{bmatrix}
2100366 & 0 & 1 \\
2100368 & 0.96 & 0.04 \\
2010937 & 0 & 0
\end{bmatrix}$
And the fundamental matrix $N$ is

$$N = \begin{bmatrix} 33.33 & 33.33 \\ 32.33 & 33.33 \end{bmatrix}$$

And the fundamental matrix $T$ is

$$T = \begin{bmatrix} 66.66 \\ 65.66 \end{bmatrix}$$

Because the initial state of the attack is the alert 17335, the number of steps of the scenario (from the start to the end) is $66.66 \approx 67$. Note that the alert 648 is a critical node.

Figure 13(a) shows an attack scenario that tries to open a new port by attacking the port 4343 on the machine with IP: 10.31.6.2. The intruder (IP: 10.31.4.151) first sends the Rothenburg Shell code encrypted with the XOR decoder. Actually, the Rothenburg code raises two alerts, 17322 and 2009247. The first alert (17322) is caused by the detection of the FNSTENV get EIP byte XOR decoder for x86 OS. The unxor’d code corresponds to the Rothenburg Shell code (alert 2009247). The Rothenburg code uses the WS2 library in order to create a socket, accepts incoming connections via a specific port and creates a thread for each accepted connection. The scenario ends by sending a NOP operation (alert 648) to sled the execution to future code adjusted according to the NOP instruction location. The intruder will then connect to the new port in order to execute further codes.

The scenario in Figure 13(b) is the result of the aggregation of all alerts that correspond to the same target (IP: 10.31.6.2). This scenario shows that the previous scenario (Figure 13(a)) is repeated twice and followed by thousands of XOR decoded codes (alert 17335). The alert 17335 is not accompanied with other alerts. This means that the codes are regular operations executed on the compromised machine.

The scenario in Figure 13(b) summarizes thousands of alerts in a graph with four nodes and six edges. The duration of the attack is 77129 s, about 21 h, which gives an idea about the high vulnerability of the port.
Thus, eight steps are required to open the remote port and execute regular codes via the port 4343. The alert 2009247 is a critical alert.

### 7. CONCLUSIONS AND FUTURE WORKS

The paper has stated the current distributed IDS architectures and their current challenges. One of the most critical components in IDS is the alert correlation. We introduced a novel approach of alert correlation based on graphs and absorbing Markov chains. The method answers most of the challenges that a correlation system is faced to; the context-based management system ensures the portability of the system and reduces false positive alerts; the correlation system guarantees real-time and scalability properties, and it also recognizes new multi-step attacks. The graph-based correlation approach provides to the administrator a simple, intuitive, and in-depth analysis of the attacks.

The performance results are very encouraging. During a relatively small duration, the system can correlate many thousands of alerts into dozens of attack graphs without ignoring any alert. The system can rank the intruders according to their gravity. It shows also the critical alerts and near-critical alerts, the attack objectives, and the time to reach them using a graphical way.

A number of works is still open for future investigations. Regarding the current work, the definition of the intruder must be improved. The NAT and the border firewall traversal should be taken into account. Second, an optimized graph method for visualizing the correlation results should be proposed. Finally, the current work should be compared with other existing methods using a unique data set and a unique set of benchmark criteria.

Regarding the correlation system in general, a formal model that characterizes both the alert generation and the traffic flow remains not defined. Moreover, the correlation system might be enhanced to react automatically depending on the past, present, and predicted states of the whole organization. Even with a good correlation system, the administrator will not be able to handle the huge amount of alerts, especially for future converged networks and applications. An automatic reaction system would be appreciated by the administrators.

### REFERENCES

3. IDS. http://www.prelude-technologies.com
Research Article

A realistic graph-based alert correlation system

Ouissem Ben Fredj

This paper introduces a graph-based attack description that comes with different analysis methods for alert correlation. The system encompasses an attack scenario detection method, an alert correlation method that recognizes multistep attacks, and a graph-based classification method to extract different properties of the attacks. The performance analysis shows that the system is real time and scalable.