ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Alert Aggregation Agent

Asra Sarwath[*1], Raafiya Gulmeher[2]
[*1]P.G.Student , Department Of Computer Science & Engineering,KBNCE, Gulbarga, Karnataka, India
[2]Assistant Professor , Department of Computer Science & Engineering, KBNCE, Gulbarga, Karnataka, India
asra.sarwath2003@gmail.com

Abstract

Intrusion detection technique is important subtask that aggregates alert. Alert aggregation goal is to identify & to cluster different alert belonging to a specific attack instance which has been initiated by an attacker at a certain point in time. Meta-alerts may then be the basis for reporting to security experts or for communication within a distributed intrusion detection system. Alert aggregation which is based on a dynamic, probabilistic model of the current attack situation, it can be regarded as a data stream version of a maximum likelihood approach for the estimation of the model parameters. Meta-alerts are generated with a delay of typically only a few seconds after observing the first alert belonging to a new attack instance. We make the system more efficient in identifying the intrusion alerts and also we extend this work by sending the Alerts as Message to the Network Administrator who governs the Network or Intrusion Detection System.

Keywords: Intrusion detection, alert aggregation, generative modeling, data stream algorithm.

Introduction

Intrusion detection systems are the `burglar alarms' of the computer security field. IDS usually focus on detecting attack types, but not on distinguishing between different attack instances. In addition, even low rates of false alerts could easily result in a high total number of false alerts if thousands of network packets or log file entries are inspected. As a consequence, the IDS Data Stream Intrusion Alert Aggregation creates many alerts at a low level of abstraction. It is extremely difficult for a human security expert to inspect this flood of alerts, and decisions that follow from single alerts might be wrong with a relatively high probability.

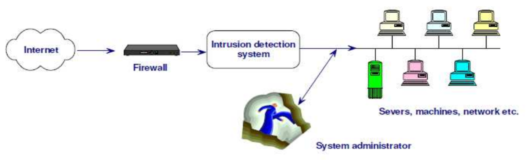


Figure 1. Network protection using conventional IDS

Fig 1: illustrates a simple network, which is protected using IDS.

a “perfect” IDS should be situation-aware[2] in the sense that at any point in time it should “know” what is going on in its environment regarding attack instances (of various types) and attackers. In this paper, we make an important step toward this goal by introducing and evaluating a new technique for alert aggregation. Alerts may originate from low-level IDS such as those mentioned above, from firewalls (FW), etc. Alerts that belong to one attack instance must be clustered together and meta-alerts must be generated for these clusters. The main goal is to reduce the amount of alerts substantially without losing any important information which is necessary to identify ongoing attack instances.

Our approach has the following distinct properties:

1. It is a generative modeling approach [3] using probabilistic methods. Assuming that attack instances can be regarded as random processes “producing” alerts, we aim at modeling these processes using approximate maximum likelihood parameter estimation techniques. Thus, the beginning as well as the completion of attack instances can be detected.
2. It is a data stream approach, i.e., each observed alert is processed only a few times [4]. Thus, it can be applied online and under harsh timing constraints.

Related Work

IDS are optimized to detect attacks with high accuracy. However, they still have various disadvantages that have been outlined in a number of

ISSN: 2277-9655
Impact Factor: 1.852

publications and a lot of work has been done to analyze IDS in order to direct future research (cf. [5], for instance). Besides others, one drawback is the large amount of alerts produced. Recent research focuses on the correlation of alerts from (possibly multiple) IDS. If not stated otherwise, all approaches outlined in the following present either online algorithms or—as we see it—can easily be extended to an online version. Probably, the most comprehensive approach to alert correlation is introduced in [6]. One step in the presented correlation approach is attack thread reconstruction, which can be seen as a kind of attack instance recognition. No clustering algorithm is used, but a strict sorting of alerts within a temporal window of fixed length according to the source, destination, and attack classification (attack type). In [7], a similar approach is used to eliminate duplicates, i.e., alerts that share the same quadruple of source and destination address as well as source and destination port. In addition, alerts are aggregated (online) into predefined clusters (so-called situations) in order to provide a more condensed view of the current attack situation. The definition of such situations is also used in [8] to cluster alerts. In [9], alert clustering is used to group alerts that belong to the same attack occurrence. Even though called clustering, there is no clustering algorithm in a classic sense. The alerts from one (or possibly several) IDS are stored in a relational database and a similarity relation—which is based on expert rules—is used to group similar alerts together. Two alerts are defined to be similar, for instance, if both occur within a fixed time window and their source and target match exactly. As already mentioned, these approaches are likely to fail under real-life conditions with imperfect classifiers (i.e., low-level IDS) with false alerts or wrongly adjusted time windows.

In [15], three different approaches are presented to fuse alerts. The first, quite simple one groups alerts according to their source IP address only. The other two approaches are based on different supervised learning techniques. Besides a basic least-squares error approach, multilayer perceptrons, radial basis function networks, and decision trees are used to decide whether to fuse a new alert with an already existing meta-alert (called scenario) or not. Due to the supervised nature, labeled training data need to be generated which could be quite difficult in case of various attack instances.

An offline clustering solution based on the CURE algorithm is presented. The solution is restricted to numerical attributes. In addition, the number of clusters must be set manually. This is problematic, as in fact it assumes that the security expert has knowledge about the actual number of ongoing attack instances. The alert clustering solution described in [11] is more related to ours. A link-based clustering approach is used to repeatedly fuse alerts into more generalized ones. The intention is to discover the reasons for the existence of the majority of alerts, the so called root causes, and to eliminate them subsequently. An attack instance in our sense can also be seen as a kind of root cause, but in [11] root causes are regarded as “generally persistent” which does not hold for attack instances that occur only within a limited time window. Furthermore, only root causes that are responsible for a majority of alerts are of interest and the attribute-oriented induction algorithm is forced “to find large clusters” as the alert load can thus be reduced at most. Attack instances that result in a small number of alerts (such as PHF or FFB) are likely to be ignored completely. The main difference to our approach is that the algorithm can only be used in an offline setting and is intended to analyze historical alert logs. In contrast, we use an online approach to model the current attack situation. The alert clustering approach described in [12] is based on [11] but aims at reducing the false positive rate. The created cluster structure is used as a filter to reduce the amount of reated alerts. Those alerts that are similar to already known false positives are kept back, whereas alerts that are considered to be legitimate (i.e., dissimilar to all known false positives) are reported and not further aggregated. The same idea—but based on a different offline clustering algorithm—is presented in [21].

Online Alert Aggregation

To outline the preconditions and objectives of alert aggregation, we start with a short sketch of our intrusion framework. Then, we briefly describe the generation of alerts and the alert format. We continue with a new clustering algorithm for offline alert aggregation which is basically a parameter estimation technique for the probabilistic model. After that, we extend this offline method to an algorithm for data stream clustering which can be applied to online alert aggregation. Finally, we make some remarks on the generation of meta-alerts

A.Collaborating Intrusion Detection Agents

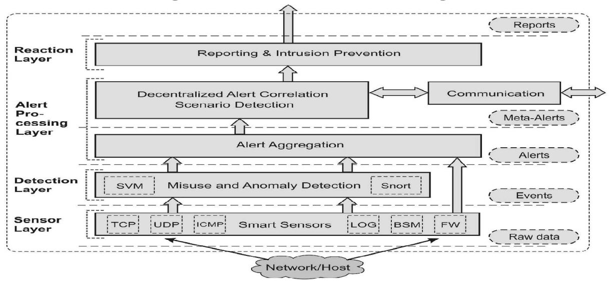


Fig.2 Architecture of an intrusion detection agent.

Fig. 2 outlines the layered architecture of an ID agent:The sensor layer provides the interface to the network and the host on which the agent resides. Sensors acquire raw data from both the network and the host, filter incoming data, and extract interesting and potentially valuable (e.g., statistical) information which is needed to construct an appropriate event. At the detection layer, different detectors, e.g., classifiers trained with machine learning techniques such as support vector machines (SVM) or conventional rule-based systems such as Snort [24], assess these events and search for known attack signatures (misuse detection) and suspicious behavior (anomaly detection). In case of attack suspicion, they create alerts which are then forwarded to the alert processing layer. Alerts may also be produced by FW or the like. At the alert processing layer, the alert aggregation module has to combine alerts that are assumed to belong to a specific attack instance. Thus, so called meta-alerts are generated. Meta-alerts are used or enhanced in various ways, e.g., scenario detection or decentralized alert correlation. An important task of the reaction layer is reporting.

The overall architecture of the distributed intrusion detection system and a framework for large-scale simulations are described in [25], [26] in more detail.

In our layered ID agent architecture, each layer assesses, filters, and/or aggregates information produced by a lower layer. Thus, relevant information gets more and more condensed and certain, and, therefore, also more valuable. We aim at realizing each layer in a way such that the recall of the applied techniques is very high, possibly at the cost of a slightly poorer precision [27].

B. Alert Generation and Format

In this section, we make some comments on the information contained in alerts, the objects that must be aggregated, and on their format. At the sensor layer,sensors determine the values of attributes that are used as input for the detectors as well as for the alert clustering module. Attributes in an event that are independent of a particular attack instance can be used for classification at the detection layer. Attributes that are (or might be) dependent on the attack instance can be used in an alert aggregation process to distinguish different attack instances. A perfect partition into dependent and independent attributes, however, cannot be made. Some are clearly dependent (such as the source IP address which can identify the attacker), some are clearly independent such as the destination port which usually is 80 in case of web based attacks), and lots are both (such as the destination port which can be a hint to the attacker's actual target service as well as an attack tool specifically designed to target a particular service only). In addition to the attributes produced by the sensors, alert aggregation is based on additional attributes generated by the detectors. Examples are the estimated type of the attack instance that led to the generation of the alert (e.g., SQL injection, buffer overflow, or denial of service), and the degree of uncertainty associated with that estimate.

C. Offline Alert Aggregation

In this section, we introduce an offline algorithm for alert aggregation which will be extended to a data stream algorithm for online aggregation in Assume that a host with an ID agent is exposed to a certain intrusion situation. One or several attackers launch several attack instances belonging to various attack types. The attack instances each cause a number of alerts with various attribute values. The task of the alert aggregation module is now to estimate the assignment to instances by using the unlabeled observations only and by analyzing the cluster structure in the attribute space. That is, it has to reconstruct the attack situation. Then, meta-alerts can be generated that are basically an abstract description of the cluster of alerts assumed to originate from one attack instance. Thus, the amount of data is reduced substantially without losing important information.There may be different potentially problematic situations:

1. False alerts are not recognized as such and wrongly assigned to clusters: This situation is acceptable as long as the number of false alerts is comparably low.
2. True alerts are wrongly assigned to clusters: This situation is not really problematic as long as the majority of alerts belonging to that cluster is correctly assigned. Then, no attack instance is missed.
3. Clusters are wrongly split: This situation is undesired but clearly unproblematic as it leads to redundant meta-alerts only. Only the data reduction rate is lower, no attack instance is missed.
4. Several clusters are wrongly combined into one:This situation is definitely problematic as attack instances may be missed.

EM(Expectation Maximization) procedure for our attack situation model is shown in Algorithm 1. It iteratively maximizes the likelihood with two alternating computation steps: E (expectation) and M maximization). The E step assigns the alerts to components—resulting in a partition of the set A with J clusters—and the M step optimizes the parameters of the mixture model.

Some additional remarks must be made:

Initialization of model parameters: The aim of the

ISSN: 2277-9655
Impact Factor: 1.852

initialization is to find good initial values. Instead of using a random initialization which results in higher runtimes and sub-optimal solutions, we use a heuristic which we have successfully applied to the training of radial basis function neural networks [31].

Hard assignment of alerts to components: More general EM algorithms make a gradual assignment of alerts to components in the E step (cf. responsibilities in [3]). In practical applications, a hard assignment reduces the runtimes significantly at the cost of slightly worse solutions in some situations. In our case, this is acceptable as we do not want to find the optimal model parameters at the end, but to generate the optimal set of meta-alerts.

Stopping criterion. An EM algorithm guarantees that the set of parameters is improved in each step. In addition, due to the hard assignment of alerts, there exists a limited number of possible assignments. For the sake of simplicity, however, we usually run the algorithm for a fixed number of iterations.

Fixed mixing coefficients. One of the main difficulties in alert aggregation is the wide range of possible cluster sizes. There are clusters that contain thousands of alerts, but there are also clusters that consist of a few alerts only. For instance, a Neptune attack instance may result in 200,000 alerts whereas a PHF attack instance may consist of only five alerts [32]. Thus, in contrast to a more general EM approach, it is important to fix the mixing coefficients.Otherwise, if the mixing coefficients were estimated from the observed samples, the EM algorithm would focus on the optimization of the parameters of "heavy" components while neglecting the "light" ones.

D. Data Stream Alert Aggregation

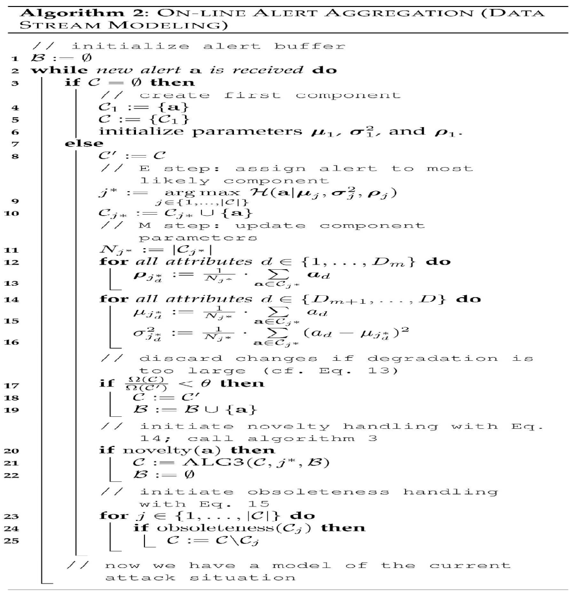
In this section, we describe how the offline approach is extended to an online approach working for dynamic attack situations.

Assume that in the environment observed by an ID agent attackers initiate new attack instances that cause alerts for a certain time interval until this attack instance is completed.

1. Component adaption: Alerts associated with already recognized attack instances must be identified as such and assigned to already existing clusters while adapting the respective component parameters.

2. Component creation: The occurrence of new attack instances must be stated. New components must be parameterized accordingly.

3. Component deletion: The completion of attack instances must be detected and the respective components must be deleted from the mode



Algorithm 2: ON-LINE ALERT AGGREGATION (DATA STREAM MODELING)

```
// initialize alert buffer
$\mathcal{B} := \emptyset$
while new alert a is received do
    if $\mathcal{C} = \emptyset$ then
        // create first component
        $\mathcal{C}_1 := \{\mathbf{a}\}$
        $\mathcal{C} := \{\mathcal{C}_1\}$
        initialize parameters $\mu_1$, $\sigma_1^2$, and $\rho_1$.
    else
        $\mathcal{C}' := \mathcal{C}$
        // E step: assign alert to most likely component
        $j^* := \arg\max_{j \in \{1,\ldots,|\mathcal{C}|\}} \mathcal{H}(\mathbf{a}|\mu_j, \sigma_j^2, \rho_j)$
        $\mathcal{C}_{j*} := \mathcal{C}_{j*} \cup \{\mathbf{a}\}$
        // M step: update component parameters
        $N_{j^*} := |\mathcal{C}_{j^*}|$
        for all attributes $d \in \{1, \ldots, D_m\}$ do
            $\rho_{j_d^*} := \frac{1}{N_{j^*}} \cdot \sum_{\mathbf{a} \in \mathcal{C}_{j*}} a_d$
        for all attributes $d \in \{D_{m+1}, \ldots, D\}$ do
            $\mu_{j_d^*} := \frac{1}{N_{j*}} \cdot \sum_{\mathbf{a} \in \mathcal{C}_{j*}} a_d$
            $\sigma_{j_d^*}^2 := \frac{1}{N_{j*}} \cdot \sum_{\mathbf{a} \in \mathcal{C}_{j*}} (a_d - \mu_{j_d^*})^2$
        // discard changes if degradation is too large (cf. Eq. 13)
        if $\frac{\Omega(\mathcal{C})}{\Omega(\mathcal{C}')} < \theta$ then
            $\mathcal{C} := \mathcal{C}'$
            $\mathcal{B} := \mathcal{B} \cup \{\mathbf{a}\}$
        // initiate novelty handling with Eq. 14; call algorithm 3
        if novelty(a) then
            $\mathcal{C} :=$ ALG3$(\mathcal{C}, j^*, \mathcal{B})$
            $\mathcal{B} := \emptyset$
        // initiate obsoleteness handling with Eq. 15
        for $j \in \{1, \ldots, |\mathcal{C}|\}$ do
            if obsoleteness$(\mathcal{C}_j)$ then
                $\mathcal{C} := \mathcal{C} \backslash \mathcal{C}_j$
    // now we have a model of the current attack situation
```

Algorithm 2 describes the online alert aggregation. If a new alert is observed we first have to decide whether a first component has to be created. In this case, we initialize its parameters with information taken from this alert. Random, small values are added, for example, to prevent any subsequent optimization steps from running into singularities of the respective likelihood function [3]. Otherwise, we have to decide whether the alert has to be associated with an existing component or not, i.e., whether we believe that it belongs to an ongoing attack instance or not. Provisionally, we assign the alert to the most likely component (E step) and optimize the parameters of this component (M step).

This procedure is initiated either when the temporal spread of the buffer content is too large or when the content is no longer homogeneous in the sense that we assume that another new attack instance may have been initiated:

1. Temporal spread: As the rate of incoming alerts depends on the current attack situation, it changes heavily over time ranging from thousands of alerts per minute to only a few alerts per hour. Thus, to keep the response time short, we have to take into account the temporal spread of the buffer content.

2. Homogeneity: The goal is to ensure that only alerts that are similar to each other are stored in the buffer. Thus, it is possible that

the novelty handling conducts—for temporal performance reasons.

```
Algorithm 3: COMPONENT CREATION IN CASE OF DETECTED NOVELTY
Input : partition $\mathcal{C}$, specific cluster number $j^*$, buffer $\mathcal{B}$
Output: updated partition $\mathcal{C}$
1 $\mathcal{C}' := \mathcal{C} \backslash \mathcal{C}_{j^*}$
  // test several component numbers
2 for $k$=1 to $K$ do
      // conduct off-line clustering with
         algorithm 1 for buffer and most
         likely component
3     $\mathcal{C}^{(k)} := \mathrm{ALG1}(\mathcal{C}_{j^*} \cup \mathcal{B}, k)$
      // assess result with Eq. 13
4     $\Omega^{(k)} := \Omega(\mathcal{C}' \cup \mathcal{C}^{(k)})$
  // determine number of components
5 $k^* := \arg\max_{k \in \{1,\dots,K\}} \Omega^{(k)}$
  // update model
6 $\mathcal{C} := \mathcal{C}' \cup \mathcal{C}^{(k^*)}$
```

In order to reduce the runtime of this algorithm further,we may reduce the number of alerts that have to be processed by means of an appropriate subsampling.

Algorithm 3 describes the novelty handling itself. Basically, to adapt the overall model, we run the offline aggregation algorithm several times with different possible component numbers to chose the optimal number. However, due to the homogeneity of the buffer, we may]

E. Meta-Alert Generation and Format

With the creation of a new component, an appropriate metaalert that represents the information about the component in an abstract way is created. Every time a new alert is added to a component, the corresponding meta-alert is updated incrementally, too. That is, the meta-alert "evolves" with the component. Meta-alerts may be the basis for a whole set further tasks . Sequences of meta-alerts may be investigated further in order to detect more complex attack scenarios(e.g., by means of hidden Markov models).Meta-alerts may be exchanged with other ID agents in order to detect distributed attacks such as one-to many attacks. . Based on the information stored in the meta-alerts, reports may be generated to inform a human security expert about the ongoing attack situation. Meta-alerts could be used at various points in time from the initial creation until the deletion of the corresponding component (or even later). For instance, reports could be created immediately after the creation of the component or—which could be more preferable in some cases—a sequence of updated reports could be created in regular time intervals. Another example is the exchange of metaalerts between ID agents: Due to high communication costs,meta-alerts could be exchanged based on the evaluation of their interestingness.

Experimental Results

In the following, the results for the alert aggregation are presented. For all experiments, the same parameter settings are used. We set the threshold _ that decides whether to add a new alert to an existing component or not to five percent, and the value for the threshold _ that specifies the allowed temporal spread of the alert buffer to 180 seconds. _ was set that low value in order to ensure that even a quite small degrade of the cluster quality, which could indicate a new attack instance, results in a new component.

First of all, it must be stated there is an operation point of the SVM at the detection layer (OP 1) where we do not miss any attack instances at all (at least in addition to those already missed at the detection layer). The reduction rate is with 99.87 percent extremely high, and the detection delay is only 5.41 s in the worst case (d100%). Average and worst case runtimes are very good, too. All OP will now be analyzed in much more detail. All attack instances for which the detector produces at least a single alert are detected in the idealized case and with OP 1 and OP 2. Choosing another OP, the rate of detected instances drops to 98.04 percent (OP 3) and 99.02 percent (OP 4). In OP 3, a FORMAT instance and a MULTIHOP instance are missed. In OP 4, only the FORMAT instance could not be detected. A further analysis identified the following reasons:

The main reason in the case of the FORMAT instance is the small number of only four alerts. Those alerts are created by the detector layer for all OP, i.e., there is obviously no benefit from choosing an OP with higher FPR. By increasing the FPR, the true FORMAT alerts are erroneously merged with false alerts into one cluster. Hence, as the false alerts easily outnumber the four true FORMAT alerts within this cluster, the FORMAT instance gets lost. . For the MULTIHOP instance, for which we have 19 alerts, the situation is more complex. The instance is only missed in OP 3 and not in OP 4. In OP 3, the downside of a higher FPR outweighs the benefit of a higher TPR—the MULTIHOP alerts are merged with a large number of false alerts. Further increasing the FPR (OP 4) leads to more false alerts as well, but, in this case, also to a further split of clusters such that the false alerts and the MULTIHOP alerts are placed into separate clusters. Next, we analyze the number of meta-alerts MA and the reduction rate r. In the idealized case, 324 meta-alerts are created. Compared to the about 1.6 million alerts, we get a reduction rate of 99.98 percent, which is a reduction of almost three orders of magnitude. Unfortunately, with exception of the first of seven weeks, it was not possible to achieve the ideal case with exactly one meta-alert for every attack instance. Basically, there are four reasons:

Distinguishable steps of an attack type: Often, a split of attack instances into more meta-alerts is caused by the nature of the attacks themselves. Actually, many attack types consist of different, clearly distinguishable steps. As an example, the FTP-WRITE attack exhibits three such steps: an FTP login on port 21, an FTP data transfer on port 20, and a remote login on port 513. Thus, a split into three related meta-alerts is quite natural. Subsequent tasks at the alert processing layer are supposed to handle such multistep attack scenarios (cf. Fig. 1).

Several independent attackers: In the DARPA data set, some attack instances are labeled as a single attack instance although they are in fact comprised of the actions of several independent attackers.

Long attack duration: Attack instances with a long duration are often split into several meta-alerts. Typical examples are slow or hidden port scans or (distributed) denial of service attacks which can last several hours.

Bidirectionalcommunication:TCP/IP-based communication between two hosts results in packets transmitted in both directions. If the detector layer produces alerts for both directions (e.g., due to malicious packets), the source and destination IP address are swapped, which in the end results in two meta-alerts. This problem could be solved with an appropriate preprocessing step.

Conclusion

The experiments demonstrated the broad applicability of the proposed online alert aggregation approach. We analyzed three different data sets and showed that machine-learning-based detectors, conventional signaturebased detectors, and even firewalls can be used as alert generators. In all cases, the amount of data could be reduced substantially.especially clusters that are wrongly split—the instance detection rate is very high: None or only very few attack instances were missed. Runtime and component creation delay are well suited for an online application.

References

[1] S. Axelsson, "Intrusion Detection Systems: A Survey and Taxonomy," Technical Report 99-15, Dept. of Computer Eng., Chalmers Univ. of Technology, 2000.

[2] M.R. Endsley, "Theoretical Underpinnings of Situation Awareness: A Critical Review," Situation Awareness Analysis and Measurement, M.R. Endsley and D.J. Garland, eds., chapter 1, pp. 3-32, Lawrence Erlbaum

[3] C.M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[4] M.R. Henzinger, P. Raghavan, and S. Rajagopalan, Computing on Data Streams. Am. Math. Soc., 1999.

[5] A. Allen, "Intrusion Detection Systems: Perspective," Technical Report DPRO-95367, Gartner, Inc., 2003.

[6] F. Valeur, G. Vigna, C. Kru¨ gel, and R.A. Kemmerer, "A Comprehensive Approach to Intrusion Detection Alert Correlation," IEEE Trans. Dependable and Secure Computing, vol. 1, no. 3, pp. 146-169, July-Sept. 2004.

[7] H. Debar and A. Wespi, "Aggregation and Correlation of Intrusion-Detection Alerts," Recent Advances in Intrusion Detection, W. Lee, L. Me, and A. Wespi, eds., pp. 85-103, Springer, 2001.

[8] D. Li, Z. Li, and J. Ma, "Processing Intrusion Detection Alerts in Large-Scale Network," Proc. Int'l Symp. Electronic Commerce and Security, pp. 545-548, 2008.

[9] F. Cuppens, "Managing Alerts in a Multi-Intrusion Detection Environment," Proc. 17th Ann. Computer Security Applications Conf. (ACSAC '01), pp. 22-31, 2001.

[10]A. Valdes and K. Skinner, "Probabilistic Alert Correlation," Recent Advances in Intrusion Detection, W. Lee, L. Me, and A. Wespi, eds. pp. 54-68, Springer, 2001.

[11]K. Julisch, "Using Root Cause Analysis to Handle Intrusion Detection Alarms," PhD dissertation, Universita¨ t Dortmund, 2003.

[12]T. Pietraszek, "Alert Classification to Reduce False Positives in Intrusion Detection," PhD dissertation, Universita¨ t Freiburg, 2006.

[13]F. Autrel and F. Cuppens, "Using an Intrusion Detection Alert Similarity Operator to Aggregate and Fuse Alerts," Proc. Fourth Conf. Security and Network Architectures, pp. 312-322, 2005

[14] G. Giacinto, R. Perdisci, and F. Roli, "Alarm Clustering for Intrusion Detection Systems in Computer Networks," Machine Learning and Data Mining in Pattern Recognition, P. Perner and A. Imiya, eds. pp. 184-193, Springer, 2005.

[15] O. Dain and R. Cunningham, "Fusing a Heterogeneous Alert Stream into Scenarios," Proc. 2001 ACM Workshop Data Mining for Security Applications, pp. 1-13, 2001.