www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2. Issue 10 October 2013 Page No. 3051-3060

Generating Meta Alert with Intrusion Framework on IDS

Sivaramaiah.Y [1] * ***S.Nagarjuna Reddy*** [2] *

[1] M.Tech, Department of CSE

[2] Associate Professor, Department of CSE

Lakireddy Balireddy College of Engineering (Autonomous), Mylavaram , Affiliated to JNTUK, Kakinada

sivaramaiah999@gmail.com* ;+91 8099999559.

ABSTRACT:

Alert aggregation is an important subtask of intrusion detection. The goal is to identify and to cluster different alerts—produced by low-level intrusion detection systems, firewalls, etc. belonging to a specific attack instance which has been initiated by an attacker at a certain point in time. Thus, meta-alerts can be generated for the clusters that contain all the relevant information whereas the amount of data (i.e., alerts) can be reduced substantially. Meta-alerts may then be the basis for reporting to security experts or for communication within a distributed intrusion detection system. We propose a novel technique for online alert aggregation which is based on a dynamic, probabilistic model of the current attack situation. Basically, it can be regarded as a data stream version of a maximum likelihood approach for the estimation of the model parameters. With three benchmark data sets, we demonstrate that it is possible to achieve reduction rates of up to 99.96 percent while the number of missing meta-alerts is extremely low. In using simulation of mobile device intrusion will attack to just display on the device. In addition, meta-alerts are generated with a delay of typically only a few seconds after observing the first alert belonging to a new attack instance.

KEY WORDS:

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

INTRODUCTION:

Intrusion detection systems (IDS) are besides other protective measures such as virtual private networks, authentication mechanisms, or encryption techniques very important to guarantee information security. They help to defend against the various threats to which networks and hosts are exposed to by detecting the actions of attackers or attack tools in a network or host-based manner with misuse or anomaly detection techniques.

At present, most IDS are quite reliable in detecting suspicious actions by evaluating TCP/IP connections or log files, for instance. Once an IDS finds a suspicious action, it immediately creates an alert which contains information about the source, target, and estimated type of the attack(e.g., SQL injection, buffer overflow, or denial of service). As
the intrusive actions caused by a single attack instance—which is the occurrence of an attack of a particular type that has been launched by a specific attacker at a certain point in time are often spread over many network connections or log file entries, a single attack instance often results in hundreds or even thousands of alerts. 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 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.

In our opinion, 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 oneattack 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 on going attack instances. We want to have no missing meta alerts, but in turn we accept false or redundant meta-alerts to a certain degree. This problem is not new, but current solutions are typically based on a quite simple sorting of alerts, e.g., according to their source, destination, and attack

type. Under real conditions such as the presence of classification
errors of the low-level IDS (e.g., false alerts), uncertainty with respect to the source of the attack due to spoofed IP addresses, or wrongly adjusted time windows, for instance, such an approach fails quite often.
Our approach has the following distinct properties:

- 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 approximative. maximum likelihood parameter estimation techniques. Thus, the beginning as well as the completion of attack instances can be detected.
- 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.

  The remainder of this paper is organized as follows: In Section 2 some related work is presented. Section 3 describes the proposed alert aggregation approach, and Section 4 provides experimental results for the alert aggregation using various data sets. Finally, Section 5 summarizes the major findings.

RELATED WORK:

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

Another approach to alert correlation is presented .A weighted, attribute-wise similarity operator is used to decide whether to fuse two alerts or not. However, as already stated in and , this approach suffers from the high number of parameters that need to be set. The similarity operator presented in [13] has the same disadvantage there are lots of parameters that must be set by the user and there is no or only little guidance in order to find good values. In , another clustering algorithm that is based on attribute-wise similarity measures with user defined parameters is presented. However, a closer look at the parameter setting reveals that the similarity measure, in fact, degenerates to a strict sorting according to the source and destination IP addresses and ports of the 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 perceptions, 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. the learned model parameters.
The same or quite similar techniques as described so far are also applied in many other approaches to alert correlation, especially in the field of intrusion scenario detection. Prominent research in scenario detection.

for example. More details can be found in 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 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 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 is based on but aims at reducing the false positive rate. The created cluster structure is used as a filter to reduce the amount of created 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.

A completely different clustering approach is presented in There, the reconstruction error of an auto associate neural network (AA-NN) is used to distinguish different types of alerts. Alerts that yield the same (or a similar) reconstruction error are put into the same cluster. The approach can be applied online, but an offline training phase and training data are needed to train the AA-NN and also to manually adjust intervals for the reconstruction error that
Determine which alerts are clustered together. In addition, it turned out that due to the dimensionality reduction by the AA-NN, alerts of different types can have the same reconstruction error which leads to erroneous clustering. In our prior work, we applied the well-known c-means Clustering algorithm in order to identify attack instances. However, this algorithm also works in a purely offline manner.

.

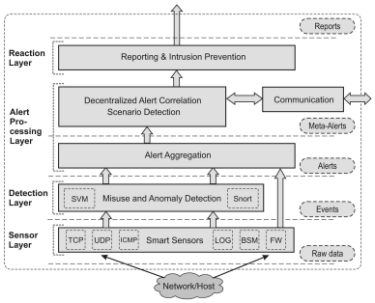

Fig. 1. Architecture of an intrusion detection agent.

A NOVEL ONLINE ALERT AGGREGATION TECHNIQUE

In this section, we describe our new alert aggregation approach which is at each point in time based on a probabilistic model of the current situation. To outline the Pre conditions 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.

Algorithm 1: EXPECTATION MAXIMIZATION ALGORITHM FOR OFF-LINE ALERT AGGREGATION

Input : set of alerts $\mathcal{A}$, number of components J
Output: optimized model parameters $\boldsymbol{\mu}_j$, $\boldsymbol{\sigma}_j^2$, $\boldsymbol{\rho}_j$, assignment of alerts to components

1 $\pi_j := \frac{1}{J}$
2 initialize the remaining model parameters
3 **while** *stopping criterion is not fulfilled* **do**
  // E step: assign alerts to components
4 **for** *all alerts* $\mathbf{a}^{(n)} \in \mathcal{A}$ **do**
5 $j^* := \underset{j \in \{1,\ldots,J\}}{\operatorname{argmax}} \mathcal{H}(\mathbf{a}^{(n)} | \boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2, \boldsymbol{\rho}_j)$
6 assign alert $\mathbf{a}^{(n)}$ to component j^*
  // M step: update model parameters
7 **for** *all components* $j \in \{1, \ldots, J\}$ **do**
8 $N_j :=$ number of alerts assigned to j
9 **for** *all attributes* $d \in \{1, \ldots, D_m\}$ **do**
10 $\boldsymbol{\rho}_{j_d} := \frac{1}{N_j} \cdot \sum_{\mathbf{a}^{(n)} \text{ assigned to } j} \boldsymbol{a}_d^{(n)}$
11 **for** *all attributes* $d \in \{D_m + 1, \ldots, D\}$ **do**
12 $\mu_{j_d} := \frac{1}{N_j} \cdot \sum_{\mathbf{a}^{(n)} \text{ assigned to } j} a_d^{(n)}$
13 $\sigma_{j_d}^2 := \frac{1}{N_j} \cdot \sum_{\mathbf{a}^{(n)} \text{ assigned to } j} (a_d^{(n)} - \mu_{j_d})^2$

Some additional remarks must be made:
Initialization of model parameters. The aim of the 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 . This heuristic selects a initial cluster centers a set of alerts with a large spread in the attribute space.
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. Thus, it can be guaranteed that the algorithm converges. 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 . Thus, in contrast to a more general EM approach, it is important to fix the mixing coefficients (here: $\pi_j = \frac{1}{J}$).

Otherwise, if the mixing coefficients were estimated from the observed samples, the EM algorithm would focus on the One important task has not been mentioned so far: The estimation of the number of components (clusters) J from the set of observed samples A. Up to now, we assumed that this number is given. In our case, the number of clusters shall correspond to the number of attack instances in an ideal case. To solve this problem, cluster validation measures can be used
to assess the quality of a clustering result. A clustering algorithm is started several times with a varying number of clusters, the quality for each clustering result is assessed numerically, and finally the results are compared and the
number that optimizes this measure is chosen (so-called relative criterion). Examples The idea that forms the basis of those measures is that they search for a clustering result that consists of clusters that are both, compact and separable. Although the usability of such measures has been proven in many (offline) application examples, they cannot be applied in a data streaming case as they suffer from high runtimes, e.g., $O(N^2)$ in the case of the Dunn index. We need a measure that can be updated incrementally when a new alert arrives. Therefore, we chose a density-based approach to estimate separability and compactness.
As a final result of the EM algorithm, we not only know the optimized parameter values of the model, but also the final assignment of alerts to components. These clusters define a partition of the sample set. First, we define the compactness of a single cluster associated with a component j by

$$\Gamma_j = \min_{\mathbf{a}^{(n)}\text{assigned to } j} \mathcal{H}\left(\mathbf{a}^{(n)}|\boldsymbol{\mu}_j, \boldsymbol{\sigma}_j^2, \boldsymbol{\rho}_j\right). \quad (8)$$

Then, the compactness of the overall partition is

$$\Gamma = \min_{j\in\{1,\ldots,J\}} \Gamma_j. \quad (9)$$

That means, the compactness is influenced by the alert with the lowest likelihood. As a consequence, we get a bias toward a higher number of components, which is desirable
in our application (high recall required, lower precision accepted). Note that we can easily update this measure in O(1) if a new alert is assigned to a component. Second, we define the separability of a pair of clusters associated with component i and j by

$$\Sigma(i,j) = \frac{1}{2}\left(\frac{\mathcal{H}\left(\mathbf{a}_i^{(*)}|\boldsymbol{\mu}_j, \sigma_j^2, \rho_j\right)}{\mathcal{H}\left(\mathbf{a}_j^{(*)}|\boldsymbol{\mu}_j, \sigma_j^2, \rho_j\right)} + \frac{\mathcal{H}\left(\mathbf{a}_j^{(*)}|\boldsymbol{\mu}_i, \sigma_i^2, \rho_i\right)}{\mathcal{H}\left(\mathbf{a}_i^{(*)}|\boldsymbol{\mu}_i, \sigma_i^2, \rho_i\right)}\right), \quad (10)$$

where we use the alerts with the highest likelihoods

$$\mathbf{a}_h^{(*)} = \arg\max_{\mathbf{a}^{(n)}\text{assigned to } h} \mathcal{H}\left(\mathbf{a}^{(n)}|\boldsymbol{\mu}_h, \sigma_h^2, \rho_h\right). \quad (11)$$

Then, the separability of the overall partition is

$$\Sigma = \min_{i,j\in\{1,\ldots,J\}, i\neq j} \Sigma(i,j). \quad (12)$$

An incremental version of this measure with O(1) time complexity is also straightforward.
Finally, the overall cluster validation measure Ω for a clustering result is

$$\Omega = \frac{\Gamma}{\Sigma}. \quad (13)$$

We search for a number J that maximizes this measure. In Data Stream Alert Aggregation, it will become clear that only a small number of choices for J must actually be tested by our online aggregation algorithm.

Data Stream Alert Aggregation

optimization of the parameters of "heavy" components while neglecting the "light" ones.

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. Thus, at any point in time the ID agent—which is assumed to have a model of the current situation, cf. Fig. 3a—has
several tasks, cf. Fig. 3b:
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 (novelty detection): The occurrence of new attack instances must be stated. New components must be parameterized accordingly.
3. Component deletion (obsoleteness detection): The completion of attack instances must be detected and the respective components must be deleted from the model.
That is, the ID agent must be situation-aware and try to keep his model of the current attack situation permanently up to date, see Fig. 3c.
Clearly, there is a trade-off between runtime (or reaction time) and accuracy. For example, it is hardly possible to decide upon the existence of a new attack instance when only one observation is made. From the viewpoint of our objectives (cf. Section 3.1), the tasks 1 and 2 are more time critical than task 3.
From a probabilistic viewpoint we can state that our overall random process is non stationary in a certain sense which can be regarded as being equivalent to changing the mixing coefficients at certain points in time. A mixing coefficient is either zero or the reciprocal of the number of
"active" components (for the time interval of the respective attack instance). With appropriate novelty and obsoleteness detection mechanisms, we aim at detecting these points in time with both sufficient certainty and timeliness. The offline aggregation algorithm provides the basic idea for the online case, too. We have to consider, however, that we want to have a data stream algorithm with short runtimes. In the following, we do not distinguish between the following two views on solutions of the modeling problem: The probabilistic model with J components with parameters μ_j, α^2_j, Ƥj and j €{1,…,J} and mixing coefficients 1/J , and the partition of the overall set of alerts into a set of clusters C = $\{C_1\ldots,C_j\}$ That is, we equate the
terms partition and model as well as cluster and component, respectively. For online clustering, we require that each alert a has a time stamp ts (A) (time of observation) and that we always know the current point in time t.
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 . 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). For the reason of temporal efficiency, we do not conduct a sequence of E and M steps for the overall model. it turned out that our main goal not to miss any attack instances, Collaborating Intrusion Detection Agents can be achieved this way with substantially lower runtimes but at the cost of some redundant meta-alerts (due to split of clusters, Offline Alert Aggregation).

The assignment of the alert to an existing component is not accepted in any case, only if the quality of the model increases or does not decrease too much, e.g., not more than 15 percent (realized by means of threshold _). Otherwise, we consider the possibility that a new attack instance may have been initiated, do not modify the model, and temporarily store the alert in a buffer until this belief is corroborated by additional observations.

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

```
// initialize alert buffer
1  B := ∅
2  while new alert a is received do
3      if C = ∅ then
           // create first component
4          C_1 := {a}
5          C := {C_1}
6          initialize parameters μ_1, σ_1^2, and ρ_1.
7      else
8          C' := C
           // E step: assign alert to most
              likely component
           j* := argmax_{j∈{1,...,|C|}} H(a|μ_j, σ_j^2, ρ_j)
9          C_j* := C_j* ∪ {a}
10         // M step: update component
              parameters
11         N_j* := |C_j*|
12         for all attributes d ∈ {1, ..., D_m} do
13             ρ_{j_d^*} := (1/N_j*) · Σ_{a∈C_j*} a_d
14         for all attributes d ∈ {D_{m+1}, ..., D} do
15             μ_{j_d^*} := (1/N_j*) · Σ_{a∈C_j*} a_d
16             σ^2_{j_d^*} := (1/N_j*) · Σ_{a∈C_j*} (a_d − μ_{j_d^*})^2
           // discard changes if degradation is
              too large (cf. Eq. 13)
17         if Ω(C)/Ω(C') < θ then
18             C := C'
19             B := B ∪ {a}
           // initiate novelty handling with Eq.
              14; call algorithm 3
20         if novelty(a) then
21             C := ALG3(C, j*, B)
22             B := ∅
           // initiate obsoleteness handling
              with Eq. 15
23         for j ∈ {1, ..., |C|} do
24             if obsoleteness(C_j) then
25                 C := C\C_j
       // now we have a model of the current
          attack situation
```

Novelty handling basically means that we empty the buffer by assigning its content either to existing components or to new components. 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:

. 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. The spread is defined as the difference between the time stamps of the oldest and the most recent alert in the buffer.

. 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 only a local optimization of the model

(see below). We regard the alerts in the buffer as being similar if they all have the same most likely component. That is, to initiate the novelty handling for a given model C and a buffer B, we use the function

$$\text{novelty}(\mathbf{a}) = \begin{cases} \text{true}, & \text{if } \left(\max\limits_{\mathbf{b}\in\mathcal{B}} \{\text{ts}(\mathbf{a}) - \text{ts}(\mathbf{b})\} \geq \gamma \right) \\ & \vee \left(\operatorname{arg\,max}\limits_{j\in\{1,\ldots,|\mathcal{C}|\}} \mathcal{H}(\mathbf{a}|\boldsymbol{\mu}_j, \sigma_j^2, \rho_j) \right. \\ & \neq \operatorname{arg\,max}\limits_{j\in\{1,\ldots,|\mathcal{C}|\}} \mathcal{H}(\mathbf{b}|\boldsymbol{\mu}_j, \sigma_j^2, \rho_j) \\ & \left. \text{for any } \mathbf{b} \in \mathcal{B} \right), \\ \text{false}, & \text{otherwise.} \end{cases} \quad (14)$$

with a user-defined threshold ∂ to state novelty. Note, that spread and homogeneity can both be checked with low temporal effort.

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 restrict the optimization to the alerts in the buffer and in one "neighbor" cluster on the one hand and a relatively small user-defined maximum number of components K on the

other without violating our main goal . The result of this local optimization is finally fused with the unmodified

parts of the model.

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 $\mathcal{C}^{(k)} := \text{ALG1}(\mathcal{C}_{j^*} \cup \mathcal{B}, k)$

// assess result with Eq. 13

4 $\Omega^{(k)} := \Omega(\mathcal{C}' \cup \mathcal{C}^{(k)})$

// determine number of components

5 $k^* := \operatorname{arg\,max}\limits_{k\in\{1,\ldots,K\}} \Omega^{(k)}$

// 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 technique.

To initiate the obsoleteness handling for a given model C and component Cj, we use

$$\text{obsoleteness}(\mathcal{C}_j) = \begin{cases} \text{true}, & \text{if } t \geq \max\limits_{\mathbf{c}\in\mathcal{C}_j} \{\text{ts}(\mathbf{c})\} + \delta, \\ \text{false}, & \text{otherwise,} \end{cases} \quad (15)$$

where ∂ is a convex combination of a multiple of the mean interarrival time between the alerts in the cluster and prior knowledge which is faded out when the number of alerts in this cluster gets larger. That is, we use a kind of estimate of the arrival time of the next alert belonging to the corresponding attack instance. If obsoleteness is stated, the corresponding

component is simply deleted from the model.

Note that the mixing coefficients are always defined implicitly by the number of currently existing components.

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 (cf. Fig. 1):

. 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-tomany 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

According to the task for which meta-alerts are used, they may contain different attributes. Examples for those attributes are aggregated alert attributes (e.g., lists or intervals of source addresses or targeted service ports, or a time interval that marks the beginning and the end if available of the attack instance), attributes extracted from the probabilistic model (e.g., the distribution parameters or the number of alerts assigned to the component), an aggregated alert assessment provided by the detection layer (e.g., the attack type classification or the classification confidence), and also information about the current attack situation (e.g., the number of recent attacks of the same or a similar type, links to attacks originating from the same or a similar source).

EXPERIMENTAL RESULTS

This section evaluates the new alert aggregation approach. We use three different data sets to demonstrate the

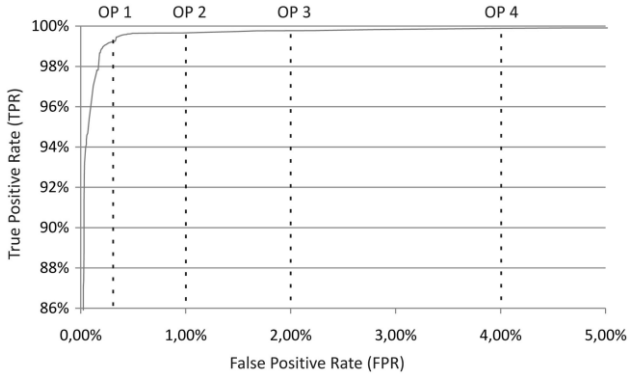

Fig. 4. ROC curve for the SVM detector.

feasibility of the proposed method: The first is the wellknown DARPA intrusion detection evaluation data set [32], for the second we used real-life network traffic data collected at our university campus network, and the third

contains firewall log messages from a commercial Internet service provider. All experiments were conducted on an PC with 2.20 GHz and 2 GB of RAM.

Internet Service Provider Firewall Logs

The third data set used here differs from the previous ones as we actually do not have a detector layer that performs a classification and searches for known attacks. Instead, we use log messages that were generated by a CISCO PIX system, which is a well-known commercial FW and network address translation (NAT) device. Typical PIX log messages are DENY or ACCEPT messages—depending on whether the incoming or outgoing network packets or connections are forbidden or allowed according to the firewall rule set. Here, the alerts consist of the source and destination IP address, the source and destination port, the creation time differences, and the PIX message type. Details on the log message format can be found in [43]. The PIX was installed in the network of an Internet service provider within a real-life environment. During the six hours of collection time, 4,898 log messages—we also use the term alerts in the following—were created. As the data were not collected within a controlled simulation environment, we do not have any information whether there occurred attacks or not and specifications of TPR and FPR are not possible. Nonetheless, this data set can be used to demonstrate the broad applicability of the proposed online alert aggregation technique.

Performance Measures

In order to assess the performance of the alert aggregation, we evaluate the following measures:

Percentage of detected instances (p). We regard an attack instance as being detected if there is at least one metaalert that predominantly contains alerts of that particular instance. The percentage of detected attack instances p can thus be determined by dividing the number of instances that are detected by the total number of instances in the data set. The measure is computed with respect to the instances covered by the output of the detection layer, i.e., instances missed by the detectors are not considered.

Number of meta-alerts (MA) and reduction rate (r). The number of meta-alerts (MA) is further divided into the number of attack meta-alerts MA_{attack} which predominantly contain true alerts and the number of nonattack meta-alerts $MA_{nonattack}$ which predominantly contain false alerts. The reduction rate r is 1 minus the number of created meta-alerts MA divided by the total number of alerts N.

Average runtime (t_{avg}) and worst case runtime (t_{worst}). The average runtime is measured in milliseconds per alert. Assuming up to several hundred thousand alerts a day, t_{avg} should stay clearly below 100 ms per alert. The worst case runtime t_{worse}, which is measured in seconds, states how long it takes at most to execute the while loop of Algorithm 2, which may include the execution of Algorithms 3 and 1 (the latter several times).

Meta-alert creation delay (d). It is obvious that there is a certain delay until a meta-alert is created for a new attack instance. The meta-alert creation delay d measures the delay between the actual beginning of the instance (i.e., the creation time of the first alert) and the creation of the first meta-alert for that instance. We investigate, how many seconds the algorithm needs to create 90 percent ($d_{90\%}$), 95 percent ($d_{95\%}$), and 100 percent ($d_{100\%}$) of the meta-alerts.

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 Θ, that decides whether to add a new alert to an existing component or not to five percent, and the value for the threshold γ that specifies the allowed temporal spread of the alert buffer to 180 seconds. Θ 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. A small value of Θ, of course, results in more components and, thus, in a lower reduction rate, but it also reduces the risk of missing attack instances. The parameter Θ, which is used in the novelty assessment function, controls the maximum time that new alerts are allowed to reside in the buffer B. In order to keep the response time short, we set it to 180 s which we think is a reasonable value. For both parameters, there were large intervals in which parameter values could be chosen without deteriorating the results.

DARPA Data

Results for the DARPA data set are given in Table 2. 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 ($d_{100\%}$). 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. With in 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. A typical example is a WAREZCLIENT instance in week four: There, attackers download illegally provided software from a compromised FTP server. As there is no cooperation between the attackers as in the case of a coordinated denial of service attack, for example, there is an individual meta-alert created for every attacker.

TABLE2

Results of the Online Alert Aggregation for Three Benchmark Data Sets

OP	p [%]	MA	MA_{attack}	$MA_{non\text{-}attack}$	r [%]	t_{avg} [ms]	t_{worst} [s]	$d_{90\%}$ [s]	$d_{95\%}$ [s]	$d_{100\%}$ [s]
DARPA Data										
Idealized	100.00	324	324	0	99.98	0.13	8.54	1.79	3.17	96.2
OP 1	100.00	1976	368	1598	99.87	0.19	11.00	1.64	1.82	5.41
OP 2	100.00	3186	369	2817	99.80	0.28	11.83	1.18	1.95	7.92
OP 3	98.04	7946	339	7607	99.50	0.81	19.50	1.73	2.47	17.1
OP 4	99.02	8588	348	8240	99.46	0.97	16.05	1.94	2.47	16.2
Campus Network Data										
n/a	100.00	52	20	32	99.96	0.75	2.87	1.35	1.35	1.61
Internet Service Provider Firewall Logs										
n/a	n/a	56	n/a	n/a	98.86	1.53	0.27	n/a	n/a	n/a

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.

Bidirectional communication. 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.

With an increasing FPR, the number of meta-alerts also increases from 1,976 meta-alerts in OP 1 to 8,588 meta-alerts in OP 4. Even though the number of meta-alerts increases clearly, we still have a reduction rate of 99.46 percent in OP 4, which would heavily reduce the—possibly manual—effort of a human security expert that analyzes that information flood. It is also interesting to see that, although the overall number of meta-alerts $MA_{increases}$, the number of attack clusters MA_{attack} remains roughly constant. The majority of additionally created meta-alerts contains only false alerts.

With more meta-alerts, the runtime increases. Even in OP 4, with an average runtime of 0.97 ms per alert, the proposed technique is still very efficient. Fig. 5 shows the component creation delays for the four operating points. The figure depicts the percentage of attack instances for which a meta-alert was created after a time not exceeding the value specified at the x-axis. It can be seen that the creation delay is With in the range of a few seconds and, thus, meets the requirements for an online application. Table 2 displays the time after which for 90, 95, and 100 percent of the attack instances meta-alerts were created. Note that these values correspond to points on the curve in Fig. 5. Interestingly, in the idealized case, the delay is much higher which can be explained by the novelty detection mechanism (cf. (14)). In

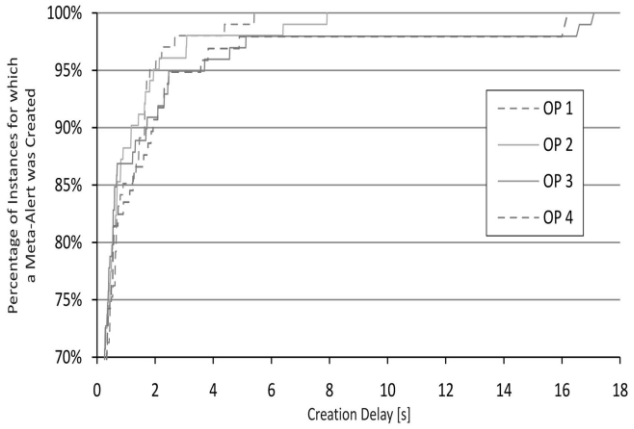

Fig.5.Cumulative creation delays for meta-alerts

the case of a more homogeneous alert stream, the novelty handling is mainly influenced by the temporal spread whereas in the case of a heterogeneous alert stream due to false alerts, the novelty handling is started more often which results in lower component creation delay times.

Campus Network Data

For the campus network data, for which the IDS Snort was used to create alerts, quite similar results could be achieved (see Table 2). All attack instances that have been launched were correctly detected. For the 17 attack instances with 128,816 alerts, 52 meta-alerts were created, which is equivalent to a reduction rate of 99.96 percent. Again, the majority of meta-alerts is caused by false alerts. We have 20 attack meta-alerts and 32 nonattack meta-alerts. For three of the 17 attack instances, a redundant meta-alert was created. The reasons are similar to the ones described for the DARPA data set. It is worth mentioning that for the attack instance where the attacker changed his IP address during the attack, only one meta-alert was created. The results for the runtime as well as for the cluster creation delay are similar to that of the DARPA data, too.

Internet Service Provider Firewall Logs

For the firewall log data, the proposed alert aggregation could also be applied successfully. As Table 2 shows, 56 meta-alerts were created for the 4,989 alerts, which is a reduction rate of 98.86 percent. As it is not possible to specify a percentage of detected attack instances, we analyzed the content of the 56 resulting meta-alerts: In many cases, it is possible to find a particular reason for the meta-alerts, e.g., we identified meta-alerts that subsume all alerts that are caused by obviously forbidden accesses from hosts within the DMZ to an external printer using the Printer PDL Datastream Port (HP JetDirect, port 9,100). Another example is meta-alerts that subsume alerts for illegal network time protocol (ntp, port 123) accesses from hosts outside the DMZ. The runtime turns out to be a little bit lower with 1.53 ms per alert, but, interestingly, the worst case runtime tworst is with 0.27 s much smaller than for the previous data sets. This effect might be explained by the more heterogeneous alert stream in case of the firewall log data which results in an increased number of calls of Algorithm 3 due to the novelty handling: On the one side, the more heterogeneous the alert stream is, the more often the (time expensive) Algorithm 3 is called which increases the overall runtime: On the other side, due to the increased number of calls of Algorithm 3, every call is conducted on an alert buffer that contains only a few new alerts which reduces the runtime of a single call of Algorithm 3—and, thus, the worst case runtime.

Conclusion

The experiments demonstrated the broad applicability of the proposed online alert aggregation approach. To approaching all result should be find out very easy to identified.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. Although there are situations as described in Section 3.3—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.

SUMMARY AND OUTLOOK

We presented a novel technique for online alert aggregation and generation of meta-alerts. We have shown that the sheer amount of data that must be reported to a human security expert or communicated within a distributed intrusion detection system, for instance, can be reduced significantly. The reduction rate with respect to the number of alerts was up to 99.96 percent in our experiments. At the same time, the number of missing attack instances is extremely low or even zero in some of our experiments and the delay for the detection of attack instances is within the range of some seconds only. In the future, we will develop techniques for interestingness- based communication strategies for a distributed IDS. This IDS will be based on organic computing principles [44]. In addition, we will investigate how human domain knowledge can be used to improve the detection processes further. We will also apply our techniques to benchmark data that fuse information from heterogeneous sources (e.g., combining host and network-based detection).

ACKNOWLEDGMENTS

This work was partly supported by the German Research Foundation (DFG) under grant number SI 674/3-2. The authors would like to thank D. Fisch for his support in preparing one of the data sets. The authors highly appreciate the suggestions of the anonymous reviewers that helped them to improve the quality of the article.

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

[3] 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.

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

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

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

[7] F. Cuppens and R. Ortalo, "LAMBDA: A Language to Model a Database for Detection of Attacks," Recent Advances in Intrusion Detection, H. Debar, L. Me, and S.F. Wu, eds. pp. 197-216, Springer,2000.

[8] M.S. Shin, H. Moon, K.H. Ryu, K. Kim, and J. Kim, "Applying Data Mining Techniques to Analyze Alert Data," Web Technologies and Applications, X. Zhou, Y. Zhang, and M.E. Orlowska, eds. pp. 193-200, Springer, 2003.

[9] R. Smith, N. Japkowicz, M. Dondo, and P. Mason, "Using Unsupervised Learning for Network Alert Correlation," Advances in Artificial Intelligence, R. Goebel, J. Siekmann, and W. Wahlster, eds. pp. 308-319, Springer, 2008.
[10] O. Buchtala, W. Grass, A. Hofmann, and B. Sick, "A Distributed Intrusion Detection Architecture with Organic Behavior," Proc. First CRIS Int'l Workshop Critical Information Infrastructures (CIIW '05), pp. 47-56, 2005.
[11] J. Postel, "RFC 790—Assigned numbers," http://www.faqs.org/ rfcs/rfc790.html, Sept. 1981.
[12] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E. Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman, "Evaluating Intrusion Detection Systems: The 1998 DARPA Offline Intrusion Detection Evaluation," Proc. DARPA Information Survivability Conf. and
Exposition (DISCEX), vol. 2, pp. 12-26, 2000.
[13] M. Halkidi and M. Vazirgiannis,"Clustering Validity Assessment Using Multi Representatives," Proc. SETN Conf., vol. 2, pp.237- 249, 2002.
[14] M.V. Mahoney and P.K. Chan, "An Analysis of the 1999 DARPA/ Lincoln Laboratory Evaluation Data for Network Anomaly Detection," Recent Advances in Intrusion Detection, G. Vigna, E. Jonsson, and C. Kru¨ gel, eds., pp. 220-237, Springer, 2003.
[15] CISCO Systems, Inc., "Cisco PIX Firewall System Log Messages, Version 6.3," http://www.cisco.com/en/US/docs/security/pix/ pix63/system/message/pixemsgs.html, 2009.
[16] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations
as Performed by Lincoln Laboratory," ACM Trans. Information and System Security, vol. 3, no. 4, pp. 262-294, 2000.
[17] M. Halkidi and M. Vazirgiannis, "Clustering Validity Assessment Using Multi Representatives," Proc. SETN Conf., vol. 2, pp. 237-249, 2002.