ISSN:0975-9646

J Swathi et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (4) , 2012,4723 - 4730

Detection of Intrusion using Alert Aggregation in DataStream Modelling with Constructive Basis

J Swathi[#1], Prof.P.Pradeep[*2], P Mahesh Kumar[#3]

[#1]*MTech (CS), Vivekananda Institute of Technology and Science, Karimnagar, A.P, INDIA*
[*2]*Head of the Department CSE, Vivekananda Institute of Technology and Science, Karimnagar, A.P, INDIA*
[#3]*MTech (CSE), Bandari Srinivas Institute of Technology and Science, Hyderabad, A.P, INDIA*

***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 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.**

***Keywords*— Intrusion detection, alert aggregation, generative modelling, data stream algorithm.**

I. 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 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 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 using prob-abilistic 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. 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.

II. 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. 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). 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 to cluster alerts. 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.

A weighted, attribute-wise similarity operator is used to decide whether to fuse two alerts or not. However, as already stated, this approach suffers from the high number of parameters that need to be set. The similarity operator presented 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. 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 drawbacks that arise thereof are the same as those mentioned above.

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 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 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 is described, for example. More details can be found.

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 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 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. The same idea but based on a different offline clustering algorithm is presented.

A completely different clustering approach is presented. There, the reconstruction error of an auto associator 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.

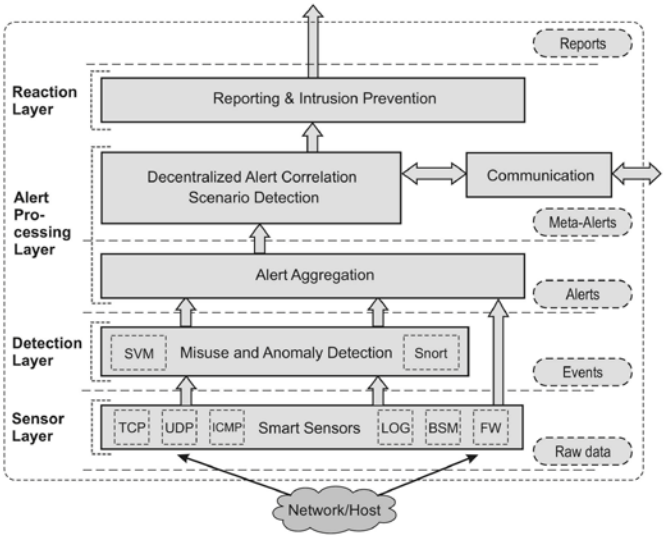

Fig. 1. Architecture of an intrusion detection agent.

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.

III. 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 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

In our work, we focus on a system of structurally very similar so-called intrusion detection (ID) agents. Through self-organized collaboration, these ID agents form a distributed intrusion detection system (DIDS).

Fig. 1 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 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. In other words, with the alert aggregation module on which we focus in this paper we want to have a minimal number of missing meta-alerts (false negatives) and we accept some false meta- alerts (false positives) and redundant meta-alerts in turn.

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. As the concrete content and format depend on a specific task and on certain realizations of the sensors and detectors, some more details will be given in Section 4 together with the experimental conditions.

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. Data Stream Alert Aggregation

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

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, 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.

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

```
// initialize alert buffer
B := ∅
while new alert a is received do
    if C = ∅ then
        // create first component
        C_1 := {a}
        C := {C_1}
        initialize parameters μ_1, σ_1^2, and ρ_1.
    else
        C' := C
        // E step: assign alert to most likely component
        j* := arg max_{j∈{1,...,|C|}} H(a|μ_j, σ_j^2, ρ_j)
        C_j* := C_j* ∪ {a}
        // M step: update component parameters
        N_j* := |C_j*|
        for all attributes d ∈ {1,...,D_m} do
            ρ_{j*_d} := (1/N_j*) · Σ_{a∈C_j*} a_d
        for all attributes d ∈ {D_{m+1},...,D} do
            μ_{j*_d} := (1/N_j*) · Σ_{a∈C_j*} a_d
            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)
        if Ω(C)/Ω(C') < θ then
            C := C'
            B := B ∪ {a}
        // initiate novelty handling with Eq. 14; call algorithm 3
        if novelty(a) then
            C := ALG3(C, j*, B)
            B := ∅
        // initiate obsoleteness handling with Eq. 15
        for j ∈ {1,...,|C|} do
            if obsoleteness(C_j) then
                C := C\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. 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. In some tests, it turned out that our main goal not to miss any attack instances, can be achieved this way with substantially lower runtimes but at the cost of some redundant meta-alerts (due to split of clusters). 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 &).

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. How- ever, 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 $\mathcal{C}$

```
C' := C\C_j*
// test several component numbers
for k=1 to K do
    // conduct off-line clustering with algorithm 1 for buffer and most likely component
    C^(k) := ALG1(C_j* ∪ B, k)
    // assess result with Eq. 13
    Ω^(k) := Ω(C' ∪ C^(k))
// determine number of components
k* := arg max_{k∈{1,...,K}} Ω^(k)
// update model
C := C' ∪ 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 sub sampling technique.

A. Meta-Alert Generation and Format

With the creation of a new component, an appropriate meta- alert 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 meta- alerts 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).

IV. EXPERIMENTAL RESULTS

This section evaluates the new alert aggregation approach. We use three different data sets to demonstrate the feasibility of the proposed method: The first is the well- known DARPA intrusion detection evaluation data set, 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.

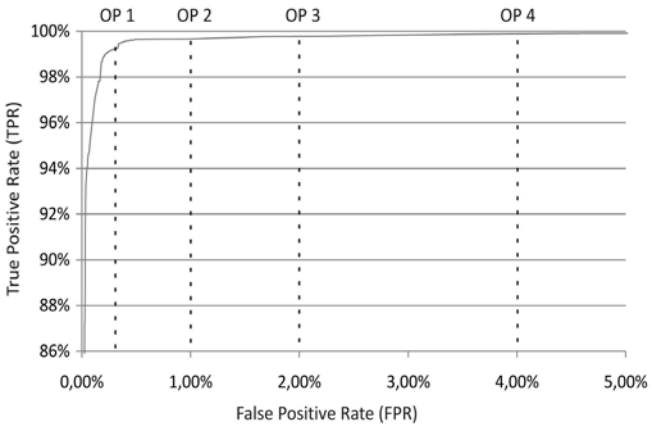

Fig. 2. ROC curve for the SVM detector.

A. Campus Network Data

To assess the performance of our approach in more detail, we also conducted own attack experiments. We launched several brute force password guessing attacks against the mail server (POP3) of our campus network and recorded the network traffic. The attack instances differed in origin, start time, duration, and password guessing rate. The attack schedule was designed to reflect situations which we regard as being difficult to recognize. In particular, we have

1. several concurrent attack instances (up to seven),
2. partially and completely overlapping attack in- stances,
3. several instances within a short time interval,
4. different attack instances from similar sources,
5. different attack durations, and
6. an attacker that changes his IP address during the attack.

In order to demonstrate that the proposed technique can also be used with a conventional signature-based detector, the captured traffic was analyzed by the open source IDS Snort, which detected all 17 attack instances that have been launched and produced 128,816 alerts. The alert format equals the one used for the SVM detector, i.e., the alerts exhibit the source and destination IP address, the source and destination port, the attack type, and creation time differences. Snort was configured to match our network topology and we turned off rudimental alert aggregation features. In order to achieve a high recall, we activated all available rule sets the official rule sets as well as available community rules, which both are available at the Snort web page. Activating all rules leads to a false alert rate of 0.33 percent. The FPR is based on the assumption that all alerts that are not classified with the attack type that we launched are false alerts. There is no missing alert rate given in Table 1 for this data set for two reasons: First, it cannot be guaranteed that there are unknown attacks in the data set that were started by real attackers and second, we do not know exactly how many alerts should be created by the attacks we launched.

B. 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 meta- alert 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. 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.

C. 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 μ 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. A small value of &, 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 μ, 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. A detailed analysis and discussion on the effects of different parameter settings can be found.

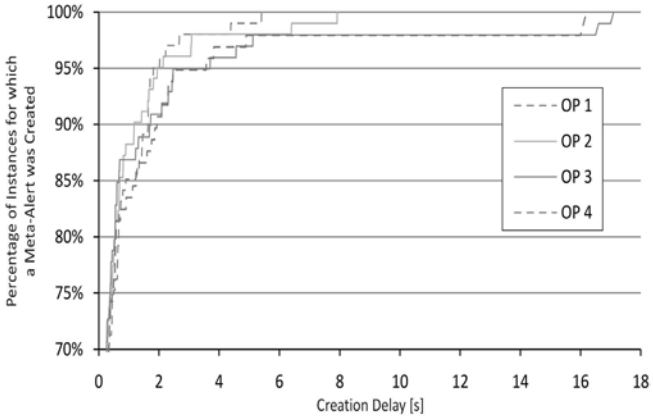

Fig. 3. Cumulative creation delays for meta-alerts.

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. 3 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 within 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 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.

D. 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 signature- based 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 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.

V. 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] 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 Assoc., 2000.

[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. Krü gel, and R.A. Kemmerer, "A Comprehensive Approach to Intrusion Detection Alert Correla-tion," 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, Universität Dortmund, 2003.

[12] T. Pietraszek, "Alert Classification to Reduce False Positives in Intrusion Detection," PhD dissertation, Universität 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.

[16] P. Ning, Y. Cui, D.S. Reeves, and D. Xu, "Techniques and Tools for Analyzing Intrusion Alerts," ACM Trans. Information Systems Security, vol. 7, no. 2, pp. 274-318, 2004.

[17] 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.

[18] S.T. Eckmann, G. Vigna, and R.A. Kemmerer, "STATL: An Attack Language for State-Based Intrusion Detection," J. Computer Security, vol. 10, nos. 1/2, pp. 71-103, 2002.

[19] A. Hofmann, "Alarmaggregation und Interessantheitsbewertung in einem dezentralisierten Angriffserkennungsystem," PhD dis- sertation, Universität Passau, under review.

[20] 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.

[21] J. Song, H. Ohba, H. Takakura, Y. Okabe, K. Ohira, and Y. Kwon, "A Comprehensive Approach to Detect Unknown Attacks via Intrusion Detection Alerts," Advances in Computer Science—ASIAN 2007, Computer and Network Security, I. Cervesato, ed., pp. 247-253, Springer, 2008.

[22] 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.

[23] A. Hofmann, D. Fisch, and B. Sick, "Identifying Attack Instances by Alert Clustering," Proc. IEEE Three-Rivers Workshop Soft Computing in Industrial Applications (SMCia '07), pp. 25-31, 2007.

[24] M. Roesch, "Snort—Lightweight Intrusion Detection for Networks," Proc. 13th USENIX Conf. System Administration (LISA '99), pp. 229-238, 1999.

[25] 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.

[26] D. Fisch, A. Hofmann, V. Hornik, I. Dedinski, and B. Sick, "A Framework for Large-Scale Simulation of Collaborative Intrusion Detection," Proc. IEEE Conf. Soft Computing in Industrial Applica- tions (SMCia '08), pp. 125-130, 2008.

[27] R.O. Duda, P.E. Hart, and D.G. Stork, Pattern Classification, second ed. Wiley Interscience, 2001.

[28] IANA, "Port Numbers," http://www.iana.org/assignments/port-numbers, May 2009.

[29] Y. Rekhter, B. Moskowitz, D. Karrenberg, and G. de Groot, "RFC 1597—Address Allocation for Private Internets," http://www.faqs.org/rfcs/rfc1597.html, Mar. 1994.

[30] J. Postel, "RFC 790—Assigned numbers," http://www.faqs.org/rfcs/rfc790.html, Sept. 1981.

[31] O. Buchtala, A. Hofmann, and B. Sick, "Fast and Efficient Training of RBF Networks," Artificial Neural Networks and Neural Information Processing—ICANN/ICONIP 2003, O. Kaynak, E. Alpaydin, E. Oja, and L. Xu, eds., pp. 43-51, Springer, 2003.

[32] 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.

[33] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "On Clustering Validation Techniques," J. Intelligent Information Systems, vol. 17, nos. 2/3, pp. 107-145, 2001.

[34] J.C. Dunn, "Well Separated Clusters and Optimal Fuzzy Partitions," J. Cybernetics, vol. 4, pp. 95-104, 1974.

[35] D.L. Davies and D.W. Bouldin, "A Cluster Separation Measure," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 1, no. 2, pp. 224-227, Apr. 1979.

[36] M. Halkidi and M. Vazirgiannis, "Clustering Validity Assessment Using Multi Representatives," Proc. SETN Conf., vol. 2, pp. 237-249, 2002.

[37] A. Hofmann, I. Dedinski, B. Sick, and H. de Meer, "A Novelty-Driven Approach to Intrusion Alert Correlation Based on Distributed Hash Tables," Proc. 12th IEEE Symp. Computers and Comm. (ISCC '07), pp. 71-78, 2007.

[38] F. Provost and T. Fawcett, "Analysis and Visualization of Classifier Performance: Comparison under Imprecise Class and Cost Distributions," Proc. Third Int'l Conf. Knowledge Discovery and Data Mining (KDD '97), pp. 43-48, 1997.
[39] 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.
[40] 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. Krü gel, eds., pp. 220-237, Springer, 2003.
[41] A. Hofmann, D. Fisch, and B. Sick, "Improving Intrusion Detection Training Data by Network Traffic Variation," Proc. IEEE Three-Rivers Workshop Soft Computing in Industrial Applications, pp. 25-31, 2007.
[42] Sourcefire, Inc., http://www.snort.org/,2009.
[43] CISCO Systems, Inc., "Cisco PIX Firewall System Log Messages, Version6.3," http://www.cisco.com/en/US/docs/security/pix/pix63/system/message/pixemsgs.html, 2009.
[44] Organic Computing, R.P. Wü rtz, ed. Springer, 2008.