Data Mining and Machine Learning—Towards Reducing False Positives in Intrusion Detection[*]

Tadeusz Pietraszek[a] and Axel Tanner[a]

[a]IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland

Intrusion Detection Systems (IDSs) are used to monitor computer systems for signs of security violations. Having detected such signs, IDSs trigger alerts to report them. These alerts are presented to a human analyst, who evaluates them and initiates an adequate response. In practice, IDSs have been observed to trigger thousands of alerts per day, most of which are mistakenly triggered by benign events (i.e., false positives). This makes it extremely difficult for the analyst to correctly identify alerts related to attacks (i.e., true positives).

In this paper we present two orthogonal and complementary approaches to reduce the number of false positives in intrusion detection using alert postprocessing by data mining and machine learning. Moreover, these two techniques, because of their complementary nature, can be used together in an alert-management system. These concepts have been verified on a variety of data sets, and achieved a significant reduction of the number of false positives in both simulated and real environments.

1. Introduction

The explosive increase in the number of networked machines and the widespread use of the Internet in organizations have led to an increase in the number of unauthorized activities, not only by external attackers but also by internal sources, such as fraudulent employees or people abusing their privileges for personal gain or revenge. As a result, intrusion detection systems (IDSs), as originally introduced by Anderson [1] and later formalized by Denning [11], have received increasing attention in recent years.

On the other hand, with the massive deployment of IDSs, their operational limits and problems have become apparent [2,4,19,26]. False positives, i.e., alerts that mistakenly indicate security issues and draw attention from the intrusion detection analyst, are one of the most important problems faced by intrusion detection today [29]. In fact, it has been estimated that up to 99% of alerts reported by IDSs are not related to security issues [2,4,19]. Reasons for this include the following:

Runtime limitations: In many cases an intrusion differs only slightly from normal activities, sometimes even only the context in which the activity occurs determines whether it is intrusive. Owing to harsh real-time requirements, IDSs cannot analyze the context of all activities to the extent required [37].

Specificity of detection signatures: Writing signatures for IDSs is a very difficult task [35]. In some cases, the right balance between an overly specific signature (which is not able to capture all attacks or their variations) and an overly general one (which recognizes legitimate actions as intrusions) can be difficult to determine.

Dependency on environment: Actions that are normal in certain environments may be malicious in others [3]. For example, performing a network scan is malicious unless the computer performing it has been authorized to do so. IDSs deployed with the standard out-of-the-box configuration will most likely identify many normal activities as malicious.

Base-rate fallacy: From the statistical point of view and following the example by Axelsson [2], we can calculate the probability that

[*]Appearing in Information Security Technical Report, 10(3), 2005

an alert is a false positive using Bayes theorem. Assuming that we analyze 1,000,000 packets per day containing only 20 packets of intrusions, it turns out that even with the unrealistic perfect detection rate of 1.0 and a very low *false positive rate* on the order of 10^{-5} we obtain 10 false positives, leading to a *Bayesian detection rate* of true positives of only 66%. This means that one third of the alerts will be false positive, i.e., will not be related to intrusive activities. With the same false alarm rate and a more realistic detection rate of 0.7, we obtain that 42% of alerts will be false positives.

This shows that building IDS having a small number of false positives is an extremely difficult task. In this paper we present two *orthogonal* and *complementary* approaches to reduce the number of false positives in intrusion detection by using alert postprocessing. The basic idea is to use existing IDSs as an alert source and then apply either off-line (using data mining) or on-line (using machine learning) alert processing to reduce the number of false positives. Moreover, owing to their complementary nature, both approaches can also be used together. Concepts presented here have been published in [19,17,20,18] (data mining) and in [36] (machine learning).

The paper is organized as follows: After presenting the related work in IDS alert-management space in Section 2, we take a look at a global picture of alert management (Section 3). Subsequently, we present the two approaches to alert management: using data mining (Section 4) and machine learning (Section 5). Finally, we discuss why combining these two methods makes sense (Section 6), and present conclusions and future work (Section 7).

2. Related Work—Towards Reducing False Positives

The related work in alert management can be grouped into two categories: (*i*) improving the quality of alerts and (*ii*) alert correlation.

2.1. Improving the Quality of Alerts

In the first category, there are approaches that try to improve the quality of alerts by leveraging additional information, such as alert context and vulnerability statements. Such an approach has been used by Sommer and Paxson [42] to enhance Bro's [35] byte-level alert signatures with regular expressions and context derived from protocol analysis.

Another approach, also performed by commercial alert correlation products, is to match alerts with vulnerability reports. Lippmann et al. [24] show how such an information can be used to prioritize alerts, depending on the vulnerabilities of the target: Even correctly identified signs of intrusions can be given a lower priority or even be discarded if the target has been identified not to be vulnerable to that specific attack. A formal model for alert correlation and leveraging such information has been proposed by Morin et al. [29].

2.2. Alert Correlation

The second category are approaches trying to solve a more ambitious goal than alert classification—the reconstruction of high-level incidents from low-level alerts. Note that in the general case, it is not possible to reconstruct incidents from alerts with certainty. To illustrate this problem, let us consider a set of alerts that were triggered by various source hosts. Knowing this, but having no additional background knowledge, it is not possible to decide with certainty whether these attacks constitute a single coordinated attack or independent attacks that happen to be interleaved. This said, alert correlation systems perform well in many cases, greatly helping with the analysis of the alerts.

The simplest form of incident recognition is based on similarities between alert features. This approach is intuitive, simple to implement, and proved to be effective in real environments. As proposed by Debar and Wespi [10,15], given three alert attributes (namely, attack class, source address and destination address), we can group alerts into scenarios depending on the number of matching attributes from the most general (only one attribute matches) to the most specific case (all three attributes match). Each of these scenarios has a practical meaning. This approach is successfully used in alert pre-processing, although it can be limited when correlating more complex attacks (e.g., an attack compromising the host and then launching another attack from there). Another problem is that this technique relies on some numerical parameters that can only be set empirically.

The work by Valdes and Skinner [44] presents a heuristic approach to alert correlation using a weighted sum of similarities of attributes, which allows to group alerts into scenarios. Dain and Cunningham [9] show how parameters for a similar heuristic model can be learned from manually classified so-called scenarios from DEF CON 2000 CTF data. In their approach the system sequentially processes the alerts and either joins them into an already existing scenarios or creates a new scenario. Such scenarios can be then investigated by the analyst.

It has been shown how other forms of knowledge, such as prerequisites and consequences of attack steps, can improve high-level correlation, even if it is not possible to formalize and capture all those prerequisites–consequences relations perfectly. The work by Cuppens et al. [6,8,7] implements this concept using Prolog predicates, whereas Ning et al. [32,31,33,30] consider correlation graphs implemented in a relational database.

Formal model checking (as proposed by Ritchey and Ammann [38]) and attack graphs analysis allow the vulnerability of a network of hosts to be evaluated and global vulnerabilities to be identified. Attack graphs can serve as a basis for detection, defense and forensic analysis. As claimed by Jha et al. [16,41], attack graphs can enhance both heuristic and probabilistic correlation approaches. Given the graph describing all likely attacks and IDSs, we can match individual alerts to attack edges in a graph. Being able to match successive alerts to individual paths in the attack graph significantly increases the likelihood that the network is under attack. It is possible to predict attackers' goals, aggregate alarms and reduce the false alarm rates. Formal techniques exhibit a great potential, but at the current stage of research can serve more as a proof-of-concept rather than a working implementation. It can be expected that with more formal definitions of vulnerabilities and security properties, formal analysis techniques will be gaining in significance.

The main focus of our approach is to improve the quality of alerts and identify true and false positives using alert preprocessing utilizing the experise of a human domain experts. Thus, our work fits into the first category above. Our approach can also be used with most of alert correlation systems discussed in the second category, possibly resulting in a better correlation accuracy.

3. Alert Management—The Global Picture

In a standard setup (Fig. 1), alerts generated by IDSs are passed to a human analyst. The analyst uses his or her knowledge to distinguish between false and true positives and to understand the severity of the alerts. Depending on this classification, the analyst may report security incidents, investigate intrusions, or identify network and configuration problems. The analyst may also try to modify bad IDS signatures or install alert filters to remove alerts matching some predefined criteria.

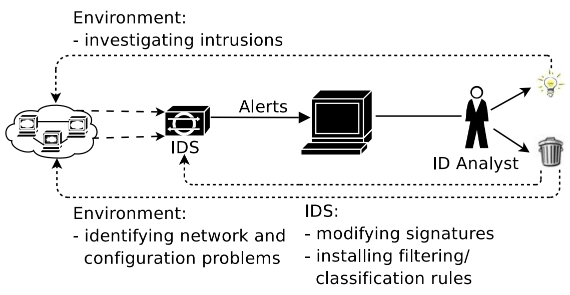
Environment:
- investigating intrusions
Alerts
IDS
ID Analyst
Environment:
- identifying network and configuration problems
IDS:
- modifying signatures
- installing filtering/ classification rules

Figure 1. The global picture of alert management.

Note that in the standard setup, manual knowledge engineering (shown by dashed lines in the figures) is separated from, or not directly used for, classifying alerts. The conventional setup does not take advantage of two facts: First, that usually a large database of historical alerts exists that can be used for automatic knowledge engineering and, second, that the analyst is analyzing the alerts in real time.

These two facts form the basis of the new alert-handling paradigm using two orthogonal approaches: *data mining* and *machine learning*. The characteristics of the two approaches are presented in Table 1 and will be discussed in more detail in the following sections.

As a general remark, note that alert-management systems, such as these presented in this paper, process only alerts generated by an IDS and are therefore not capable of detecting attacks that the underlaying IDS missed. Therefore the investigation of such missed attacks is beyond the scope of our systems and this paper.

4. Data-Mining Approach—Unlabeled Alerts

The first approach to address the problem of false positives is to use data mining of the historical alert logs in an off-line fashion to discover common reasons for large numbers of alerts [19,18]. Underlying this method is the concept that for every observed alert there is a *root cause*, which is the reason for its generation. The hypothesis, verified by [19], which makes this approach interesting is (*i*) that large groups of alerts have a common root cause, (*ii*) that few of these root causes account for a large volume of alerts, and (*iii*) that these root causes are (relatively) persistent over time. This means that discovering and removing few most prominent root causes can safely and significantly reduce the number of alerts the analyst has to handle.

The goal of the *CLARAty* (CLustering Alerts for Root Cause Analysis) is therefore to efficiently detect large clusters of alerts and to describe them in a generalized way in order to make commonalities between the different alerts explicit and understandable for a human. Ideally, these generalized descriptions correspond to root causes and are in turn actionable for the IDS security analysts.

Figure 2 shows how CLARAty fits into the standard alert-management setup, its components and workflow: historical alerts, which are directly generated by the IDSs in the computing environment and are therefore unlabeled, are mined for large clusters of alerts (see below for details). The descriptions of these clusters are presented in the form of *generalized alerts* to a human analyst to search for the underlying root causes. If these root causes are considered to be false positives rather than signs of real attacks, the alerts should not burden the operators anymore in the future and should therefore be removed. The ideal solution is to eliminate the corresponding root cause either by fixing the computing environment (e.g., in the case of a misconfiguration of a network device) or by tuning the components of the IDS (e.g., signatures can be made more specific). If this is not possible, the corresponding alerts can be filtered out of the stream of alerts by means of alert filters.

4.1. Background Knowledge—Clustering Hierarchies

In general, alerts are described as tuples of attribute-value pairs. Alerts can have many different attributes depending on the type of the alert, but in practice, certain basic attributes are always part of the alert, e.g., a timestamp, source and destination addresses, and a description or type of the alert.

To allow the clustering of alerts in the abstract alert space, a measure of similarity is needed. To define this measure we use our knowledge of the specific environment and general properties. This *background knowledge* is represented through *generalization hierarchies* of the important attributes of the alerts.

For example, the network topology of the specific environment can be captured in a hierarchy of IP address descriptions, which, possibly through multiple levels, generalize a specific IP addresses into generalized address labels. In this way, specific IP addresses could be labeled according to their role (`Workstation`, `Firewall`, `HTTPServer`), then grouped according to their network location (`Intranet`, `DMZ`, `Internet`, `Subnet1`) with a final top-level gen-

Table 1
Characteristics of the two alert management approaches presented in this paper.

	Data mining **CLARAty** [19,17,20,18]	**Machine Learning** **ALAC** [36]
Type	Off-line	On-line
Input	Unclassified (raw) alerts	Classified alerts
Size of input	Large	Small–Medium
Background knowledge	Clustering hierarchies	Attribute-value
Interpretable results	Cluster descriptions	Classification rules
Targets	Mostly false positives with identifiable root causes	Both false positives and true positives

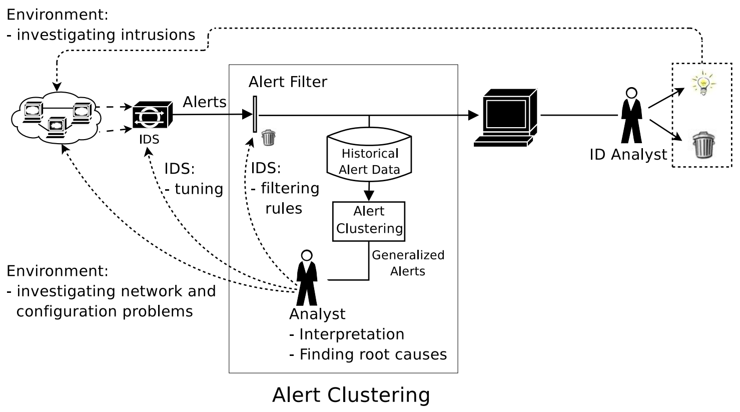

Figure 2. Alert management system using CLARAty (data mining) to discover root causes and reduce false positives in intrusion detection.

eralized address `AnyIP` (see Fig. 3).

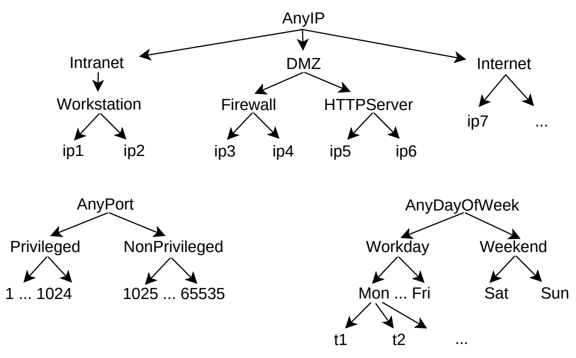

Figure 3. Sample generalization hierarchies for address, port and time attributes.

Other attributes will have different generalization hierarchies, depending on the type and our interests. For example, the source and destination ports of port-oriented IP connections can be generalized into `Privileged` (1-1024) and `NonPrivileged` (1025-65535), with a top-level category of `AnyPort`. And a timestamp attribute could be generalized into `Workday` and `Weekend`, or also `OfficeHours` and `NonOfficeHours`.

Generalization hierarchies like these are static, but dynamic hierarchies are also possible, for example frequently occurring substrings in free-form string attributes (e.g., in context information), which can be generated and evaluated during the runtime of the data mining.

Although the generalization hierarchies described here are represented as trees, CLARAty can be extended to allow the use of *directed acyclic graphs* (DAGs) [18].

4.2. Alert Clustering

Many different data-mining techniques exist for cluster analysis and the suitability of the different methods strongly depends on the area of application and its properties. For our problem of alert clustering, where we look for human-understandable descriptions of alert clusters corresponding to root causes, *attribute-oriented induction (AOI)* [14,13] with extensions for this specific application area is the most fitting.

The modified AOI algorithm as described in [19] uses the generalization hierarchies describing the background knowledge to combine alerts into *generalized alerts* iteratively. These generalized alerts contain at least partially *generalized attributes*, i.e., attributes generalized through the above hierarchies beyond the lowest level. As an example, see the first generalized alert in Table 2: It describes a cluster of alerts of the type `WWW IIS view source attack` coming from a `NonPrivileged` port of a machine in the `Internet`, targeting the specific destination address `ip5` on the specific port `80`. In this case the timestamp was generalized to the highest level `AnyTime`, indicating that the alerts occur at random times and no preference for specific days of the week or times during the day could be observed.

Conceptually, the modified AOI algorithm works as follows: Given a large set of alerts with attributes $A_1, ...A_n$, a generalization hierarchy $\mathcal{G}_i$ for each attribute of the alerts and a parameter $N_{\min}$, the process of alert clustering to create one generalized alert consists of the following steps:

1. Select an attribute A_i to generalize (heuristically).
2. For all alerts: replace the value of attribute A_i with the parent value of the corresponding generalization hierarchy for A_i.
3. Group all alerts that have become identical after this generalization and maintain an associated *count* of the corresponding original alerts.
4. Iterate the above steps until one of the generalized alerts has a count larger than $N_{\min}$.

The control parameter $N_{\min}$ needs to be specified by the user. As the output of the above algorithm is one generalized alert that covers at least $N_{\min}$ original alerts, this parameter must be chosen carefully: if it is chosen too large, distinct root causes are merged and therefore the result will be too generalized. If it is chosen too small, a single root cause might be represented by multiple generalized alerts. Although this parameter basically has to be defined heuristically, a stability test is made by varying $N_{\min}$: the above algorithm will be executed for a $N'_{\min} = (1 \pm \epsilon)N_{\min}$ and the resulting generalized alerts will be compared. If the results are the same, the generalized alert is considered *robust* and therefore will be accepted for further analysis; otherwise, the algorithm is repeated with a slightly lower $N_{\min}$. For more details, see [19].

This algorithm will explicitly construct one generalized alert that defines a cluster of at least $N_{\min}$ alerts. Iterations of the above algorithm will deliver multiple generalized alerts, describing a large part of the original set of alerts in a compact way.

As can be seen, this approach focuses on identifying the root causes for large groups of alerts, which typically correspond to problems in the computing infrastructure that lead to many false positives (with the potential exception of large-scale automated attacks). It does not look for small, stealthy attacks in the alert logs, but aims to reduce the noise in the raw alerts to make it easier to identify real attacks in the subsequent analysis.

4.3. Results

The data-mining approach was tested with real world alert logs from IBM's Managed Security Services. Alert logs from 16 commercial network-based, misuse IDSs placed in operational environments of a variety of companies and network locations were analyzed. The data was collected over a period of one year, totalling more than 35 million alerts. Complete results for this analysis can be found in [19].

The alerts have been grouped by IDS and month into distinct data sources for the alert clustering. The generalized alerts resulting from the analysis with CLARAty were evaluated by an intrusion detection specialist to identify root causes of the corresponding cluster of alerts described by

Table 2
Examples of generalized alerts in the intrusion detection logs.

Alert Type	**Source-Port**	**Source-IP**	**Dest-Port**	**Dest-IP**	**Time**	**Size** (total 156380)
`WWW IIS view source attack`	NonPrivileged	Internet	80	ip5	AnyTime	54310
`FTP SYST command attempt`	NonPrivileged	Firewall	21	Internet	AnyTime	6439
`IP fragment attack`	n/a	ip6	n/a	Firewall	Workday	4581
`TCP SYN host sweep`	NonPrivileged	Firewall	25	AnyIP	AnyTime	761

the generalized alert.

Examples of generalized alerts found by CLARAty for a specific IDS and one month of log data are shown in Table 2.

WWW IIS view source attack: Analysis shows that such an alert is created by the IDS when it sees an `http-GET` request containing the string "%2E", which is the hex-encoding for a ".". It was found that in this specific environment a web server (destination port 80) with the address `ip5` offers a search engine that contains this substring for all search results—so if clients on the `Internet` follow these links, their request to the web server will contain this string and will trigger the (false) alert. This specific generalized alert covered about 54,000 out of about 156,000 alerts of a specific IDS and month.

FTP SYST command attempt: This generalized alert corresponds to the fact that many FTP clients (destination port 21) on the `Internet` are configured to issue the SYST command at the beginning of each FTP session. This is legal behavior and therefore is to be considered a false alert.

IP fragment attack: This generalized alert can result either from `ip6` maliciously sending fragmented packets to the `Firewall`, or from a router that fragments packets between `ip6` and the `Firewall`. Investigation has shown that the latter was the case in this specific environment.

TCP SYN host sweep: The IDS believes in this case that the `Firewall` is scanning multiple IP addresses on port 25. This results from the fact that the firewall relays SMTP (destination port 25) requests from clients. As sometimes multiple clients contact different SMTP servers at nearly the same time, it looks to the IDS as if the relaying firewall is scanning IP addresses on port 25.

Together with nine additional generalized alerts, these generalized alerts cover 149,134 of the total 156,380 raw alerts for this month. This clearly exemplifies that CLARAty is able to describe a large number of raw alerts in a very compact and understandable way, and that the generalized alerts are indeed meaningful and allow the deduction of root causes.

Further evaluation in [19] for the much larger data set shows that the hypotheses stated in the beginning of this section indeed holds: in the 192 distinct alert logs (average size about 180,000 alerts), on average 18 generalized alerts cover more than 90% of the alerts.

The CLARAty approach is also very efficient—on a modest PC it takes on average a few minutes to cluster 100,000 alerts, including the stability analysis with varying $N_{\min}$.

To estimate the alert load reduction that can be achieved by CLARAty, we used the results of the analysis of every month to identify root causes and then use these to reduce the alerts for the following month by filtering. On average this yielded a reduction of about 75% of the alerts in the following month, thereby proving that root causes are indeed persistent in a real environment on this timescale and thus enable a very effective load reduction.

In a few cases, this month-to-month reduction of alerts was much below the average. All of these cases corresponded to very unusual events in the environment, e.g., a networking problem, or the rise of new massive threat such as the NIMDA worm. In these cases, the specific problems created large number of *new* types of alerts, which were not available at the time of analysis, and therefore this previous analysis could not help in the reduction of the new alerts.

Clearly, instead of a timescale of one month, one can choose longer or shorter intervals for the analysis-review-implementation cycles depending on the environment. Timescales are bound from below by the necessity to have a large enough amount of log data and bound from above by the timescale of the persistence of the root causes.

5. Machine-Learning Approach—Labeled Alerts

The second approach addresses the problem of false positives in intrusion detection by building an alert classifier that tells true from false positives. We define *alert classification* as attaching a label from a fixed set of user-defined labels to an alert. In the simplest case, alerts are classified into false and true positives, but the classification can be extended to indicate the category of an attack, the causes of a false positive or anything else.

Alerts are classified by a so-called *alert classifier* (or *classifier* for short). Alert classifiers can be built automatically using machine-learning techniques or they can be built manually by human experts. The Adaptive Learner for Alert Classification (ALAC) introduced in [36] uses the former approach. Most importantly, ALAC learns alert classifiers having an explicit classification logic so that a human expert can inspect it and verify its correctness. In that way, the analyst can gain confidence in ALAC by understanding how it operates.

ALAC classifies alerts into true positives and false positives, and presents these classifications to the intrusion detection analyst, as shown in Fig. 4. In contrast to the standard alert-management approach, ALAC uses the feedback of the analyst, who is classifying the alerts at the alert console, to create labeled alerts. These labeled alerts are used by the system to generate training examples, which are used by machine-learning techniques to first build and then update the classifier. The classifier is then used to classify new alerts. This process is continuously repeated to improve the alert classification. The analyst can review the classifier at any time.

More precisely, ALAC operates in two modes. In the first mode, the so-called *recommender mode*, the system does not process any alerts automatically, but only predicts the labels and forwards the alerts to the analyst together with this recommendation. This means that the analyst has to review all alerts as before, but the suggested classification can be used, for example, to prioritize alerts and also to assess the accuracy of classification. This is the conservative mode of ALAC.

In the second mode, the so-called *agent mode*, the system assesses the confidence of classification and, if it is above a predefined threshold value (c_{th}), the system processes alerts automatically. In the case of false positives, this would simply mean discarding them. In the case of true positives, the system can, e.g., report the event to the administrator or initiate an automated response. Moreover, to ensure the stability of the system in the agent mode, we introduced a *random sampling* of alerts for which the classification confidence is high. This means that, even if the system predicts the classification with high confidence, the alert will nonetheless be forwarded to the analyst with the probability of k (a user-defined rate). This is an important property of ALAC, as the sampling rate k is a parameter that will be used in the evaluation of the system in the next section.

Note that, unlike the data-mining approach used by CLARAty, this approach hinges on the analyst's ability to classify alerts correctly. This assumption is justified because the analyst must be an expert in intrusion detection to perform incident analysis and initiate appropriate responses.

This raises the question of why analysts do not write alert classification rules themselves or do not write them more frequently. An explanation of these issues can be based on the following facts:

Analysts' knowledge is implicit: Analysts find it hard to generalize, i.e., to formulate more general rules, based on individual alert classifications. For example, an analyst might be able to individually classify some alerts as false positives, but may not be able to write a general rule that characterizes the whole set of these alerts.

Environments are dynamic: In real-world environments, the characteristics of alerts change, e.g., different alerts occur as new computers and services are installed or as certain worms or attacks gain and lose popularity. The classification of alerts may also change. As a result, rules need to be maintained and managed. This process is labor-intensive and error-prone.

5.1. Why it is a Hard Problem—The Choice of Machine-Learning Techniques

As stated above, we use machine-learning techniques to build an alert classifier that tells true from false positives. Viewed as a machine-learning problem, alert classification poses several challenges.

First, the distribution of classes (true positives vs. false positives) is usually very skewed, i.e., false positives are more frequent than true positives. Second, it is also common that the cost of misclassifying alerts is asymmetrical, i.e., misclassifying true positives as false positives is usually more costly than vice versa. Third, ALAC classifies alerts in real-time and updates its classifier as new alerts become available. The learning technique should be efficient enough to work in real time and incrementally, i.e., to be able to modify its logic as new data becomes available. Fourth, we require the machine -learning technique to use background knowledge, i.e., additional information such as network topology, alert database, alert context, etc., which is not contained in alerts, but allows us to build more accurate classifiers (e.g., classifiers using generalized concepts, similar to those used in CLARAty). In fact, research in machine learning has shown that the use of background knowledge frequently leads to more natural and concise rules [21]. However, the use of background knowledge increases the complexity of a learning task, and only some machine-learning techniques support it.

In our system we decided to use RIPPER [5]—a fast and effective rule learner. It has been successfully used in intrusion detection (e.g., on system call sequences and network connection data [22,23]) as well as related domains, and has proved to produce concise and intuitive rules. As reported by Lee [22], RIPPER rules have two very desirable properties for intrusion detection: a good generalization accuracy and concise conditions. Another advantage of RIPPER is its efficiency with noisy data sets.

RIPPER is well documented in the literature and its description is beyond the scope of this paper. However, for the sake of better understanding the system, we will briefly explain what kind of rules RIPPER builds.

Given a set of training examples labeled with a class label (in our case false and true alerts), RIPPER builds a set of rules discriminating between classes. Each rule consists of conjunctions of attribute value comparisons followed by a class label and, if the rule evaluates to *true* a prediction is made.

RIPPER can produce *ordered* and *unordered* rule sets. Very briefly, for a two-class problem, an unordered rule set contains rules for both classes (for both false and true alerts), whereas an ordered rule set contains rules for only one class, assuming that all other alerts fall into another class (so-called default rule). Both ordered and unordered rule sets have advantages and disadvantages. We decided to use ordered rule sets because they are more compact and easier to interpret. We will discuss this issue further in Sect. 5.3.4.

Unfortunately, the standard RIPPER algorithm is not cost-sensitive and does not support incremental learning. We therefore converted it to a cost-sensitive learner by implementing instance weighting [43]. For the incremental learning, we adopted a "batch-incremental" approach, where new instances extend the existing training set and together are used for training a new classifier. To prevent the training set from growing to infinity, a windowing function can be implemented. Moreover, retraining is only performed, when the classification accuracy drops below a user-defined threshold. In this way, the learning process is initiated only when it can improve the classification accuracy.

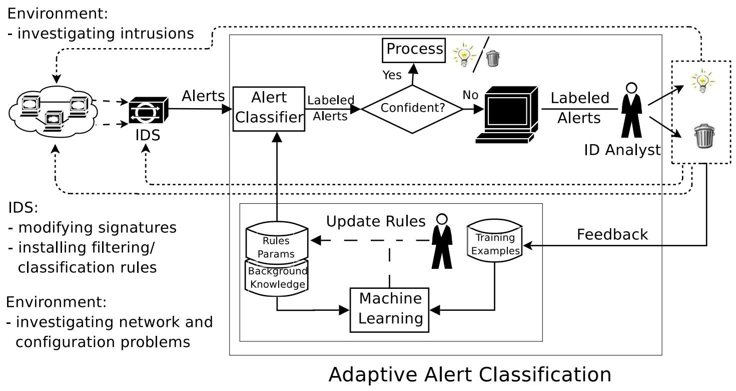

Figure 4. Alert-management system using ALAC (machine learning) to build an adaptive alert classifier.

5.2. Background Knowledge

Recall that we use machine-learning techniques to build the classifier. In machine learning, if the learner has no prior knowledge about the learning problem, it learns exclusively from examples. However, difficult learning problems typically require a substantial body of prior knowledge [21], which makes it possible to express the learned concept in a more natural and concise manner. In the field of machine learning such knowledge is referred to as *background knowledge*, whereas in the field of intrusion detection it is quite often called *context information* (e.g., [42]). The use of background knowledge is also very important in intrusion detection [29]. Examples of background knowledge include:

Network Topology. Network topology contains information about the structure of the network, IP addresses assigned, etc. It can be used to better understand the function and role of computers in the network. Similar to the generalization hierarchies used for CLARAty (see Sec. 4.1), possible classifications can be based on the location in the network (e.g., `Internet`, `DMZ`, `Intranet`, `Subnet1`, `Subnet2`, `Subnet3`) or the function of the computer (e.g., `HTTPServer`, `FTPServer`, `DNSServer`, `Workstation`). In the context of machine learning, the network topology can be used to learn rules that make use of generalized concepts such as `Subnet1, Intranet, DMZ, HTTPServer`.

Alert Context. Alert context, i.e., other alerts *related* to a given alert, is for some attacks (e.g., portscans, password guessing, repetitive exploit attempts) crucial to their classification. In intrusion detection various definitions of alert context are used. Typically, the alert context has been defined to include all alerts similar to the original alert, however the definition of similarity varies greatly [9,6,44].

Alert Semantics and Installed Software. By alert semantics we mean how an alert is interpreted by the analyst. For example, the analyst knows what type of intrusion the alert refers to (e.g., scan, local attack, remote attack) and the type of system affected (e.g., Linux 2.4.20, Internet Explorer 6.0). Such information can be obtained from proprietary vulnerability databases or public sources such as Bugtraq [40] or ICAT [34].

Typically the alert semantics is correlated with the software installed (or the device type, e.g., `Cisco PIX`) to determine whether the system is vulnerable to the reported attack [24]. The result of this process can be used as additional background knowledge for classifying alerts.

Note that the information about the installed software and alert semantics can be used even if no alert correlation is performed, as it allows us to learn rules that make use of generalized concepts such as `OS Linux, OS Windows, etc.`

In the remainder of this section we will briefly present the background knowledge we used in ALAC. For more details, see [36]. We decided to focus on the first two types of background, namely, network topology and alert context. Owing to a lack of required information concerning installed software, we decided not to implement matching alert semantics with installed software. This would also be a repetition of the experiments by Lippmann et al. [24].

Our machine-learning method uses training examples in a propositional form. Therefore, we used an *attribute-value* representation of alerts with the background knowledge represented as additional attributes. Specifically, the background knowledge resulted in 19 attributes, which are constructed as follows: the classification of IP addresses (network topology), the type of operating system and the host function (total three attributes per IP address); alert context computed as aggregates—the number of similar alerts in a specified time window.

For our purposes, similar alerts are defined as follows (each resulting in one additional attribute): (*i*) alerts with the same source IP address, (*ii*) alerts with the same destination IP address, (*iii*) alerts with the same source or destination IP address, (*iv*) alerts with the same signature, and (*v*) alerts classified as intrusions. We computed these aggregates in three time windows of 1, 5 and 30 min., resulting in the above total of 15 attributes.

This choice of background knowledge, which was motivated by heuristics used in alert-correlation systems, is necessarily a bit *ad-hoc* and reflects our expertise in classifying IDS attacks. As this background knowledge is not especially tailored to training data, it is natural to ask how useful it is for alert classification. To answer this question, we performed ROC analysis, which confirmed that that our background knowledge was indeed useful and improved the classification accuracy [36].

5.3. Results

To evaluate the performance of ALAC, we tested it with Snort [39]—an open-source network-based IDS. As ALAC required that labeled examples, we were not able to reuse the data sets we used with CLARAty. Instead, we used two other data sets: a synthetic one and a real-world one producing comparable results, which allows the performance of ALAC on other networks to be estimated.

The synthetic data set was the DARPA1999 data set [28], collected from a simulated medium-sized computer network in a fictitious military base. The DARPA1999 data set has many well-known weaknesses (e.g., [25,27]), and contains some simulation artifacts, which may affect our analysis. This said, the DARPA1999 data set is nonetheless valuable for evaluation of our research prototype and, to the best of our knowledge, the only publicly available data set that can be used to reproduce our results.

The second data set, Data Set B, is a real-world data set collected over a period of one month in a medium-sized corporate network. The network connects to the Internet through firewalls and to the rest of the corporate intranet, and does not contain any externally accessible machines. Our Snort sensor recorded information exchanged between the Internet and the intranet. Owing to privacy issues, this data set cannot be shared with third parties. We do not claim that it is representative for all real-world data sets, but it is an example of a real data set on which our system could be used. Hence, we are using this data set for a second validation of our system.

For both data sets we recorded and classified the alerts generated by a Snort sensor. In the case of DARPA1999 data set, the classification was based on the attack truth tables. In the case of Data Set B, we did the classification manually. Note that in both cases, Snort might have missed some attacks, either because of a lack of proper signatures or because of some evasion technique being used. It should be noted, however, that assessing the performance of Snort was not the goal of our prototype. Instead, we used generated alerts as an input to ALAC and focused on the alert processing.

For the evaluation, it is important to understand the notion of two types of misclassification made by ALAC: false positives and false negatives. A *false positive* occurs when an alert that is not related to a security issue gets classified as a real one. Conversely, when an alert related to a security issue gets classified as an nonimportant one, it is called a *false negative.* Note that for the evaluation of ALAC false positives and false negatives are different from false positives and false negatives of an underlaying IDS, as ALAC sees only the alerts generated by the IDS.

5.3.1. Setting ALAC parameters

ALAC is controlled by a number of parameters, which we had to set in order to evaluate its performance. To evaluate the performance of ALAC as an incremental classifier, we first selected the parameters of its base classifier.

The performance of the base classifier at various costs and class distributions is depicted by the ROC curve, and it is possible to select an optimal classifier for a certain cost ratio and class distribution [12]. As these values are not defined for our task, we could not select an optimal classifier using the above method. Therefore we arbitrarily selected a base classifier that gives a good tradeoff between false positives and false negatives for cost ratio $ICR = 50$. For this classifier we obtained the baseline false positive and false negative rates of $FN = 0.0076$ and $FP = 0.059$ with 10-fold cross-validation. These baseline rates are shown as dashed lines in Figs. 5(a) and 5(b).

We assumed that in real-life scenarios the system would work with an initial model and only use new training examples to modify its model. To simulate this, we used 30% of the input data to build the initial classifier and the remaining 70% to evaluate the system. We evaluated ALAC in two modes as discussed in Section 5: the recommender and the agent mode.

In the remainder of this section, we present the detailed results of our evaluation of ALAC on the DARPA1999 data set. The results for the other data set, Data Set B, can be found in [36].

5.3.2. ALAC in Recommender Mode.

In recommender mode, the analyst reviews each alert and corrects ALAC misclassifications. We plotted the number of misclassifications, i.e., , false positives (Fig. 5(a)) and false negatives (Fig. 5(b)) calculated as rates, as a function of processed alerts.

The resulting overall false negative rate ($FN = 0.024$) is much higher than the false negative rate for the batch classification on the entire data set ($FN = 0.0076$) marked by a dashed line. At the same time, the overall false positive rate ($FP = 0.025$) is less than half of the false positive rate for batch classification ($FP = 0.059$). These differences are expected because of the different learning and evaluation methods used, i.e., batch incremental learning vs. 10-fold cross-validation. Note that both ALAC and a batch classifier have a very good classification accuracy and yield comparable results in terms of accuracy.

5.3.3. ALAC in Agent Mode.

In agent mode, ALAC processes alerts autonomously based on criteria defined by the analyst, as described in Sect. 5. We configured the system to forward all alerts classified as true alerts and false alerts classified with low confidence ($confidence < c_{\text{th}}$) to the analyst. The system discarded all other alerts, i.e., false alerts classified with high confidence, except for a fraction k of randomly chosen alerts, which were also forwarded to the analyst.

Similarly to the recommender mode, we calculated the number of misclassifications made by the system. We experimented with different values of c_{th} and the sampling rate k. We then chose $c_{\text{th}} = 90\%$ and three sampling rates, $k = 0.1$, 0.25 and 0.5. Our experiments show that the sampling rates below 0.1 make the agent misclassify too many alerts and also significantly change the class distribution in the training examples. On the other hand, with sampling rates much higher than 0.5, the system works similarly as in the recommender mode and is less useful for the analyst.

Note that there are two types of false negatives in agent mode — the ones corrected by the analyst and the ones the analyst is not aware of because the alerts have been discarded. We plot-

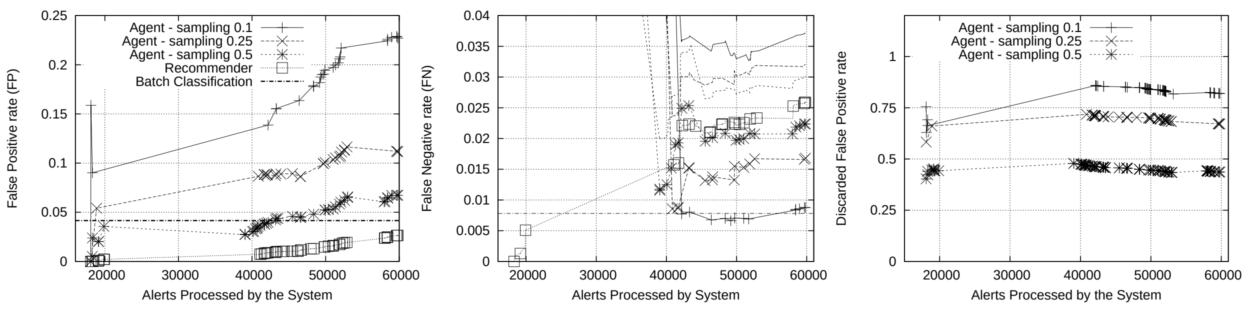

Figure 5. False positive rate, false negative rate and the fraction of discarded alerts for ALAC in agent and recommender modes (DARPA1999 data set, $ICR = 50$).

ted the second type of misclassification as additional series (with no data points) in Fig. 5(a). Intuitively, with lower sampling rates, the agent will have fewer false negatives of the first type, in fact missing more alerts (larger difference between the two series). As expected the total number of false negatives is lower with higher sampling rates. Note that the initial spike of FN at approx. 20,000 alerts (not shown in the figure) is a transient effect at system startup. As the rates are calculated relative to the number of alerts processed, they can be misleadingly high when this number is low. This is only a transient effect, however, and the system has a stable and low false negative rate during its normal operation.

We were surprised to observe that the recommender and the agent have higher false positive rates, but similar false negative rates, even with low sampling rates ($FN = 0.028$ for $k = 0.25$ vs. $FN = 0.025$). This seemingly counterintuitive result can be explained if we note that the automatic processing of alerts classified as false positives effectively changes the class distribution in the training examples in favor of true alerts. As a result, the agent performs comparably to the recommender.

As shown in Fig. 5(c), with the sampling rate of 0.25, more than 70% of false alerts were processed and discarded by ALAC (note that we do not use the initial 30% of alerts used for the initial training in the rate calculations). With the current class distribution, this means that the system autonomously processed over 50% of all input alerts, thus reducing analyst's workload by this factor. At the same time the number of unnoticed false negatives is half the number of mistakes for the recommender mode. Our experiments show that the system is useful for intrusion detection analysts as it significantly reduces number of false positives, without making many mistakes.

5.3.4. Understanding the Rules

One requirement of our system was that the rules can be reviewed by the analyst and their correctness can be verified. The rules built by RIPPER are generally human-interpretable and thus can be reviewed by the analyst. Here is a representative example of two rules used by ALAC:

```
(cnt_intr_w1 <= 0) and (cnt_sign_w3 >= 1)
 and (cnt_sign_w1 >= 1)
 and (cnt_dstIP_w1 >= 1)
   => class=FALSE
(cnt_srcIP_w3 <= 6)
 and (cnt_int_w2 <= 0)
 and (cnt_ip_w2 >= 2)
 and (sign = ICMP PING NMAP)
   => class=FALSE
```

The first rule reads as follows: If a number of alerts classified as intrusions in the last minute (window `w1`) equals zero and there have been other alerts triggered by a given signature and targeted at the same IP address as the current alert, then the alert should be classified as false positive. The second rule says that if the number of NMAP PING alerts originating from the same IP address is fewer than six in the last 30 min. (window w3), there have been no intrusions in the last 5 min. (window w2) and there has been at least 1 alert with identical source or destination IP address, then the current alert is false positive.

These rules are intuitively appealing: If there have been similar alerts recently and they were all false alerts, then the current alert is also a false alert. The second rule says that if the number of NMAP PING alerts is small and there have been no intrusions recently, then the alert is a false alert.

We observed that the comprehensibility of rules depends on several factors, including the background knowledge and the cost ratio. With less background knowledge, RIPPER learns more specific and difficult to understand rules. The effect of varying the cost ratio is particularly apparent for rules produced while constructing the ROC curve, where RIPPER induces rules for either true or false alerts. This is due to the use of RIPPER running in ordered rule set mode.

6. Putting it All together

As we stated in the introduction, CLARAty and ALAC are two complementary approaches and can be used together in a two-staged alert filtering and classification system (Fig. 6). Such a system would use CLARAty in the first stage to periodically mine raw alerts, discover their root causes and either remove them or install alert filters. The output from the first stage would then be forwarded to ALAC interacting with an operator.

The benefit of this approach is that alert filters from CLARAty remove the most prevalent and uninteresting false positives, which effectively improves class distribution in favor of true positives in the alerts passed on to the second stage. Moreover, ALAC receives fewer alerts to process, which is important because of runtime requirements.

In the current prototype, we only evaluated CLARAty and ALAC separately, because the data sets used for evaluation of CLARAty were not labeled and thus could not be used with ALAC. On the other hand, the data sets used for the evaluation of ALAC did not contain a sufficient number of alerts to enable an efficient data mining with CLARAty. An evaluation of the two-stages system is left as future work.

7. Conclusions and Future Work

We showed the global picture of alert management in intrusion detection and presented two orthogonal and complementary approaches to reduce the number of false positives in intrusion detection: CLARAty, based on data mining and root-cause discovery, and ALAC, based on machine learning. We also showed how both techniques fit into standard alert-management systems.

CLARAty is alert-clustering approach using data mining with a modified version of attribute-oriented induction. Using background knowledge in the form of generalization hierarchies for the alert attributes, it analyzes historical alert logs for large clusters of alerts describable by a generalized alert which a human expert can interpret to identify root causes. Experiments with real-world data sets have shown that already few dozens of generalized alerts cover over 90% of the raw alerts. These generalized alerts can indeed be understood as root causes, and therefore be used, through fixing or filtering of these root causes, for subsequent alert reduction (achieving on average a 75% reduction filtering every month).

ALAC is an an adaptive alert classifier based on the feedback of an intrusion detection analyst and machine-learning techniques. We discussed the importance of background knowledge and why the classification of IDS alerts is a difficult machine-learning problem. Finally, we presented a prototype implementation of ALAC and evaluated its performance on a synthetic and a real-world intrusion data set.

ALAC can operate in two modes: a recommender mode, in which all alerts are labeled and passed onto the analyst, and an agent mode, in which some alerts are processed automatically.

We showed that the system is useful in recommender mode, where it adaptively learns the classification from the analyst. In this mode we obtained false negative and false positive rates

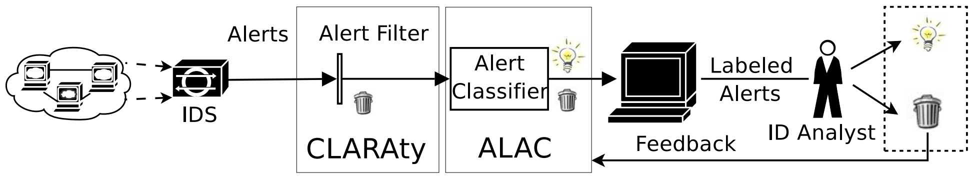

Figure 6. Combining CLARAty and ALAC into a two-staged alert filtering and classification system.

comparable to those of batch classification. We also found that our system is useful in the agent mode, where some alerts are autonomously processed (e.g., false positives classified with high confidence are discarded). More importantly, for both data sets the false negative rate of our system is comparable to that in the recommender mode. At the same time, the number of alerts for the analyst to handle has been reduced by more than 50%.

We also discussed how both CLARAty and ALAC can be used in a two-staged alert classification and filtering system. As future work we are planning to evaluate the performance of such a system to understand the possible interactions and synergies. We are also aware of the limitations of the data sets used with ALAC. We aim to evaluate the performance of the system on the basis of more realistic intrusion detection data and to integrate an alert-correlation system to reduce redundancy in alerts. We are also looking at the background knowledge and how it can be better represented for machine-learning algorithms.

Acknowledgments

The authors would like to thank Klaus Julisch for his readiness and support to place his work on CLARAty into this broader context.

REFERENCES

1. James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co, 1980.
2. Stefan Axelsson. The base-rate fallacy and its implications for the intrusion detection. In *Proceedings of the 6th ACM conference on Computer and Communications Security*, pages 1–7, Kent Ridge Digital Labs, Singapore, 1999.
3. Steven M. Bellowin. Packets Found on an Internet. *Computer Communications Review*, 23(3):26–31, 1993.
4. Eric Bloedorn, Bill Hill, Alan Christiansen, Clem Skorupka, Lisa Talbot, and Jonathan Tivel. Data Mining for Improving Intrusion Detection. Technical report, MITRE Corporation, 2000.
5. William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Proceedings of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, 1995. Morgan Kaufmann.
6. Frédéric Cuppens. Managing alerts in multi-intrusion detection environment. In *Proceedings 17th Annual Computer Security Applications Conference*, pages 22–31, New Orleans, 2001.
7. Frédéric Cuppens, Fabien Autrel, Alexandre Miège, and Salem Benferhat. Correlation in an intrusion detection process. In *Proceedings SÉcurité des Communications sur Internet (SECI02)*, pages 153–171, 2002.
8. Frédéric Cuppens and Alexandre Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–215, 2002.
9. Olivier Dain and Robert K. Cunningham. Fusing a heterogeneous alert stream into scenarios. In *Proc. of the 2001 ACM Workshop on Data Mining for Security Application*, pages 1–13, Philadelphia, PA, 2001.
10. Hervé Debar and Andreas Wespi. Aggregation and correlation of intrusion-detection alerts. In *Recent Advances in Intrusion De-*

tection (RAID2001), volume 2212 of *Lecture Notes in Computer Science*, pages 85–103. Springer-Verlag, 2001.

11. Dorothy E. Denning. An intrusion detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, 1987.
12. Tom Fawcett. ROC graphs: Note and practical considerations for researchers (HPL-2003-4). Technical report, HP Laboratories, 2003.
13. J. Han, Y. Cai, and N. Cercone. Data-Driven Discovery of Quantitative Rules in Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(1):29–40, 1993.
14. Jiawei Han, Yandong Cai, and Nick Cercone. Knowledge Discovery in Databases: An Attribute-Oriented Approach. In *Proceedings 18th International Conference on Very Large Databases (VLDB)*, pages 547–559, Vancouver, Canada, Aug. 1992.
15. IBM. IBM Tivoli Risk Manager. Tivoli Risk Manager User's Guide. Version 4.1, 2002.
16. S. Jha, O. Sheyner, and J. Wing. Two formal analyses of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW 2002)*, pages 49–63, 2002.
17. Klaus Julisch. Mining Alarm Clusters to Improve Alarm Handling Efficiency. In *Proceedings 17th Annual Computer Security Applications Conference*, pages 12–21, New Orleans, LA, Dec. 2001.
18. Klaus Julisch. Clustering intrusion detection alarms to support root cause analysis. *ACM Transactions on Information and System Security (TISSEC)*, 6(4):443–471, 2003.
19. Klaus Julisch. *Using Root Cause Analysis to Handle Intrusion Detection Alarms.* PhD thesis, University of Dortmund, Germany, 2003.
20. Klaus Julisch and Marc Dacier. Mining Intrusion Detection Alarms for Actionable Knowledge. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 366–375, Edmonton, Alberta, Canada, 2002.
21. Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood, 1994.
22. Wenke Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems.* PhD thesis, Columbia University, 1999.
23. Wenke Lee, Wei Fan, Matthew Miller, Salvatore J. Stolfo, and Erez Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1–2):5–22, 2002.
24. Richard Lippmann, Seth Webster, and Douglas Stetson. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 307–326. Springer-Verlag, 2002.
25. Matthew V. Mahoney and Philip K. Chan. An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection. In *Recent Advances in Intrusion Detection (RAID2003)*, volume 2820 of *Lecture Notes in Computer Science*, pages 220–237. Springer-Verlag, 2003.
26. Stefanos Manganaris, Marvin Christensen, Dan Zerkle, and Keith Hermiz. A data mining analysis of RTID alarms. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 34(4):571–577, Oct 2000.
27. Johh McHugh. The 1998 Lincoln Laboratory IDS evaluation. A critique. In *Recent Advances in Intrusion Detection (RAID2000)*, volume 1907 of *Lecture Notes in Computer Science*, pages 145–161. Springer-Verlag, 2000.
28. MIT Lincoln Laboratory. 1999 DARPA Intrusion Detection Evaluation Data Set. Web page at `http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html`, 1999.
29. Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducasse. M2D2: A formal data model for IDS alert correlation. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–137. Springer-Verlag, 2002.

30. Peng Ning and Yun Cui. An intrusion alert correlator based on prerequisities of intrusions (TR-2002-01). Technical report, North Carolina State University, Raleigh, NC, 2002.
31. Peng Ning, Yun Cui, and Douglas S. Reeves. Analyzing intrusion alerts via correlation. In *Recent Advances in Intrusion Detection (RAID2002)*, volume 2516 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
32. Peng Ning, Yun Cui, and Douglas S. Reeves. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 245–254, 2002.
33. Peng Ning, Douglas S. Reeves, and Yun Cui. Correlating alerts using prerequisites of intrusions (TR-2001-13). Technical report, North Carolina State University, Raleigh, NC, 2001.
34. NIST. ICAT Metabase. Web page at `http://icat.nist.gov/`, 2000–2004.
35. Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.
36. Tadeusz Pietraszek. Using adaptive alert classification to reduce false positives in intrusion detection. In *Recent Advances in Intrusion Detection (RAID2004)*, volume 3324 of *Lecture Notes in Computer Science*, pages 102–124, Sophia Antipolis, France, 2004. Springer-Verlag.
37. Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc., 1998.
38. Ronald W. Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2000)*, pages 156–165, 2000.
39. Martin Roesch. SNORT. The Open Source Network Intrusion System. Web page at `http://www.snort.org`, 1998–2003.
40. SecurityFocus. BugTraq. Web page at `http://www.securityfocus.com/bid`, 1998–2004.
41. Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P 2002)*, pages 254–265, 2002.
42. Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM conference on Computer and Communication Security*, pages 262–271, Washington, DC, 2003.
43. K.M. Ting. Inducing cost-sensitive trees via instance weighting. In *Proceedings of The Second European Symposium on Principles of Data Mining and Knowledge Discovery*, volume 1510 of *Lecture Notes in AI*, pages 139–147. Springer-Verlag, 1998.
44. Alfonso Valdes and Keith Skinner. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection (RAID2001)*, volume 2212 of *Lecture Notes in Computer Science*, pages 54–68. Springer-Verlag, 2001.