Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection

Tadeusz Pietraszek

IBM Zurich Research Laboratory
Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
pie@zurich.ibm.com

Abstract. 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 false positives (i.e., alerts mistakenly triggered by benign events). This makes it extremely difficult for the analyst to correctly identify the true positives (i.e., alerts related to attacks).
In this paper we describe ALAC, the Adaptive Learner for Alert Classification, which is a novel system for reducing false positives in intrusion detection. The system supports the human analyst by classifying alerts into true positives and false positives. The knowledge of how to classify alerts is learned adaptively by observing the analyst. Moreover, ALAC can be configured to process autonomously alerts that have been classified with high confidence. For example, ALAC may discard alerts that were classified with high confidence as false positive. That way, ALAC effectively reduces the analyst's workload.
We describe a prototype implementation of ALAC and the choice of a suitable machine learning technique. Moreover, we experimentally validate ALAC and show how it facilitates the analyst's work.
Key words: Intrusion detection, false positives, alert classification, machine learning

1 Introduction

The explosive increase in the number of networked machines and the widespread use of the Internet in organizations has 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. As a result, intrusion detection systems (IDSs), as originally introduced by Anderson [1] and later formalized by Denning [9], 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 [2,4,17,26]. False positives, i.e., alerts that mistakenly indicate security issues and require attention from the intrusion detection analyst, are one of the most important problems faced by intrusion detection today [32]. In fact, it has been estimated that up to 99% of alerts reported by IDSs are not related to security issues [2,4,17]. Reasons for this include the following:

- In many cases an intrusion differs only slightly from normal activities. Sometimes it is only the context in which the activity occurs that determines whether it is intrusive. Owing to harsh real-time requirements, IDSs cannot analyze the context of all activities to the extent required [38].
- 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) and an overly general one (which recognizes legitimate actions as intrusions) can be hard to determine.
- 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.

In this paper we address 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 this paper uses the former approach. Moreover, ALAC learns alert classifiers whose classification logic is explicit 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 works.

ALAC classifies alerts into true positives and false positives and presents these classifications to the intrusion detection analyst, as shown in Fig. 1 on page 6. Based on the analyst's feedback, the system generates training examples, which are used by machine learning techniques to initially build and subsequently update the classifier. The classifier is then used to classify new alerts. This process is continuously repeated to improve the alert classification. At any time the analyst can review the classifier.

Note that 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.

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 often skewed, i.e., false positives are more frequent than true positives. Second, it is also common that the cost of misclassifying alerts is most often asymmetrical i.e., misclassifying true positives as false positives is usually more costly than the other way round. Third, ALAC classifies alerts in real-time and updates its classifier as new alerts become available. The learning technique should be efficient enough to perform in real-time and work 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). In fact, research in machine learning has shown that the use of background knowledge frequently leads to more natural and concise rules [18]. However, the use of background knowledge increases the complexity of a learning task and only some machine learning techniques support it.

We revisit these challenges in Sect. 2.2, where we discuss them and present a suitable learning technique. The point made here is that we are facing a highly challenging machine learning problem that requires great care to solve properly.

1.1 Related Work

To the best of our knowledge, machine learning has not previously been used to incrementally build alert classifiers that take background knowledge into account. However, some of the concepts we apply here have been successfully used in intrusion detection and related domains.

Building IDSs. In intrusion detection, machine learning has been used primarily to build systems that classify network connections (e.g., 1999 KDD CUP [15]) or system call sequences (e.g., [25]) into one of several predefined classes.

This task proved to be very difficult because it aimed at building IDSs only from training examples. Lee [19] developed a methodology to construct additional features using data mining. He also showed the importance of domain-specific knowledge in constructing such IDSs. The key differences of our work is the real-time use of analyst feedback and that we classify alerts generated by IDSs, whereas other researchers used machine learning to build a new IDS.

Fan [12] performed a comprehensive study of cost-sensitive learning using classifier ensembles with RIPPER, therefore his work is particularly relevant to ours. The work differs from ours in design goals: we developed a system to assist human users to classify alerts generated by an IDS, whereas Fan built an IDS using machine learning techniques. We also used a simplified cost model, in order to reduce the number of variable parameters in the system. Finally, the type of learning methods used is also different: ensemble-based learning methods vs. a single classifier in our case.

Alert Classification. The methods used to classify alerts can be divided into two categories: first, methods that identify true positives and second, methods that identify false positives.

Methods that identify true positives have been studied particularly well and can be summarized as follows:

– In environments with multiple IDSs, some methods enhance the confidence of alerts generated by more than one IDS (based on the assumption that the real attack will be noticed by multiple IDSs, whereas false positives tend to be more random) [32],
– Couple sensor alerts with background knowledge to determine whether the attacked system is vulnerable [22,32],
– Create groups of alerts and use heuristics to evaluate whether an alert is a false positive [7,45]. The work by Dain and Cunningham [7] is particularly relevant to us as it uses machine learning techniques: neural networks and decision trees to build a classifier grouping alerts into so-called scenarios. They also discuss domain-specific background knowledge used to discover scenarios. In contrast, our work focuses on alert classification, uses different background knowledge and different machine learning algorithms.

The second category of alert classification methods identifies false positives and can be based on data mining and include root cause analysis [17], or on statistical profiling [26]. For example, Julisch [17] shows that the bulk of alerts triggered by an IDS can be attributed to a small number of root causes. He also proposes a data mining technique to discover and understand these root causes. Knowing the root causes of alerts, one can easily design filters to remove alerts originating from benign root causes. Our work differs from the above in that we use real-time machine learning techniques that take advantage of background knowledge.

Interface agents. The concept of learning from an analyst has been introduced in interface agents [23]. The classic example is Magi, an interface agent that learns how to classify e-mails by observing into which folders the user classifies e-mails [36]. Similar work in e-mail classification has been done by Mizoguchi and Ohwada [31]. Parson and

Moreover, the issues of interface agents that assist the analyst have been investigated in network management and telecommunication networks [11,34].

Our work differs from the above because first, we operate in a different domain (namely intrusion detection, rather than e-mail classification or network management), and second, we use different machine learning techniques, which among other things take advantage of background knowledge.

1.2 Paper Overview

The remainder of this paper is organized as follows. In Section 2 we present the design of our system and analyze machine learning techniques and their limitations with regard to the learning problem we are facing. Section 3 describes the prototype implementation of the system and shows results obtained with synthetic and real intrusion detection data. In Section 4 we present conclusions and future work.

2 ALAC — An Adaptive Learner for Alert Classification

In this section we describe the architecture of the system and contrast it to a conventional setup. We introduce two modes in which the system can operate, namely recommender mode and agent mode. We then focus on machine learning techniques and discuss how suitable they are for alert classification.

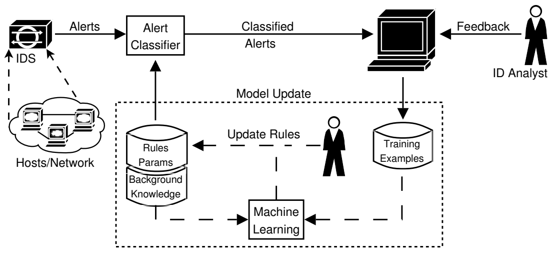
2.1 ALAC Architecture

In a conventional setup, 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. Note that conventional systems may use manual knowledge engineering to build an alert classifier or may use no alert classifier at all. In any case, the conventional setup does not take advantage of the fact that the analyst is analyzing the alerts in real-time: the manual knowledge engineering is separated from analyzing alerts.

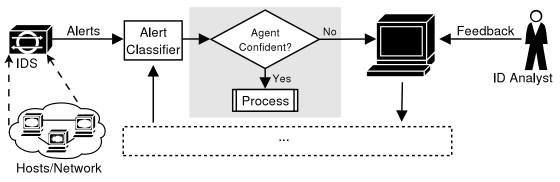
As shown in Fig. 1, our system classifies alerts and passes them to the analyst. It also assigns a *classification confidence* (or *confidence* for short), to alerts, which shows the likelihood of alerts belonging to their assigned classes. The analyst reviews this classification and reclassifies alerts, if necessary. This process is recorded and used as training by the machine learning component to build an improved alert classifier.

Currently we use a simple human-computer interaction model, where the analyst *explicitly* classifies alerts into true and false positives. More sophisticated interaction techniques are possible and will be investigated as part of our future work. In addition to the training examples, we use background knowledge to learn improved classification rules. These rules are then used by ALAC to classify alerts. The analyst can inspect the rules to make sure they are correct.

The architecture presented above describes the operation of the system in *recommender mode*. The second mode, *agent mode*, introduces autonomous processing to reduce the operator's workload.



(a) Recommender mode



(b) Agent mode

Fig. 1. Architecture of ALAC in agent and recommender mode.

In recommender mode (Fig. 1(a)), ALAC classifies alerts and passes all of them to the console to be verified by the analyst. In other words, the system assists the analyst suggesting the correct classification. The advantage for the analyst is that each alert is already preclassified and that the analyst has only to verify its correctness. The analyst can prioritize his or her work, e.g., by dealing with alerts classified as true positives first or sorting the alerts by classification confidence. It is important to emphasize that at the end, the analyst will review all classifications made by the system.

In agent mode (Fig. 1(b)), ALAC autonomously processes some of the alerts based on criteria defined by the analyst (i.e., classification assigned by ALAC and classification confidence). By processing alerts we mean that ALAC executes user-defined actions associated with the class labels and classification confidence values. For example, attacks classified as false positives can be automatically removed, thus reducing the analyst's workload. In contrast, alerts classified as true positives and successful attacks can initiate an automated response, such as reconfiguring a router or firewall. It is important to emphasize that such actions should be executed only for alerts classified with high confidence, whereas the other alerts should still be reviewed by the analyst.

Note that autonomous alert processing may change the behavior of the system and negatively impact its classification accuracy. To illustrate this with an example, suppose the system classifies alerts into true and false positives and it is configured to autonomously discard the latter if the classification confidence is higher than a given threshold value. Suppose the system learned a good classifier and classifies alerts with high confidence. In this case, if the system starts classifying all alerts as false positives then these alerts would be autonomously discarded and would never be seen by the analyst. These alerts would not become training examples and would never be used to improve the classifier.

Another problem is that alerts classified and processed autonomously cannot be added to the list of training examples as the analyst has not reviewed them. If alerts of a certain class are processed autonomously more frequently than alerts belonging to other classes (as in the above example), we effectively change the class distribution in the training examples. This has important implications as machine learning techniques are sensitive to class distribution in training examples. In the optimal case, the distribution of classes in training and testing examples should be identical.

To alleviate these problems, we propose a technique called *random sampling.* In this technique we randomly select a fraction k of alerts which would normally be processed autonomously and instead forward them to the analyst. This ensures the stability of the system. The value of k is a tradeoff between how many alerts will be processed autonomously and how much risk of misclassification is acceptable.

Background Knowledge Representation. 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 [18], 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 [32]. Examples of background knowledge include:

Network Topology. Network topology contains information about the structure of the network, assigned IP addresses, etc. It can be used to better understand the function and role of computers in the network. 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, 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 one, is in the case of some alerts (e.g., portscans, password guessing, repetitive exploits 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 it, however the definition of similarity varies greatly [7,6,44].

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 [41] or ICAT [33].

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 [22]. The result of this process can be used as additional background knowledge used to classify alerts.
Note that the information about the installed software and alert semantics can be used even when alert correlation is not performed, as it allows us to learn rules that make use of generalized concepts such as `OS Linux, OS Windows, etc.`

2.2 Machine Learning Techniques

Until now we have been focusing on the general system architecture and issues specific to intrusion detection. In this section we focus on the machine learning component in our system. Based on the discussion in Sect. 1 and the proposed system architecture, we can formulate the following requirements for the machine learning technique:

1. Learn from training examples (alert classification given by the analyst).
2. Build the classifier whose logic can be interpreted by a human analyst, so its correctness can be verified.
3. Be able to incorporate the background knowledge required.
4. Be efficient enough to perform real-time learning.
5. Be able to assess the confidence of classifications. Confidence is a numerical value attached to the classification, representing how likely it is to be correct.
6. Support cost-sensitive classification and skewed class distributions.
7. Learn incrementally.

Learning an Interpretable Classifier from Examples. The first requirement yields *supervised* machine learning techniques, that is techniques that can learn from training examples. The requirement for an understandable classifier further limits the range of techniques to *symbolic* learning techniques, that is techniques that present the learned concept in a human readable form (e.g., predictive rules, decision trees, Prolog clauses) [25].

Background Knowledge and Efficiency. The ability to incorporate background knowledge differentiates two big groups of symbolic learners: inductive logic programming and symbolic attribute-value learners. In general, inductive logic programming provides the framework for the use of background knowledge, represented in the form of logic predicates and first-order rules, whereas attribute-value learners exclusively learn from training examples. Moreover, training examples for attribute-value learners are limited to a fixed number of attributes.

The inductive logic programming framework can easily handle the background knowledge introduced in Sect. 2.1, including alert context as well as arbitrary Prolog clauses. As the search space is much bigger than in other machine learning techniques, such as rule and decision tree learners, the size of problems that can be solved efficiently by inductive logic programming is smaller and these learners are much less efficient. This may make such a system unsuitable for real-time learning.

On the other hand, attribute-value learners can use a limited form of background knowledge using so-called *feature construction* (also known as propositionalization [18]) by creating additional attributes based on values of existing attributes or existing background knowledge.

Given that most background knowledge for intrusion detection can be converted to additional features using feature construction, and considering the runtime requirement, symbolic attribute-value learners seem to be a good choice for alert classification.

Confidence of Classification. Symbolic attribute-value learners are decision tree learners (e.g., C4.5 [39]) and rule learners (e.g., AQ [29], C4.5rules [39], RIPPER [5]). Both of these techniques can estimate the confidence of a classification based on its performance on training examples. However, it has been shown that rules are much more comprehensible to humans than decision trees [30,39]. Hence, rule learners are particularly advantageous in our context.

We analyzed the characteristics of available rule learners, as well as published results from applications in intrusion detection and related domains. We have not found a good and publicly available rule learner that fulfills all our requirements, in particular cost-sensitivity and incremental learning.

Of the techniques that best fulfill the remaining requirements, we chose 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 [19,20]) as well as related domains and it has proved to produce concise and intuitive rules. As reported by Lee [19], RIPPER rules have two very desirable conditions for intrusion detection: a good generalization accuracy and concise conditions. Another advantage of RIPPER is its efficiency with noisy data sets.

RIPPER has been well documented in the literature and its description is beyond the scope of this paper. However, for the sake of a better understanding of 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 one class only, 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. 3.6.

Unfortunately, the standard RIPPER algorithm is not cost-sensitive and does not support incremental learning. We used the following methods to circumvent these limitations.

Cost-Sensitive Classification and Skewed Class Distribution. Among the various methods of making a classification technique cost-sensitive, we focused on those that are not specific to a particular machine learning technique: Weighting [43] and MetaCost [10]. By changing costs appropriately, these methods can also be used to address the problem of skewed class distribution. These methods produce comparable results, although this can be data dependent [10,27]. Experiments not documented here showed that in our context Weighting gives better run-time performance. Therefore we chose Weighting for our system.

Weighting resamples the training set so that a standard cost-insensitive learning algorithm builds a classifier that optimizes the misclassification cost. The input parameter for Weighting is a cost matrix, which defines the costs of misclassifications for individual class pairs. For a binary classification problem, the cost matrix has only one degree of freedom — the so-called cost ratio. These parameters will be formally defined in Sect. 3.

Incremental Learning. Ours is an incremental learning task which is best solved with an incremental learning technique, but can also be solved with a batch learner [14]. As we did not have a working implementation of a purely incremental rule learner (e.g., AQ11 [29], AQ11-PM [25]) we decided to use a "batch-incremental" approach.

In this approach we add subsequent training examples to the training set and build the classifier using the entire training set as new examples become available. It would not be feasible to rebuild the classifier after each new training example, therefore we handle training examples in batches. The size of such batches can be either constant or dependent on the current performance of the classifier. In our case we focused on the second approach. We evaluate the current classification accuracy and, if it drops below a user-defined threshold, we rebuild the classifier using the entire training set. Note that the weighted accuracy is more suitable than the accuracy measure for cost-sensitive learning. Hence, the parameter controlling "batch-incremental" learning is called the *threshold weighted accuracy*. It will be formally defined in Sect. 3.

The disadvantage of this technique is that the size of the training set grows infinitely during a system's lifetime. A future work item of ours will be to limit the number of training examples to a certain time window and use a technique called partial memory [25] to reduce the number of training examples.

Summary. To summarize, we have not found a publicly available machine learning technique that addresses all our requirements, in particular cost-sensitivity and incremental learning. Considering the remaining requirements the most suitable techniques are rule learners. Based on desirable properties and successful applications in similar domains, we decided to use RIPPER as our rule-learner. To circumvent its limitations with regard to our requirements, we used a technique called Weighting to implement cost-sensitivity and adjust for skewed class distribution. We also implemented incremental learning as a "batch-incremental", approach, whose batch size dependent on the current classification accuracy.

3 Experimental Validation

We have built a prototype implementation of ALAC in recommender and agent mode using the Weka framework [46]. The prototype has been validated with synthetic and real intrusion detection data and we summarize the results obtained in this section.

Similar to the examples used throughout this paper, our prototype focuses on binary classification only, that is on classifying alerts into true and false positives. This does not affect the generality of the system, which can be used in multi-class classification. However, it simplifies the analysis of a system's performance. We have not evaluated the classification performance in a multi-class classification.

So far we have referred to alerts related to attacks as *true positives* and alerts mistakenly triggered by benign events as *false positives.* To avoid confusion with the terms used to evaluate our system, we henceforth refer to true positives as *true alerts* and false positives as *false alerts*, respectively. This allows us to use the terms true and false positives for measuring the quality of the alert classification.

More formally, we introduce a *confusion matrix* C to evaluate the performance of our system. Rows in C represent actual class labels and columns represent class labels assigned by the system. Element $C[i, j]$ represents the number of instances of class i classified as class j by the system. For a binary classification problem, the elements of the matrix are called true positives (tp), false negatives (fn), false positives (fp) and true negatives (tn) as shown in Table 1(a).

For cost-sensitive classification we introduce a *cost matrix* Co with identical meaning of rows and columns. The value of $Co[i, j]$ represents the cost of assigning a class j to an example belonging to class i. Most often the cost of correct classification is zero, i.e., $Co[i, i] = 0$. In such cases, for binary classifications (Table 1(b)), there are only two values in the matrix: c_{21} (cost of misclassifying a false alert as a real one) and c_{12} (cost of misclassifying a true alert as a false one).

classified / actual	$+$	$-$
$+$	tp	fn
$-$	fp	tn

(a) Confusion matrix C

classified / actual	$+$	$-$
$+$	0	c_{12}
$-$	c_{21}	0

(b) Cost matrix Co

Table 1. Confusion and cost matrices for alert classification. The positive class ($+$) denotes true alerts and the negative class ($-$) denotes false alerts. The columns represent classes assigned given by the system; the rows represent actual classes.

In the remainder of the paper we use the following measures defined on cost and confusion matrices: true positive rate (TP), false positive rate (FP), false negative rate (FN). We also use cost ratio (CR), which represents the ratio of the misclassification cost of false positives to false negatives, and its inverse — inverse cost ratio (ICR), which we found more intuitive for intrusion detection. For cost-sensitive classification we used a commonly used evaluation measure — so-called weighted accuracy (WA). Weighted accuracy expresses the accuracy of the classification with misclassifications weighted by their misclassification cost.

$$TP = \frac{tp}{tp + fn}, \quad FP = \frac{fp}{fp + tn}, \quad FN = \frac{fn}{tp + fn}$$

$$CR = \frac{c_{21}}{c_{12}}, \quad ICR = \frac{1}{CR}, \quad WA = \frac{CR \cdot tn + tp}{CR \cdot (tn + fp) + tp + fn}$$

3.1 Data Sources

We used Snort [40] — an open-source network–based IDS — to detect attacks and generate alerts. We purposely used the basic out-of-the box configuration and rule set to demonstrate the performance of ALAC in reducing the amount of false positives and therefore reducing time-consuming IDS tuning.

Snort was run on two data sources: a synthetic one known as DARPA 1999 [21], and a real-world one — namely the traffic observed in a medium-sized corporate network (called *Data Set B*). Alerts are represented as tuples of attribute values, with the following seven attributes: signature name, source and destination IP addresses, a flag indicating whether an alert is a scan, number of scanned hosts and ports and the scan time.

DARPA 1999 Data Set is a synthetic data set collected from a simulated medium-sized computer network in a fictitious military base. The network was connected to the outside world by a router. The router was set to open policy, i.e., not blocking any connections. The simulation was run for 5 weeks which yielded three weeks of training data and two weeks of testing data. Attack truth tables describing the attacks that took place exist for both periods. DARPA 1999 data consists of: two sets of network traffic (files with tcpdump [16] data) both inside and outside the router, BSM and NT audit data, and directory listings.

In our experiments we ran Snort in batch mode using traffic collected from outside the router for both training and testing periods. Note that Snort missed some of the attacks in this dataset. Some of them could only be detected using host-based intrusion detection, whereas for others Snort simply did not have the signature. It is important to note that our goal was not to evaluate the detection rate of Snort on this data, but to validate our system in a realistic environment.

The DARPA 1999 data set has many well-known weaknesses (e.g., [24,28]) and we want to make sure that using it we get representative results for how ALAC performs in real-world environments. To make this point we analyze how the weaknesses identified by McHugh [28], namely the generation of attack and background traffic, the amount of training data for anomaly based systems, attack taxonomy and the use of ROC analysis; can affect ALAC.

With respect to the training and test data, we use both training and test data for the incremental learning of ALAC, so that we have sufficient data to train the system. With respect to attack taxonomy, we are not using the scoring used in the original evaluation, and therefore attack taxonomy is of less significance. Finally, we use ROC analysis correctly.

The problem of the simulation artifacts is more thoroughly analyzed by Mahoney and Chan [24] thus we use their work to understand how these artifacts can affect ALAC. These artifacts manifest themselves in various fields, such as the TCP and IP headers and higher protocol data. Snort, as a signature based system, does not take advantage of these artifacts and ALAC sees only a small subset of them, namely the source IP address. We verified that the rules learned by ALAC seldom contain a source IP address and therefore the system does not take advantage of simulation artifacts present in source IP addresses. On the other hand, we cannot easily estimate how these regularities affect aggregates used in the background knowledge. This is still an open issue.

We think that the proper analysis of these issues is beyond the scope of our work, and would also require comparing multiple real-world data sets. DARPA1999 data set is nonetheless valuable for evaluation of our research prototype.

Data Set B is a real-world data set collected over the 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 as a second validation of our system.

3.2 Alert Labeling

Our system assumes that alerts are labeled by the analyst. In this section we explain how we labeled alerts used to evaluate the system (the statistics for both datasets are shown in Table 2).

	DARPA 1999	Data Set B
Duration of experiment:	5 weeks	1 month
Number of IDS alerts:	59812	47099
False alerts:	48103	33220
True alerts:	11709	13383
Unidentified:	—	496

Table 2. Statistics generated by the Snort sensor with DARPA 1999 data set and Data set B.

DARPA 1999 Data Set. In a first step we generated alerts using Snort running in batch mode and writing alerts into a relational database. In the second step we used automatic labeling of IDS alerts using the provided attack truth tables.

For labeling, we used an automatic approach which can be easily reproduced by researchers in other environments, even with different IDS sensors. We consider all alerts meeting the following criteria related to an attack: (i) matching source IP address, (ii) matching destination IP address and (iii) alert time stamp in the time window in which the attack has occurred. We masked all remaining alerts as false alerts. While manually reviewing the alerts we found that, in many cases, the classification is ambiguous (e.g., a benign PING alert can be as well classified as malicious if it is sent to the host being attacked). This may introduce an error in class labels.

Note that different attacks triggered a different number of alerts (e.g., wide network scans triggered thousands of alerts). For the evaluation of our system we discarded the information regarding which alerts belong to which attack and labeled all these alerts as true alerts.

Data Set B. We generated these alerts in real-time using Snort. As opposed to the first data set we did not have information concerning attacks. The alerts have been classified based on the author's expertise in intrusion detection into groups indicating possible type and cause of the alert. There was also a certain number of alerts that could not be classified into true or false positives. Similarly to the first data set we used only binary classification to evaluate the system, and labeled the unidentified alerts as true positives.

Note that this data set was collected in a well maintained and well protected network with no direct Internet access. We observed a low number of attacks in this network, but many alerts were generated. We observed that large groups of alerts can be explained by events such as a single worm infection and unauthorized network scans. The problem of removing such redundancy can be solved by so-called *alert correlation systems* [6,8,44], where a group of alerts can be replaced by a meta-alert representative of the alerts in the group, prior to classification. The topic of alert correlation is beyond the scope of this paper and will be addressed as a future work item.

Another issue is that the classification of alerts was done by only one analyst and therefore may contain errors. This raises the question of how such classification errors affect the performance of ALAC. To address this issue, one can ask multiple analysts to classify the dataset independently. Then the results can be compared using interrater reliability analysis.

3.3 Background Knowledge

We decided to focus on the first two types of background knowledge presented in Sect. 2.1, 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. [22].

As discussed in Sect. 3.1, we used an attribute-value representation of alarms with the background knowledge represented as additional attributes. Specifically, the background knowledge resulted in 19 attributes, which are calculated as follows:

Classification of IP addresses resulted in an additional attribute for both source and destination IP classifying machines according to their known subnets (e.g., Internet, intranet, DMZ).

Classification of hosts resulted in additional attributes indicating the operating system and the host type for known IP addresses.

Aggregates1 resulted in additional attributes with the number of alerts in the following categories (we calculated these aggregates for alerts in a time window of 1 minute):
- alerts with the same source IP address,
- alerts with the same destination IP address,
- alerts with the same source or destination IP address,
- alerts with the same signature,
- alerts classified as intrusions.

Aggregates2,3 were calculated similarly to the first set of attributes, but in time windows of 5 and 30 minutes, respectively.

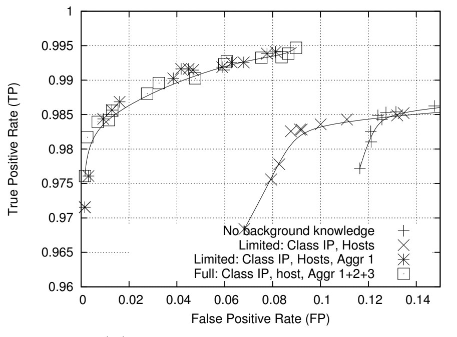
This choice of background knowledge, which was motivated by heuristics used in alert correlation systems, is necessarily a bit *ad-hoc* and reflects the author's expertise in classifying IDS attacks. As this background knowledge is not especially tailored to training data, it is natural to ask how useful it is for alert classification. We discuss the answer to this question in the following sections.

3.4 Results Obtained with DARPA 1999 Data Set

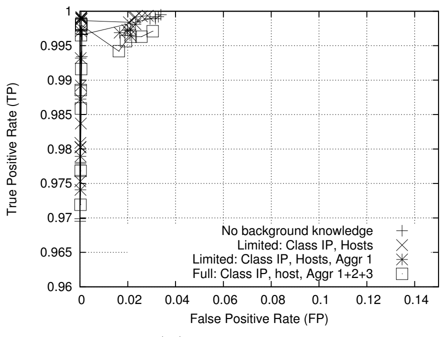
Our experiments were conducted in two stages. In the first stage we evaluated the performance of the classifier and the influence of adding background knowledge to alerts on the accuracy of classification. The results presented here allowed us to set some parameters in ALAC. In the second stage we evaluated the performance of ALAC in recommender and agent mode.

Background Knowledge and Setting ALAC Parameters. Here we describe the results of experiments conducted to evaluate background knowledge and to set ALAC parameters. Note that in the experiments we used only the machine learning component of ALAC, namely a RIPPER module, to build classifiers for the entire data set. Hereafter we refer to these results as *batch classification.*

Since the behavior of classifiers depends on the assigned costs, we used ROC (Receiver Operating Characteristic) analysis [37] to evaluate the performance of our classifier for different misclassification costs. Figure 2(a) shows the performance of the classifier using data with different amounts of background knowledge. Each curve was plotted by varying the cost ratio for the classifier. Each point in the curve represents results obtained from 10-fold cross validation for a given misclassification cost and type of background knowledge.



(a) DARPA1999 data set



(b) Data set B

Fig. 2. ROC curves for classifier used with different types of background knowledge.

As we expected, the classifier with no background knowledge (plus series) performs worse than the classifier with simple classifications of IP addresses and operating systems running on the machines (cross series) in terms of false positives. Using the background knowledge consisting of the classifications above and aggregates introduced in Sect. 3.3 significantly reduces the false positive rate and increases the true positive rate (star series). Full background knowledge (having additional aggregates in multiple time windows) performs comparably to the reduced one (star vs. box series). In our experiments with ALAC we decided to use full background knowledge.

ROC curves show the performance of the system under different misclassification costs, but they do not show how the curve was built. Recall from Sect. 2.2 that we use the inverse cost ratio in Weighting to make RIPPER cost sensitive and varied this parameter to obtain a multiple points on the curve. We used this curve to select good parameters of our model.

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 and class distribution [13]. 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 $ICR = 50$.

The second parameter is the threshold weighted accuracy (WA) for rebuilding the classifier (see Sect. 2.2). The value of threshold weighted accuracy should be chosen carefully as it represents a tradeoff between classification accuracy and how frequently the machine learning algorithm is run. We chose the value equal to the accuracy of a classifier in batch mode. Experiments not documented here showed that using higher values increases the learning frequency with no significant improvement in classification accuracy.

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 input data to build the initial classifier and the remaining 70% to evaluate the system.

ALAC in Recommender Mode. In recommender mode the analyst reviews each alert and corrects ALAC misclassifications. We plotted the number of misclassifications: false positives (Fig. 3(a)) and false negatives (Fig. 3(b)) 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 ($FN = 0.0076$) as shown in Fig. 2(a). At the same time, the overall false positive rate ($FP = 0.025$) is less than half of the false positive rate for batch classification ($FP = 0.06$). These differences are expected due to 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.

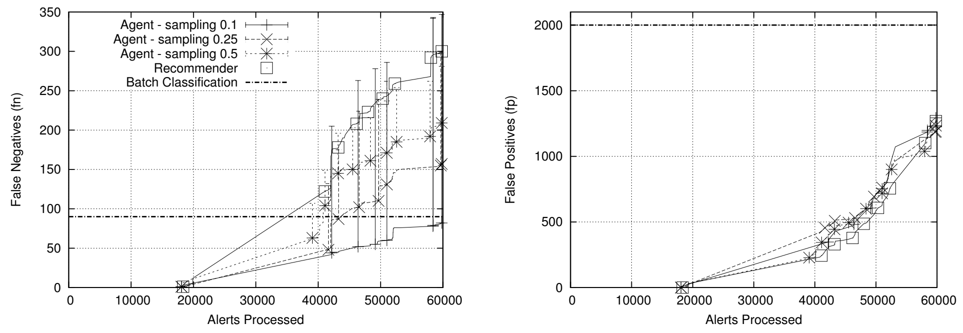


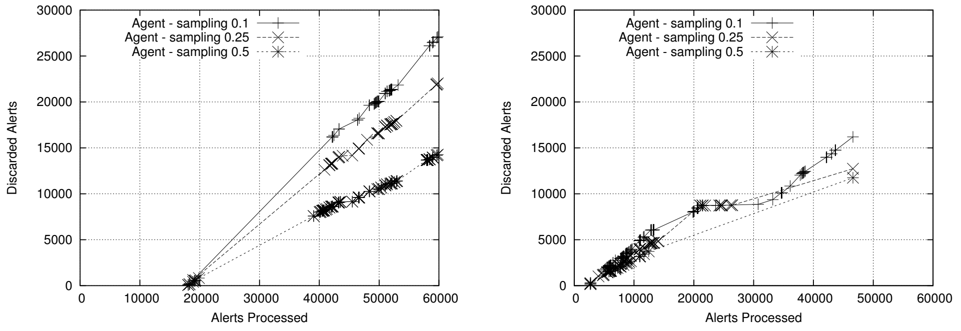
Fig. 3. False negatives and false positives for ALAC in agent and recommender modes (DARPA1999 data set, ICR=50).

ALAC in Agent Mode. In agent mode ALAC processes alerts autonomously based on criteria defined by the analyst, described in Sect. 2.1. We configured the system to forward all alerts classified as true alerts and false alerts classified with low confidence ($confidence < c_{th}$) 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 sampling rates k. We then chose $c_{th} = 90\%$ 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 significantly changes the class distribution in the training examples. On the other hand, with sampling rates much higher than 0.5, the system works similarly to recommender mode and is less useful for the analyst.

Notice 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 plotted the second type of misclassification as an error bar in Fig. 3(a). Intuitively with lower sampling rates, the agent will have fewer false negatives of the first type, in fact missing more alerts. As expected the total number of false negatives is lower with higher sampling rates.

We were surprised to observe that the recommender and the agent have similar false positive rates ($FP = 0.025$ for both cases) and similar false negative rates, even with low sampling rates ($FN = 0.026$ for $k = 0.25$ vs. $FN = 0.025$). This seemingly counterintuitive result can be explained if we note that automatic processing of alerts classified as false positives effectively changes the class distribution in training examples in favor of true alerts. As a result the agent performs comparably to the recommender.



(a) DARPA1999 data set, ICR=50 (b) Data set B, ICR=50

Fig. 4. Number of alerts processed autonomously by ALAC in agent mode.

As shown in Fig. 4(a), with the sampling rate of 0.25, more than 45% of false alerts were processed and discarded by ALAC. At the same time the number of unnoticed false negatives is half the number of mistakes for 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.

3.5 Results Obtained with Data Set B

We used the second dataset as an independent validation of the system. To avoid "fitting the model to the data" we used the same set of parameters as for the first data set. However, a ROC curve in Fig. 2(b) shows that the classifier achieves much higher true positive rate and much lower false negative rate than for the first data set, which means that Data Set B is easier to classify. The likely explanation of this fact is that Data Set B contains fewer intrusions and more redundancy than the first data set.

Notice that the ROC curve consists of two distinct parts. An analysis shows that the left part corresponds to RIPPER run for small ICRs, where it learns the rules describing true alerts. The right part of the curve corresponds to high ICRs, where RIPPER learns the rules describing false alerts. Better performance in the first case can be explained by the fact that the intrusions in this data set are more structured and therefore easier to learn. On the other hand, false alerts are more difficult to describe and hence the performance is poorer.

Background Knowledge and Setting ALAC Parameters. Results with ROC analysis (Fig. 2(b)) show that the classifier correctly classifies most of the examples, and adding background knowledge has little effect on classification. To have the same conditions as with the first data set, we nonetheless decided to use the full background knowledge. We also noticed that $ICR = 50$ is not the optimal value for this dataset as it results in a high false positive rate ($FN = 0.002, FP = 0.05$).

We observed that ALAC, when run with 30% of the alerts as an initial classifier, classified the remaining alerts with very few learning runs. Therefore, to demonstrate its incremental learning capabilities, we decided to lower the initial amount of training data from 30% to 5% of all the alerts.

ALAC in Recommender Mode. Figure 5 shows that in recommender mode the system has a much lower overall false negative rate ($FN = 0.0045$) and a higher overall false positive rate ($FP = 0.10$) than for DARPA 1999 data set, which is comparable to the results of the classification in batch mode. We also observed that the learning only took place for approximately the first 30% of the entire data set and the classifier classified the remaining alerts with no additional learning. This phenomena can also be explained by the fact that Data Set B contains more regularities and the classifier is easier to build.

This is different in the case of the DARPA1999 data set, where the classifier was frequently rebuilt in the last 30% of the data. For DARPA1999 data set the behavior of ALAC is explained by the fact that most of the intrusions actually took place in the last two weeks of the experiment.

Note that the high number of false positives can be attributed to too high value of ICR. Experiments not documented here showed that changing ICR to lower values and increasing threshold accuracy to values higher than 99% improves the results with regard to false positives.

ALAC in Agent Mode. In agent mode we obtained results similar to those in recommender mode, with a great number of alerts being processed autonomously by the system ($FN = 0.0065, FP = 0.13$). As shown in Fig. 4(b), with the sampling rate of 0.25, more than 27% of all alerts were processed by the agent. At the same time the actual number of unnoticed false negatives is one third smaller than the number of false negatives in recommender mode. This confirms the usefulness of the system tested with an independent data set.

Similarly to observation in Sect. 3.4 with lower sampling rates, the agent will have seemingly fewer false negatives, in fact missing more alerts. As expected the total number of false negatives is lower with higher sampling rates. This effect is not as clearly visible as with DARPA1999 data set.

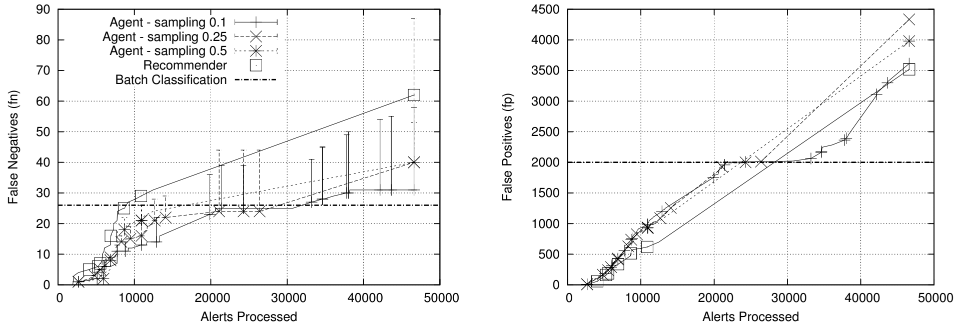


Fig. 5. False negatives and false positives for ALAC in agent and recommender modes (Data set B, ICR=50).

3.6 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 less than six in the last 30 minutes (window `w3`), there have been no intrusions in the last 5 minutes (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 has not been any 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 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.

4 Conclusions and Future Work

We presented a novel concept of building an adaptive alert classifier based on an intrusion detection analyst's feedback using machine learning techniques. We discussed the issues of human feedback and background knowledge, and reviewed machine learning techniques suitable for alert classification. Finally, we presented a prototype implementation and evaluated its performance on synthetic as well as real intrusion data.

We showed that background knowledge is useful for alert classification. The results were particularly clear for the DARPA 1999 data set. For the real-world dataset, adding background knowledge had little impact on the classification accuracy. The second set was much easier to classify, even with no background knowledge. Hence, we did not expect improvement from background knowledge in this case. We also showed that the system is useful in recommender mode, where it adaptively learns the classification from the analyst. For both datasets we obtained false negative and false positive rates comparable to batch classification. Note that in recommender mode all system misclassifications would have been corrected by the analyst.

In addition, we found that our system is useful in 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 false positives has been reduced by approximately 30%.

The system has a few numeric parameters that influence its performance and should be adjusted depending on the input data. In the future, we intend to investigate how the value of these parameters can be automatically determined. We are also aware of the limitations of the data sets used. 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. Our system uses RIPPER, a noise-tolerant algorithm, but the extent to which ALAC can tolerate errors in the data, is currently unknown. We will address this issue by introducing an artificial error and observing how it affects the system.

The topic of learning comprehensible rules is very interesting and we plan to investigate it further. We are currently looking at learning multiple classifiers for each signature and using RIPPER in unordered rule set mode.

In the machine learning part we intend to focus on the development of incremental machine learning technique suitable for learning a classifier for intrusion detection. Initially we want to perform experiments with partial memory technique and batch classifiers. Later we will focus on truly incremental techniques. It is important that such techniques be able to incorporate the required background knowledge.

Acknowledgments

Many thanks to Klaus Julisch and Anderas Wespi for their contribution to the system and valuable comments. Thanks also goes to other members of the Global Security Analysis Laboratory and anonymous reviewers for their insightful remarks.

References

1. Anderson, J.P.: Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co (1980).
2. Axelsson, S.: The base-rate fallacy and its implications for the intrusion detection. In: Proceedings of the 6th ACM conference on Computer and Communications Security, Kent Ridge Digital Labs, Singapore (1999) 1–7.
3. Bellowin, S.M.: Packets Found on an Internet. Computer Communications Review **23** (1993) 26–31.
4. Bloedorn, E., Hill, B., Christiansen, A., Skorupka, C., Talbot, L., Tivel, J.: Data Mining for Improving Intrusion Detection. Technical report, MITRE (2000).
5. Cohen, W.W.: Fast effective rule induction. In Prieditis, A., Russell, S., eds.: Proceedings of the 12th International Conference on Machine Learning, Tahoe City, CA, Morgan Kaufmann (1995) 115–123.
6. Cuppens, F.: Managing alerts in multi-intrusion detection environment. In: Proceedings 17th Annual Computer Security Applications Conference, New Orleans (2001) 22–31.
7. Dain, O., Cunningham, R.K.: Fusing a heterogeneous alert stream into scenarios. In: Proc. of the 2001 ACM Workshop on Data Mining for Security Application, Philadelphia, PA (2001) 1–13.
8. Debar, H., Wespi, A.: Aggregation and correlation of intrusion-detection alerts. In: Recent Advances in Intrusion Detection (RAID2001). Volume 2212 of Lecture Notes in Computer Science., Springer-Verlag (2001) 85–103.
9. Denning, D.E.: An intrusion detection model. IEEE Transactions on Software Engineering **SE-13** (1987) 222–232.
10. Domingos, P.: Metacost: A General Method for Making Classifiers Cost-Sensitive. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, California (1999) 155–164.
11. Esfandiari, B., Deflandre, G., Quinqueton, J.: An interface agent for network supervision. In: Proceedings of the ECAI-96 Workshop on Intelligent Agents for Telecom Applications, Budapest, Hungary (1996).
12. Fan, W.: Cost-Sensitive, Scalable and Adaptive Learning Using Ensemble-based Methods. PhD thesis, Columbia University (2001).
13. Fawcett, T.: ROC graphs: Note and practical considerations for researchers (HPL-2003-4). Technical report, HP Laboratories (2003).
14. Giraud-Carrier, C.: A Note on the Utility of Incremental Learning. AI Communications **13** (2000) 215–223.
15. Hettich, S., Bay, S.D.: The UCI KDD Archive. Web page at `http://kdd.ics.uci.edu` (1999).
16. Jacobson, V., Leres, C., McCanne, S.: TCPDUMP public repository. Web page at `http://www.tcpdump.org/` (2003).
17. Julisch, K.: Using Root Cause Analysis to Handle Intrusion Detection Alarms. PhD thesis, University of Dortmund (2003).
18. Lavrač, N., Džeroski, S.: Inductive Logic Programming: Techniques and Applications. Ellis Horwood (1994).
19. Lee, W.: A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems. PhD thesis, Columbia University (1999).
20. Lee, W., Fan, W., Miller, M., Stolfo, S.J., Zadok, E.: Toward cost-sensitive modeling for intrusion detection and response. Journal of Computer Security **10** (2002) 5–22.

21. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 DARPA Off-Line Intrusion Detection Evaluation. Computer Networks: The International Journal of Computer and Telecommunications Networking **34** (2000) 579–595.
22. Lippmann, R., Webster, S., Stetson, D.: 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., Springer-Verlag (2002) 307–326.
23. Maes, P., Kozierok, R.: Learning interface agents. In: Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93), Washington, DC (1993) 459–465.
24. Mahoney, M.V., Chan, P.K.: 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., Springer-Verlag (2003) 220–237.
25. Maloof, M.A., Michalski, R.S.: Incremental learning with partial instance memory. In: Proceedings of Foundations of Intelligent Systems: 13th International Symposium, ISMIS 2002. Volume 2366 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2002) 16–27.
26. Manganaris, S., Christensen, M., Zerkle, D., Hermiz, K.: A Data Mining Analysis of RTID Alarms. Computer Networks: The International Journal of Computer and Telecommunications Networking **34** (2000) 571–577.
27. María, J., Hidalgo, G.: Evaluating cost-sensitive unsolicited bulk email categorization. In: Proceedings of the 2002 ACM Symposium on Applied Computing, Springer-Verlag (2002) 615–620.
28. McHugh, J.: The 1998 Lincoln Laboratory IDS Evaluation. A critique. In: Recent Advances in Intrusion Detection (RAID2000). Volume 1907 of Lecture Notes in Computer Science., Springer-Verlag (2000) 145–161.
29. Michalski, R.: On the quasi-minimal solution of the general covering problem. In: Proceedings of the V International Symposium on Information Processing (FCIP 69)(Switching Circuits). Volume A3., Yugoslavia, Bled (1969) 125–128.
30. Mitchel, T.M.: Machine Learning. Mc Graw Hill (1997).
31. Mizoguchi, F., Ohwada, H.: Personalized mail agent using inductive logic programming. In Furukawa, K., Michie, D., Muggleton, S., eds.: Machine Intelligence 15. Intelligent Agents. Oxford University Press (1999) 154–175.
32. Morin, B., Mé, L., Debar, H., Ducasse, M.: M2D2: A formal data model for IDS alert correlation. In: Recent Advances in Intrusion Detection (RAID2002). Volume 2516 of Lecture Notes in Computer Science., Springer-Verlag (2002) 115–137.
33. NIST: ICAT Metabase. Web page at `http://icat.nist.gov/` (2000–2003).
34. Nock, R., Esfandiari, B.: Oracles and assistants: Machine learning applied to network supervision. In: Proceedings of the 12th in Canadian Conference on Artificial Intelligence. Volume 1418 of Lecture Notes in Computer Science., Springer-Verlag (1998) 86–.
35. Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks **31** (1999) 2435–2463.
36. Payne, T.R., Edwards, P., Green, C.L.: Experience with Rule Induction and k-Nearest Neighbor Methods for Interface Agents that Learn. IEEE Transactions on Knowledge and Data Engineering **9** (1997) 329–335.
37. Provost, F., Fawcett, T.: Robust classification for impresice environments. Machine Learning Journal **42** (2001) 203–231.
38. Ptacek, T.H., Newsham, T.N.: Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks Inc. (1998).

39. Quinlan, R.: C4.5: Programs for Machine Learning. Morgan Kaufman (1993).
40. Roesch, M.: SNORT. The Open Source Network Intrusion System. Web page at `http://www.snort.org` (1998–2003).
41. SecurityFocus: BugTraq. Web page at `http://www.securityfocus.com/bid` (1998–2003).
42. Sommer, R., Paxson, V.: Enhancing Byte-Level Network Intrusion Detection Signatures with Context. In: Proceedings of the 10th ACM conference on Computer and Communication Security, Washington, DC (2003) 262–271.
43. Ting, K.: 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., Springer-Verlag (1998) 139–147.
44. Valdes, A., Skinner, K.: Probabilistic alert correlation. In: Recent Advances in Intrusion Detection (RAID2001). Volume 2212 of Lecture Notes in Computer Science., Springer-Verlag (2001) 54–68.
45. Wang, J., Lee, I.: Measuring false-positive by automated real-time correlated hacking behavior analysis. In: Information Security 4th International Conference. Volume 2200 of Lecture Notes in Computer Science., Springer-Verlag (2001) 512–.
46. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools with Java implementations. Morgan Kaufmann, San Francisco (2000).