Toward Cost-Sensitive Modeling for Intrusion Detection

Wenke Lee
Computer Science Department
North Carolina State University
Raleigh, NC 27695-7550
wenke@csc.ncsu.edu

Matthew Miller, Sal Stolfo, Kahil Jallad, Christopher Park, Erez Zadok, Vijay Prabhakar
Computer Science Department
Columbia University, Mail Code 0401
1214 Amsterdam Avenue
New York, NY 10027
{mlm46,sal,kdj4,cpark,ezk,weasel}@cs.columbia.edu

Abstract

Intrusion detection systems need to maximize security while minimizing costs. In this paper, we study the problem of building cost-sensitive intrusion detection models. We examine the major cost factors: development costs, operational costs, damage costs incurred due to intrusions, and the costs involved in responding to intrusions. We propose cost-sensitive machine learning techniques to produce models that are optimized for user-defined cost metrics. We describe an automated approach for generating efficient run-time versions of these models. Empirical experiments in off-line analysis and real-time detection show that our cost-sensitive modeling and deployment techniques are effective in reducing the overall cost of intrusion detection.

1 Introduction

Intrusion Detection (ID) is an important component of infrastructure protection mechanisms. Many intrusion detection systems (IDSs) are emerging in the market place, following research and development efforts in the past two decades. They are, however, far from the ideal security solutions for customers. Lack of accuracy due to low detection rate and high false alarm rate is still a major shortcoming of IDSs. Even the more accurate IDSs often waste time and resources to detect and respond to attacks that the customers aren't concerned with because the potential damage is so small; they overload security staff and increase the maintenance and expense of IDSs.

Investment in IDSs should bring the highest possible benefit. IDSs need to maximize user-defined security goals while minimizing costs. This requires the ID models to be sensitive to cost factors, that at the minimum should include the development cost, the operational cost (i.e., the needed resources), the cost of damages of an intrusion, and the cost of detecting and responding to a potential intrusion. Currently these cost factors are ignored as unwanted complexities in the development process of IDSs because developing an effective and accurate IDS is already challenging given the complexities of today's network environments and the manual effort of knowledge-engineering approaches (e.g., encoding expert rules). The cost factors are not sufficiently considered in the deployment of IDSs either because many organizations are not educated about the cost benefits of security systems, and analyzing site-specific cost factors is very difficult. Therefore, we believe that the community as a whole needs to study the cost-sensitive aspects of IDSs to help make intrusion detection a more successful technology.

We developed a data mining framework for building intrusion detection models in an effort to automate the process of IDS development and lower its development cost. The framework uses data mining algorithms to compute activity patterns and extract predictive features, and then it applies machine learning algorithms to generate detection rules [8, 9]. Results from the 1998 DARPA Intrusion Detection Evaluation showed that our ID model was one of the best performing of all the participating systems, most of which were knowledge-engineered [9, 7].

In this paper, we report the initial results of our current research in extending our data mining framework to build cost-sensitive models for intrusion detection. We examine the relevant cost factors, cost models, and cost metrics related to IDSs. We propose to use cost-sensitive machine learning techniques that can automatically construct detection models optimized for the overall cost metrics instead of relying solely on statistical accuracy. We are not suggesting that accuracy is not an important goal, but rather that cost factors need to be included in the process of developing and evaluating IDSs.

We are also developing automatic translation systems to convert our models, which are learned off-line, into modules of programmable real-time IDSs (e.g., Bro [14] and NFR [12]). Although cost-sensitive learning algorithms already produce models with low computational costs, they can be improved. For example, each intrusion can have low-cost *necessary conditions* associated with it. Checking these necessary conditions and eliminating the need for checking a large portion of the detection rules can reduce overall operational costs of an IDS.

The rest of the paper is organized as follows. Section 2 examines the major cost factors related to IDSs, and the problem of modeling and measuring the relationship among these factors. Section 3 describes the cost-sensitive version of the RIPPER rule learning algorithm [4] which produces low computational cost detection rules, a MetaCost [5] procedure that can be used to minimize damage costs and response costs, and our experiments in using these algorithms. Section 4 describes our rule translation system, and the experiments in implementing and comparing various cost-sensitive models using Bro and NFR. Section 5 compares our research with related efforts. Section 6 outlines future research directions.

2 Cost Factors, Models, and Metrics in IDSs

In order to build cost-sensitive models, we first analyze the cost factors related to IDSs. Borrowing ideas from related fields of credit card and cellular phone fraud detection, we identify the following major types of costs: damage costs, response costs, and operational costs.

2.1 Damage Costs

The damage cost characterizes the maximum amount of damage inflicted by an attack when intrusion detection is unavailable or not completely effective. This cost is important but difficult to define since it is likely a function of the risks that need to be analyzed by a site that seeks to protect itself. The defined cost function per attack (or attack type) and per service (e.g., http, ftp, email) should be used to measure the cost of damage. Rather than simply measuring false negatives (*FN*) as a rate of missed intrusions, we should measure the total damage loss of all missed attacks, because different intrusions cost differently, and the goal of an IDS is to minimize damage to the business.

2.2 Response Costs

The response cost is the cost to act upon an alarm that indicates a potential intrusion. For an IDS, one might consider dropping or suspending a suspicious connection and attempting to check, by analyzing the service request, if any system data was compromised, or if any system resources were abused by attackers or blocked from other legitimate users. Other costs can be included, such as the time spent by personnel gathering evidence for prosecution purposes if the intruder can be traced. These costs can be estimated, as a first approximation, by the amount of CPU and disk resources needed to respond to a suspicious connection. For simplicity, instead of estimating the response cost for each intrusive connection, we assign a typical, averaged, response cost for each specific *type* of attack.

Response cost and damage cost form the basis for sites deciding what counts as an intrusion and thus needs to be reported and acted upon. Since there is trade-off between costs and benefits, intrusions that have lower damage cost than response cost could be ignored[1]. For example, many businesses, consider port-scanning a harmless attack that need not be detected or reported.

2.3 Operational Costs

The operational cost is the cost of running an IDS. The main cost here is the amount of resources needed to extract and test features from raw traffic data[2]. Some features cost more to gather than others. However, costlier features are often more informative for detecting intrusions. For example, to detect accurately some denial-of-service attacks, an IDS needs to compute and examine some temporal and statistical features of active connections. Computing such features requires the storage and look-up of potentially many active connections, and is more costly than just examining features of a single connection.

An IDS should detect an on-going attack and generate an alarm as quickly as possible so that damage could be minimized. A slower IDS which uses features with higher computational costs should therefore be penalized. Next, we consider the trade-off between accuracy and operational cost.

2.4 Cost Models

The cost model of an IDS formulates the total expected cost of the IDS. The cost model depends on the detection performance of the IDS. We examine the costs associated with each of these outcomes: false negative (FN), false positive (FP), true positive (TP), true negative (TN), and misclassified hits.

FN Cost is the cost of not detecting an attack, and represents the most dangerous case. It is incurred by most systems today that do not field IDSs. Here, the IDS falsely decides that a connection is not an attack and does not respond to the attack. This indicates that the attack will succeed and some service or data might be lost. The FN Cost is therefore defined as the damage cost associated with the particular type of attack a, DCost(a).

TP Cost is the cost of detecting an attack and doing something about it (i.e., challenging it). Here, one hopes to stop an attack from damaging the service. In the case of a correctly classified attack, the response cost, *RCost*, must also be considered. When some event triggers an IDS to predict correctly that a true attack is underway, or has happened, we are faced with several choices. If the cost to respond to the attack is RCost, but the attack affected a service whose value is less than RCost, then ignoring these attacks reduces costs. Therefore, for a true positive, if RCost$(a) \geq$ DCost(a), the intrusion is not acted upon and the loss is DCost(a), but if RCost$(a) <$ DCost(a), the intrusion is acted upon and the loss is limited to RCost(a). In reality, by the time enough evidence is gathered and analyzed by an IDS, some damage may have incurred. In such cases, the TP cost is RCost$(a) + \epsilon$DCost(a), where $0 \leq \epsilon \leq 1$. RCost is actually related to DCost, and we can also associate another ϵ_r with RCost. For simplicity, we fix RCost to its maximum cost ($\epsilon_r = 1$).

FP Cost. When an IDS falsely determines an event as an attack, and the attack type is regarded as high cost, a response will ensue. We pay the response cost, RCost. In addition, false alarms can be penalized using various measures. For our discussion, we use PCost(c) to represent the penalty cost of treating a legitimate connection c as an intrusion. For example, if c is terminated, PCost(c) can have the same cost as denial-of-service (DOS) because the service is denied from a legitimate user.

TN Cost. An IDS correctly decides that a connection is normal and not an attack. We therefore bare no cost that is dependent on the action of an the IDS.

[1] These intrusions can simply be recorded in a log file for future reference.

[2] For simplicity, we omit the discussion of personnel cost involved in maintaining an IDS.

Outcome	Consequential Cost *CCost(c)*	Condition
Miss (False Negative, *FN*)	DCost(a)	
False Alarm (False Positive*FP*)	RCost(a) + PCost(c)	if DCost(a) > RCost(a) or
	0	if DCost(a) ≤ RCost(a)
Hit (True Positive, *TP*)	RCost(a) + ϵ_1DCost(a), $0 \leq \epsilon_1 \leq 1$	if DCost(a) > RCost(a) or
	DCost(a)	if DCost(a) ≤ RCost(a)
Normal (True Negative, *TN*)	0	
Misclassified Hit	RCost(a) + ϵ_2DCost(a_t), $0 \leq \epsilon_2 \leq 1$	if DCost(a) > RCost(a) or
	DCost(a_t)	if DCost(a) ≤ RCost(a)

Table 1: The Targeted Model for Consequential Cost

Misclassified Hit Cost. Since we have more than two classes of intrusions, we need to analyze the cost when an incorrect type of intrusion is detected: when a is detected instead of a_t. First, there is the damage cost of ϵDCost(a_t), where $0 \leq \epsilon \leq 1$, since the incorrect response action may not be effective. Second, depending on whether the IDS responds to the attack a or not, there can also be the response cost of $RCost(a)$.

So far we only considered costs that depend on the outcome of an IDS. We now incorporate the operational cost, *OpCost*, that is independent of the IDS's predictive performance. OpCost measures mainly the cost of computing values of features and checking the rules in the IDS when examining a connection. We denote *OpCost(c)* as the operational cost for a connection c.

We now describe the cost model for the IDS. When evaluating an IDS over some labeled test set S, where each connection, $c \in S$, has a label of "normal" or one of the intrusions, we define the cumulative cost for an IDS as follows:

$$CumulativeCost(S) = \sum_{c \in S} (CCost(c) + OpCost(c)) \tag{1}$$

where CCost(c), the consequential cost (i.e., cost of the consequence) of the prediction by the IDS on c, is defined in Table 1, and a is the attack type detected by the IDS for connection c.

Note that a higher operational cost, OpCost(c), could be incurred by employing expensive features; this may improve the predictive performance of the IDS and thus lower CCost(c). As discussed in later sections, to minimize CumulativeCost(S), we need to investigate and quantify the trade-off between OpCost(c) and CCost(c) in Equation 1.

2.5 Cost Metrics

Cost-sensitive models can be constructed and evaluated only using given cost metrics. This problem is related to the fields of computer risk analysis and security assessment. The literature [1, 2, 6] suggests that attempts to fully quantify all factors involved in cost modeling usually generate misleading results because not all factors can be reduced to discrete dollars (or some common unit of measurement) and probabilities. It is recommended that qualitative analysis be used to measure the relative magnitudes of the cost factors.

Research results in intrusion taxonomy also influence our work in cost measurements. Intrusions can be categorized and analyzed from different perspectives. Lindqvist and Jonsson introduced the concept of the *dimension* of an intrusion and used several dimensions to classify intrusions [10]. The *intrusion results* dimension can be used to assess the damage costs and response costs, while the *intrusion techniques* dimension affects the operational costs and the response costs.

When measuring the cost factors, we only consider individual attacks detectable by IDSs. For example, a continuous attack that involves first port-scanning a network, then gaining user-level access to the network illegally, and finally acquiring the root access, would normally be detected and responded to by an IDS as three separate attacks because IDSs are designed to respond quickly. It is therefore reasonable to measure the attacks individually.

Next, we describe how the relative scales of the cost factors can be determined.

2.5.1 Damage Costs

For simplicity, we measure the damage cost by the type of the attack. We therefore use some intrusion taxonomy to categorize the attacks. Following the conventions used by MIT Lincoln Lab, where DARPA Intrusion Detection Evaluation is conducted [7], we partition attacks into these four categories:

1. Probing attacks such as port-scanning.
2. DOS attacks such as syn-flood.
3. R2L (remotely gaining illegal access to local systems) such as with password guessing.
4. U2R (gaining illegal root access) such as buffer-overflow.

All attacks in the same category have the same damage cost. This corresponds to the top level of the intrusion results dimension of attack taxonomy [10]. Finer grained and more comprehensive taxonomies can be used for more elaborate cost analysis and measurement. For example, each attack type can be further broken down by the types of targets, e.g., servers or user workstations, and further by the types of the services, e.g., email, telnet, Web, etc. Using finer grained classification schemes can result in more precise cost measurements but requires a more complex cost analysis process.

We use the knowledge from the intrusion results dimension to define the relative scales of these four attack types. We only need to fix the starting point. For example, if we consider U2R as the most damaging attack type, then its cost can be set to a maximum value of 100, and other attack types can be assigned relative costs, e.g., the cost of R2L can be 50.

2.5.2 Response Costs

As shown in Table 1, the response cost of a connection c, RCost(c), is compared to damage cost DCost(c) to decide CCost(c). Since we define DCost using a taxonomy of intrusion results, we need to define RCost using the same taxonomy. The responses for different attack types are normally different. For example, for an R2L or U2R attack, since access to a server is involved, the response may be to kill the relevant processes on the target host system; but for a Probing or DOS attack, the response may be to instruct the Firewall to terminate the relevant connections. We can estimate the relative complexities of the response actions so that we can later define the relative scales of RCost for the different attack types. For example, we can consider that for both U2R and R2L, the response is twice as complex as the response for both DOS and Probing.

The damage cost and response cost are not folded into the same measurement unit (e.g., dollar), instead, each is in its own relative scale. We have to combine the two so that we can compute CCost(c) and use our cost model (see Equation 1). One way is to decide first under what conditions to ignore some intrusions. For example, assuming that probing attacks can be ignored and that the damage cost for probing is 2, then the response cost for probing must be greater, say, 20. Similarly, if the attack type with lowest damage cost should not be ignored, then the corresponding lowest response cost should be a smaller value. Once this starting value is defined, remaining values can be defined following the relative scales.

2.5.3 Operational Costs

The main operational cost we consider is the cost of computing the features required for checking the rules. We begin with measuring the cost of computing each individual feature. Based on our experience in extracting and constructing predictive features from network audit data, we classify features into three relative levels, based on their computational costs:

- Level 1 features are computed using at most the first three packets of a connection. For example, the "destination service" can be determined using the first packet.
- Level 2 features are computed in the middle of or near the end of a connection using only the information of the connection. For example, the "number of data bytes from the source to the destination" can be computed when the connection terminates.
- Level 3 features are computed using information from several connections within a given past time window. For example, the feature denoting "the percentage of connections to the same destination host as the current connection within the past 2 seconds" can be computed by examining all the connections of the past 2 seconds.

We can assign relative magnitudes to these features to represent the different computational costs. For example, level 1 features may cost 1, level 2 features may cost 10, and level features may cost 100.

OpCost(c) in Equation 1 can be computed as the sum of the computational costs of all the features used in the actual rule checking. This cost includes the costs for processing rules that failed to match. Since OpCost(c) and CCost(c) use two different measurement units and there is no possibility of comparing the two, as in the case with damage cost and response cost, we can use Equation 1 at a conceptual level. That is, when evaluating IDSs, we can consider both the cumulative OpCost and accumulated CCost, but actual comparisons are performed separately using the two costs. This inconvenience can not be overcome easily unless all cost factors can be represented using a common measurement unit, or there is a reference (e.g., comparison) relationship for all the factors. Site-specific policies can be used to determine uniformally how, if ever, to measure these cost factors.

In closing this section, we note that it is very important but challenging to define, model, and measure the cost factors related to IDSs. Further research is needed to provide better understanding and tools to handle the complexities and impreciseness of the cost analysis task. For the purpose of our research, which is focused on building cost-sensitive models given the cost metrics that are given as input, we simplified cost factors and models as a first attempt in this study.

3 Cost-Sensitive Modeling

As suggested in Anderson's paper [1], risk analysis is likely a repeated activity because it has to be performed periodically to take into account changed assets and policies. The same is true for cost-sensitive modeling in IDSs because the cost metrics may have changed. It is important to have an automated approach that can take input cost metrics and output the appropriate updated models. We propose to use machine learning techniques to build cost-sensitive models.

3.1 Optimizing Operational Costs

We study the problem of how to maintain comparable accuracy of an ID model while minimizing its operational cost. Operational cost is defined to be the total cost of the features used by the learned ID model.

In an ordinary machine learning process, the features that provide the maximum *information gain* are selected into the classification rules. These are the features that can best separate data items into their respective classes. The computational costs of the features are not considered in the process because accuracy is the only concern. A simple way of incorporating a sensitivity to feature costs into a machine learning algorithm is to modify the search heuristic, its information gain calculation, to also consider cost.

We experimented with modifying RIPPER's search heuristics to make it sensitive to feature costs. The method selected for experimentation was that used by Nunez [13] in the field of medical diagnosis: Let $Gain(A)$ be the information gain calculation in RIPPER that evaluates a conjunct involving feature A, and let Cost(A) be the cost of computing a value of A, we can use the following formula as the evaluation function:

$$\frac{(2^{Gain(A)} - 1)}{(Cost(A) + 1)^{\omega}} \tag{2}$$

Here $\omega \in [0, 1]$ is a constant that determines how sensitive the feature cost is in relation to information again. Different types of intrusions can use different ω values to enforce a notion of per-class sensitivity. For example, for the very damaging intrusions, it is desirable to produce a model that is as accurate as possible, without much consideration to the costs of the features. For intrusions that cause little harm, it is desirable to have a model that is very cheap even if accuracy is sacrificed. Experiments on data gathered by MIT Lincoln Lab and on our real-time network environment have shown that the modified RIPPER builds models requiring less costly feature sets while maintaining the original level of accuracy [11].

3.2 Optimizing Consequential Costs

Table 1 suggests an ideal cost model used by a cost-sensitive IDS. The IDS responds to a detected intrusion if and only if the damage cost is greater than the response cost. For ignored intrusions, the effect of detection (TP) is the same as miss a (FP). Instead of encoding these cost-sensitive decisions manually into IDSs, we use a MetaCost procedure [5] to produce automatically those models that would purposely "misclassify" (e.g., produce a miss) data in order to minimize cost.

Given a *cost matrix*, C, where $C(i, j)$ is the cost of predicting that a record belongs to class i when in fact it belongs to class j, the *Bayes optimal* prediction for a record x is the class i that minimizes the *conditional risk* [15]:

$$R(i|x) = \sum_j P(j|x)C(i, j) \tag{3}$$

where $R(i|x)$, the conditional risk, is the expected cost of predicting that x belongs to class i. $P(j|x)$ is the probability that x belongs to class j. The main idea behind MetaCost is to first relabel each record in the training dataset using the record's optimal class that minimizes Equation 3 (which may not be the same as its actual class), then apply a standard machine learning algorithm to the relabeled training set. The resultant model is optimal because the training data is already relabeled optimally according to the cost matrix, and all machine learning algorithms compute models that best fit data. The quality of the estimates on class probabilities $P(j|x)$ is important only insofar as it leads to the determination of the correct optimal class for x. As suggested in the paper by Domingos [5], a simple way of obtaining $P(j|x)$ is to create multiple bootstrap replicas of the training set, learn a classifier on each, and then estimate each class's probability on each record by the fraction of votes, i.e., the number of predictions that match the class, that it receives from the set of classifiers.

For the intrusion detection domain, the *cost matrix*, C, describes the costs of FN, FP, TP, TN, and Misclassified Hit for normal and all intrusions. As illustrated in Section 3.3, using the metrics for damage cost (DCost) and response cost (RCost), and following similar analysis steps in Table 1, we can compute the cost matrix, and apply the MetaCost procedure to learn ID models with optimized prediction costs.

3.3 Experiments

Our experiments use data that was distributed by the 1998 DARPA Intrusion Detection Evaluation Program, which was conducted by MIT Lincoln Lab. The data were gathered from a military network with a wide variety of intrusions injected into the network over a period of 7 weeks. The data was processed into connection records using MADAM ID (a system for Mining Audit Data for Automated Models for Intrusion Detection) [9]. A 10% sample was taken which maintained the same distribution of attacks and normal connections as the original data. We used 80% of this sample as training data. For infrequent intrusions in the training data, the records for those connections were repeatedly injected to prevent the learning algorithm

Category	Damage Cost	Response Cost
u2r	100	40
r2l	50	40
dos	20	20
probe	2	20
normal	0	0

Table 2: Cost Metrics

from neglecting them as statistically insignificant data points. The remaining 20% of our sample data was left unaltered and were used as test data for evaluation of the learned models.

To evaluate the effectiveness of our approach of building a Feature-Cost-Sensitive (FCS) ruleset, we set each feature's specified costs to that feature's category cost as described in Section 2.3. Multiple rulesets were learned by holding ω constant for all classes of intrusions, but varying that constant over the interval $\omega \in [0, 1]$.

Table 2 shows the damage and response costs used for our experiments with a varied cost sensitivity per intrusion. Although the cost metrics were simplified, it is important to note that the main focus of our research is not in security assessment, but rather in developing mechanisms that can produce the desirable cost-sensitive detection models for given cost metrics. It is sufficient here to show the results of using our techniques on the chosen cost metrics.

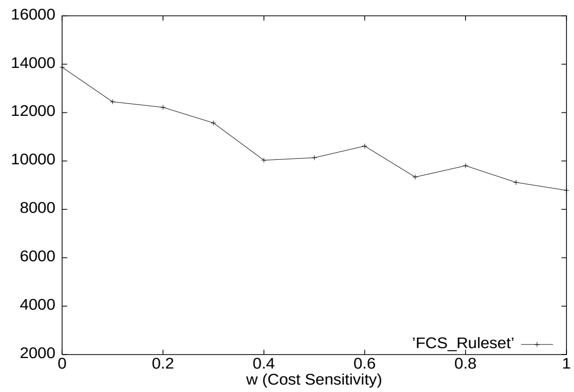

Figure 1: Operational Cost of FCS Ruleset vs. ω.

The operational and consequential costs incurred when evaluating each ruleset over the test data are shown in Figure 1 and Figure 2. The operational cost (OpCost) is computed as sum of the feature computation costs of all rules in the ruleset up to and including the rule resulting in a prediction, summed over all predictions made on the data. Likewise, the consequential cost (CCost) is computed as the sum of the cost matrix entries for all rules leading to a prediction, summed over all predictions. These definitions of the measurements taken over the test data attempt to mimic a worst-case real-time implementation of a rule-evaluating system. They assume that each attribute would be re-computed for each connection to be tested every time a comparison must be made on that attribute. Because this same methodology is used to compute the measurements for all learned rule-sets, they yield a sufficient basis for cost comparisons.

As expected, the results show a general trend of decrease in the cumulative operational cost on the test data as the cost sensitivity ω is increased.

For experimentation with MetaCost, we need to construct the cost matrix, C, and the probability of each class in order to compute the conditional risk using Equation 3. A procedure that follows the analysis steps described in Table 3 is used to compute the cost matrix. We can not use Table 1 here because it already describes the *effects* of the desired cost-sensitive model that we would like to produce using MetaCost. For

Outcome	CCost(c)
Miss (*FN*)	DCost(a)
False Alarm (*FP*)	RCost(a)) + PCost(c)
Hit (*TP*)	RCost(a) + ϵ_1DCost(a), $0 \leq \epsilon_1 \leq 1$
Normal (*TN*)	0
Misclassified Hit	RCost(a) + ϵ_2DCost(a_t), $0 \leq \epsilon_2 \leq 1$

Table 3: A Baseline Model of Consequential Cost for Computing Cost Matrix

example, we may wish to ignore probing attacks because the response cost is greater than damage cost. In this case, we would like the MetaCost procedure to use "normal" as the optimal class when relabeling probing attacks. However, using Table 2, Table 1 indicates that probing is already the optimal class for itself because a TP has the same cost as an FN. On the other hand, with Table 3, which describes the effects of a standard cost-insensitive ID model, probing attacks are relabeled as normal through MetaCost procedure. This results in the lower consequential cost of the resulting cost-sensitive ID model. See Figure 2.

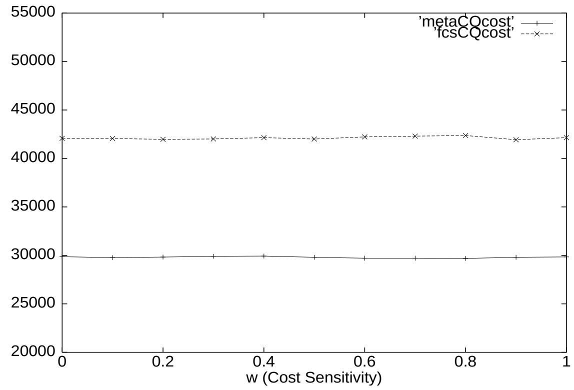

Figure 2: Consequential Costs of MetaCost (metaCQ) and Feature-Cost-Sensitive (fcs) rulesets

MetaCost always tries to relabel a connection record with a class that incurs less consequential cost. The policy with which MetaCost decides its re-classification scheme is dependent upon both the definition of consequential cost (CCost) and the distribution of each intrusion in the data. Our specification of the cost metrics used in the computation of CCost (RCost and DCost) are at the level of the intrusion categories.

From our results, we also observed that the ID models produced by MetaCost have effectively lower the operational costs. In the MetaCost procedure, probing attacks are relabeled as "normal", and therefore no detection rules produced for them. Such rules have high operational costs because they require temporal and statistical features. Eliminating these rules substantially reduces the overall operational costs.

For the purpose of simplicity in our experimentation with models learned using MetaCost rulesets, we used $\epsilon_1 = 0$ and $\epsilon_2 = 1$. The first condition corresponds to the case where an intrusion is correctly classified and the DCost is 0 because we can fully prevent damage. The second condition corresponds to the case where we penalize a misclassified hit because we are unable to prevent damage. Using other values of ϵ produced rulesets in which all of the intrusions in higher cost categories were subsumed by lower cost categories. This resulted in either a null ruleset (when all intrusions were relabeled to "normal") or a ruleset in which all intrusions were classified as an arbitrary member of the class of DOS attacks. This is an artifact of the data, in that our cost specifications and the probability distributions of the data dictated which intrusion from the class of DOS attacks would be used for relabeling. The ability for this to occur is a product of the low granularity used for cost specifications of intrusion categories. A finer granularity would produce a more meaningful class relabeling. Hence, higher performance ID models could be learned from the MetaCost dataset.

4 Cost-Sensitive Deployment

Our intrusion detection models are produced off-line. Effective intrusion detection should be in real-time to minimize security compromises. We therefore need to study how our models perform in a real-time environment.

4.1 Automatic Conversion to Real-time Modules

We have been working on translating RIPPER rules into real-time detection modules in Network Flight Recorder(NFR) [12], a system that includes a packet capturing engine and N-code programming support for specifying packet "filtering" logic.

NFR offers a fairly simple framework for network monitoring. It sniffs packets on the network, reassembles them, and then passes them to filter functions for further processing, e.g., calculating network traffic statistics. Filter functions are written in N-code, a preemptive, event-driven scripting language that supports a wide range of operations on network packet data. In order to use NFR for network intrusion detection, one needs to implement an extensive set of N-code filters that look for evidence in the network packets.

The purposes of our NFR project are therefore not just to simply evaluate how off-line learned models can be utilized in a real-time environment. More importantly, we seek to automate the creation of real-time N-code detection filters which can be easily incorporated into a running IDS provided that the features used by the ruleset have been implemented.

The set of features required in the RIPPER rules has been converted into N-code as detailed in the discussion of our implementation. A translator has also been implemented that automatically translates each RIPPER rule into an N-code function that can be called to determine whether or not a given connection is an attack. We envision that when NFR is shipped with our data mining system to a customer site, the site-specific intrusion detection rules can be learned using locally gathered audit data, and be automatically converted into N-code "rule filters". We believe that our NFR project will showcase how to rapidly automatically deploy, and customize, an IDS.

4.2 Efficient Execution of ID Models

Although often ignored in off-line analysis, efficiency is a very important consideration in real-time intrusion detection. In our first experimental implementation of N-code "rule filters", we essentially tried to follow the off-line analysis steps in a real-time environment. A connection is not inspected (i.e., classified using the rules) until its connection record is completely formulated, that is, all packets of the connection have arrived and have been summarized, and all the temporal and statistical features are computed. This scheme is not efficient. Worse yet, when there is a large volume of network traffic, the amount of time taken to process the connection records within the past 2 seconds and calculate the statistics is also very large. Many connections may have terminated (and thus completed with attack actions) when the current connection is finally inspected by the RIPPER rules. That is, the detection of intrusions is severely delayed. Ironically, DOS attacks, which typically generate a large amount a traffic in a very short period time, are often used by intruders to first overload an IDS, and use the detection delay as a window of opportunity to quickly perform their malicious intent. For example, they can seize control of the operating system and "kill" the IDS.

Note that we cannot simply order the rules by their costs for real-time execution for the following reasons. First, the rules output by RIPPER are in a strict sequential order (e.g., "if rule 1 else rule 2 else ..."), and hence reordering the rules may result in unintended classification errors. Furthermore, even if the rules can be tested in strictly cost order without introducing classification errors, many rules will still be tested (and fail to match) before a classification can be made. Therefore, ordering the rules by their costs alone is not necessarily the optimal solution for quick model evaluation. We thus seek to compute an "optimal schedule" for feature computation and rule testing to minimize model evaluation costs, and increase the response rate for real-time detection.

4.2.1 Low Cost "Necessary" Conditions

Ideally, we can have a few tests involving the low cost (i.e., level 1 and level 2) features to eliminate the majority of the rules that need to be checked, and thus eliminating the needs to compute some of the high cost features.

In order to eliminate the need for testing a rule for intrusion I, we need a test of the form of $F \rightarrow \neg I$ which can be derived from $I \rightarrow \neg F$ We can compute the association rules that have the intrusion labels on the LHS and the low cost features on the RHS, and a *confidence* of 100%.

We discovered several such associations for the RIPPER rules, for example,

$$\begin{array}{l} \textit{port\_scan} \rightarrow \textit{src\_bytes} = 0\ [c = 1.0],\ \text{and} \\ \textit{syn\_flood} \rightarrow \textit{flag} = \textit{S0}\ [c = 1.0],\ \text{etc.} \end{array}$$

Note that most of these feature values, for example, *src_bytes* $= 0$, are not in the RIPPER rules because they are prevalent in the normal data. That is, they don't have predictive power. However, these associations are the "necessary" conditions for the intrusions, for example, "this connection is a port-scan attack **only if** *src_bytes* is 0", which is equivalent to "if the *src_bytes* is **not** 0, then this connection is **not** a port-scan attack".

When a RIPPER rule for an intrusion is excluded because of the failure of its necessary condition, the features of the rule need not be computed, unless they are needed for other candidate (remaining) rules. We next discuss how to do efficient bookkeeping on the candidate rules and features to determine a schedule for feature computation and rule condition testing.

4.2.2 Real-time Rule Filtering

Suppose that we have n RIPPER rules. We use a n-bit vector, with the bit order corresponding to the order of the rules output by RIPPER, as the "remaining" vector to indicate which rules still need to be checked. Initially, all bits are 1's. Each rule has a "invalidating" n-bit vector, where only the bit corresponding to the rule is 0 and all other bits are 1's. Each of the high cost features, i.e., the level 3 temporal and statistical feature, has a "computing" n-bit vector, where only the bits corresponding to the rules that require this feature are 1's.

For each intrusion label, we record its "lowest cost necessary condition" (if there are such conditions), according to the costs of the features involved. We sort all these necessary conditions according to the costs to produce the order for real-time condition checking.

When examining a packet, or a (just completed) connection, if a "necessary" condition of an intrusion is violated, the corresponding "invalidating" bit vectors of the RIPPER rules of the intrusion are used to AND the "remaining" vector and all the "computing" vectors for the high cost features. After all the necessary conditions are checked, we get all the features with non-zero "computing" vectors. These features are potentially useful because of the remaining rules that need to be checked. A single function call is made to N-code modules to compute all these features at once. This execution strategy reduces memory or disk access since these features compute statistical information on the past (stored) connections records. The "remaining" vector is then used to check the remaining rules one by one. This approach was implemented within N-code and is used in various experiments.

4.3 Implementation

Three basic components have been developed in order to implement the system: the connection database, the feature set, and the rulesets. We modeled the system after the off-line analysis approach to intrusion detection, with the connection database providing a data set for the rules to inspect.The connection database provides the IDS data that is required by our learned models, and is maintained by a set of N-Code filters that keep track of information about active and inactive network connections that have occurred within a

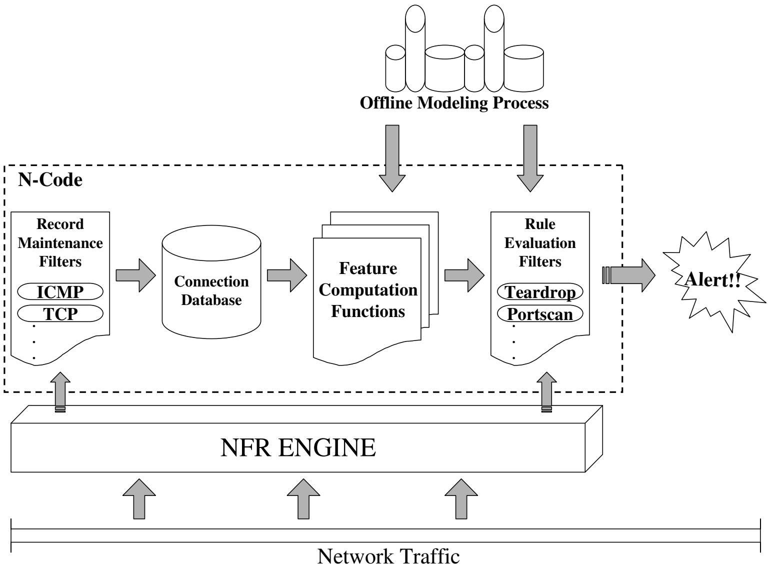

Figure 3: Architecture of Real-Time Rule Evaluation System

certain time window. The feature set consists of N-Code functions that operate on data provided by the connection database and return values to the rules; the rules in turn use these values to evaluate the traffic.

The implementation of the system within the NFR environment led to a logical separation of certain types of connection data. In particular, NFR provides high-level filters that allow code to be executed only upon receipt of certain type of packets. In this case, connection data is logically separated into three types before inclusion in the connection database : ICMP, TCP and UDP. This separation shortens the decision tree that our IDS uses to detect the occurrence of particular attacks. The system does not examine every packet to find its type, instead, the detection procedure for a given attack is placed inside an N-Code filter that corresponds to the type

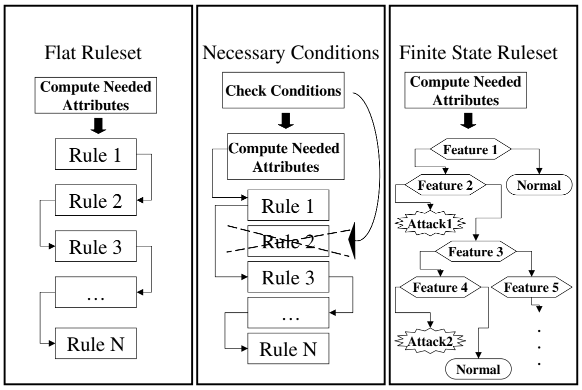

Figure 4: Baseline, NC, and FSM Rulesets

4.4 Experiments

We determined computation times for features in real time. These experimental results provide the justification for the feature level cost estimates provided in Section 2.3. The execution time for functions calculating the appropriate feature increases by roughly an order of magnitude for each feature level. See Table 4

Feature	Level	Average Execution Time	σ
rejFlag	1	3.8×10^{-5}	3.9×10^{-5}
wrongFrag	1	3.5×10^{-5}	3.5×10^{-5}
srcBytes	2	2.1×10^{-5}	2.0×10^{-4}
rerrorRate	3	5.0×10^{-4}	1.0×10^{-4}
sameSrvRate	3	9.0×10^{-4}	10×10^{-4}

Table 4: Feature Computation Time

The large variation in the level one features, *rejFlag* and *wrongFrag*, as well as the level two feature *srcBytes*, is result of the low cost of computation of these features. When the NFR engine was relatively free, the feature was calculated nearly instantly. However, given some other process using CPU time, a small amount of lag would lead to a comparatively high computation time.

We developed several implementations of RIPPER-generated rulesets for experimentation:

- A procedural, all inclusive version (FLAT).
- A necessary condition (NC) checking version utilizing an optimized feature computation algorithm.
- A finite state machine (FSM) version that builds a tree of decision points for a given rule and computes only the feature necessary at a given point in the tree.
- Two implementations based on rulesets learned using varying ω with cs-RIPPER,one implementation with and and one without necessary conditions checking.
- Two implementations based on rulesets learned using the MetaCost method, also with and without necessary conditions checking

The FLAT version is our baseline model, and is inherited directly from the offline approach: within their respective categories (ICMP, TCP, UDP), we analyze each connection by evaluating each rule checking function until the end of the ruleset is reached or a value is returned. They are kept in the same order that they are produced by the RIPPER algorithm, and are in fact translated directly into N-Code functions by a separate program. The NC ruleset decides which rules to compute based on lowest cost necessary conditions and scans the database only once to compute all of the features that will be needed in testing these rules. We refer to this feature computation process as the *collapsed attribute* function, it is conceptually analogous to the feature computation process used in off-line analysis.

The FSM approach is a hand coded attempt to model the decision tree approach used in many IDS applications within the bounds that our offline modeling process has imposed upon the system. At the top level, packets are classified by protocol and then proceed in the FSM. All rules within a category are grouped into a single decision tree, with leaves of the tree classified as attacks or normal connections. Features are computed at each point in the tree, cheapest first in an attempt to detect ”normal” traffic as early as possible and eliminate computation of additional features.A path from the root to a leaf is the execution of one of the RIPPER generated rules.

We ensured that all of the implementations were produced from the same set of cost-sensitive and cost-insensitive rulesets such that they all detect the same attacks and parallel the experiments done with RIPPER and Bro. Additionally, the features that they use to detect the attacks are essentially the same, even though the ordering or methods for feature computation are different. We developed and tested several variations on the three basic implementations. The first of these variations is the addition of the collapsed attribute function to all of the algorithms, with the aim of testing the differences in the three approaches while using the most efficient method to compute features. The three rulesets defined within this scheme are detailed in Figure 4 . The rulesets were all loaded into the NFR engine and run on a live network, with a relatively constant sustained traffic flow. The execution times of all of the different versions were measured over the span of an hour and a half using NFR statistic gathering tools. The total execution times, in user wall time, were gathered and compared to determine whether or not gains in operational costs were also realized in the real time environment. Total execution times are detailed in Figure 5.

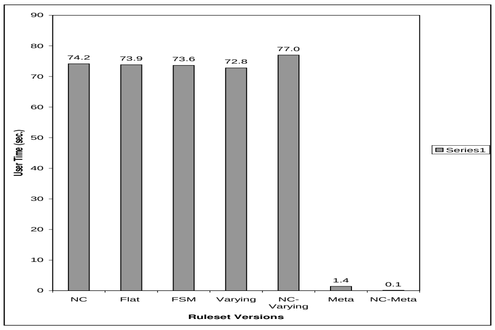

Figure 5: Total Execution Times of Implemented Rulesets

Some of the results run contrary to results found in offline testing. For example, the necessary conditions checking rulesets ran slower than the flat, or baseline, models. This result is apparently due to the fact our implementation uses only two or fewer rules within the category features produced by NFR - the necessary conditions checking logic in effect becomes overhead when it is applied to a very small number of rules.Additionally the number of connections in the database was relatively small, with from 75 to 125 connections active at a given time. The total number of incoming connections was much higher, however, and the necessary conditions logic is checked on all incoming connections within each category.The overhead of scanning the entire database to compute third order features is relatively low for these experiments, and the extra time taken by computing the necessary conditions logic on all connections seems to slightly outweigh the cost of simply computing all the rules on all the connections in some instances.For the MetaCost-based ruleset, the necessary conditions checking performed as expected, and this version of the ruleset was significantly faster than the original. The rules in the MetaCost based version were oriented towards ICMP or exception traffic, and in normal traffic the throughput of this kind of data is low compared to TCP traffic throughput. It seems that when the number of incoming connections is high compared to the average number of connections in the database, necessary conditions checking may make a ruleset less efficient since time gained by skipping computation of high - cost features is outweighed by the time lost computing the extra logic on the high number of incoming connections.

The rest of the results agree with those determined in offline testing. The MetaCost based ruleset, by reclassifying some scanning - type intrusions as normal, displayed a running time that was approximately 2% of the next lowest-cost ruleset. The ruleset based on cs-RIPPER computations with varying ω was more efficient that its cost insensitive counterpart, and also slightly outperformed the hand - coded finite state machine model.

Note: the data gathering tools used to gather these data were potentially somewhat uncertain, and exact timing within this environment is difficult. Part of future work with NFR will be to develop better metrics and statistics gathering tools for real - time intrusion detection systems.

4.5 Bro Experiments

In addition to the development and experimentation of real-time ID modules using NFR, we are conducting similar research using Bro. The motivation is to verify the results of our experiments in different systems. More importantly, we are working on an experimental CIDF (Common Intrusion Detection Framework) [3] implementation where a data mining system (i.e., MADAM ID) acts as a center for audit data analysis, model construction and distribution for other real-time IDSs (e.g., NFR and Bro).

Like NFR, Bro is a customizable IDS where intrusion detection rules can be programmed into Bro "policy scripts", in the form of event handlers. These are invoked at run-time by the Bro engine. For the purpose of this study, we compared the the run-time performance of two implementation schemes of RIPPER rules. The first was FLAT, which corresponds to a mechanical translation of RIPPER rules where the rules are checked one by one. The second is NC, which corresponds to first checking the violation of necessary conditions to eliminate the need of checking more costly rules, and then checking the qualifying rules one by one. We implemented a cost-insensitive RIPPER ruleset, which was learned using the 7 weeks of training data from the 1998 DARPA Intrusion Detection Evaluation, as Bro scripts. The ruleset contains rules to detect a total of more than 30 types of intrusions, covering all 4 attack categories, i.e., Probing, DOS, R2L, and U2R. Using the first week of testing data from the DARPA evaluation, we compared the two implementations using the total processing time required for evaluation of different test data sets.

For the 5 data sets of the week, one for each weekday, we measured the percentage run time improvement that NC has over FLAT. We saw a wide range of values, from 17% to 3%. Analysis of the actual audit data showed that for a dataset that has a large amount of Probing and DOS attacks, the improvement is small. The process of repeatedly checking necessary conditions results in additional computational cost. A positive result of these experiments is the reduction of execution time of the remaining ruleset because a number of rules may be eliminated. The running time improvement is therefore the difference between the gain and the cost. When there are a large amount of Probing and DOS attacks in the data, the number of violations to the necessary conditions are small, and thus only a small number of rules are eliminated. Hence the improvement is small. We plan to conduct more thorough experiments and we will report the results in the final version of the paper.

5 Related Work

As discussed throughout the paper, our work in cost-sensitive modeling for IDSs is closely related to research in computer security assessment and intrusion taxonomy. In particular, Glaseman et al [6] discussed a model for evaluating the total expected cost in using a security system s as $\mathrm{C}(s) = \mathrm{O}(s) + \mathrm{D}(s)$, where $\mathrm{O}(s)$ is the operational cost of s and $\mathrm{D}(s)$ is the expected loss. $\mathrm{D}(s)$ is calculated by summing over all possible threats, the products of exposed value and the probability of safeguard failure. This model is very similar to our cost model for IDSs, which is defined in Equation 1. However, our definition of consequential cost allows cost-based optimization strategies to be explored because it includes the response cost and models the relationship between the damage cost and response cost.

Credit card fraud detection and cellular phone fraud detection are related to intrusion detection because because they also deal with detecting abnormal behavior. Both of these applications are motivated by cost-saving and therefore use cost-sensitive modeling techniques. Take credit card fraud detection as an example. The cost factors include operation cost, challenge cost, i.e., the personnel cost of putting a staff to investigate before authorizing a transaction, and loss (damage cost). If the dollar amount of a suspected transaction is lower than the challenge cost, the transaction is authorized, that is, the credit card company will take the potential loss. Since the cost factors can all be folded into dollar amount, the cost-sensitive analysis and modeling task is a lot more simpler than that for IDSs.

Cost-sensitive modeling is an active research area for data mining and machine learning because of the demand from application domains such as medical diagnosis, fraud and intrusion detection, etc. Several techniques, e.g., MetaCost, have been proposed for building models optimized for given cost metrics. In our research we extend these general techniques and develop new approaches according to the characteristics of the cost models for IDSs.

6 Conclusion and Future Work

Our main contributions to the field of Intrusion Detection is in the development of a framework for analyzing and building cost sensitive models for classification. We examine a number of difficulties in the development of cost models that must be considered by both the Intrusion Detection community and Security and Risk Analysis communities if our framework is to be further developed and utilized. These difficulties are the identification of cost factors, their relationships (the cost model), and their measurement. Our work explores the process of building Intrusion Detection models that effectively utilize this information.

It is clearly possible for a human expert to hand-craft a detection model which takes into account the operational and consequential costs involved in the detection of network intrusions. This model may even be more efficient than our learned and implemented models, as we have shown that hand-crafted finite state implementations can reduce the involved costs over our computer translated implementations (NC and FLAT evaluation). However, our work develops a framework for not only automating the generation of these models, but for automating their deployment while considering the operational and consequential costs. Thus, the total development and deployment costs incurred in the intrusion detection process are dramatically reduced.

6.1 Future Work

Our framework currently has the restriction that when a site's cost factors change, it is necessary to re-generate the cost matrix and build a new cost-based model for detection. We will study the science and engineering methods for building dynamic models that do not need re-training based on changes in the pertinent cost factors. Discoveries in this area will reduce the cost of re-learning models for both changes in intra-site cost analysis and deployment at diverse sites with inherently different cost models.

Our experimental specifications for damage and response costs are based on a very coarse dimension of intrusion categories. These specifications often result' in the subsumption of higher cost categories by those of lower or nonexistent cost (i.e. "normal"). By increasing the granularity allowed, and possibly allowing for the specification of these metrics in heterogeneous dimensions, it would be possible to learn more customized models of detection for sites with diverse risk analysis measurements. For example, a site may wish to specify that a DOS attack to their mission critical Web server would have a damage cost that is on the order of 1000 times the damage cost of a DOS attack to the site's firewall machine, which may be more robustly equipped to thwart the attack.

Our results in experimentation with RIPPER datasets and offline tcpdump analysis show improvements in both operational and consequential costs. However, it is necessary that a more adequate set of tools and test methods be developed for the evaluation of real-time detection models. Although it is difficult to quantitatively measure the accuracy and operational costs associated with the evaluation of learned models by our real-time system, it is clear that a number of improvements can be made in the implementation. Specifically, it is possible to implement the connection database and feature function set inside the NFR Engine. This approach would most likely show a uniform decrease in operational cost across all of the different rulesets we have tested in this work. In the addition of new features to our system, we will also be adding the capability of examining the data portion of both packets and re-assembled connections in order to provide a richer feature set capable of detecting a larger set of intrusions.

References

[1] A. M. Anderson. Comparing risk analysis methodologies. In D. T. Lindsay and W. L. Price, editors, *Information Security*. Elsevier Science Publishers, 1991.

[2] R. P. Campbell and G. A. Sands. A modular approach to computer security risk management. In *AFIPS Conference Proceedings*. AFIPS Press, 1979.

[3] CIDF. Common intrusion detection framework. http://gost.isi.edu/cidf.

[4] W. W. Cohen. Fast effective rule induction. In *Machine Learning: the 12th International Conference*, Lake Taho, CA, 1995. Morgan Kaufmann.

[5] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-99)*, August 1999.

[6] S. Glaseman, R. Turn, and R. S. Gaines. Problem areas in computer security assessment. In *Proceedings of the National Computer Conference*, 1977.

[7] MIT Lincoln Lab. 1998 darpa intrusion detection evaluation. http://www.cs.columbia.edu/sal/JAM/PROJECT/MIT/mit-index.html.

[8] W. Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, June 1999.

[9] W. Lee, S. J. Stolfo, and K. W. Mok. A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, May 1999.

[10] U. Lindqvist and E. Jonsson. How to systematically classify computer security intrusions. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland CA, May 1997.

[11] Matthew Miller. Learning cost-sensitive classification rules for network intrusion detection using ripper. Technical report, Computer Science Department, Columbia University, June 1999.

[12] Network Flight Recorder Inc. Network flight recorder. http://www.nfr.com, 1997.

[13] M. Nunez. The use of background knowledge in decision tree induction. *Machine Learning*, 6(3):231–250, 1991.

[14] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, San Antonio, TX, 1998.

[15] R. O. Ruda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.