Weighting versus Pruning in Rule Validation for Detecting Network and Host Anomalies

Gaurav Tandon
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901, USA
gtandon@fit.edu

Philip K. Chan
Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901, USA
pkc@cs.fit.edu

ABSTRACT

For intrusion detection, the LERAD algorithm learns a succinct set of comprehensible rules for detecting anomalies, which could be novel attacks. LERAD validates the learned rules on a separate *held-out* validation set and removes rules that cause false alarms. However, removing rules with possible high coverage can lead to missed detections. We propose to retain these rules and associate weights to them. We present three weighting schemes and our empirical results indicate that, for LERAD, rule weighting can detect more attacks than pruning with minimal computational overhead.

Categories and Subject Descriptors

I.2.6 [**Learning**]; K.6.5 [**Security and Protection**]

General Terms

Algorithms

Keywords

Machine learning, anomaly detection, rule pruning, and rule weighting

1. INTRODUCTION

Intrusion detection has two general approaches – *signature detection* (also known as *misuse detection*), where patterns signaling well-known attacks are searched; and *anomaly detection*, where deviations from normal behavior are flagged. Signature detection works reliably for known attacks, but has a limitation of missing new attacks. Though anomaly detection can detect novel attacks, it has the drawback of not being able to discern intent; it can only signal that some event is unusual, but not necessarily hostile, thus generating false alarms. This paper focuses on anomaly detection.

Rules for normal behavior can be introduced manually by a human expert, a tedious task which incurs significant effort and cost; or automatically learned from positive data using machine learning. One such technique, called LERAD (LEarning Rules for Anomaly Detection)[20], efficiently learns a succinct set of comprehensible rules and detects attacks unknown to the algorithm. To reduce false alarms, these rules are validated on normal *held-out* data and all violated rules are discarded. However, these rules were selected initially to cover a relatively large number of training examples and their elimination could possibly lead to missed detections. Instead of outright rejection of such rules (called *Pruning* in this paper), we propose weights estimating rule support. A conformed rule increases our belief in it and hence its weight is increased. On the other hand, weight is decreased upon rule violation symbolizing decrease in trust. We present three weighting schemes – *Winnow-specialist-based Weighting*, *Equal Reward Apportioning* and *Weight of Evidence*. These schemes inherently differ in nature in terms of weight update functions, incremental vs. batch updation, and the number of anomaly rules retained. We also incorporate the weight into the anomaly scoring mechanism - each rule assigns an anomaly score proportional to its weight, and all the scores are aggregated to compute the total anomaly score.

We present empirical evaluation and comparison of the weighting and pruning variants on various network and host data sets. Results show that weighted rules detect upto 60% more attack-based anomalies than rule pruning at less than 1% false alarm rates. Our claim about rule retention was reinforced by the fact that most of the new attacks detected were due to violations of previously discarded rules. Since previously eliminated rules are retained, weighted rule sets are expected to increase the computational and storage overhead. But the size of the rule set is still fairly small - generally less than 100 rules per week of each network data. The computational overhead of weighting is only a fraction of a millisecond per instance of data. The improvement in accuracy is thus obtained at the cost of small computational overhead and reasonable space requirements.

In Section 2 we discuss some network and host-based anomaly detection systems and rule learning algorithms. Section 3 briefly describes the LERAD algorithm and motivates and details rule weighting applied to LERAD. Three weighting strategies are described in Section 4. Section 5 evaluates and compares the accuracy and the CPU time requirements of *Pruning* with the three weighting strategies on multiple data sets. New attacks detected by weighting are also analyzed. Section 6 summarizes the results and presents some future research directions.

KDD'07, August 12–15, 2007, San Jose, California, USA.

2. RELATED WORK

Prior research on anomaly-based intrusion detection has focussed on monitoring sniffed network data as well as audit logs. Network anomaly detection systems can warn of attacks launched from the outside at an earlier stage, before the attacks actually reach the host. SNORT [26] and BRO [22] are simple rule-based systems. Rules can be hand-coded to restrict access to specific hosts and services. But manual updation of rules is deemed impractical. Intrusion detection systems (IDSs) such as NIDES [2], ADAM [3], and SPADE model features of the network and transport layer, such as port numbers, IP addresses, and TCP flags. Web based attacks are detected by monitoring web request parameters in [25]. Some anomaly detection algorithms are for specific attacks (e.g., portscans [29]) or services (e.g., DNS[16]). A host-based anomaly detector can detect some attacks (for example, *inside* attacks) that do not generate network traffic. Host based anomaly detection generally uses system call sequences [10, 32, 33] and have been represented using finite state automata [28] and neural networks [14]. Other features used include system call arguments [21, 30, 4] and call stack information [8, 13].

Associating weights with rules attempts to characterize the quality of the rules. One aspect of quality is *predictiveness*, which quantifies how likely the consequent occurs when the antecedent is observed; that is, how accurate the antecedent predicts the consequent. *Predictiveness* is commonly measured by estimating $P(consequent|antecedent)$. Another aspect of quality is *belief*, which measures the level of trust for the rule; that is, how believable the entire rule is. For example, in association rules [1], each rule has a confidence value and a support value — the confidence value estimates *predictiveness*, while the support value approximates *belief*. Many learning algorithms, including RIPPER [7] and CN2 [6], use *predictiveness* to formulate rules during the learning (training) process and/or provide confidence values for their predictions during the prediction (test) process. Ensemble methods, including Weighted Majority [19] and Boosting [27, 11], usually use *belief* to combine predictions from multiple learned models. Pruning is an approach to reduce overfitting the training data. After learning a decision tree and converting each path in the tree into rules, Quinlan [24] removes conditions from the antecedent of a rule if the estimated accuracy improves. Furnkranz [12] has a review of various rule pruning techniques.

For rule learning algorithms, many studies demonstrate the efficacy of using weights (*predictiveness* and/or *belief*) over not using weights as well as pruning over not pruning. However, we are not aware of studies in comparing using weights and pruning, particularly in anomaly detection. In this paper, we study how rule weighting compares to pruning in a rule learning algorithm for anomaly detection.

3. PRUNING AND WEIGHTING IN LERAD

LEarning Rules for Anomaly Detection (LERAD) [20] is an efficient randomized algorithm that forms conditional rules of the form:

$$a_1 = v_{11} \bigwedge a_2 = v_{23} \bigwedge ... => a_c \in \{v_{c1}, v_{c2}, ...\} \quad [p] \quad (1)$$

where a_i is the i^{th} attribute and v_{ij} is the j^{th} value for a_i. LERAD adopts a probabilistic framework and estimates

Input: sample set (D_s), training set (D_t), and validation set (D_v)
Output: LERAD rule set R

1. generate candidate rules from D_s and evaluate them
2. select a "minimal" set of candidate rules that covers D_s
3. train the selected candidate rules on D_t
4. eliminate the rules that cause false alarms on D_v

Figure 1: Main steps of LERAD algorithm

$P(C|A)$, where A is the antecedent and C is the consequent of the rule $A \Rightarrow C$. During training, a set of rules R that "minimally" describes the training data are generated and their $p = P(\neg C|A)$ is estimated, where C, though expected, is not observed when A is observed. An estimate for novel events from data compression [34] is used:

$$p = P(NovelEvent) = \frac{r}{n}. \quad (2)$$

where n is the total number of observed events and r is the number of unique observed events. A sample network anomaly rule for LERAD is:

$$\begin{aligned} &SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \\ &\Rightarrow DestPort \in \{21, 25, 80\} \quad [p = r/n = 3/100] \end{aligned} \quad (3)$$

Thus, 100 instances satisfy the above rule which claims 3 distinct destination ports (21-FTP, 25-SMTP, 80-HTTP) for the given source and destination IP addresses.

For describing the LERAD algorithm, we use the following notation. Let D be the entire data set, and D_T be the training set with normal behavior and D_E be the evaluation (test) data set with normal behavior as well as attacks such that $D_T \cup D_E = D$ and $D_T \cap D_E = \emptyset$. Training data is further partitioned into subsets D_t (training data set) and D_v (validation *held-out* data set) respectively such that $D_t \cup D_v = D_T$, $D_t \cap D_v = \emptyset$, and $|D_t| > |D_v|$. Also, let R be the rule set learned after training.

The LERAD algorithm consists of four main steps as illustrated in Figure 1. Step 1 intends to generate and evaluate candidate rules from a small data sample D_s (such that $|D_s| \ll |D_t|$), which allows efficient training. Step 2 selects a small set of predictive rules that sufficiently describe D_s. This allows learned models to be small. The selected rules are then trained on the much larger set D_t in Step 3. The validation set D_v is used in Step 4, and is described next in context of *pruning* and *weighting* strategies.

3.1 Validating Rules

To reduce overfitting the training data, machine learning algorithms use a separate *held-out* data to validate the trained model. LERAD uses validation set D_v for the rules learned from D_t. For each rule $r_k \in R$ and instance $d \in D_v$, one of three cases apply:

1. The rule is *conformed* when all conditions in the antecedant as well as the consequent are satisfied by the instance. For example, instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 80] conforms to the rule in Eq. 3.

2. The rule is *violated* if the antecedant holds true but the consequent does not. The rule in Eq. 3 is violated by the instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 23].

3. The rule is not applicable for the instance if any condition in the antecedant is not satisfied, an example instance being [SrcIp = 128.1.5.7, DestIp = 128.4.5.6, DestPort = 21].

Pruning and weighting differ in how the *conformed* (case 1) and *violated* (case 2) rules are treated. They do not differ in rules that are not applicable (case 3) and no action is taken.

3.1.1 Pruning Rules

A conformed rule (case 1) is not updated but the associated p value is modified. Given instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 80], the rule in Eq. 3 has new p value of 3/101 upon conformance. For rule violation (case 2), the rule is eliminated from the rule set (Step 4 in Fig. 1) since D_v is normal and each anomaly is a false alarm. Inapplicable rules (case 3) are left unchanged alongwith their p values.

3.1.2 Weighting Rules

LERAD performs a coverage test to minimize the number of rules (Step 2 in Fig. 1). Thus each selected rule covers a relatively large number of examples in the training set D_t. But removing a rule that causes false alarms also removes coverage on a relative large number of training examples, which can lead to missed detections. Thus, there is a trade off between decreasing false alarms and increasing missed detections. There are two possible solutions. We can either backtrack to find rules that cover the training examples that should be covered, or lessen the belief in the rule instead of eliminating it. For large amounts of data, the latter option is more efficient.

We propose associating a weight with each rule in the rule set to symbolize rule support. Violated rules are penalized by reducing their weights, whereas conformed rules are rewarded with increase in their respective weights. A sample rule using our method is of the form:

$$\begin{aligned} &SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \\ &\Rightarrow DestPort \in \{21, 25, 80\}[p = 3/100, w = 1.0] \end{aligned} \tag{4}$$

The semantics of this rule is similar to the rule in Eq. 3, but a new w value is introduced for the rule weight to represent belief in the rule. p and w are distinct and independent entities — p is the probability of not seeing a value in the consequent when the conditions in the antecedant hold true (i.e. probability of the rule being violated) and corresponds to *predictiveness* from Section 2; weight w, on the other hand, approximates the support of the entire rule (i.e. *belief* from Section 2).

Instead of making a binary decision of retaining or eliminating a rule, we may keep a rule but update its associated weight. The rule consequent and p value may also be updated. For conformed rules (case 1), p is updated similar to rule pruning but has additional w value. Rule violation (case 2) results in updating the rule as well as probability p. For the instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 23], the rule in Eq. 4 is modified as:

$$\begin{aligned} &SrcIp = 128.1.2.3 \bigwedge DestIp = 128.4.5.6 \\ &\Rightarrow DestPort \in \{21, 23, 25, 80\}[p = 4/101, w] \end{aligned} \tag{5}$$

How weights are updated for conformed (case 1) and violated (case 2) rules is discussed in Section 4.

3.2 Scoring Anomalies

During the monitoring stage, LERAD uses the learned rules to assign an anomaly score to each data instance.

3.2.1 Scoring based on Rule Pruning

During detection, given a data instance d, an anomaly score is generated if d violates any of the rules. Let $R' \subset R$ be the set of rules that d violates. The anomaly score is calculated as:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{1}{p_k}, \tag{6}$$

where r_k is a rule in R' and p_k is the p value of rule r_k representing its *predictiveness*. The reciprocal of p_k reflects a surprise factor that is large when anomaly has a low likelihood (small p_k). Intuitively, we are less surprised if we have observed a novel value in a more recent past. Let t_k be the duration since the last novel value was observed in the consequent of rule r_k. A non-stationary model is proposed and each violated rule r_k assigns a score:

$$Score_k = \frac{t_k}{p_k}. \tag{7}$$

Total anomaly score is accumulated over all violated rules:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{t_k}{p_k}. \tag{8}$$

The t_k factor also accommodates the "bursty" nature of network traffic [23], so that multiple successive anomalies generate a single high scoring alarm.

3.2.2 Scoring based on Rule Weighting

Each rule assigns an anomaly score to a test instance $d \in D_T$, a higher score implying more critical aberration. Different algorithms adopt different scoring schemes, the simplest being incrementing the anomaly score by unity. The anomaly score for the test instance is aggregated over all the rules in the rule set. A rule may abstain from assigning a score if it is not applicable (i.e. the antecedent does not hold true). We incorporate the weight representing rule trust to compute the anomaly score:

$$AnomalyScore(d) = \sum_{r_k \in R'} (w_k \times Score_k), \tag{9}$$

where $Score_k$ is due to violation of rule r_k and w_k is the weight of the violated rule. Thus, each rule assigns an anomaly score proportional to its weight, and all the scores are aggregated to compute the total anomaly score. Modified anomaly score for LERAD follows from Eqs. 7, 9:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{w_k t_k}{p_k}. \tag{10}$$

Thus, rule weighting incorporates both the *predictiveness* and *belief* aspects of rule quality.

4. WEIGHTING STRATEGIES

We propose associating a weight to each rule $r \in R$, where weights symbolize rule support. Violated rules are penalized by reducing their weights, whereas conformed rules are rewarded with increase in their respective weights. Next, we present three weighting schemes used in our experiments.

4.1 Winnow-Specialist-based Weight Update

Winnow is an incremental weight updating algorithm for voting experts [18], which correspond to rules in our case. Our first weighting strategy is similar to the Winnow specialist variant of [5]. Initially all rule weights are assigned a value 1, signifying equality of *belief* across the rule set. For any data instance $d \in D_v$, a rule $r \in R$ must either hold good or be inapplicable (in which case it abstains from voting). Any rule violation in D_v corresponds to a false alarm (since D_v comprises of non-attack data) and reduces trust in the culprit rule. If a rule formed during training is not useful, it is likely to be violated many times. Such rules are penalized by multiplicative decay of their weight. On the other hand, if a rule is conformed by a data instance $d \in D_v$ when other rule(s) were violated, it stresses upon validity of the rule and increases trust. Since the rule formed during training is expected to hold true in validation as well, we increase its weight by a small fraction. The intent is to levy a heavy penalty by decreasing the weight by a factor α when the rule is violated, but increase the weights by factor β for a conformed rule.

The strategy to update weights is formally defined by the following weight update function:

$$w_k = \begin{cases} w_k \times \alpha, & \text{if } r_k \in R \text{ is violated} \\ w_k(1+\beta), & \text{if } r_k \in R \text{ is conformed but} \\ & r_j \in R \text{ is violated } (j \neq k) \end{cases} \quad (11)$$

where α, $\beta \in \mathbb{R}$, $0 \leq \alpha < 1$ and $0 \leq \beta \leq 1$. Assuming $\alpha = 0.5$ and inital weight 1, the weight is equal to 0.5 the first time the rule is violated. It is reduced to 0.25 upon second violation and so on. On the other hand, weight is updated as 1.5, 2.25, 3.375 ($\beta = 0.5$) for the first three conformances respectively, when there was atleast one rule violation for the same data instance. Theoretical bounds for the parameters have been presented in [18, 5]. It can be noted that *Pruning* is a special case of this weighting strategy, with $\alpha = \beta = 0$.

4.2 Equal Reward Apportioning

This is a variant of the Winnow-Specialist-based approach explained above. One can observe from Eq. 11 that the weights for correct rules are incremented by a constant factor β. This results in varied weight increments across conforming rules. For example, given $\alpha = \beta = 0.5$, current weights 1.0 and 0.5 of two conforming rules r_1 and r_2 are updated as 1.5 and 0.75 respectively. The Winnow-Specialist-based scheme thus favors rules with already higher weights by increasing their weights even more, resulting in potential imbalance. Moreover, the amount of weight increase is independent of whether a high or low support rule was violated.

The *Equal Reward Apportioning* scheme adopts an impartial approach towards all conforming rules, irrespective of their current weights. This weighting scheme aggregates the total weight reduction due to violation of rules, and rewards the conforming rules by equally distributing the consolidated weight mass amongst them. For each instance in $d \in D_v$, the total penalty TP is computed as:

$$TP = \sum_{r_k \in R_v} (1-\alpha) w_k, \quad (12)$$

where $R_v \subseteq R$ is the set of rules violated by d and $\alpha \in \mathbb{R}$ ($0 \leq \alpha < 1$). Let $R_c \subseteq R$ be the set of conformed rules. The weights are updated as follows:

$$w_k = \begin{cases} w_k \times \alpha, & \text{if } r_k \in R \text{ is violated} \\ w_k + \frac{TP}{|R_c|}, & \text{if } r_k \in R \text{ is conformed} \end{cases} \quad (13)$$

The amount of weight increase for conforming rules is thus dependent on the amount of weight decreased for violated rules. Following the example above, if the violated rule r_3 has weight 0.6, weights for conformed rules r_1 and r_2 are incremented by the same amount (0.15), resulting in weights 1.15 and 0.65 respectively. On the other hand, if a higher trust rule is violated, say rule r_4 with weight 1.0, it provides greater boost to the conforming rules r_1 and r_2 by incrementing their weights by 0.25 each.

4.3 Weight of Evidence

Weight of evidence is defined as the measure of evidence provided by an observation in favor of a target attribute value as opposed to other values for the same target attribute. This measure is based on information theory and has been applied in classification tasks based on event associations [31]. Mathematically, it is the difference in the mutual information when the target attribute Y takes a certain value y and when it doesn't, given some observed value x for the attribute X:

$$W(Y = y / Y \neq y \mid X = x) = \begin{array}{l} I(Y = y; X = x) \\ -I(Y \neq y; X = x), \end{array} \quad (14)$$

where $I(a;b)$ is the mutual information of a and b and is computed as:

$$I(a;b) = P(a,b) log \frac{P(a,b)}{P(a)P(b)}. \quad (15)$$

We cannot apply Eq. 14 directly to our problem since we are not trying to predict a single target value. Rather, we want to measure the gain provided by an observation for the target value to be from a finite set of values. The weight of evidence for the k^{th} rule is reformulated as:

$$w_k(Y \in \{y_1, y_2, \ldots, y_n\} / Y \notin \{y_1, y_2, \ldots, y_n\} \mid \underline{X})$$

$$= I(Y \in \{y_1, y_2, \ldots, y_n\}; \underline{X}) - I(Y \notin \{y_1, y_2, \ldots, y_n\}; \underline{X}) \quad (16)$$

where $\{y_1, y_2, \ldots, y_n\}$ is the set of values for the target attribute Y of the rule r_k; and $\underline{X}$ corresponds to the conditions in the antecedent.

We used this scheme to associate weights with the rules in the rule set. The weight is computed for each rule $r \in R$ based on the evidence in D_v. Contrary to the previous two incremental weighting techniques, this involves batch weighting where evidence is consolidated from D_v as a whole. Moreover, weight of evidence can be positive, negative or

zero. A positive value reflects high trust in the rule whereas a negative or zero value implies otherwise. Only rules with positive weights are kept and the remaining may be eliminated. One can also scale the values by a linear shift of the axis such that all weights are positive. Now the high support (positive weight of evidence) rules have high positive weights, whereas the low (negative/zero weight of evidence) trust rules have low positive weights. Due to its simplicity and intuitiveness, we used the former approach for our experiments.

5. EMPIRICAL EVALUATION

In this section, we evaluate and compare the pruning and weighting schemes for anomaly detection.

5.1 Experimental Data

We evaluated the techniques on five different data sets:
(a) The DARPA/Lincoln Laboratory intrusion detection evaluation network data set (IDEVAL) [17] contains 201 labeled instances of 58 attacks. Since one day of inside traffic is missing, and there are one queso and four snmpget attacks against the router which are not visible from inside the local network, the total number of detectable attacks is 185. Refer [15] for attack taxonomy.
(b) Over 600 hours of network traffic collected on a university departmental server (UNIV) over 10 weeks, comprising of six labeled attacks - port/security scan from inside the firewall, an external HTTP proxy scan, an external DNS version probe, Nimda HTTP worm, Code Red II HTTP worm, and the Scalper worm. The port/security scan has two parts; first an attempt to retrieve the password file by a cgi-bin/htsearch exploit, followed by a port scan, with open ports probed further to test for vulnerabilities.
(c) The BSM audit log from the 1999 DARPA evaluation (IDEVAL BSM) obtained from a Solaris host. Data corresponding to 11 different applications is extracted to get a good mix of benign and malicious behavior. The total number of distinct attacks is 33.
(d) University of New Mexico (UNM) data set, comprising of *lpr*, *login* and *ps* applications contains 3 distinct attacks. *lpr* comprises of 2703 normal and 1001 attack traces from hosts running SUNOS 4.1.4. Traces from the *login* and *ps* applications were obtained from Linux machines.
(e) Florida Tech and University of Tennessee at Knoxville (FIT-UTK) macro execution traces comprise 36 normal and 2 malicious traces that correspond to a distributed denial of service (DDoS) attack, modifying registry settings and execute some other application. The behavior is similar to that exhibited by the "Love bug" worm which opens up the web browser to a specified website and executes a program, modifying registry keys and corrupting user files.

5.2 Experimental Procedures

We considered three attribute sets for each of the two network data sets: reassembled TCP streams (TCP) which reads attributes of the inbound side of unsolicited (client to server) reassembled TCP sessions; inbound client IP packets (PKT) which uses the first 32 pairs of bytes in each IP packet as attributes; and the combination of the two (COMBINED). The data sets will hereafter be referred to as IDEVAL TCP, IDEVAL PKT, IDEVAL COMBINED, UNIV TCP, UNIV PKT and UNIV COMBINED respectively. For the IDEVAL data, we performed training on week 3, which contains no attacks, and testing on weeks 4 and 5. For UNIV data, we tested on weeks 2 through 10, using the previous week as training. By chance, there are no known attacks in week 1. However, there are generally attacks in the training data which could mask detections in the test data.

For host based data sets, we used system calls and related attributes to create application-based models, consisting of return value, error status and other arguments. Only IDEVAL BSM data set had complete argument information. For the UNM and FIT-UTK datasets, the sliding window of contiguous system calls was used, with a window size of 6, as this is claimed to give best results [32].

5.3 Evaluation Criteria

We evaluate and provide comparison for accuracy of models, computational and storage overheads.

Accuracy. For IDEVAL data set (both network and host), an attack is counted as detected if one or more alarms identifies the target address within 60 seconds of any portion of the attack (same as the 1999 DARPA evaluation criterion). Any other alarm is a false alarm. For the UNIV network traffic, we use the criterion that the technique must exactly identify at least one of the packets or TCP sessions involved in the attack. For the UNM and FIT-UTK host data sets, flagging an anomaly anywhere within the attack trace was used to be consistent with previous evaluations.

A Receiver Operating Characteristic (ROC) curve is an effective representation for model evaluation. We use ROC curves for studying the trend in percentage of attacks detected at different false alarm rates. We also list the areas under the ROC curve, where higher area implies better performance [9]. The area under the curve is normalized for the false alarm rate. Since the drawback of anomaly detection is the generation of false alarms, we focus on small false alarm rates (up to 1%).

Storage and Computational Overhead. To evaluate the viability of our technique for online usage, we measure its space and computational requirements. The storage overhead includes the size of the stored model, i.e. rules learned. We also measure the CPU time during the training and testing phases to determine the effectiveness of the techniques.

5.4 Accuracy of Rule Weighting: Number of Attack Detections

The ROC curves for the *Pruning* and weighting (*Winnow-specialist-based*, *Equal Reward Apportioning* and *Weight of Evidence*) variants of LERAD are presented in Fig. 2. The respective areas under ROC curve are listed in Table 1 - weighted area values greater than that of *Pruning* are highlighted. The values in the table are not the Y axis (detection rate) on the ROC curve, but represent the percentage of the maximum area under the curve upto the respective false alarm rate. The random detector has the same false alarm rate and true positive rate for any threshold (x=y line for ROC). For the IDEVAL network data, the ROC curves of Figs. 2(a)-(c) suggest that all techniques generally detect same number of attacks. Even their area under ROC curves

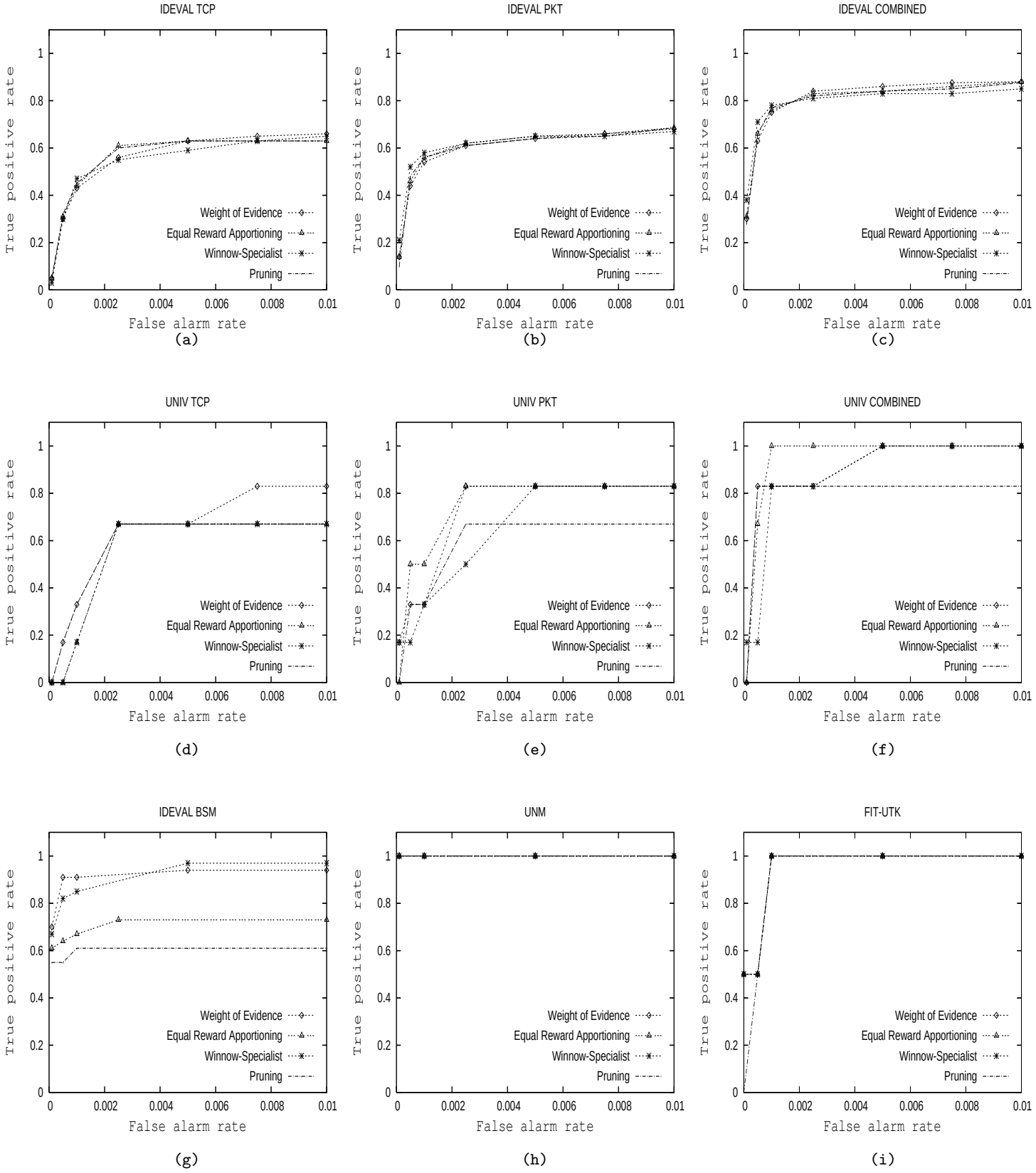

Figure 2: ROC curves (upto 1% false alarm rates) for pruning and weighting strategies on network and host data sets.

Table 1: Area under ROC curve (in %) upto 0.1% and 1% false alarm rates. Results better than *Pruning* are in bold-face. Random detector has area = 0.05% (at 0.1% false alarm rate), 0.5% (at 1% false alarm rate).

Data set	0.1% False Alarm Rate				1% False Alarm Rate			
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Pruning*	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*
IDEVAL TCP	27.2	26.2	26.5	25.8	57.5	55.8	57.5	57.3
IDEVAL PKT	38.6	**44.2**	**38.9**	37.5	61.1	**62.1**	**61.8**	61.0
IDEVAL COMBINED	57.4	**62.9**	**58.0**	56.1	81.1	80.5	**81.6**	**82.4**
UNIV TCP	15.9	4.3	4.3	15.9	59.3	57.0	57.0	**65.3**
UNIV PKT	23.1	21.0	**35.0**	**28.2**	60.1	**66.5**	**75.7**	**73.8**
UNIV COMBINED	58.1	33.5	55.2	58.1	80.5	**88.7**	**95.5**	**91.1**
IDEVAL BSM	56.5	**78.3**	**63.9**	**84.7**	60.6	**92.7**	**71.6**	**92.5**
UNM	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
FIT-UTK	50.0	**95.0**	**95.0**	**95.0**	62.5	**96.3**	**96.3**	**96.3**
Number of times better/tie/worse than Pruning	—	4/1/4	5/1/3	3/3/3	—	5/1/3	6/2/1	6/1/2

are close to each other. But looking at the actual number of detections at 1% false alarm rate, the weighted variants detected 7 new attacks for IDEVAL TCP, including *yaga*, *sechole*, *arppoison*, *syslogd*, *perl*, *crashiis*, and *secret* attacks. For IDEVAL COMBINED, one additional *tcpreset* intrusion was flagged by *Equal Reward Apportioning* and *Weight of Evidence* weighting schemes.

For the UNIV data set (Figs. 2(d)-(f)), *Weight of Evidence* generally outperformed *Pruning* in all cases, though *Equal Reward Apportioning* performed the best for UNIV PKT and UNIV COMBINED data sets. In all UNIV data sets, all three weighted schemes detected one more attack than their rule pruning counterpart at 1% false alarm rate. The *Code Red II worm* was detected using tcp streams whereas the packet data detected the *DNS version probe*, and both were detected when the attribute sets were combined. Hence 100% detection at 1% false alarm rate for weighted LERAD variants in Fig. 2(f).

An interesting observation for IDEVAL and UNIV network data sets was that PKT data detected more attacks than TCP data for all false alarm rates $\leq 1\%$. This was true for all the weighted as well as pruning LERAD variants. Moreover, the COMBINED attribute set detected the maximum attacks suggesting that TCP and PKT data detect attacks which are not detected by the other. Evaluating the attacks detected by TCP and PKT, we saw a significant overlap between the two with some attacks being detected by only one of the attribute set. Combining the two (IDEVAL COMBINED) maximized accuracy by detecting attacks observable in packet data as well as tcp streams.

The ROC curves for the host datasets are presented in Figs. 2(g)-(i). For the IDEVAL BSM data, *Weight of Evidence* detected most attacks at 0.1% false alarm rate (approx. 50% more attacks than *Pruning*) whereas *Winnow-Specialist* had maximum area under curve at 1% false alarm rate, detecting 60% additional attacks than *Pruning*. A total of 12 new attacks were detected (at 1% false alarm rate) across the three weighted variants, including *fdformat*, *ffbconfig*, *guest*, *syslogd*, *httptunnel*, 4 distinct *secret* attacks, *portsweep*, *eject* and *selfping* exploits. Accuracy was the same for the UNM data set, where all techniques detected 3 attacks. On the FIT-UTK data, weighted variants had greater area under ROC curve than *Pruning* at 0.1% and 1% false alarm rates, though all techniques successfully captured the 2 malicious macro executions at 1% false alarm rate.

The last row of Table 1 lists the number of times the respective weighting strategy did better, same and worse than *Pruning*. Results indicate that rule weighting generally has greater accuracy and suggest that the rules discarded by LERAD might be effective in detecting attack based anomalies. It also stresses on the effectiveness of weight updation and its incorporation to score anomalies. At 0.1% false alarm rate, *Equal Reward Apportioning* outperformed *Pruning* 5 times and was worse 3 times. Other weighting techniques drew even. At 1% false alarm rate, *Equal Reward Apportioning* was worse than *Pruning* once but better 6 times. *Weight of Evidence* also did better than *Pruning* in 6 cases and was twice marginally below the pruning counterpart. *Winnow-specialist* performed better on 5 occasions and was worse thrice. All four techniques were tied in one case (UNM data). At 0.1% false alarm rate, *Winnow-specialist* was the best (amongst all 4 techniques) twice whereas all the remaining techniques were best once (ties not included). But all three weighting techniques emerged winners twice each for 1% false alarm rate, and tied for two other data sets.

Table 2: New attacks detected by weighting schemes at 1% false alarm rate.

Factor contributing to attack detection	Data set: attack(s) detected
Conformed rule(s) with increased support	IDEVAL TCP: yaga, sechole
Violated rule(s) with reduced support	IDEVAL TCP: arppoison, syslogd, perl, crashiis, secret IDEVAL COMBINED: tcpreset UNIV TCP: codered UNIV PKT: bindver UNIV COMBINED: codered IDEVAL BSM: fdformat, ffbconfig, guest, syslogd, httptunnel, secret, portsweep, eject, selfping

Table 3: Computational overhead: training phase.

Data set	Data set size (no. of instances)	Total training time (seconds)			
		Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*
IDEVAL TCP	35452	2.06	2.52	2.18	2.56
IDEVAL PKT	280281	3.98	6.27	7.86	12.60
UNIV TCP	141162	8.69	9.57	9.16	10.56
UNIV PKT	1305873	18.15	23.48	23.33	45.25
IDEVAL BSM	1261252	90.55	107.15	107.81	101.27
UNM	3128	0.05	0.04	0.04	0.06
FIT-UTK	94759	0.91	0.95	0.98	1.27

Table 4: Storage requirements: size of rule set.

Data set	Size of rule set = number of rules per week of data			
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*
IDEVAL TCP	52	71	71	71
IDEVAL PKT	100	108	108	106
UNIV TCP	45	88	88	88
UNIV PKT	48	80	80	75
IDEVAL BSM	155	176	176	176
UNM	36	36	36	36
FIT-UTK	11	12	12	12

5.5 Attacks Detected by Rule Weighting

The increase in detections for all weighted variants (*Winnow-specialist*, *Equal Reward Apportioning* and *Weight of Evidence*) is caused by an increase in the anomaly score, which could result from: (a) increased support for conformed rules, and/or (b) scores from rules discarded by *Pruning* but retained (with reduced support) by the weighted variants.

We analyzed the attacks detected using the 3 weighting schemes that were missed by *Pruning* at 1% false alarm rate. The results are listed in Table 2. Most of the new attacks detected are due to rules that were eliminated by LERAD, supporting our claim for retaining the rules but reducing their support. The *Perl* attack was detected in IDEVAL TCP due to an anomalous payload attribute that was part of the exploit. *Syslogd* is a Denial of Service attack that was flagged due to an invalid source whereas *crashiis* involved an unusual request. The *Code Red II* HTTP requests for /default.ida (GET /default.ida?NNNN...) in the UNIV TCP data set are captured by anomaly in the application payload. *fdformat* and *ffbconfig* vulnerabilities are buffer overflow attacks that are detected by encountering unusual arguments in the IDEVAL BSM data. The *syslogd* exploit violated a rule due to syslog segmentation fault.

Two attacks were detected by increasing the weight of existing rules: *yaga* is detected by long duration times due to the TCP connection not being closed after crashing and rebooting the target; whereas the *sechole* exploit is detected by an anomaly in the application payload.

Rule weighting also reinforced the detection of attacks already detected by *Pruning*. This was attributed to large rule weights for some rules, resulting in further increase of the anomaly score. Also, there were multiple alarms for the same attack due to violation of rules introduced by the weighted variants but absent in *Pruning*.

5.6 Computational and Storage Overhead

Besides using different weight updation formulae, there are two key distinctions between the three weighting schemes discussed in Section 4. First, *Winnow-specialist* and *Equal Reward Apportioning* involve incremental weight updation (with every data instance in D_v), compared to batch weight computation for *Weight of Evidence*. Second, *Winnow Specialist* and *Equal Reward Apportioning* schemes suggest keeping all the rules that were previously discarded by *Pruning*, whereas *Weight of Evidence* keeps a subset thereof. These characteristics result in a larger rule set and increased execution times. To check the viability of the techniques for online usage, we studied the overhead involved in rule weighting, both in terms of storage (size of rule set) and the CPU times for training and testing. Experiments were performed on a SUN Ultra 60 workstation with 450 MHz clock speed and 512 MB RAM.

The time requirements for training are listed in Table 3. Most notable difference existed for IDEVAL PKT data set, where *Equal Reward Apportioning* was twice and *Weight of Evidence* took thrice the time than *Pruning*. Since training can be performed offline, higher training times are acceptable. But even in these worst cases, the overheads were not significant considering the amount of training data - approx. 14 μsec/instance for *Pruning* vs. 45 μsec/instance for *Weight of Evidence* in the case of IDEVAL PKT; and 14 μsec/instance (*Pruning*) compared to 36 μsec/instance (*Weight of Evidence*) for UNIV PKT. Training overhead due to weighting was smaller for other data sets.

Storage of the model is determined by the number of rules

Table 5: Computational overhead: testing phase.

Data set	Data set size (no. of instances)	Total testing time (seconds)			
		Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*
IDEVAL TCP	178099	7.72	8.76	8.26	8.65
IDEVAL PKT	534763	3.02	3.90	3.68	3.20
UNIV TCP	143403	7.64	8.50	8.21	8.41
UNIV PKT	1310493	7.70	8.64	8.21	8.18
IDEVAL BSM	1889680	113.93	121.34	121.63	120.97
UNM	7283	0.06	0.06	0.06	0.06
FIT-UTK	13745	0.10	0.10	0.10	0.09

in the rule set. Table 4 lists the number of rules generated for the various data sets. Amongst the network data sets, the least overhead was obtained for IDEVAL PKT where the increase was roughly 6-8%. UNIV TCP presented the maximum overhead, where the number of rules almost doubled for all weighted schemes. Considering the large amount of data used during training (1-9 weeks) and the number of attributes involved, the size of the weighted rule set formed is fairly small, generally less than 100 rules for one week of network training data. Additionally, we could limit the rule set size by eliminating a rule which has been violated a certain number of times or with weight below a threshold. Host data sets displayed lower storage overhead. For the IDEVAL BSM data, the weighted rule set was over 13% larger than *Pruning*. Since 11 different applications were modeled in IDEVAL BSM data, this corresponds to an average of 16 rules per application, which is small for one week of training data. Number of rules were same for UNM data and weighted rule set size exceeded by one rule for FIT-UTK data set.

The time taken during test phase is also dependent on the rule set size. The more the rules, the higher is the number of sanity checks to be made for each test instance. Typically, the time taken should be low for online detection. The results obtained from our experiments are presented in Table 5. Due to larger rule sets, the weighted schemes have longer execution times, making them computationally more expensive than *Pruning*. The maximum overhead was 5.99 μsec/instance for UNIV TCP, followed by 5.84 μsec/instance in the case of IDEVAL TCP, and 3.90 μsec/instance for IDEVAL BSM. Remaining data sets had insignificant overheads. Thus, the weighting overhead is only a fraction of a millisecond per instance, reasonable for an online system.

6. CONCLUDING REMARKS

Machine learning research has been pursued to learn anomaly rules for intrusion detection. LERAD is one such algorithm that can characterize normal behavior in logical rules by finding associations among nominal attributes. It forms a small set of "easy to comprehend" rules that characterize the data. The algorithm is very efficient and effective in capturing anomaly based attacks. A separate *held-out* data is used to validate the rules. Any violations result in the rule being eliminated. We conjecture that discarding rules with possibly high coverage can lead to missed detections. In this paper we propose keeping rules in the rule set and associating a support value with each rule. Weights are representative of rule *support* in our strategy. A conformed rule increases rule trust and hence the weight is increased. On the other hand, weight is decreased upon rule violation.

We present three weighting schemes - *Winnow-specialist-based weighting*, *Equal Reward Apportioning* and *Weight of Evidence*. Besides using different weight updation formulae, the weighting schemes have two key differences. Whereas *Winnow-specialist-based* and *Equal Reward Apportioning* update weight incrementally, *Weight of Evidence* involves batch computation. Further, all the rules $\in R$ are kept in *Winnow-specialist-based* and *Equal Reward Apportioning*, but only a subset are retained in *Weight of Evidence*. We also incorporate the weight into the anomaly scoring mechanism - each rule assigns an anomaly score proportional to its weight, and all the scores are aggregated to compute the total anomaly score. Our technique adds new consequent values to LERAD anomaly rules and recomputes the associated probabilities.

We evaluated pruning and weighting LERAD variants on various network and host data sets. Empirical results show that weighted rules detect more attack-based anomalies than pruning at less than 1% false alarm rates. The weighted strategies accounted for 7 more attack detections for IDEVAL TCP data set. *Code Red II worm* and *DNS version probe* were additionally detected in the UNIV data set, resulting in 100% detections for UNIV COMBINED. But the most significant improvement was in the case of IDEVAL BSM data, where detected 12 new attacks (60% more than *Pruning*) were detected. At 0.1% false alarm rate, *Equal Reward Apportioning* outperformed *Pruning* in 5 data sets and generally performed the best. But all weighted variants were better in terms of attack detections at 1% false alarm rate – *Equal Reward Apportioning* and *Weight of Evidence* did better than *Pruning* in 6 cases whereas *Winnow-specialist* performed better on 5 occasions. We also observed that among the network data sets, COMBINED consistently outperformed TCP and PKT individually, stressing on using both attribute sets to detect a wider range of attacks.

We also analyzed the new attack detected by weighted LERAD variants, which were attributed to high anomaly scores resulting from (a) violations of rules discarded by *Pruning* but retained by weighted variants with reduced support; and (b) increased support for existing rules due to the weight update functions. The former factor contributed to most new attack anomalies. We also computed overheads incurred due to weighting. Since previously discarded rules are retained,

weighted rule sets tend to be larger. But the size of the rule set is still fairly small - 176 rules for one week of IDEVAL BSM training data comprising 11 different applications; and less than 100 rules per week of each network data. The computational overhead of weighting is also minimal, the worst in our our experiments being 30 μsec/instance for training and 6 μsec/instance for testing, making it reasonable for real-time usage.

For future work, we intend to limit the rule set size by eliminating a rule which has been violated many times and its weight falls below a user-defined threshold. We are also exploring other linear weight update functions. Additionally, we intend to incorporate our weighting schemes with other anomaly detection algorithms. An alternate approach for learning is to minimize the rule set after pruning the violated rules. This might reduce the training time, but we suspect that it will also eliminate high coverage (more general) rules, resulting in a larger rule set comprising more specific rules, thereby increasing the test time. We intend to evaluate and compare the accuracy of such a learner with the current technique.

7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 1994.

[2] D. Anderson, T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. Detecting unusual program behavior using the statistical component of the next generation intrusion detection expert system (nides). Technical Report SRI-CSL-95-06, Computer Science Laboratory SRI, 1995.

[3] D. Barbara, J. Couto, S. Jajodia, L. Popyack, and N. Wu. Adam: Detecting intrusions by data mining. In *IEEE Workshop on Information Assurance and Security*, 2001.

[4] S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *IEEE Security and Privacy*, 2006.

[5] A. Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(5):5–23, 1997.

[6] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–285, 1989.

[7] W. Cohen. Fast effective rule induction. In *ICML*, pages 115–123, 1995.

[8] H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *IEEE Security and Privacy*, 2003.

[9] P. Flach. The many faces of roc analysis in machine learning. In *ICML Tutorial*, 2004.

[10] S. Forrest, S. Hofmeyr, A.Somayaji, and T. Longstaff. A sense of self for unix processes. In *IEEE Security and Privacy*, 1996.

[11] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

[12] J. Furnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27:139–171, 1997.

[13] D. Gao, M. Reiter, and D. Song. On gray-box program tracking for anomaly detection. In *USENIX Security Symposium*, 2004.

[14] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *USENIX Security Symposium*, 1999.

[15] K. Kendell. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, MIT, Cambridge, MA, 1999.

[16] C. Krugel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *ACM SAC*, 2002.

[17] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 2000.

[18] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[19] N. Littlestone and M. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[20] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *ICDM*, 2003.

[21] D. Mutz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous system call detection. *ACM Trans. Information and System Security*, 2006.

[22] V. Paxson. Bro: A system for detecting network intruders in real time. In *USENIX Security Symposium*, 1998.

[23] V. Paxson and S. Floyd. The failure of poisson modeling. *IEEE/ACM Trans. Networking*, 3:226–244, 1995.

[24] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[25] W. Robertson, G. Vigna, C. Kruegel, and R. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *NDSS*, 2006.

[26] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA*, 1999.

[27] R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–226, 1990.

[28] R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *IEEE Security and Privacy*, 2001.

[29] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10:105–136, 2002.

[30] G. Tandon and P. Chan. On the learning of system call attributes for host-based anomaly detection. *Intl. Journal on AI Tools*, 15(6):875–892, 2006.

[31] Y. Wang and A. Wong. From association to classification: inference using weight of evidence. *IEEE Trans. Knowledge and Data Engineering*, 15(3), 2003.

[32] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Security and Privacy*, 1999.

[33] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection*, 2000.

[34] I. Witten and T. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Info. Theory*, 1991.