Increasing coverage to improve detection of network and host anomalies *

Gaurav Tandon and Philip K. Chan

Department of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
gtandon@fit.edu, 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 three techniques for increasing coverage - *Weighting*, *Replacement* and *Hybrid*. *Weighting* retains previously pruned rules and associate weights to them. *Replacement*, on the other hand, substitutes pruned rules with other candidate rules to ensure high coverage. We also present a *Hybrid* approach that selects between the two techniques based on training data coverage. Empirical results from seven data sets indicate that, for LERAD, increasing coverage by *Weighting*, *Replacement* and *Hybrid* detects more attacks than *Pruning* with minimal computational overhead.

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.

* Portion of this work appeared in *Proceedings of ACM KDD 2007*[36]. The *KDD* conference paper proposes *Weighting*, this journal submission proposes additional techniques: *Replacement* and *Hybrid*.

Rules for normal behavior can be introduced manually by a human expert, a tedious task that incurs significant effort and cost; or automatically learned from normal data using machine learning. One such technique, called LERAD (LEarning Rules for Anomaly Detection)[22], efficiently learns a succinct set of comprehensible rules from normal data 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. Outright rejection of such rules (called *Pruning* in this paper) reduces rule set coverage over training data. We propose two methods to improve rule coverage: we can either lessen the belief in the violated rule instead of eliminating it (*Weighting*), or backtrack to find rules that cover the training examples that should be covered (*Replacement*). In *Weighting*, rules are associated with weights estimating rule belief. 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. In *Replacement*, coverage is increased by including candidate rules that cover attribute values which have lost coverage due to pruned rules. Additionally, new rules are learned from attribute values that are not covered. We conjecture that increasing coverage over training data would increase the number of attack detections. Thus, we also present a third technique, called *Hybrid*, that chooses between *Weighting* and *Replacement*, the one that has higher coverage on training data.

Our main contributions are:

- to increase the detection rate of LERAD, we propose coverage augmenting techniques (*Weighting*, *Replacement* and *Hybrid*) that overcome the limitations of *Pruning*;
- we compare the techniques on various network and host data sets and demonstrate the efficacy of our models in terms of accuracy and computational overhead - at less than 1% false alarm rate, *Weighting*, *Replacement* and *Hybrid* are more accurate than *Pruning* by 23%, 21% and 29% respectively;
- we study the effect of coverage on accuracy for our data sets - *Hybrid* selects higher coverage technique and generally detects more attacks;
- we analyze the new attack detections - most of them are attributed to rules ignored by *Pruning* during validation (included in *Weighting*) and coverage test (included in *Replacement*).

In Section 2 we discuss some network and host-based anomaly detection systems and rule learning algorithms. Section 3 briefly describes pruning in the LERAD algorithm. Weighting for LERAD and three weighting strategies are described in Section 4. The replacement technique is detailed in Section 5, whereas Section 6 motivates and presents our hybrid model. Section 7 evaluates and compares the accuracy of *Pruning*, *Weighting*, *Replacement* and *Hybrid* on multiple network and host data sets. The effect of coverage on

accuracy is studied for *Hybrid* and new attacks detected by *Weighting* and *Replacement* are analyzed. The rule set size and CPU time requirements for training and testing are also presented. Section 8 summarizes the results and presents some future research directions.

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 [29] and BRO [25] are examples of rule-based network anomaly detection systems. Rules can be handcoded to restrict access to specific hosts and services. But manual update of rules is deemed impractical. Intrusion detection systems (IDSs) such as NIDES [3], ADAM [4], 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 [28]. Some anomaly detection algorithms are for specific attacks (e.g., portscans [34]) or services (e.g., DNS[18]). More recent work used pointwise mutual information for anomaly detection [9]. Since a large number of dependencies of attribute values is stored, the storage and computational overhead for the technique can be quite high. Complementing network systems, 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 [12, 38, 39] and have been represented using finite state automata [31] and neural networks [16]. Other features used include system call arguments [24, 35, 5] and call stack information [10, 15].

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 accurately 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 [2], 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 [8] and CN2 [7], 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 [21] and Boosting [30, 13], 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 [27] removes conditions from the antecedent of a rule if the estimated accuracy

improves. Furnkranz [14] has a review of various rule pruning techniques. For rule learning algorithms, studies cited above 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.

Rules can generally be learned in two ways: generate and test strategy vs. data driven approach. Generate and test adpots a hypothesis driven approach, where a general to specific search is usually performed in the hypothesis space. CN2 [7] is a general to specific beam search algorithm that maintains the k best candidates at each step, where rules are refined based on their accuracy on the data set. Apriori [1] learns all association rules above user defined confidence and support thresholds. It is also a generate and test approach, with only a subset of rules generated at each step are considered for specialization in the subsequent iteration. Using a general-to-specific search, ITRULE [32, 33] learns k (user-specified) rules that have the highest information content based on the J-measure (mutual-information based). Different candidate rules are formed with each attribute as the consequent. According to information-theoretic bounds, specializations of the current rules are not explored if they will not yield higher information content than the top k rules found so far.

Contrary to generate and test strategy, data driven approach involves generation of hypotheses that is constrained by specific data instances. AQ15 [23] generates a rule to cover a specific attribute value, and these rules are specialized at each step. After each rule, the algorithm picks another attribute value not covered to initiate search in the hypothesis space. For the LERAD algorithm, two instances are randomly chosen and rules are generated using matching attribute values, making it a data driven approach. Hypotheses search is from general to specific, and rules are updated by adding values over entire training data. LERAD differs from AQ15 in that it allows different attributes in the rule consequent, whereas AQ15 learns classes based on a single attribute. Also, AQ15 generates all rules that cover a data instance. In contrast, LERAD randomly choses matching attributes, thus generating a subset of the rules and making the algorithm more efficient. In this paper, we present *Replacement* variant for LERAD that revisits candidate rules to replace rules discarded during validation. Since rules generated previously are checked against the validation set for false alarms, this step is hypotheses driven. In addition, *Replacement* takes into account attribute values not covered and learns rules based on those target values, making this step a data driven strategy and the technique a mixture of data and hypotheses driven methods.

Table 1. Example data set and rule set.

d_i	$SrcPort$	$DestPort$	$SrcIp$	$DestIp$
d_1	80	80	128.1.2.3	128.4.5.6
d_2	80	80	128.1.2.3	128.4.5.6
d_3	80	25	128.3.2.1	128.4.5.6

r_1: $* \Rightarrow DestPort \in \{25, 80\}[p = 2/3]$
r_2: $SrcIp = 128.1.2.3 \Rightarrow DestIp \in \{128.4.5.6\}[p = 1/2]$
r_3: $DestIp = 128.4.5.6 \Rightarrow SrcPort \in \{80\}[p = 1/3]$

3 Rule Pruning in LERAD

LEarning Rules for Anomaly Detection (LERAD) [22] is an efficient randomized algorithm that forms conditional rules of the form:

$$a_1 = v_{11} \wedge a_2 = v_{23} \wedge ... => a_c \in \{v_{c1}, v_{c2}, ...\}\ [p] \tag{1}$$

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 $P(C|A)$, 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 [40] is used:

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

where n is the total number of observed events and r is the number of unique observed events. That is, n is the number of data instances that conform to the rule, and r is the cardinality of the set of possible attribute values in the consequent. A sample network anomaly rule for LERAD is:

$$SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\}[p = \frac{r}{n} = \frac{3}{100}] \tag{3}$$

A data instance is a feature vector comprised of all feature attribute values, and multiple data instances form a data set. For example, a network data set may be composed of data instances represented by the feature vector $< SrcPort, DestPort, SrcIp, DestIp >$. The rule of Eq. 3, which claims three distinct destination ports (21-FTP, 25-SMTP, 80-HTTP) given the source and destination IP addresses, is satisfied by 100 data instances. One or more such rule(s) forms the rule set. A synthetic network data set and rule set is presented in Table 1, with the semantics of data instance and rule as explained above.

Definition 1. Let i be a data instance and j be an attribute. $cover_{ij}$ is 1 if there exists a rule r_k (Eq. 1) in the rule set R such that instance i satisfies

the condition(s) in the antecedent of rule r_k and the value of attribute j in instance i is a member of the set of values of the same attribute j in the consequent of rule r_k. That is, $cover_{ij}$ indicates if the value of attribute j in instance i is covered by a rule in the rule set. More formally:

$$cover_{ij} = \begin{cases} 1, \text{ if } j^{th} \text{ attribute value is in consequent} \\ \quad \text{of a rule satisfied by } i^{th} \text{ instance} \\ \\ 0, \text{ otherwise} \end{cases} \tag{4}$$

In Table 1, rule r_1 is applicable for all three data instances, and all $DestPort$ values are contained in the consequent. Hence, $cover_{12} = cover_{22} = cover_{32}$=1. The attribute values not covered by the rule set are $SrcIp$ of data instances $\{d_1, d_2, d_3\}$, each of which has a cover value of 0. Similarly, $cover_{34}$=0, since $DestIp$ of data instance d_3 is not covered by any rule in the rule set.

Definition 2. For the entire data set, *coverage* is defined as the fraction of attribute values covered by the rule set. Let N be the total number of instances, and M the number of distinct attributes, then *coverage* is formally defined as:

$$coverage = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} cover_{ij} \tag{5}$$

The example in Table 1 has three data instances ($d_1 - d_3$) and four attributes. Eight attribute values are covered by the 3 rules in the rule set, resulting in a *coverage* value of 0.67.

3.1 Training Candidate Rules and Coverage Test

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*

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. perform coverage test - select a "minimal" set of candidate rules that covers D_s:
   a) sort candidate rules in increasing order of probability of being violated
   b) discard rules that do not cover any attribute values in D_s
3. train the selected candidate rules on D_t
4. eliminate the rules that cause false alarms on D_v

Fig. 1. Main steps of LERAD algorithm

Table 2. Example training data subset $D_s = \{d_i\}$ for $i = 1..3$ and rules r_k ($k = 1..3$) generated from D_s. Consequent attribute values in data instances are marked by (r_k) in coverage test.

d_i	$SrcPort$	$DestPort$	$SrcIp$	$DestIp$
d_1	80	80 (r_2)	128.1.2.3	128.4.5.6
d_2	80	80 (r_2)	128.1.2.3	128.4.5.6
d_3	80	25 (r_1)	128.3.2.1	128.4.5.6

r_1: $* \Rightarrow DestPort \in \{25, 80\}[p = 2/3]$
r_2: $SrcIp = 128.1.2.3 \Rightarrow DestPort \in \{80\}[p = 1/2]$
r_3: $SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{80\}[p = 1/2]$

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 Fig. 1. Step 1 generates and evaluates candidate rules from a small data sample D_s (such that $|D_s| \ll |D_t|$), which allows efficient training. Table 2 contains example data and candidate rules to help illustrate the algorithm. A pair of data instances, say d_1 and d_2, are picked at random. Rules are generated by selecting matching attributes in a random order. In the example, d_1 and d_2 have all matching attributes. Randomly selecting them in the order $DestPort$, $SrcIp$, and $DestIp$, we get the following rules:
r_1: $* \Rightarrow DestPort \in \{80\}$
r_2: $SrcIp = 128.1.2.3 \Rightarrow DestPort \in \{80\}$
r_3: $SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{80\}$

A rule so generated implies that the attribute in the consequent can have a value from a set of values only if the conditions in the antecedent are satisfied. It may so happen that there is a consequent but no antecedent in a rule formed by LERAD. This means that an attribute can take any value from its set of values without the need to satisfy any other condition. Such a situation is presented in rule r_1 where the antecedent is represented by a wildcard character *.

Step 2 extends the rules from previous step to the entire set D_s. It selects a small set of predictive rules that sufficiently describe D_s. This allows learned models to be small. This step, called *coverage test*, is based on two heuristics. First, rules with lower $p = P(\neg C|A)$ are preferred. Second, a rule can cover multiple instances in D_s, but an instance does not need to be covered by more than one rule. Hence, rules are sorted based on p and evaluated in ascending order (Step 2a). For each rule, instances covered by the rule are marked. If a rule cannot mark any remaining unmarked instances, it is removed. That is, rules with lower p are retained and rules that do not contribute to covering instances not covered by previous rules with lower p values are discarded (Step 2b). More specifically, since rules can have different attributes in the

consequent, the attribute of an instance, not the entire instance, is marked. Hence, a rule is removed only if it cannot mark any unmarked attribute of any instance.

Table 2 contains example data and candidate rules to help illustrate the algorithm. Rules r_2 and r_3 in the table have a lower p (1/2) than r_1, so r_2 or r_3 is evaluated first. Rule r_2 is arbitrarily picked before r_3, and marks *DestPort* of d_1 and d_2. Then r_3 cannot mark any attribute and is removed. Finally, r_1 marks *DestPort* of d_3. At this point, rules r_2 and r_1 are selected. This procedure guarantees at least one attribute of all instances in D_s are marked by a subset of candidate rules. Also, rules that are subsumed by more general rules are automatically removed due to a higher p.

The selected rules are then updated using the much larger set D_t (Step 3 in Fig. 1) by updating the consequent and p. If the antecedent of a rule matches an instance in D_t and the consequent of the rule does not contain the corresponding value, the value is added to the consequent. p is updated by updating the number of instances that match the antecedent in D_t (n) and values in the consequent (r). The validation set D_v is used in Step 4, and is described next in context of *Pruning*.

3.2 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].

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 along with their p values. This version of LERAD is referred to as *Pruning* for the remainder of the paper.

3.3 Scoring Anomalies

During the monitoring stage, LERAD uses the learned rules to assign an anomaly score to each data instance. 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 [26], so that multiple successive anomalies generate a single high scoring alarm.

We claim that *Pruning* reduces rule coverage, resulting in lower accuracy. We propose two solutions to increase coverage: retain pruned rules with lower rule belief using *Weighting*, or revisit candidate rules to replace pruned rules, called *Replacement*. Our hypothesis is based on the putative link between train set coverage and test set accuracy – increasing the coverage might result in increased accuracy. Resampling techniques such as cross-fold validation or bootstrap are not applicable due to the temporal aspects of the data sets. *Weighting* and *Replacement* are discussed in the next two sections.

4 Rule Weighting

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.

We propose associating a weight with each rule in the rule set to symbolize rule belief. Violated rules are penalized by reducing their weights, whereas

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. coverage test - select a "minimal" set of candidate rules that covers D_s:
   a) sort candidate rules in increasing order of probability of being violated
   b) discard rules that do not cover any attribute values in D_s
3. train the selected candidate rules on D_t and assign weight of unity
4. validate rules on D_v
   a) increase weight on rule conformance (increase rule belief)
   b) decrease weight on rule violation (reduce rule belief)

Fig. 2. Rule Weighting in LERAD

conformed rules are rewarded with increase in their respective weights. A sample rule using our method is of the form:

$$\begin{aligned} &SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \\ &\Rightarrow DestPort \in \{21, 25, 80\}[p = 3/100, w = 1.0] \end{aligned} \tag{9}$$

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 *belief* of the entire rule.

4.1 Validating Rules

The training is similar to *Pruning*, as seen in Fig. 2. The main difference lies in the validation step (Step 4 in Fig. 2). Instead of making a binary decision of retaining or eliminating a rule, we keep a rule but update its associated weight. The rule consequent and p value may also be updated. For conformed rules (case 1 in Sec. 3.2), p is updated similar to *Pruning* but has an additional w value. Rule violation (case 2 in Sec. 3.2) 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. 9 is modified as:

$$\begin{aligned} &SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \\ &\Rightarrow DestPort \in \{21, 23, 25, 80\}[p = 4/101, w] \end{aligned} \tag{10}$$

How weights are updated for conformed and violated rules (case 1 and 2 respectively from Sec. 3.2) is discussed next. No action is taken for inapplicable rules (case 3).

4.2 Weighting Strategies

We propose associating a weight to each rule $r_k \in R$, where weights symbolize rule *belief*. 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.

Winnow-Specialist-based Weight Update

Winnow is an incremental weight updating algorithm for voting experts [20], which correspond to rules in our case. Our first weighting strategy is similar to the Winnow specialist variant of [6]. 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 $r_k \in R$ 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:

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

where $\alpha, \beta \in \Re$, $0 \leq \alpha < 1$ 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 at least one rule violation for the same data instance. Theoretical bounds for the parameters have been presented in [20, 6]. It can be noted that *Pruning* is a special case of this weighting strategy, with $\alpha = \beta = 0$.

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 belief 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:

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

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

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

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. In the event that no rule is violated, weight for conforming rule remains unchanged.

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 [37]. 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:

$$W(Y = y / Y \neq y \mid X = x) = I(Y = y; X = x) - I(Y \neq y; X = x), \tag{14}$$

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

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

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:

$$\begin{aligned} & w_k(Y \in \{y_1, y_2, \ldots, y_n\} / Y \notin \{y_1, y_2, \ldots, y_n\} \mid \underline{\mathrm{X}}) \\ & = I(Y \in \{y_1, y_2, \ldots, y_n\}; \underline{\mathrm{X}}) - I(Y \notin \{y_1, y_2, \ldots, y_n\}; \underline{\mathrm{X}}) \end{aligned} \tag{16}$$

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 $r_k \in R$ 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 belief (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.

4.3 Scoring Anomalies

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:

$$AnomalyScore(d) = \sum_{r_k \in R'} (w_k \times Score_k), \tag{17}$$

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, 17:

$$AnomalyScore(d) = \sum_{r_k \in R'} \frac{w_k t_k}{p_k}. \tag{18}$$

Thus, anomaly score in *Weighting* incorporates both the *predictiveness* and *belief* aspects of rule quality.

5 Rule Replacement

Weighting introduces the additional aspect of rule belief and increases rule coverage over training data by retaining previously pruned rules. Alternatively, one can revisit rules rejected during coverage test (Step 2 in Fig. 1). In this section, we present the *Replacement* technique that substitutes pruned rules with new rules to increase coverage over training data. *Replacement* increases coverage in two ways:

- by considering rules with lower predictiveness that were rejected during coverage test, and
- by generating new candidate rules from instances in the training data set with attribute values not covered.

The main steps of *Replacement* are presented in Fig. 3. Steps 1-4 are same as *Pruning* with the exception of Step 2b, where rules that do not increase coverage are retained in a rule pool R_{pool}, instead of eliminating them. The validation phase (Step 4) may prune rules with high predictiveness, resulting in significant loss of coverage. Rules from R_{pool} can then be re-evaluated to increase coverage over training data (Step 5). Rules that increase the coverage are added to the rule set in Step 5a. It can be noted that such rules will have lower predictivess than the rules in Rule Pruning. Step 5b validating the new rules against D_v. Rules causing false alarms are eliminated, and remaining rules from R_{pool} are considered in the subsequent iteration. This loop terminates when no rules remain the pool, or all attribute values are covered, or the entire rule set conforms to the validation data set. At the end of each iteration, coverage increases or remains the same. To maximize coverage,

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 (i.e. coverage test):
   a) sort candidate rules in increasing order of probability of being violated
   b) R_{pool} = rules that do not cover any consequent attribute values
   c) select remaining rules as candidate rules
3. train the selected candidate rules on D_t
4. eliminate the rules that cause false alarms on D_v
5. while (rule violations in $D_v \wedge R_{pool} \neq \emptyset \wedge$ candidate rules $\in R_{pool} \wedge$ coverage < 1.0)
   a) if (candidate rule increases coverage in D_t)
      add candidate rule to R and remove it from R_{pool}
   b) validate the rule set on D_v and eliminate rules violated
6. generate new candidate rules from D_u, which are instances in D_s with attribute values not covered
7. select a "minimal" set of candidate rules that covers D_u
8. eliminate violating rules on D_v and add remaining rules to R

Fig. 3. Rule Replacement in LERAD

Table 3. Example training data subset $D_u = d_i$ $i = 4, 5$, representing attribute values not covered for *DestPort*.

d_i	*SrcPort*	*DestPort*	*SrcIp*	*DestIp*
d_4	25	80	128.1.2.3	128.7.8.9
d_5	25	80	128.1.2.3	128.0.3.5

all candidate rules are considered in each iteration, except those already in the rule set or pruned in validation in previous iterations. Additionally, new candidate rules are learned from instances with attribute values that are not covered (Step 6). Only data instances with not yet covered values are used in this step to generate new candidate rules, which are constrained to have those values in the rule consequent. Generalizations and rules with higher predictiveness are preferred to keep the rule set small (Step 7). Step 8 involves pruning rules that cause false alarms on the validation data set. Anomalies are scored as shown in Eq. 8 for the *Pruning* strategy.

Consider the synthetic data and rules from Table 2. Coverage test removes r_3 ($SrcIp = 128.1.2.3 \wedge DestIp = 128.4.5.6 \Rightarrow DestPort \in \{80\}[p = 1/2]$), which is now added to the rule pool R_{pool}. Assume rule r_2 ($SrcIp = 128.1.2.3 \Rightarrow DestPort \in \{80\}[p = 1/2]$) is pruned during validation step. This reduces the *cover* value of attribute *DestPort* from three to one instance. *Replacement* allows rules from the pool to substitute for pruned rules in order to increase coverage. Thus, r_3 is added to the rule set, increasing the attribute *Cover* back to three instances in Table 2. In addition, assume a couple more data instances of Table 3 that lose coverage due to pruned rules r_2. Note that r_3 does not apply to these data instances. New rules are generated from these samples (Step 6 in Fig. 3) by constraining the rule consequent to include *DestPort* values of d_4 and d_5 and generating the antecedent through matching attributes such that the rule is satisfied by these instances. An example rule that covers the two data instances is:

$$SrcIp = 128.1.2.3 \wedge SrcPort = 25 \Rightarrow DestPort \in \{80\}[p = 1/2]. \quad (19)$$

If the new candidate rule conforms to D_v, it is added to the final rule set R.

6 Hybrid Approach

In the previous sections, we presented *Weighting* and *Replacement* for increasing rule coverage. In this section, we present the *Hybrid* approach that chooses among the two techniques based on which one has higher coverage on the training data. *Weighting* rule set comprises of rules with high predictiveness but can have low belief. *Replacement* may constitute many rules with low predictiveness. Combining the two approaches can result in rules with low predictiveness and low belief, and thus avoided. Thus, *Hybrid* picks one or the other based on coverage.

Input: sample set (D_s), training set (D_t), and validation set (D_v)
Output: LERAD rule set R
1. generate rule set R_w using *Weighting* steps of Fig. 2
2. generate rule set R_r using *Replacement* steps of Fig. 3
3. compute c_w = attribute value coverage of R_w over D_t
4. compute c_r = attribute value coverage of R_r over D_t
5. if ($c_w \geq c_r$)
   $R \leftarrow R_w$
   else
   $R \leftarrow R_r$

Fig. 4. Hybrid approach in LERAD

The main steps of the hybrid approach are presented in Fig. 4. Rule sets R_w and R_r are generated in Steps 1 and 2 using algorithms of Figs. 2 and 3 respectively. Let c_w be the coverage of weighting rule set and c_i be the coverage of i^{th} iteration in *Replacement* (c_r after all iterations). Thus, coverage for pruning is c_1. It can be noted that $c_w \geq c_1$ and $c_{i+1} \geq c_i$. But the relation between c_w and c_r depends on the data set, as depicted in Figs. 5a-b. Fig. 5a shows coverage over all iterations for UNIV TCP data set (data set details presented in Section 7.1). Rule weighting has higher coverage than *Replacement*. On the other hand, *Replacement* betters the coverage of *Weighting* in the fourth iteration for UNIV PKT data set, as evident from Fig. 5b. Assuming higher coverage leads to higher accuracy, the *Hybrid* approach selects

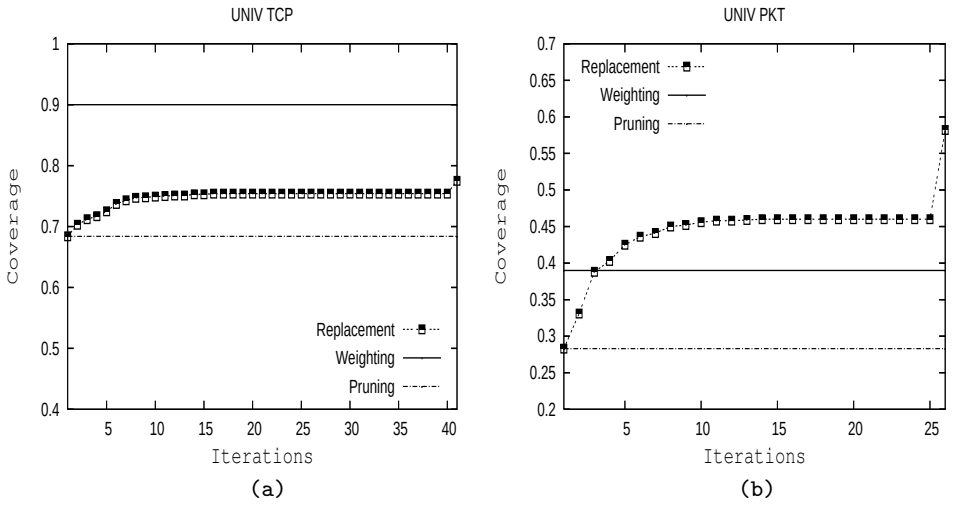

Fig. 5. Coverage comparsion for *Pruning*, *Weighting* and *Replacement* for (a) UNIV TCP and (b) UNIV PKT.

the technique with the higher coverage. Anomalies are scored using Eq. 8 for *Replacement* or Eq. 18 if *Weighting* is selected. For *Replacement*, a jump in coverage can be noted during the last iteration in Fig. 5. This increase is due to new candidate rules learned from attribute values that were previously not covered (Steps 6-8 in Fig. 5).

7 Empirical Evaluation

In this section, we evaluate and compare the *Pruning*, *Weighting*, *Replacement*, and *Hybrid* schemes for anomaly detection.

7.1 Experimental Data

We evaluated the techniques on five different data sets:

1. The DARPA/Lincoln Laboratory intrusion detection evaluation network data set (IDEVAL) [19] 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 to Kendell's thesis [17] for attack taxonomy.
2. 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.
3. The BSM audit log from the DARPA evaluation obtained from a Solaris host (BSM data set). 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.
4. 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. 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.

7.2 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. The data sets will hereafter be referred to as IDEVAL TCP, IDEVAL PKT, UNIV TCP, and UNIV PKT 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 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 [38].

7.3 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. This is achieved by varying the anomaly threshold in our techniques. We also list the areas under the ROC curve, where higher area implies better performance [11]. 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.

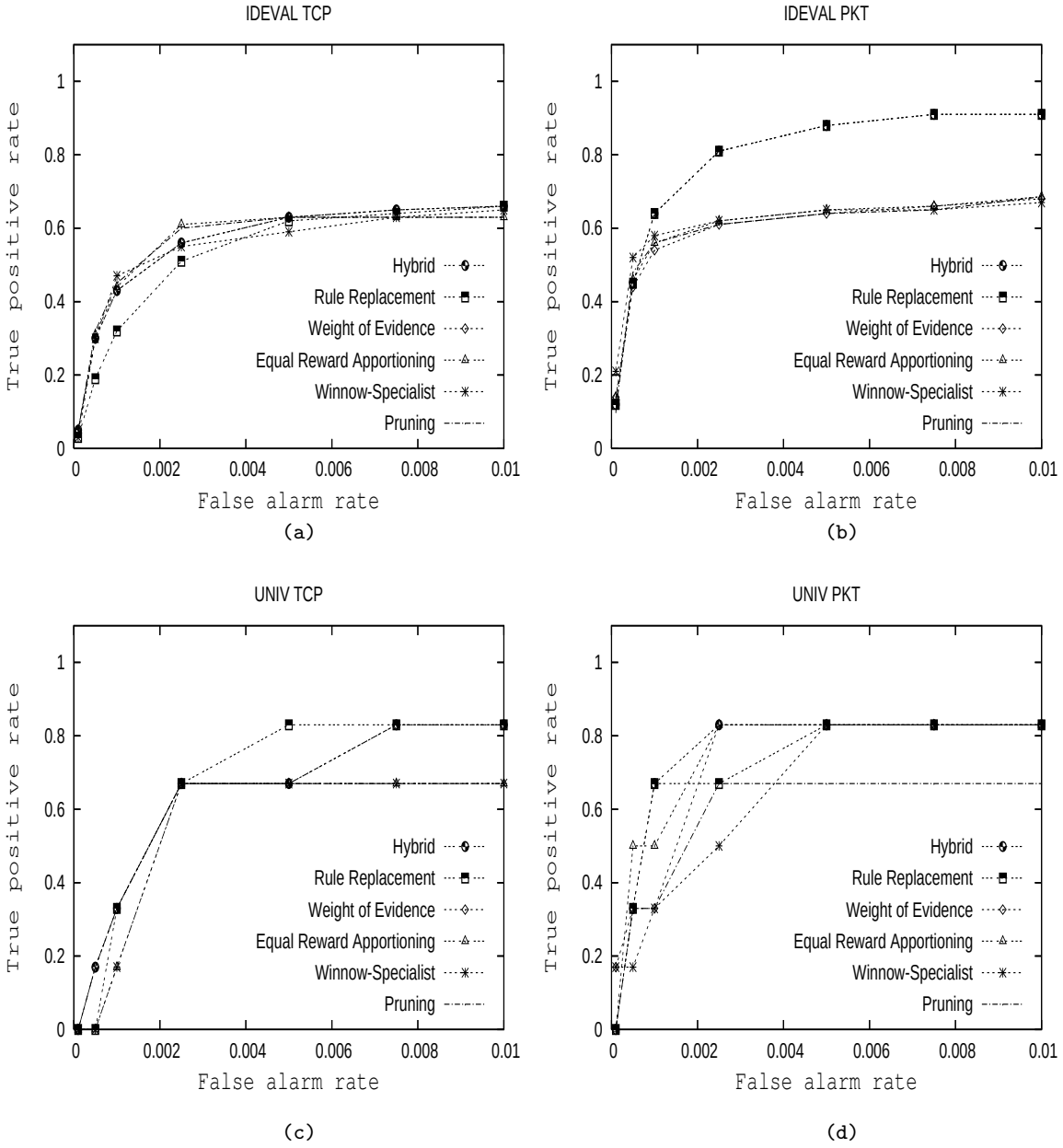

Fig. 6. ROC curves (up to 1% false alarm rates) for *Pruning*, *Weighting*, *Replacement* and *Hybrid* strategies for LERAD on network data sets.

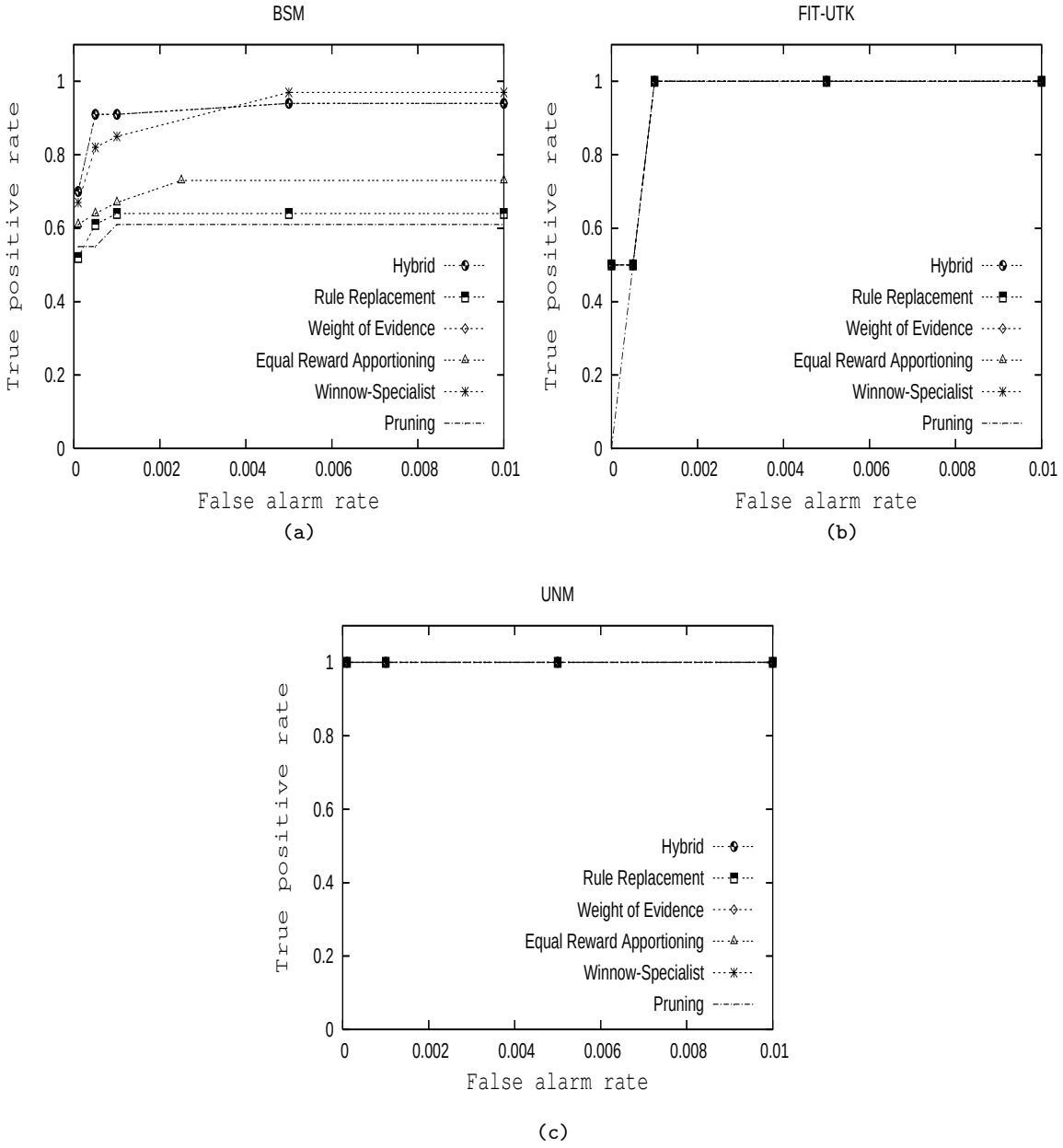

Fig. 7. ROC curves (up to 1% false alarm rates) for *Pruning*, *Weighting*, *Replacement* and *Hybrid* strategies for LERAD on host data sets.

7.4 Accuracy of Weighting, Replacement and Hybrid

The ROC curves for the *Pruning*, *Weighting*, *Replacement* and *Hybrid* variants of LERAD are presented in Figs. 6, 7. The respective areas under ROC curve are listed in Table 4 - 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 up to 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). Since *Weight of Evidence* had the highest accuracy in weighting, it was used in conjunction with *Replacement* for *Hybrid*.

Figs. 6a-b present the ROC curves for the IDEVAL network data. Fig. 6a suggests that all techniques generally detect same number of attacks for IDEVAL TCP. Even their area under ROC curves are close to each other. But looking at the actual number of detections at 1% false alarm rate, *Weighting* and *Hybrid* variants detected 7 new attacks for IDEVAL TCP; whereas six extra attacks were detected by *Replacement*. For IDEVAL PKT, *Replacement* and *Hybrid* improved the AUC of *Pruning* by 33%.

For the UNIV data set (Figs. 6c-d), *Weight of Evidence* generally outperformed *Pruning* in all cases. *Replacement* had higher AUC than *Pruning* and all weighted variants at 1% false alarm rate. *Hybrid* selected *Weighting* for UNIV TCP and *Replacement* for UNIV PKT because of higher coverage on the resepective data sets. *Weighting*, *Replacement* and *Hybrid* schemes detected one more attack than *Pruning* for both UNIV data sets - the *Code Red II worm* was detected using tcp streams whereas the packet data detected the *DNS version probe*. An interesting observation for all LERAD variants on IDEVAL and UNIV network data sets was that PKT data detected more attacks than TCP data for all false alarm rates $\leq 1\%$. 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.

The ROC curves for the host datasets are presented in Figs. 7. For the 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*. *Replacement* did marginally better than *Pruning*, with only one additional attack detection. *Hybrid* had same accuracy as *Weighting* and a total of 12 new attacks were detected (at 1% false alarm rate), including *fdformat*, *ffbconfig*, *guest*, *syslogd*, *httptunnel*, 4 distinct *secret* attacks, *portsweep*, *eject* and *selfping* exploits. On the FIT-UTK data, *Weighting*, *Replacement* and *Hybrid* 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. Accuracy was the same for the UNM data set, where all techniques detected 3 attacks.

The second last row of Table 4 lists the number of times the respective strategy did better, same and worse than *Pruning*. Results indicate that

Table 4. Area under ROC curve (in %) up to 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**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	27.2	26.2	26.5	25.8	17.5	25.8
IDEVAL PKT	38.6	**44.2**	**38.9**	37.5	**39.9**	**39.9**
UNIV TCP	15.9	4.3	4.3	15.9	8.3	15.9
UNIV PKT	23.1	21.0	**35.0**	**28.2**	**31.6**	**31.6**
BSM	56.5	**78.3**	**63.9**	**84.7**	**59.1**	**84.7**
FIT-UTK	50.0	**95.0**	**95.0**	**95.0**	**95.0**	**95.0**
UNM	100.0	100.0	100.0	100.0	100.0	100.0
Number of times better/tie/worse than Pruning	—	3/1/3	4/1/2	3/2/2	4/1/2	4/2/1
Average improvement (%) over Pruning [excluding UNM]	—	9.6	13.3	25.7	8.6	29.2
Data set	**1% False Alarm Rate**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	57.5	55.8	57.5	57.3	54.1	57.3
IDEVAL PKT	61.1	**62.1**	**61.8**	61.0	**81.1**	**81.1**
UNIV TCP	59.3	57.0	57.0	**65.3**	**68.6**	**65.3**
UNIV PKT	60.1	**66.5**	**75.7**	**73.8**	**76.7**	**76.7**
BSM	60.6	**92.7**	**71.6**	**92.5**	**63.5**	**92.5**
FIT-UTK	62.5	**96.3**	**96.3**	**96.3**	**96.3**	**96.3**
UNM	100.0	100.0	100.0	100.0	100.0	100.0
Number of times better/tie/worse than Pruning	—	4/1/2	4/2/1	4/1/2	5/1/1	5/1/1
Average improvement (%) over Pruning [excluding UNM]	—	18.8	15.9	23.3	21.5	29.0

Weighting, *Replacement* and *Hybrid* generally have greater accuracy than *Pruning*. This suggests that the rules discarded by LERAD might be effective in detecting attack based anomalies. At 0.1% false alarm rate, *Hybrid* outperformed *Pruning* four times and was worse once. *Equal Reward Apportioning* and *Replacement* also had higher AUC on four data sets but lower AUC on 2 data sets. At 1% false alarm rate, *Equal Reward Apportioning*, *Replacement* and *Hybrid* had same or better accuracy than *Pruning* on six occassions, and were worse only once; whereas *Weight of Evidence* and *Winnow-specialist* had higher AUC five times and lower AUC for two data sets. Overall, *Hybrid* had

better accuracy than *Pruning* on most data sets at both 0.1% and 1% false alarm rates.

Though the second last row of the table gives us the number of times our techniques better, same or worse than *Pruning*, it fails to capture the magnitude of gain or loss in accuracy. This is captured in the last row of the table, which denotes the average percentage of improvement in the AUC over *Pruning*, and is defined as:

$$\frac{1}{|datasets|} \sum_{|datasets|} \left(\frac{AUC_x - AUC_{Pruning}}{AUC_{Pruning}}\right) \times 100, \tag{20}$$

where $x \in \{Weighting, Replacement, Hybrid\}$. The UNM data set is excluded from calculating the average improvement in Table 4 because improvement over *Pruning*'s 100% AUC is not possible and all the methods have 100% AUC (no loss in accuracy). Among the weighted variants, *Weight of Evidence* had the best accuracy among all weighting techniques - an improvement of 25.7% and 23.3% over *Pruning* at 0.1% and 1% false alarm rate respectively. *Replacement* had a gain of 8.6% at 0.1% false alarm rate, compared to an improvement of 21.5% at 1% false alarm rate. But *Hybrid* performed the best, with the most average improvement in accuracy of about 29%.

We also applied the paired t-test to check if the improvements in accuracy were statistically significant. We paired each of *Weighting*, *Replacement*, *Hybrid* with *Pruning* to obtain the confidence intervals. The seven data sets result in six degrees of freedom. We consider accuracy improvement with confidence level lower than 90% as not statistically significant. Since *Weight of Evidence* had the maximum improvement in accuracy, it was chosen among the three *Weighting* techniques. Results show that at 0.1% false alarm rates, improvement in accuracy for *Weighting* and *Hybrid* over *Pruning* is statistically significant at the 90% level but that of *Replacement* is not. At 1% false alarm rates, there is a statistically significant increase in accuracy for *Weighting* and *Replacement* at the 95% level, and 97.5% level for *Hybrid* over *Pruning*. Comparing the change in accuracy of *Hybrid* over *Weighting* and *Replacement*, we found that the result is statistically significant at 90% level at 1% false alarm rate but not significant at 0.1% false alarm rate.

7.5 Coverage vs. Accuracy

We measured the coverage of *Weighting* and *Replacement* techniques on the various data sets. The results are compiled in Table 5, where bold values are better. Results show that *Weighting* has higher or same coverage except for IDEVAL and UNIV PKT data sets, where *Replacement* had higher coverage.

The *Hybrid* approach selects *Weighting* or *Replacement* based on coverage on training data. It assumes higher coverage yields higher accuracy (AUC). Is this assumption supported? Which data sets have higher accuracy with higher coverage at 0.1% and 1% false alarm rates, and how does that explain

Table 5. Coverage comparison of *Weighting* and *Replacement*

Data set	Weighting	Replacement
IDEVAL TCP	**0.91**	0.86
IDEVAL PKT	0.43	**0.64**
UNIV TCP	**0.90**	0.77
UNIV PKT	0.39	**0.58**
BSM	**0.87**	0.85
FIT-UTK	**0.87**	0.85
UNM	0.88	0.88

the performance of *Hybrid*? Results are depicted in Fig. 8. The X-axis, denoted as $\Delta Coverage$, represents the difference in coverage of *Weighting* and *Replacement*:

$$\Delta Coverage = Coverage_{weighting} - Coverage_{replacement} \tag{21}$$

The Y-axis (ΔAUC) represents the difference in accuracy for the two techniques:

$$\Delta AUC = AUC_{weighting} - AUC_{replacement} \tag{22}$$

The correlation between accuracy and coverage is positive if increasing (decreasing) coverage increases (decreases) accuracy. The correlation is negative when increased (decreased) coverage leads to decreased (increased) accuracy. In Figs. 8a-b, positive correlations are represented by data points in quadrants

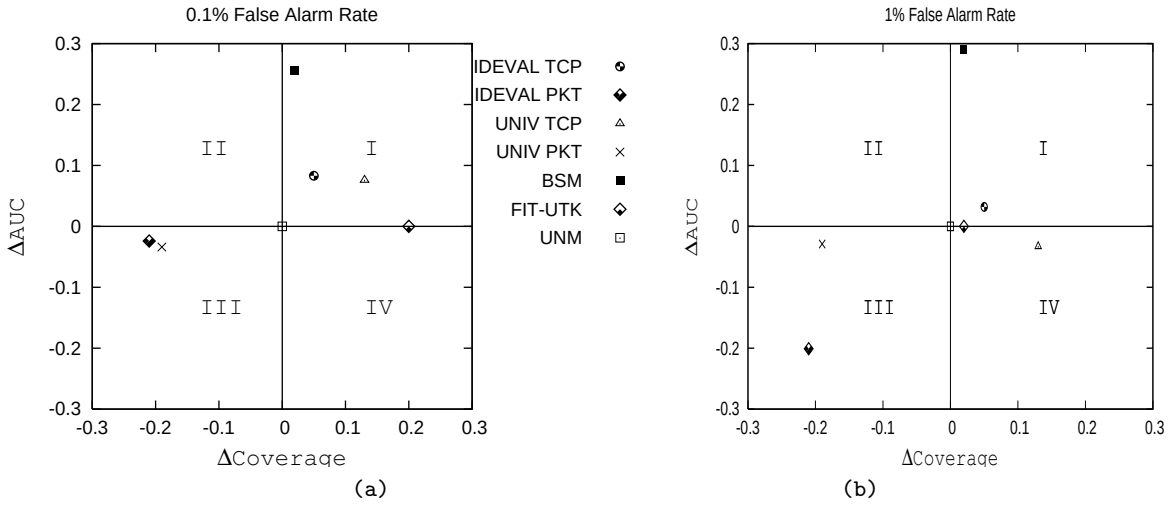

Fig. 8. Accuracy vs. Coverage at (a) 0.1% and (b) 1% false alarm rates. X-axis represents difference in coverage and Y-axis is the difference in AUC for *Weighting* and *Replacement*.

I and III, whereas negative correlations are denoted by points on quadrants II and IV. Both techniques were identical for the UNM data set, hence marked as origin in Figs. 8a-b. IDEVAL TCP and BSM data sets are represented by data points in Quadrant I, where *Weighting* had higher coverage and accuracy than *Replacement*. *Replacement* did better (in terms of accuracy) than *Weighting* with increased coverage for IDEVAL PKT and UNIV PKT, and are represented by data points in Quadrant III. The only data set showing a negative correlation was UNIV TCP at 1% false alarm rate Fig. 8b. This is due to the fact that rule weighting had higher coverage than *Replacement* (0.90 vs. 0.77 - Table 5), but lower accuracy. Table 4 depicts accuracy of 15.9% for *Weighting* and 8.3% for *replacement* at 0.1% false alarm rate, and 65.3% and 68.6% respectively at 1% false alarm rate. *Hybrid* picks the less accurate model for UNIV TCP at 1% false alarm rate. For the data set, *Weighting* has higher coverage over *Replacement*, but *Hybrid* has AUC of *Replacement* and not *Weighting*, as seen in Tables 4 and 5. But for all other cases, the higher coverage selection by *Hybrid* yields higher accuracy. In our experiments, there is only one instance of negative correlation between coverage and accuracy, indicating that increased coverage generally increases accuracy. This supports our motivation for *Hybrid*, that for a given data set, algorithm with higher coverage yields higher accuracy.

But how does coverage affect accuracy across data sets, for the same algorithm? From Tables 4 - 5 and Figs 8a-b, note that for *Hybrid*, the two TCP data sets have the highest coverage ($\geq$ 90%), but they have the least accuracy. This can be explained by the fundamental aspects that affect accuracy: (i) data representation, i.e. the features used; (ii) knowledge/model representation; and (iii) the learning algorithm, which finds the best model to fit the data. Since the knowledge/model representation (LERAD rules) and the learning algorithm (LERAD) are the same, the data representation attributes to lower AUC for TCP. Results indicate that TCP header doesn't model network data as well as PKT for intrusion detection. Since the exploits are not detectable at the TCP header level, any increase in coverage does not affect the accuracy. Thus, data sets with higher coverage might not have higher accuracy for a given algorithm. Our claim for *Hybrid* yielding higher accuracy with higher coverage is only applicable across techniques on the same data set.

7.6 Additional Attacks Detected by Weighting and Replacement beyond Pruning

Both *Weighting* and *Replacement* inherently differ on how the rule set coverage is increased over the training data. We conjecture that increased coverage can result in higher attack detections. In this section, we analyze additional attack detections and study if the additional rules are main contributors to new attack detections.

Table 6. 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 rule *belief*	IDEVAL TCP: yaga, sechole
Violated rule(s) with reduced rule *belief*	IDEVAL TCP: arppoison, syslogd, perl, crashiis, secret UNIV TCP: codered UNIV PKT: bindver BSM: fdformat, ffbconfig,guest, syslogd, httptunnel, secret, portsweep, eject, selfping

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 belief for conformed rules, and/or (b) scores from rules discarded by *Pruning* but retained (with reduced belief) 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 6. Most of the new attacks detected are due to rules that were eliminated by *Pruning*, and support our claim for retaining the rules but reducing their belief in *Weighting*. 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 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*.

Replacement substitutes violated rules with low predictiveness rules. Additional rules are also generated from attribute values previously not covered, as shown in Steps 6-8 of Fig. 3. The increase in attack detections for *Replacement* is thus attributed to (a) candidate rules replacing pruned rules, and (b) new candidate rules learned from attribute values that were not covered. We analyzed the attacks detected by the replacement scheme that were missed by *Pruning* at 1% false alarm rate. Table 7 lists all the new attacks detected

Table 7. New attacks detected by *Replacement* at 1% false alarm rate.

Factor contributing to attack detection	**Data set: attack(s) detected**
Existing candidate rule(s) from rule pool	IDEVAL TCP: ppmacro, xsnoop, fdformat, xterm IDEVAL PKT: netcat_breakin, warez, sshtrojan, warezclient, portsweep, phf, tcpreset, sendmail, ipsweep, eject, processtable, perl, crashiis, apache2, guest, anypw, xterm, guest, snmpget, back, yaga, sqlattack, syslogd, guesspop UNIV TCP: codered UNIV PKT: bindver BSM: fdformat
New rule(s) from attribute values not covered earlier	IDEVAL TCP: selfping IDEVAL PKT: mailbomb, casesen, insidesniffer, dosnuke, xterm, perl, ncftp, selfping, loadmodule, ppmacro

in each of the two categories. As seen from the table, existing candidate rules that replace pruned rules are able to detect most new attacks.

7.7 Computational and Storage Overhead

Besides using different weight updation formulae, another distinction between the three weighting schemes discussed in Section 4.2 is the number of rules retained. *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. *Replacement* reintroduces candidate rules and generates new ones from data not covered by initial candidate rules. This may result in larger rule sets and increased execution times for *Weighting*, *Replacement* and *Hybrid* techniques. To check the viability of the techniques for online usage, we studied the overhead involved with all the above techniques, 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 8. Among the weighting methods, 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*. Compared to *Pruning* and *Weighting*, *Replacement* has significantly higher times, due to the high number of iterations for the algorithm. Additional rules are also generated from data without rule set coverage. *Hybrid* is even worse, since it needs to learn both weighting and replacement models before selecting one based on higher coverage. In the worst case, the time taken by *Weighting* was approx. 45 μsec/instance for *Weight of Evidence* vs. 14 μsec/instance for *Pruning* in the case of IDEVAL PKT; *Replacement* took 1773 μsec/instance for the same data set. Since training can be performed offline, higher training times are acceptable. The time requirements

Table 8. Computational overhead: training phase.

Data set	**Data set size (number of instances)**	**Total training time (seconds)**					
		Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	35452	2.06	2.52	2.18	2.56	15.44	18.19
IDEVAL PKT	280281	3.98	6.27	7.86	12.60	496.89	514.92
UNIV TCP	141162	8.69	9.57	9.16	10.56	53.98	62.39
UNIV PKT	1305873	18.15	23.48	23.33	45.25	203.79	241.19
BSM	1261252	90.55	107.15	107.81	101.27	475.89	562.89
FIT-UTK	94759	0.91	0.95	0.98	1.27	1.08	1.45
UNM	3128	0.05	0.04	0.04	0.06	0.06	0.10

Table 9. Storage requirements: size of rule set.

Data set	**Number of rules**					
	Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	52	71	71	71	160	71
IDEVAL PKT	100	108	108	106	554	554
UNIV TCP	45	88	88	88	94	88
UNIV PKT	48	80	80	75	293	293
BSM	155	176	176	176	359	176
FIT-UTK	11	12	12	12	12	12
UNM	36	36	36	36	36	36

for training can be reduced further by terminating the loop (Step 5 in Fig. 3) early, when there is no increase in coverage even though there might be rule violations in validation phase.

Storage of the model is determined by the number of rules in the rule set. Table 9 lists the number of rules generated for the various data sets. For all data sets except FIT-UTK and UNM, the number of rules is based on per week of data. Amongst the network data sets, the least overhead was obtained for IDEVAL PKT where the increase was roughly 6-8% for various weighting strategies. UNIV TCP presented the maximum overhead, where the number of rules almost doubled for all weighted schemes, and six times for *Replacement*. *Hybrid* selects *Weighting* or *Replacement* based on higher coverage on training data. Thus, the rule set would be same as *Weighting* or *Replacement*, depending upon the technique selected. *Weighting* is selected for all data sets except IDEVAL PKT and UNIV PKT, resulting in a smaller rule set for *Hybrid* than *Replacement*. Considering the large amount of data used during training (1-9 weeks) and the number of attributes involved, the size of the weighted and replacement rule sets formed is fairly reasonable. For *Weighting*, we could additionally limit the rule set size by eliminating a rule which has been violated a certain number of times or with weight below a threshold.

Table 10. Computational overhead: testing phase.

Data set	**Data set size (number of instances)**	**Total testing time (seconds)**					
		Pruning	*Winnow Specialist*	*Equal Reward Apportioning*	*Weight of Evidence*	*Replacement*	*Hybrid*
IDEVAL TCP	178099	7.72	8.76	8.26	8.65	9.56	8.74
IDEVAL PKT	534763	3.02	3.90	3.68	3.20	13.35	13.22
UNIV TCP	143403	7.64	8.50	8.21	8.41	8.97	8.65
UNIV PKT	1310493	7.70	8.64	8.21	8.18	16.61	16.37
BSM	1889680	113.93	121.34	121.63	120.97	187.31	120.66
FIT-UTK	13745	0.10	0.10	0.10	0.09	0.10	0.10
UNM	7283	0.06	0.06	0.06	0.06	0.06	0.06

For *Replacement*, rule set size can be reduced by terminating iterations earlier and/or ignoring new rules below a certain *predictiveness* threshold. Host data sets displayed lower storage overhead. For the BSM data, the weighted rule set was over 13% larger than *Pruning*. *Replacement* produced 359 rules compared to 155 for pruning. Since 11 different applications were modeled in BSM data, this corresponds to an average of 16 rules/application for weighting and 33 rules/application for *Replacement*, which is small for one week of training data. Number of rules were same for UNM data, whereas the weighted and replacement 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 10. Due to larger rule sets, *Weighting*, *Replacement* and *Hybrid* schemes have longer execution times, making them computationally more expensive than *Pruning*. The maximum overhead for *Weighting* was 5.99 μsec/instance on UNIV TCP data set, and 19.32 μsec/instance for *Replacement* on IDEVAL PKT. Thus, the overhead is only a fraction of a millisecond per instance, reasonable for an online system.

8 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 three techniques to increase rule coverage: *Weighting*, *Replacement* and *Hybrid*.

Weighting retains violated rules in the rule set and associates a belief value with each rule. Weights are representative of rule *belief* 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. Three weighting schemes are presented - *Winnow-specialist-based weighting*, *Equal Reward Apportioning* and *Weight of Evidence*. *Replacement* collects rules ignored in coverage test in a rule pool. These rules are reevaluated to replace pruned rules and increase coverage. The steps of validation, pruning, and replacement are repeated until certain exit criterion is met. Furthermore, new rules are learned from remaining attribute values that were not covered. We also present *Hybrid* technique that selects between *Weighting* and *Replacement* based on higher coverage on training data.

We evaluated *Pruning*, *Weighting*, *Replacement* and *Hybrid* LERAD variants on various network and host data sets. Empirical results show that weighted and replacement 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, whereas *Replacement* detected 6 extra attacks than *Pruning*. For *Weighting*, the most significant improvement was in the case of BSM data, where 12 new attacks (60% more than *Pruning*) were detected. *Replacement* performed best on IDEVAL PKT data set, where it detected 32% more attacks than pruning. At 0.1% false alarm rate, *Equal Reward Apportioning* outperformed *Pruning* in 5 data sets and generally performed the best. *Replacement* had the best accuracy on our data sets at 1% false alarm rate, where it did better than *Pruning* on seven data sets. Generally, all proposed techniques were better than *Pruning* in terms of AUC as well as number of attack detections at 1% false alarm rate.

We studied the effect of coverage on accuracy. Results indicate that increased coverage generally resulted in better accuracy. That is the reason why *Hybrid* did better than *Weighting* and *Replacement*, as shown in Fig. 8. We also analyzed the new attack detected by *Weighting* and *Replacement* based LERAD variants. For *Weighting*, these were attributed to high anomaly scores resulting from (a) violations of rules discarded by *Pruning* but retained by weighted variants with reduced belief; and (b) increased belief for existing rules due to the weight update functions. The former factor contributed to most new attack anomalies. For *Replacement*, detections are attributed to (a) candidate rules replacing pruned rules, and (b) new candidate rules learned from attribute values that are not covered. Our analysis shows that most of the new attack anomalies are detected by the first factor.

We also computed overheads incurred due to *Weighting* and *Replacement*. Training times are generally higher for *Replacement* as it involves multiple iterations, the worst in our experiments being 1773 μsec/instance. For *Weighting*, it was 30 μsec/instance. But training can be performed offline. Since pre-

viously discarded rules are retained (for *Weighting*) and additional rules are added (for *Replacement*), rule sets tend to be larger. This has direct effect on the test phase - the larger the rule set, the higher is the testing overhead. But this overhead is minimal, 6 μsec/instance for *Weighting* and 19 μsec/instance for *Replacement*, 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.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD*, pages 207–216, 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *International Conference on Very Large Data Bases*, 1994.
3. D. Anderson, T.F. 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.
4. 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.
5. S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *IEEE Symposium on Security and Privacy*, 2006.
6. A. Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(5):5–23, 1997.
7. P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–285, 1989.
8. W. Cohen. Fast effective rule induction. In *International Conference on Machine Learning*, pages 115–123, 1995.
9. K. Das and J. Schneider. Detecting anomalous records in categorical datasets. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2007.
10. H. Feng, O. Kolesnikov, P. Fogla, W. Lee, and W. Gong. Anomaly detection using call stack information. In *IEEE Symposium on Security and Privacy*, 2003.
11. P.A. Flach. The many faces of roc analysis in machine learning. In *International Conference on Machine Learning Tutorial*, 2004.
12. S. Forrest, S. Hofmeyr, A.Somayaji, and T. Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, 1996.

13. Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.
14. J. Furnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27:139–171, 1997.
15. D. Gao, M. Reiter, and D. Song. On gray-box program tracking for anomaly detection. In *USENIX Security Symposium*, 2004.
16. A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *USENIX Security Symposium*, 1999.
17. K. Kendell. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
18. C. Krugel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *ACM Symposium on Applied Computing*, 2002.
19. R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 2000.
20. N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.
21. N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
22. M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *IEEE International Conference on Data Mining*, 2003.
23. R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The aq15 inductive learning system: An overview and experiments. Technical Report UIUCDCS-R-86-1260, University of Illinois, Urbana, 1986.
24. D. Mutz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous system call detection. *ACM Transactions on Information and System Security*, 2006.
25. V. Paxson. Bro: A system for detecting network intruders in real time. In *USENIX Security Symposium*, 1998.
26. V. Paxson and S. Floyd. The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.
27. J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, San Mateo, CA, 1993.
28. W. Robertson, G. Vigna, C. Kruegel, and R.A. Kemmerer. Using generalization and characterization techniques in the anomaly-based detection of web attacks. In *Network and Distributed System Security Conference*, 2006.
29. M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA*, 1999.
30. R. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–226, 1990.
31. R. Sekar, M. Bendre, D. Dhurjati, and P. Bollineni. A fast automaton-based method for detecting anomalous program behaviors. In *IEEE Symposium on Security and Privacy*, 2001.
32. P. Smyth and R. Goodman. Rule induction using information theory. *Knowledge discovery in databases*, pages 159–176, 1991.
33. P. Smyth and R. Goodman. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering*, 4(4):301–316, 1992.
34. S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10:105–136, 2002.

35. G. Tandon and P.K. Chan. On the learning of system call attributes for host-based anomaly detection. *International Journal on Artificial Intelligence Tools*, 15(6):875–892, 2006.
36. G. Tandon and P.K. Chan. Weighting vs. pruning in rule validation for detecting network and host anomalies. In *ACM International Conference on Knowledge Discovery and Data Mining*, pages 697–706, 2007.
37. Y. Wang and A.K.C. Wong. From association to classification: inference using weight of evidence. *IEEE Transactions on Knowledge and Data Engineering*, 15(3), 2003.
38. C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, 1999.
39. A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *Recent Advances in Intrusion Detection*, 2000.
40. I. Witten and T. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 1991.