

Detecting Anomalies by Learning Weighted Rules

Gaurav Tandon and Philip K. Chan

Department of Computer Sciences, Florida Institute of Technology, Melbourne, FL 32901
{gtandon, pkc}@cs.fit.edu

Abstract

Anomaly detection focuses on modeling the normal behavior and identifying significant deviations, which could be novel attacks. The previously proposed LERAD algorithm can efficiently learn a succinct set of comprehensible rules for detecting anomalies. We conjecture that LERAD eliminates rules with possibly high coverage, which can lead to missed detections. This study proposes weights that approximate rule confidence and are learned incrementally. We evaluate our algorithm on various network and host datasets. Compared to LERAD, our technique detects more attacks at low false alarm rates with minimal computational overhead.

1 Introduction

Intrusion detection has two general approaches *signature detection* (also known as *misuse detection*), where we look for patterns signaling well-known attacks, and *anomaly detection*, where we look for deviations from normal behavior. Signature detection works reliably on known attacks, but has the disadvantage of not detecting 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.

The previously proposed LERAD (LEarning Rules for Anomaly Detection) algorithm [19] can efficiently learn a succinct set of comprehensible rules and detect attacks unknown to the algorithm. To reduce false alarms, LERAD, during the validation phase, eliminates rules that cause false alarms. However, these rules were selected initially to cover a relatively large number of training examples and their elimination could lead to missed detections. Instead of eliminating these rules, we propose learning weights that approximate rule confidence (Sec. 3). We present an empirical evaluation of our algorithm and compare it with LERAD on various network and host datasets (Sec. 4). Results show that our technique detects more attacks than LERAD at low false alarm rates. The improvement in accuracy is obtained at the cost of small computational overhead and reasonable space requirements.

2 Related Work

2.1 Anomaly Detection

Network anomaly detection systems can warn of attacks launched from the outside at an earlier stage, before the attacks actually reach the host. Intrusion detection systems (IDSs) such as NIDES [1], ADAM [2], and SPADE model only features of the network and transport layer, such as port numbers, IP addresses, and TCP flags. Models built with these features could detect probes (such as port scans) and some denial of service (DOS) attacks on the TCP/IP stack, but would not detect

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 algorithm of LERAD

attacks where the exploit code is transmitted to a server in the application payload. SNORT [25] and BRO [22] use rules for detecting network attacks. Application payload has been used in [32] whereas multiple network attributes are utilized in [27]. Web based attacks are detected by monitoring web request parameters in [24]. Some anomaly detection algorithms are for specific attacks (e.g., portscans [28]) or services (e.g., DNS [14]).

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. [8, 11] create n-gram models (sequences of $n = 3$ to 6 calls) using a sliding window across system call sequences. Sequence time-delay embedding (stide) [33] memorizes all contiguous sequences of predetermined, fixed lengths. Extensions to n-gram method is provided in [34], where variable length patterns are generated by using a bioinformatics pattern-discovery algorithm. Other models of normal system call sequences have been used, such as finite state automata [26] and neural networks [10]. Instance-based methods have been proposed [15] for detecting anomalous user commands. Limitation of system call sequence based techniques have been presented in [31, 29] where mimicry attacks are shown to evade such IDSs. Other features that have been used include system call arguments [21, 30, 3] and call stack information [6, 9].

2.2 LERAD (LEarning Rules for Anomaly Detection)

LERAD [19] is an efficient randomized algorithm that forms conditional rules of the form:

$$SrcIp = 128.1.2.3, DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\} \quad [p] \quad (1)$$

In the above sample rule, given source IP address ($SrcIp$) = 128.1.2.3 and destination IP address ($DestIp$) = 128.4.5.6, destination port ($DestPort$) is either 21 (FTP), 25 (SMTP), or 80 (HTTP) in the normal training data. p is the probability of the rule being violated and is explained below.

The LERAD algorithm is based on sampling and randomization. Let D be the entire data set, and D_T be the training set with normal behavior and D_E be the evaluation (test) 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 set) and D_v (validation set) respectively such that $D_t \cup D_v = D_T$, $D_t \cap D_v = \emptyset$, and $|D_t| > |D_v|$. Also, let D_s be a random sample of D_t such that $D_s \subset D_t$ and $|D_s| \ll |D_t|$.

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 sample D_s of the data, which allows efficient training. Step 2 selects a small set of predictive rules that sufficiently describe the small sample 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, which involves eliminating rules that are not satisfied, since any alarm on validation set is a false alarm.

LERAD adopts a probabilistic framework and estimates $P(C|A)$, where A is the antecedent and C is the consequent of the rule $A \Rightarrow C$. The anomaly score depends on $P(\neg C|A)$, where C , though expected, is not observed when A is observed. During training, a set of rules R that “minimally” describes the training data are generated and their $p = P(\neg C|A)$ is estimated. During detection, given an instance x , an anomaly score is generated if x violates any of the rules ($A \Rightarrow C$). Let $R' \subset R$ be the set of rules that x violates. The anomaly score is calculated as:

$$AnomalyScore(x) = \sum_{r_k \in R'} \frac{1}{p_k}, \quad (2)$$

where r_k is a rule in R' and p_k is the p value of rule r_k . The reciprocal of p_k reflects a surprise factor that is large when anomaly has a low likelihood (small p_k).

Witten and Bell [35] proposed a few estimates for novel events in the context of data compression; LERAD uses the following one:

$$P(NovelEvent) = \frac{r}{n}. \quad (3)$$

where n is the total number of observed events and r is the number of unique observed events.

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 . Hence, LERAD incorporates a non-stationary model and the anomaly score is calculated as:

$$AnomalyScore(x) = \sum_{r_k \in R'} \frac{t_k}{p_k}. \quad (4)$$

Also, the t_k factor accommodates the “bursty” nature of network traffic [23], so that a burst of anomalies will only generate a single high scoring alarm.

3 Rule Weighting for Anomaly Detection

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 . LERAD eliminates rules that are violated by instances in the validation data set D_v (Step 4 in Fig. 1), since the data is normal and any anomaly would correspond to a false alarm. 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 confidence on the rule instead of eliminating it. For large amounts of data, the latter option is more efficient.

In this section, we propose LERAD-w (LERAD with weighted rules), which associates a weight to each rule in the rule set, where the weights symbolize confidence in the respective rules. We present a new validation phase for the LERAD algorithm, as depicted in Fig. 2. Instead of making a binary decision of retaining or eliminating a rule, we always keep a rule but update its consequent and the associated weight and probability values.

A sample rule using our method is of the form:

$$SrcIp = 128.1.2.3, DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\} \quad [p, w] \quad (5)$$

The semantics of this rule is similar to the rule in Eq. 1, but a new w value is introduced for the rule weight to represent confidence in the rule. p and w are distinct and independent entities. p is the

- | |
|---|
| <p>4. for each instance in D_v, check if the instance is consistent with the rules in R</p> <ul style="list-style-type: none"> (a) update the rule's weight w (b) update the rule's consequent (c) update the rule's probability p |
|---|

Figure 2: New validation phase. First three steps are the same as Fig. 1.

probability of not seeing a value in the consequent when the conditions in the antecedent hold true. That is, it is probability of the rule being violated. Weight w , on the other hand, approximates the confidence of the entire rule.

Given a rule and an instance in the validation phase of Fig. 2, one of three cases apply:

1. The rule is satisfied when all conditions in the antecedent as well as the consequent is satisfied by the instance. In this case, the rule is not updated but the associated p and w values are modified. For example, instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 80] complies with the rule in Eq. 5.
2. The rule may be violated if the antecedent holds good but the consequent does not. This results in updating the rule and its respective probability and weight. For example, the rule in Eq. 5 is violated by the instance [SrcIp = 128.1.2.3, DestIp = 128.4.5.6, DestPort = 23].
3. The rule may not be applicable for the instance if any condition in the antecedent is not satisfied, in which case neither the rule nor its p and w values is updated. For example, the rule in Eq. 5 does not apply to the instance [SrcIp = 128.1.5.7, DestIp = 128.4.5.6, DestPort = 21].

Rule update and modification of p and w values is discussed next.

Step 4(a) in Fig. 2 updates the rule's weight. Initially all rule weights are assigned a value 1, signifying equality of rule confidence across the rule set. Rules are penalized upon violation and rewarded upon conformance. When a rule is violated, it reduces our confidence in it. If a rule formed during training is not useful, it is likely to be violated many times. To minimize its effect, the penalty involves multiplicative decay of the weight. On the other hand, if a rule is complied to, it stresses upon its validity and increases our confidence in it. 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 conservatively by β when the rule is complied. By decreasing the weight of a rule upon violation and not eliminating them, we hope to increase the number of detections. The strategy to update weights is formally defined by the following weight update function:

$$w_k = \begin{cases} w_k \times \alpha, & \text{if } r_k \in R \text{ is violated} \\ w_k(1 + \beta), & \text{if } r_k \in R \text{ is complied,} \end{cases} \quad (6)$$

where $\alpha, \beta \in \mathbb{R}$, $0 \leq \alpha < 1$ and $0 \leq \beta \leq 1$. Assuming $\alpha = 0.5$ and initial 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 1.1, 1.21, 1.331 respectively (with $\beta = 10\%$) for the first three conformances. It can be noted that LERAD is a special case of our rule weighting strategy, with $\alpha = \beta = 0$.

The strategy of associating and decrementing weights is similar to Weighted Majority [18] and Dynamic Weighted Majority [13], but these techniques do not increment weights upon conformance,

which is intuitive in our case as it reflects the confidence in a rule. A calendar scheduling technique [4] proposed a variant of Winnow [17] to increase the weights, but their technique can be visualized as fixed sized rules whereas we support rules of variable length. Their rule set is exponentially large in the number of features used whereas our technique derives only a small rule set.

In addition to updating the weights as above, we need to update the rule consequent and the associated probability p , as mentioned in Steps 4(b) and 4(c) of Fig. 2. Rule update is required only in the event of the rule being violated. The new attribute value is added to the rule consequent in this case. The probability p of the rule being violated is computed using Eq. 3. Both n and r values are incremented to compute p when a rule is violated, whereas only the n value is updated when the rule is satisfied. Consider the following synthetic rule:

$$SrcIp = 128.1.2.3, DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\} \quad [p = 3/100, w = 1.0] \quad (7)$$

100 instances satisfy the above rule which claims 3 destination ports (21-FTP, 25-SMTP, 80-HTTP) for the given source and destination IP addresses. This rule is initially given a weight of 1. If this rule is violated by an instance (e.g. port 23-TELNET for the given source and destination IP address pair), our confidence in the rule is reduced. Hence the weight of the rule is reduced by α . Given $\alpha=0.5$, the weight becomes 0.5. The rule consequent is updated by adding the new $DestPort$ value; n and r (hence p) values are modified, resulting in the following rule:

$$SrcIp = 128.1.2.3, DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 23, 25, 80\} \quad [p = 4/101, w = 0.5] \quad (8)$$

On the other hand, the weight is incremented by a factor β of the original weight when the rule is satisfied. If $\beta=0.1$, the weight is updated to 1.1. The rule remains the same with only the n value being incremented by 1. The rule in Eq. 7 would now be represented as:

$$SrcIp = 128.1.2.3, DestIp = 128.4.5.6 \Rightarrow DestPort \in \{21, 25, 80\} \quad [p = 3/101, w = 1.1] \quad (9)$$

For a test instance, an anomaly score is computed by LERAD as per Eq. 4. Confidence in a rule entails confidence in its anomaly score. Hence each rule assigns a score proportional to its weight. Thus, the anomaly score is modified as:

$$AnomalyScore(x) = \sum_{r_k \in R'} \left(\frac{w_k t_k}{p_k} \right) \quad (10)$$

where $R' \subset R$ is the set of rules that instance x violates.

4 Empirical Evaluation

4.1 Experimental Data

We evaluated LERAD and LERAD-w on five different data sets:

(a) The DARPA/Lincoln Laboratory intrusion detection evaluation network data set (IDEVAL) [16] 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. For attack taxonomy, see [12].

(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 [20], and the Scalper worm [5]. 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.

4.2 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, 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 [33].

4.3 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 [7]. The area under the curve is normalized for the false alarm rate. The drawback of anomaly detection is the generation of false alarms, so we focus only on small false alarm rates (up to 1%).

Table 1: Area under ROC curve at 0.1% and 1% false alarm rates.

<i>Data</i>	<i>0.1% FA</i>		<i>1% FA</i>	
	<i>LERAD</i>	<i>LERAD-w</i>	<i>LERAD</i>	<i>LERAD-w</i>
IDEVAL TCP	21.80%	23.45%	54.71%	56.35%
IDEVAL PKT	38.60%	41.55%	61.14%	62.46%
IDEVAL COMBINED	54.85%	61.60%	80.96%	83.06%
UNIV TCP	8.50%	4.25%	58.68%	62.98%
UNIV PKT	23.10%	28.20%	60.06%	73.77%
UNIV COMBINED	16.60%	58.10%	80.51%	92.41%
IDEVAL BSM	54.40%	78.85%	60.34%	93.99%
UNM	100.00%	100.00%	100.00%	100.00%
FIT-UTK	50.00%	62.50%	95.00%	96.25%

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.

4.4 Accuracy of rule weighting

LERAD removes a rule that causes false alarms during validation phase, removing coverage on a relative large number of training examples, which can lead to missed detections. In Section 3, we proposed a strategy to associate weights with rules to signify rule confidence. The weight is increased when the rules is satisfied by an instance, and the weight is decreased upon rule violation. For our experiments in this section, we present results for weight update strategy of Eq. 6, with $\alpha = 50\%$ and $\beta = 5\%$. Parameter study is discussed in Section 4.4.3.

4.4.1 Number of attack detections

The ROC curves for LERAD and LERAD-w on all the data sets are depicted in Fig. 3 and the area under their ROC curves are listed in Table 1. The results for the IDEVAL network data (Fig. 3(a)-(c)) show that LERAD-w generally detects more attacks than LERAD at various false alarm rates. It is also seen that the COMBINED attribute set detects the maximum attacks suggesting that TCP and PKT data detect attacks which are not detected by the other. Looking at 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) enables the system to detect attacks observable in packet data as well as tcp streams. As depicted in Table 1, the area under the ROC curve for LERAD-w is greater than LERAD at 0.1% and 1% false alarm rates for all the three IDEVAL network data.

Results also show that the performance of LERAD-w is better than LERAD on UNIV data (Fig. 3(d)-(f)). The highest detections are for the UNIV COMBINED attribute set due to reasons mentioned above. LERAD-w detected one more attack than LERAD for both TCP and PKT at 1% false alarm rate. The HTTP proxyscan was detected using tcp streams whereas the packet data detected the port/security scan, and both were detected when the attribute sets were combined. Hence 100% detection for LERAD-w in Fig. 3(f).

The ROC curves for the host datasets are presented in Fig. 3(g)-(i). For the IDEVAL BSM data, LERAD-w detected 49% and 52% more attacks than LERAD at 0.1% and 1% false alarm

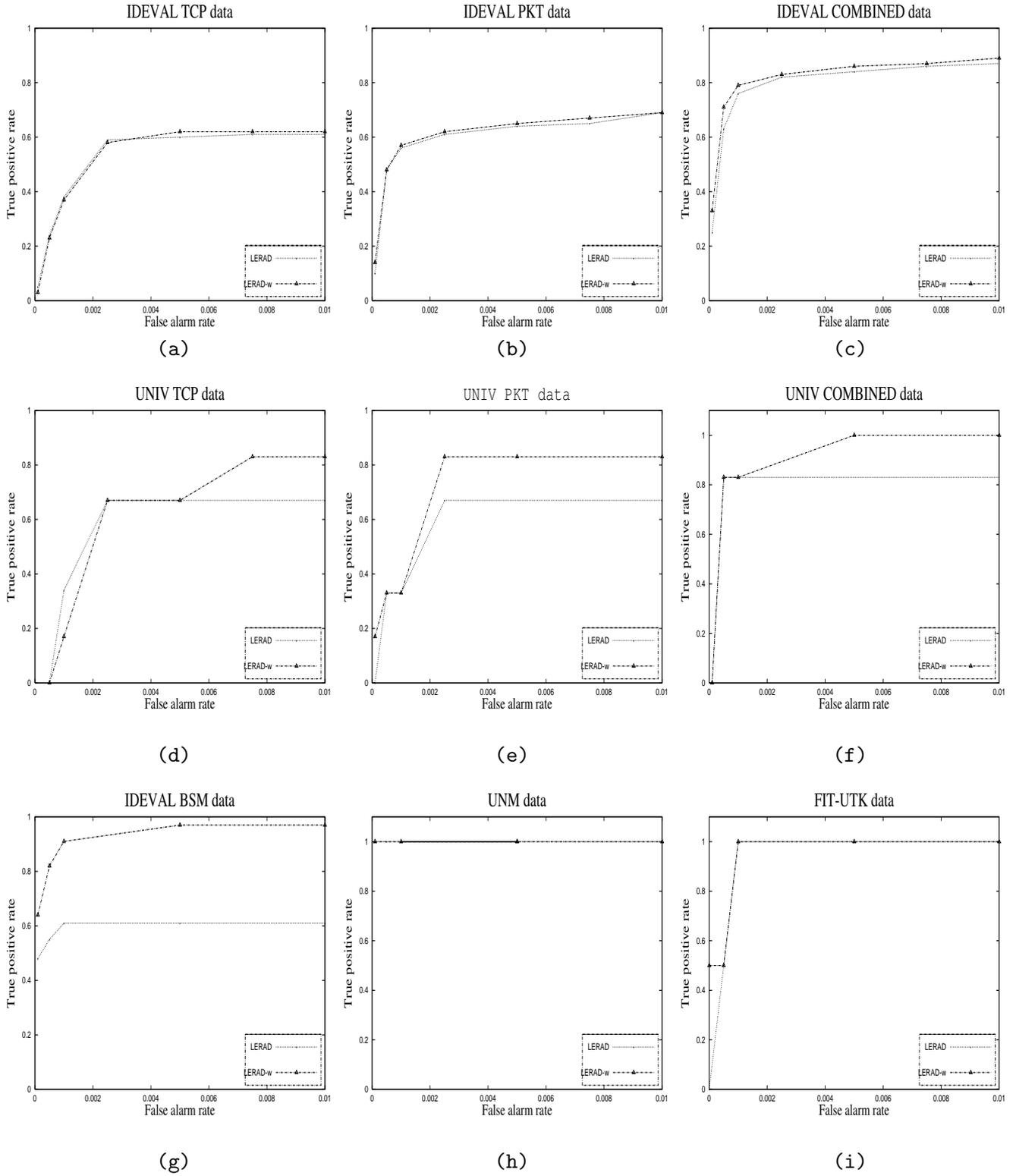


Figure 3: Accuracy comparison of LERAD and LERAD-w on (a) IDEVAL TCP, (b) IDEVAL PKT, (c) IDEVAL COMBINED, (d) UNIV TCP, (e) UNIV PKT, (f) UNIV COMBINED, (g) IDEVAL BSM, (h) UNM, and (i) FIT-UTK data sets.

Table 2: New attacks detected by LERAD-w at 1% FA.

<i>Attack</i>	<i>Data set</i>	<i>Factor contributing to attack detection</i>	
		<i>Existing rule(s) with increased confidence</i>	<i>Re-introduced rule(s) with reduced confidence</i>
codered	UNIV TCP		✓
bindver	UNIV PKT		✓
arpoison	IDEVAL TCP		✓
teardrop	IDEVAL TCP		✓
yaga	IDEVAL TCP	✓	
sechole	IDEVAL TCP	✓	
secret	IDEVAL TCP		✓
fdformat	IDEVAL BSM		✓
ffbconfig	IDEVAL BSM		✓
guest	IDEVAL BSM		✓
syslogd	IDEVAL BSM		✓
httptunnel	IDEVAL BSM		✓
secret	IDEVAL BSM		✓
portsweep	IDEVAL BSM		✓
selfping	IDEVAL BSM		✓

rates respectively. Accuracy was the same for the UNM data set, where both LERAD and LERAD-w detected all 3 attacks. On the FIT-UTK data, LERAD-w had greater area under ROC curve than LERAD at 0.1% and 1% false alarm rates. Results indicate that LERAD-w generally has greater accuracy than LERAD 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.

4.4.2 Attacks detected by rule weighting

The increase in detections for LERAD-w is caused by an increase in the anomaly score, which could result from: (a) increase in weights of rules kept by LERAD, and/or (b) scores from rules eliminated by LERAD, that are re-introduced in LERAD-w. We analysed the attacks detected by LERAD-w that were missed by LERAD at 1% false alarm rate. The results are listed in Table 2.

As observed from the table, most of the new attacks detected by LERAD-w are due to rules that were eliminated by LERAD, supporting our claim for retaining the rules but reducing their confidence. The Code Red II HTTP requests for /default.ida (GET /default.ida?NNNN...) are captured by anomaly in the application payload. The *secret* attack is detected due to a secret file being mailed to unauthorized person, though our rules do not have any knowledge of any security policy. *fdformat* and *ffbconfig* vulnerabilities are buffer overflow attacks that are detected by encountering unusual arguments. The *syslogd* exploit is a Denial of Service attack that violates 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 LERAD. This was attributed to large rule weights for some rules in LERAD-w, resulting in further increase of the anomaly score. Also, there were multiple alarms for the same attack due to violation of rules

Table 3: Area under ROC curve for weight decrease parameter α (with $\beta = 0$).

<i>Data</i>	<i>0.1% FA</i>		<i>1% FA</i>	
	<i>Auto α</i>	$\alpha = 0.5$	<i>Auto α</i>	$\alpha = 0.5$
IDEVAL TCP	21.20%	21.20%	54.90%	54.90%
IDEVAL PKT	38.60%	38.60%	61.01%	61.14%
UNIV TCP	4.25%	4.25%	62.98%	62.98%
UNIV PKT	23.10%	23.10%	66.06%	70.06%
IDEVAL BSM	54.40%	53.75%	60.34%	68.08%

Table 4: Area under ROC curve for weight increase parameter β (with $\alpha = 0.5$).

<i>Data</i>	<i>0.1% FA</i>			<i>1% FA</i>		
	$\beta = 0$	$\beta = 0.05$	$\beta = 0.25$	$\beta = 0$	$\beta = 0.05$	$\beta = 0.25$
IDEVAL TCP	21.20%	23.45%	1.45%	54.90%	56.35%	13.42%
IDEVAL PKT	38.60%	41.55%	40.05%	61.14%	62.46%	62.31%
UNIV TCP	4.25%	4.25%	4.25%	62.98%	62.98%	62.98%
UNIV PKT	23.10%	28.20%	32.45%	70.06%	73.77%	75.47%
IDEVAL BSM	53.75%	78.85%	73.90%	68.08%	93.99%	91.69%

introduced by LERAD-w but absent in LERAD.

4.4.3 Sensitivity of Parameters

LERAD-w penalizes a rule violation during validation phase by reducing the associated weight. A straightforward policy is to reduce the weight by a constant factor. Decreasing the weight by half seems intuitive and reasonable. Thus the initial weight of 1 would decay exponentially to 0.5, 0.25, 0.125 upon the first 3 rule violations. But if the validation set was used in the training phase itself, rules would only be updated along with the associated probabilities. An alternate way to decrease the weight for rule violation in validation phase would be such that the probability becomes equal to what one would have obtained if the sample was part of the training itself. Thus, weight w_k is updated as:

$$w_k = \left(\frac{n+1}{n}\right)\left(\frac{r}{r+1}\right)w_k \quad (11)$$

where w is the current weight. We call this auto decrement strategy. Since n is generally much larger than r , this is a very conservative updation of weight. We performed experiments to compare the conservative weight reduction strategy of Eq. 11 with the stricter 50% penalty. Results, depicted in Table 3, show that LERAD-w with $\beta = 0.5$ has greater area under ROC curve at 1% false alarm rates. This suggests that the 50% penalty strategy reflects rule confidence more appropriately and generally detects more attacks than its conservative counterpart.

Our weight updation technique also rewards a rule upon conformance by increasing its weight. We propose a weight update function in Eq. 6. We fixed $\alpha = 50\%$ and varied $\beta \in \{0\%, 5\%, 25\%\}$, each set representing three different values for no increment, conservative increment and liberal increment respectively. The results are presented in Table 3. For each data set, there exists a $\beta \neq 0$ with greater area under the curve than no increment strategy, indicating that rule increment is a better strategy. Generally, non zero β values performed better than no increment except IDEVAL TCP, where $\beta = 25\%$ performed the worst. But even in that case $\beta = 5\%$ detected more attacks than $\beta = 0\%$. Experiments with other increment (β) values gave similar results.

Table 5: Computational overhead: training phase.

<i>Data</i>	<i>Data set size (no. of instances)</i>	<i>Total training time (seconds)</i>		<i>Training rate (milliseconds/instance)</i>	
		LERAD	LERAD-w	LERAD	LERAD-w
IDEVAL TCP	35452	7.58	7.72	0.21	0.22
IDEVAL PKT	280281	33.15	33.32	0.12	0.12
UNIV TCP	141162	33.29	33.58	0.23	0.24
UNIV PKT	1305873	149.43	152.65	0.11	0.12
IDEVAL BSM	1261252	88.91	92.34	0.07	0.07

Table 6: Storage requirements: size of rule set.

<i>Data</i>	<i>Number of rules (per week of data)</i>	
	LERAD	LERAD-w
IDEVAL TCP	54	73
IDEVAL PKT	89	99
UNIV TCP	42	80
UNIV PKT	49	86
IDEVAL BSM	156	179

4.5 Computational and Storage Overhead

Since we suggest keeping all the rules that were previously discarded by LERAD, it would result in a larger rule set and increased execution times. To check the viability of LERAD-w for online usage, in this section we study 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 results from the experiments conducted to measure the computational overhead during the training phase is depicted in Table 5. LERAD-w and LERAD form a rule set very efficiently at approximately the same rates per instance. For the UNIV data, the results are consolidated over 10 weeks of data, hence higher total training times. But the rates are similar to IDEVAL TCP and PKT data. For the BSM data the time per instance is much faster, taking only 0.07 milliseconds/instance to train. Network data has more attributes than host data (only system call and arguments), resulting in higher training times.

Storage of the model is determined by the number of rules in the rule set. Table 6 lists the number of rules generated for the various data sets. For the IDEVAL BSM data, the LERAD-w rule set size is approximately 15% more than LERAD. Since 11 different applications were modeled in IDEVAL BSM data, LERAD-w generates an average of 12 rules per application, which is still small for one week of training data. The best results are obtained for IDEVAL PKT where the increase is 11%. In the worst case (UNIV TCP) the number of rules increased by almost 90%. Considering the large amount of data used during training (1-9 weeks) and the number of attributes involved, the size of the rule set formed by LERAD-w 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 many times and its weight falls below a threshold.

The time taken during test phase is also dependent on the rule set size. The more the rules, the higher is the number of checks to be made for each test instance. Typically, the time taken should

Table 7: Computational overhead: testing phase.

<i>Data</i>	<i>Data set size</i> (no. of instances)	<i>Total testing time (seconds)</i>		<i>Testing rate (milliseconds/instance)</i>	
		LERAD	LERAD-w	LERAD	LERAD-w
IDEVAL TCP	178099	29.61	32.53	0.17	0.18
IDEVAL PKT	534763	19.11	22.83	0.04	0.04
UNIV TCP	143403	25.77	29.66	0.18	0.21
UNIV PKT	1310493	32.52	48.37	0.02	0.04
IDEVAL BSM	1889680	114.50	119.90	0.06	0.06

be low for online detection. The results obtained from our experiments are presented in Table 7. It can be observed that the bigger rule sets for LERAD-w result in longer execution times, making LERAD-w computationally more expensive than LERAD. But this overhead is only a fraction of a millisecond per instance, which is still reasonable for an online system.

5 Concluding Remarks

LEarning Rules for Anomaly Detection (LERAD) is a learning 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. In this paper we conjecture that LERAD eliminates rules with possibly high coverage that can lead to missed detections. We propose keeping rules in the rule set and associating a confidence value with each rule. Weights are representative of rule confidence in our strategy (called LERAD-w) and updated incrementally. If a rule is satisfied by an instance, it increases our confidence in the rule and hence the weight is increased. On the other hand, weight is decreased upon rule violation symbolizing decrease in confidence. Our technique adds new values to the consequent of the rule and recomputes the associated probability. 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 evaluated LERAD and LERAD-w on various data sets. Empirical results show that LERAD-w rules detects more attack-based anomalies than LERAD at less than 1% false alarm rates. For the IDEVAL BSM data, LERAD-w detected upto 50% more attacks than LERAD. We also analysed the new attack detections by LERAD-w, which are attributed to high anomaly scores resulting from (a) violations of rules eliminated by LERAD during the validation phase (but retained in LERAD-w); and (b) higher scores for existing rules due to our weight update function of Eq. 6. Since rules eliminated by LERAD are retained in LERAD-w, it forms a larger rule set than LERAD. But the size of the rule set is still fairly small - 179 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 LERAD-w is minimal, taking only a fraction of a millisecond per instance more than LERAD. We also studied the effect of weight update parameters. Results indicate that a imposing severe penalty is better than a conservative one, whereas conservative increase in weight generally detects more attacks.

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 various linear weight update functions other than the one used in Eq. 6.

References

- [1] 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.
- [2] 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.
- [3] S. Bhatkar, A. Chaturvedi, and R. Sekar. Dataflow anomaly detection. In *IEEE Symposium on Security and Privacy*, 2006.
- [4] A. Blum. Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning*, 26(5):5–23, 1997.
- [5] CERT. Ma-044.072002: Apache worm, 2002. <http://www.mycert.org.my/advisory/MA-044.072002.html>.
- [6] 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.
- [7] P.A. Flach. The many faces of roc analysis in machine learning. In *International Conference of Machine Learning Tutorial*, 2004.
- [8] S. Forrest, S. Hofmeyr, A.Somayaji, and T. Longstaff. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, 1996.
- [9] D. Gao, M. Reiter, and D. Song. On gray-box program tracking for anomaly detection. In *USENIX Security Symposium*, 2004.
- [10] A. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *USENIX Security Symposium*, 1999.
- [11] S. Hofmeyr, S. Forrest, and A.Somayaji. Intrusion detection using sequences of system calls. *Journal of Computer Security*, 1998.
- [12] K. Kendell. A database of computer attacks for the evaluation of intrusion detection systems. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, 1999.
- [13] J.Z. Kotler and M.A. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *IEEE International Conference on Data Mining*, pages 123–130, 2003.
- [14] C. Krugel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *ACM Symposium on Applied Computing*, 2002.
- [15] T. Lane and C.E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Computer Security*, 2(3):295–331, 1999.
- [16] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 darpa off-line intrusion detection evaluation. *Computer Networks*, 2000.
- [17] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear threshold algorithm. *Machine Learning*, 2:285–318, 1988.

- [18] N. Littlestone and M.K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.
- [19] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *IEEE International Conference on Data Mining*, 2003.
- [20] D. Moore, C. Shannon, and k claffy. Code red: a case study on the spread and victims of an internet worm. In *ACM SIGCOMM Workshop on Internet Measurement*, 2002.
- [21] D. Mutz, F. Valeur, C. Kruegel, and G. Vigna. Anomalous system call detection. *ACM Transactions on 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 Transactions on Networking*, 3:226–244, 1995.
- [24] 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 Symposium*, 2006.
- [25] M. Roesch. Snort - lightweight intrusion detection for networks. In *USENIX LISA*, 1999.
- [26] 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.
- [27] J. Shavlik and M. Shavlik. Selection, combination, and evaluation of effective software sensors for detecting abnormal computer usage. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2004.
- [28] S. Staniford, J.A. Hoagland, and J.M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10:105–136, 2002.
- [29] K. Tan and R. Maxion. "why 6?" defining the operational limits of stide. In *IEEE Symposium on Security and Privacy*, pages 188–201, 2002.
- [30] G. Tandon and P.K. Chan. On the learning of system call attributes for host-based anomaly detection. *International Journal on Artificial Intelligence Tools*, 2006.
- [31] D. Wagner and P. Soto. Mimicry attacks on host-based intrusion detection systems. In *ACM Conference on Computer and Communications Security*, 2002.
- [32] K. Wang, G. Cretu, and S. Stolfo. Anomalous payload-based worm detection and signature generation. In *International Symposium on Recent Advances in Intrusion Detection*, 2005.
- [33] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*, 1999.
- [34] A. Wespi, M. Dacier, and H. Debar. Intrusion detection using variable-length audit trail patterns. In *International Symposium on Recent Advances in Intrusion Detection*, 2000.
- [35] 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.