# Incrementally Learning Rules for Anomaly Detection

**Denis Petrussenko, Philip K. Chan**

Microsoft, Florida Institute of Technology
denipet@microsoft.com, pkc@cs.fit.edu

## Abstract

LERAD is a rule learning algorithm used for anomaly detection, with the requirement that all training data has to be present before it can be used. We desire to create rules incrementally, without needing to wait for all training data and without sacrificing accuracy. The algorithm presented accomplishes these goals by carrying a small amount of data between days and pruning rules after the final day. Experiments show that both goals were accomplished, achieving similar accuracy with fewer rules.

## Introduction

Intrusion detection is usually accomplished through either signature or anomaly detection. With signature detection, attacks are analyzed to generate unique descriptions. This allows for accurate detections of known attacks. However, attacks that have not been analyzed cannot be detected. This approach does not work well with new or unknown threats. With anomaly detection, a model is built to describe normal behavior. Significant deviations from the model are marked as anomalies, allowing for the detection of novel threats. Since not all anomalous activity is malicious, false alarms become a issue.

An offline intrusion detection algorithm called LERAD creates rules to describe normal behavior and uses them to detect anomalies, comparing favorably against others on real-world data (Mahoney and Chan 2003b). However, keeping it up to date involves keeping previous data and re-learning from an ever increasing training set, which is both time and space inefficient.

We desire an algorithm that does not need to process all previous data again every time updated rules are needed. It should keep knowledge from previous runs and improve it with new data. Rules should be created more frequently (i.e. daily instead of weekly), without losing fidelity.

Our contributions include an algorithm which learns rules for anomaly detection as data becomes available, with statistically insignificant difference in accuracy from the offline version and producing fewer rules, leading to less overhead during detection.

## Related Work

Given a training set of normal data instances (one class), LERAD learns a set of rules that describe said training data. During detection, data instances that deviate from the learned rules and generate a high anomaly score are identified as anomalies. Similar to APRIORI (Agrawal & Srikant, 1994), LERAD does not assume a specific target/class attribute in the training data. Instead, it aims to learn a minimal set of rules that covers the training data, while APRIORI identifies **all** rules that exceed a given confidence and support. Furthermore, the consequent of a LERAD rule is a set of values, not a single value as in APRIORI.

Learning anomaly detection rules is more challenging than learning classification rules with multiple classes specified in the training data, which provide information for locating class boundary. WSARE (Wong et al, 2005) generates a single rule for a target day. The goal is to describe as many anomalous training records as possible with a single output rule that is the best (most statistically significant) description of all relationships in the training data. Marginal Method (Das and Schneider, 2007) looks for anomalous instances in the training set by building a set of rules that look at attributes which are determined to be statistically dependent. The rules are used to classify training data as "normal" or "abnormal".

VtPath (Feng et al, 2003) analyzes system calls at the kernel level, looking at the call stack during system calls and building an execution path between subsequent calls. This approach examines overall computer usage, not any particular application. PAYL (Wang et al, 2005) examines anomalous packets coming into and then being sent out from the same host. The resulting signatures can detect similar activity and can be quickly shared between various sites to deal with zero-day threats. Kruegel and Vigna (2003) examine activity logs for web applications, looking at a specific subset of parameterized web requests.

## Original Offline LERAD (OFF)

LEarning Rules for Anomaly Detection, or LERAD (Mahoney & Chan, 2003a), is an efficient, randomized algorithm that creates rules in order to model normal behavior. Rules describe what data should look like and are used to generate alarms when it no longer looks as expected. They have the format:

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

where $a_i$ is attribute $i$ in the dataset, $v_{ij}$ is the $j^{\text{th}}$ value of $a_i$ and $r$, $n$ and $w$ are statistics used to calculate a score upon violation.

$r$ is number of unique values in the rule's consequent, representing the likelihood of it being violated. $n$ is records that matched the rule's antecedent. Each time $n$ is incremented, $r$ can potentially be increased as well, if the consequent attribute value ($v_{ci}$) in the tuple is not already present in the rule. $w$ is the rule's validation set performance, calculated from mutual information (Tandon & Chan, 2007).

LERAD is composed of four main steps shown in the pseudocode in Fig. 1. Rule generation (st. 1) picks antecedent and consequent attributes based on similarities of tuples randomly chosen from the sample set $D^S$. After sufficient rules are generated, a coverage test is performed to minimize the number of rules (st. 2). Step 3 exposes rules to the training set, updating their $n$ and $r$ values based on how they apply to $D^T$. Lastly the weight of evidence is calculated for each rule by applying it to $D^V$ and seeing how many times it conforms or violates (st. 4). The output is a set of rules $R$ that can generate alarms on unseen data.

For each record $d \in D$, every rule $r_i \in R$ will either match or not match antecedent values. Only rules that match are used to compute the anomaly score. Let $R$ be the set of rules whose antecedents match $d$, then

$$Score(d) = \sum_{R_i \in R} t_i w_i \frac{n_i}{r_i}$$

where $t_i$ is time since $R_i$ was last involved in an alarm. The goal is to find rule violations that are most surprising. A rule that has been violated recently is more likely to be violated again, as opposed to one that has been matching records for a long time. Scores above a certain threshold are then used to trigger actual alarms.

## Basic Incremental Algorithm (INCR)

Each dataset fed to INCR is divided into three sets: training, validation and test. For day $k$, let the sample set be $D_k^S$, training set be $D_k^T$ and validation set be $D_k^V$. With OFF, training data consists of one dataset and testing data of another. For INCR, training data is split into roughly equal-sized sets (days), with test data remaining in a single set.

```
Input: sample set ($D^S$), training set ($D^T$), validation set ($D^V$)
Output: LERAD rule set R
  1. Generate and evaluate candidate rules $R'$ from $D^S$
  2. Perform coverage test – select a "minimal" set from $R'$
     that covers $D^S$:
       a. Sort $R'$ in increasing order of violation probability
       b. Discard rules from $R'$ that do not cover any attribute
          values in $D^S$
  3. Train the rest of $R'$ on $D^T$
  4. Validate $R'$ on $D^V$
       a. Increase w on rule conformance (incr. rule belief)
       b. Decrease w on rule violation (reduce belief)
```

***Fig. 1****: Offline LERAD algorithm (OFF)*
*[Adapted from Fig. 1 in Tandon & Chan, 2007]*

Regardless of being split up, in this paper the training data for INCR is exactly the same as for OFF. Sample set $D_K^S$ is generated each day by randomly copying a few records from $D_k^T$; $D_k^V$ consists of a small fraction (e.g. 10%) of all training set records, removed before training data is loaded. Training records that are moved into the validation set are chosen at random.

INCR applies the same basic algorithm to training data as OFF. The main difference is that rules and sample set are carried between, and trained on, later days (***Fig. 2***). INCR mirrors OFF and creates a new sample set $D_k^S$ for each day $k$ of data, but shrunk proportionally to the *total number of days in training data* ($m$): $|D_k^S| = \frac{|D^S|}{m}$. That way INCR won't access many more sample records than OFF when creating rules, which would result in different rules. This may not be detrimental, but it does stray from the OFF algorithm. Furthermore, in addition to using $D_k^S$, sample sets from previous days are carried over and joined together into the combined sample set $D_k^{CS}$, from which $R'_k$ (new rules for day $k$) are made. $D_k^T$ and $D_k^V$ are not carried over between days since they are huge compared to $D_k^S$. For datasets that OFF can handle they could be, but INCR is designed to work with far more data.

As before, a coverage test is performed after $R'_k$ is generated to ensure that rules keep their quality, but on $D_k^{CS}$ instead of $D_k^S$, since that is what the rules were created from (Step 2 in ***Fig. 2***). After coverage test, remaining rules in $R'_k$ are compared with $R_{k-1}$ (containing all previous days). Any rules in $R'_k$ that already exist in $R_{k-1}$ are removed from $R'_k$ (Step 3 in ***Fig. 2***). Removing duplicates from $R'_k$ is not a problem because at this point, rules in $R'_k$ have only been trained on $D_k^{CS}$, which contains mostly data that $R_{k-1}$ has been exposed to. The only thing lost by removing duplicates is what $R'_k$ gained from training on $D_k^S$ (subset of $D_k^{CS}$). This is remedied by merging new rules with old $R_k = R_{k-1} \cup R'_k$ (Step 6 in ***Fig. 2***) and training $R_k$ on $D_k^T$ (superset of $D_k^S$). Before training, rules from $R_{k-1}$ already have some statistics from previous days and rules from $R'_k$ have statistics from being trained on $D_k^{CS}$. These are not reset for $D_k^T$ training, statistics from $D_k^T$ are simply added. That way, a rule trained (for example) on days 1 to $k$ has the same information as a rule trained on $D^T$ that consists of all records from days 1 to $k$. After training, $R_k$ is validated on $D_k^V$ (Step 8 in ***Fig. 2***). Again, the previous $w$ is not reset but rather combined with the value from day $k$.

## Collecting Appropriate Statistics

Recall that LERAD generates rules strictly from the sample set, which is a comparatively small collection of records meant to be representative of the entire dataset. Antecedent and consequent attributes are picked based on similarities of tuples randomly chosen from the sample set and never changed after they are picked. Therefore the sample set is solely responsible for the structure of all rules in LERAD.

Using $D_k^{CS}$ instead of $D^S$ allowed INCR to generate more rules with the same structure as OFF, but with different statistics. Comparing rules present in both INCR and OFF but only detecting in OFF, a number of deficiencies showed up in INCR rules: statistics were off, resulting in lower alarm scores and missed detections.

To fix this, similar rules generated by both algorithms need to contain similar statistics. INCR rules generated towards the end had significantly smaller $n$ values, as well as smaller $r$ and $w$ values, because they do not have access to data from days before they were generated. To completely eliminate this problem, all days would have to be kept around so that all rules can be trained on all days. With an incremental algorithm, this is not feasible. Instead, extra statistics are kept that are represent each $D_k^T$. An additional piece of information is carried for each day $k$:

$$ScalingFactor_k = \frac{|D_k^T|}{|D_k^S|}.$$

By using tuples from $D_k^S$ and repeating each one $ScalingFactor_k$ times, a virtual training set $D_k^{VT}$ is created, representing the training set from that day. For each day $k$, all previous virtual training sets ($D_1^{VT}$ to $D_{k-1}^{VT}$) are joined together to create a combined virtual training set $D_k^{CVT} = D_1^{VT} \cup D_2^{VT} \cup \ldots \cup D_{k-1}^{VT}$.

New rules for day $k$ are still generated on $D_k^{CS}$. However, now $D_k^{CVT} \cup D_k^T$ is used for training instead of just $D_k^T$, allowing rules to obtain consequent values that may have only been present in previous days and enabling $n$ counts to reflect records not present in all days (Step 4 in ***Fig. 2***). Note that the virtual set is purely an abstract notion; sample sets are simply used in a manner that is consistent with having a virtual training set. During training, when a rule matches a record, its $n$ value is increased by $ScalingFactor_k$ instead of just 1. Consequent values are simply appended, along with $r$ being incremented, if they don't yet exist in the rule. The weight of evidence calculation for new rules is also modified to collect statistics from $D_k^{CS}$ in addition to $D_k^V$. Since rules from previous days were trained on actual training sets that $D_k^{CVT}$ attempting to approximate, they are not exposed to it, or $D_k^{CS}$. $ScalingFactor$ only benefits $n$, since it is directly affected by it (Step 4 in ***Fig. 2***).While this yields results that parallel the OFF algorithm, it will have to be changed for a purely incremental implementation. Currently, the sample set grows without bound in order to match the OFF sample set. In the real world, there will need to be a limiting mechanism on its growth, such as not keeping sample sets older than a certain number of days.

```
Input: m sample sets ($D_1^S$ to $D_m^S$), m training sets ($D_1^T$ to $D_m^T$)
and m validation sets ($D_1^V$ to $D_m^V$)
Output: LERAD rule set R
For each day $k$:
    1.  Generate and evaluate candidate rules $R'_k$ from $D_k^{CS}$
    2.  Perform coverage test – select a "minimal" set from $R'_k$
        that covers $D_k^{CS}$:
        a.  Sort $R'_k$ in increasing order of violation probability
        b.  Discard rules from $R'_k$ that do not cover any attribute
            values in $D_k^{CS}$
    3.  Delete candidate rules that were generated and saved
        during any previous days (duplicates)
        a.  Increment B values of each old rule for each deletion
            caused by it
    4.  Train the rest of $R'_k$ on $D_k^{CVT}$
    5.  Validate $R'_k$ on $D_k^{CS}$
        a.  Increase w on rule conformance (increase rule belief)
        b.  Decrease w on rule violation (reduce rule belief)
    5.  Combine candidate rules with all rules from days 1 to
        $k-1$
    6.  Train all rules on $D_k^T$ [Section 3.2]
    7.  Validate all rules on $D_k^V$ [Section 3.2]
Remove all rules with low B values. [Section 3.4]
```

*Fig. 2: Incremental LERAD algorithm (INCR) [steps different from offline are in* ***bold****]*

## Pruning Rules

Because INCR generates rules multiple times (one for each day/period), INCR generally creates more rules overall than OFF. By design, a lot are common between them. However, there are usually some extra rules unique to INCR. Some cause detections that would otherwise have been missed, others cause false alarms that drown out detections. An analysis yielded the number of generations (or birthdays) as the best predictor of inaccurate rules. Let $B$ be the number of times a rule was generated (born). Most rules unique to INCR and only causing false alarms had low $B$ values.

This heuristic allowed for removal of inaccurate rules. INCR removes rules with $B$s below a certain threshold from the final rule set as the very last step. This drops rules that were causing false alarms, increasing performance. For example, the LL/tcp dataset went from final INCR rule count of 250 to just 68 when $B$ was set to 2, compared to 77 rules in OFF. While the exact value of $B$ depends on the dataset, experiments showed that $B = 2$ tends to provide closest AUC values to OFF. $B$ is currently determined by sensitivity analysis across all possible values. Further work is needed to establish the ideal $B$ value during training.

## Empirical Evaluation

In this section, we evaluate the performance of incremental LERAD and compare it to offline. Evaluation was performed on five different datasets: DARPA / Lincoln Laboratory (LL TCP) contained 185 labeled instances of 58 different attacks (full attack taxonomy in Kendell, 1999); UNIV comprised of over 600 hours of network traffic, collected over 10 weeks from a university departmental server (Mahoney and Chan, 2003a), containing six attacks; DARPA BSM was an audit log of system calls from a Solaris host, with 33 attacks spread across 11 different applications (see Lippmann et al, 2000); Florida Tech and University of Tennessee at Knoxville (FIT/UTK) contained macro execution traces with 2 attacks (Mazeroff et al, 2003); University of New Mexico (UNM) set included system calls from 3 applications (Forrest et al, 1996) with 8 distinct attacks.

## Experimental Procedures

For all datasets, training data was entirely separate from testing. LL training consisted of 7 days, ~4700 records each, with almost 180,000 records in testing. For UNIV, week 1 was split into 5 days of training, ~2700 records each, with weeks 2 through 10 used for testing (~143,000 records). In BSM, week 3 was separated into 7 days or ~26,000 records each, with weeks 4 and 5 used for testing (~350,000 records). FIT/UTK had 7 days of training, with ~13,000 records each day and testing. UNM contained 7 days of data with ~850 records each and ~7800 for testing.

Several adjustable parameters were used. Size of $D_k^S$ was set to $\frac{100}{m}$, where 100 was the sample set in offline experiments and $m$ was the number of training days (see pg 2). This still resulted in tiny sample sets when compared to training. For example, for LL, $|D_k^S| = 0.3\%$ of $|D_k^T|$, putting sample set size well below 1% of training. Validation set $D_k^V$ was 10% of $D^T$ and contained same random records for OFF and INCR. Candidate rule set size $|R_k'|$ was set to 1000 and the maximum number of attributes per rule was 4, all to mirror Tandon & Chan (2007) experiments. Rule pruning parameter $B$ was 2, as this produced the closest performance curve to OFF.

On every dataset, both INCR and OFF ran 10 times each with random seeds. For datasets with multiple applications, a model was created for each one and results averaged together, weighted by training records for that application. As applications had vastly different amounts of training records, their results could not be simply averaged together. LERAD is looking for anomalous activity, so apps with more training records (i.e. activity) have more alarms and are more relevant to performance on the whole dataset.

For rule comparison, each INCR run is compared to all OFF runs and average counts of rules involved are taken. Then all INCR runs are averaged together for each dataset. For datasets with multiple applications, results for each one are averaged together for the whole dataset.

## Evaluation Criteria

Because false alarm (FA) rates are an issue in anomaly detection, we focused on low FA rates of 0.1%. Alarm threshold was varied in small increments between 0 and a value that resulted in FA rate of 0.1%, with percentage of valid detections measured each time. This was plotted on a receiver operating characteristic (ROC) curve, where X axis was false alarm rate and Y axis was detection rate. Area under this curve, or AUC, is absolute algorithm performance. Higher AUC values indicate better performance. Since we concentrated on the first $\frac{1}{1000}$th of the ROC curve, highest performance possible in our tests was 0.001.

To average together multiple applications from a single dataset, let $CT_{app} = |D_{app}^T| + |D_{app}^V|$ be the count of training records for an application. Then dataset $AUC$ is:

$$AUC^{avg} = \sum_{app \in Apps} \left( AUC_{app} \frac{CT_{app}}{\sum_{app \in Apps} CT_{app}} \right)$$

## Sensitivity Analysis of Parameter *B*

One way to compare performance is through $AUC$s. Let $\Delta AUC$ be the difference between INCR and OFF:

$$\Delta AUC = AUC_{INCR} - AUC_{OFF}$$

A positive $\Delta AUC$ value means INCR is more accurate than OFF. $\Delta AUC$ is negative when INCR is less accurate, with lower values for worse performance. Our goal was for INCR to be close to OFF, $\Delta AUC$s closest to 0 were desired.

Pruning threshold $B$ (pg. 3) has the largest effect on $AUC$. Rules with $B$ under a certain threshold are removed from INCR after training. There is no apparent way of picking $B$, so we analyzed all of them. Maximum $B$ value tested was 5 since most datasets did not produce any rules past that, either because they were only split into 5 days or because no rules were generated more than 5 times in a row. Lowest $B$ value was 1 since any lower is meaningless.

$B$s and resulting $\Delta AUC$s for all datasets are shown in ***Fig. 3*** (0.1% FA rate). With BSM, LL TCP and UNIV datasets, INCR performance is similar to OFF, resulting in $\Delta AUC$ values fairly close to zero. With UNM, there were no attacks detected by OFF below 0.1% false alarm rate, while INCR did detect some attacks, resulting large $\Delta AUC$ differences. The opposite situation occurred with FIT/UTK, with INCR detecting no attacks. Overall, there is no obvious relationship between $B$ and $\Delta AUC$ values across all datasets.

To determine if $\Delta AUC$s are statistically significant, and

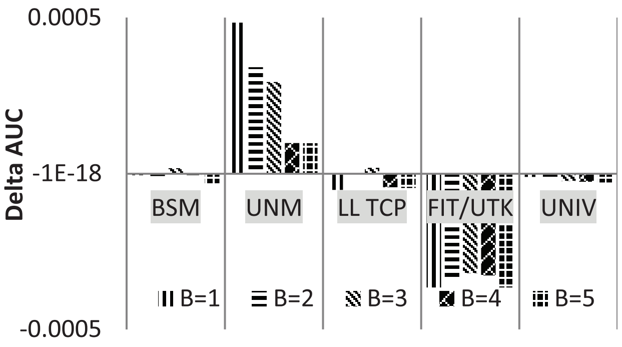

***Fig. 3****: $\Delta AUC$s versus Bs (0.1% FA rate)*

for which $B$ values, we perform the two-sample T-test on data from ***Fig. 3***. $AUC$s from 10 INCR runs for each dataset are compared against $AUC$s from 10 OFF runs. For multi-application datasets, $AUC$s are averaged, weighted by training records (see page 4). $B = 4$ had statistically insignificant $\Delta AUC$s across most datasets (***Table 1***). For FIT/UTK, there was no statistically insignificant $\Delta AUC$ due to very poor performance of INCR.

***Table 1*** shows the absolute two-tail probabilities in Student's t distribution (with degrees of freedom = 9), computed from experimental data. For each dataset and $B$, ***Table 1*** contains the probability that INCR and OFF are not significantly different. Because the goal is to have similar performance, we concentrate on results where INCR and OFF do not have a statistically significant difference. Cells with $P > 0.05$ show which instances are not significantly different. That is, they do not have a probability < 0.05 of being significantly different, which is required in order to be at least 95% confident. Note that some probabilities are 0.0 – this happens when $\Delta AUC$s are extreme. In UNM, OFF had tiny $AUCs$, very different from what INCR had with low $B$s. However, as $B$ increased and performance fell (***Fig. 3***), INCR became almost as bad as OFF, bad enough to no longer be statistically significant. In FIT/UTK, INCR had low scores, never getting near OFF.

***Table 1****: P(T<=t) two-tail for two-sample T-test*

| Dataset | B=1 | B=2 | B=3 | B=4 | B=5 |
|---|---|---|---|---|---|
| BSM | 0.27 | 0.18 | 0.02 | 0.66 | 0.01 |
| UNM | 0.00 | 0.00 | 0.01 | 0.17 | 0.17 |
| LL TCP | 0.01 | 0.57 | 0.37 | 0.08 | 0.10 |
| FIT/UTK | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| UNIV | 0.40 | 0.17 | 0.08 | 0.06 | 0.04 |

To conclusively determine which $B$s cause statistically insignificant $\Delta AUC$s across all datasets, we apply the paired T-test. For each dataset, average $AUC$ for each $B$ was paired with average OFF $AUC$ for that dataset. Again, as with ***Table 1***, numbers in ***Table 2*** represent the probabilities associated with Student's T-test. In order to be at least 95% confident that INCR and OFF $AUC$s are statistically different, values in ***Table 2*** need to be under 0.05. No $B$ yielded statistically different performance between INCR and OFF. This is due mostly to the similar variations of $AUC$s between all datasets, which were exhibited by both OFF and INCR. The two least different $B$ values were 2 and 3, and since $B = 2$ had insignificant performance difference on 3 of 5 datasets (see ***Table 1***), the rest of our analysis is based on B=2.

***Table 2****: P(T<=t) two-tail, paired two sample T-test*

| B=1 | B=2 | B=3 | B=4 | B=5 |
|---|---|---|---|---|
| 0.95 | 0.98 | 0.98 | 0.44 | 0.39 |

## Analysis of Rule Content

There are 4 cases when comparing INCR and OFF rules. Let $\alpha$ be rules that have same antecedent attributes, same consequent attribute and same consequent values. They are structurally exactly alike. $\beta$ consists of rules with same antecedent attributes and same consequent attribute, but different consequent values. $\gamma$ has rules with identical antecedent but different consequent attributes. $\lambda$ contains rules that have different antecedent attributes. Each INCR rule compared to OFF will be in either $\alpha$, $\beta$, $\gamma$ or $\lambda$, which are mutually exclusive and describe all possible outcomes.

With $B = 2$ and FA rate at 0.1% (section 4.4), performance difference between INCR and OFF is insignificant but present. To understand why, we look at sizes of $\alpha$, $\beta$, $\gamma$ and $\lambda$ (***Table 3***). Datasets with high $\alpha$, $\beta$ and $\gamma$ have smallest $\Delta AUC$s. A large percentage of similar rules should naturally lead to similar performance, as with UNIV, LL TCP and BSM. UNM's $\Delta AUC$ is large due to OFF's poor performance. FIT/UTK's $\Delta AUC$ is due to a very small number of INCR rules in general.

***Table 3****: Size of $\alpha$, $\beta$, $\gamma$ and $\lambda$ as percent of total number of OFF rules (B=2)*

| Dataset | $\alpha$ | $\beta$ | $\gamma$ | $\lambda$ | $\Delta AUC$ |
|---|---|---|---|---|---|
| **UNIV** | 17% | 5% | 19% | 60% | -0.00002 |
| **FIT/UTK** | 2% | 1% | 3% | 94% | -0.00034 |
| **LL TCP** | 24% | 5% | 22% | 49% | 0.00001 |
| **UNM** | 27% | 14% | 21% | 38% | 0.00034 |
| **BSM** | 17% | 5% | 11% | 66% | -0.00001 |

## Analysis of Rule Statistics

To determine just how close INCR rules are to OFF, their $n$, $r$ and $w$ are compared. For this to be accurate, rules have to be of the same type. Since $n$ are only dependent on the antecedent attributes, $\alpha$, $\beta$ and $\gamma$ are used. Comparing $r$ only makes sense when checking the same consequent attribute, so only $a$ and $\beta$ are used. Since $w$ describe the effectiveness of a rule as a whole, only $\alpha$ is relevant.

Because we are now analyzing subsets of rules, we look at performance of individual rules. Alarms during attacks are called detections (DETs); those triggered during normal activity are false alarms (FAs). For each DET or FA triggered on record $d$, the contribution of each rule $r_i$ is:

$$Contribution(d, r_i) = \frac{t_i w_i \frac{n_i}{r_i}}{Score(d)}$$

Then for each rule $r_i$, all contributions to detections across the whole dataset are added into $DET_{TOTAL}$ and all false alarm contributions added into $FA_{TOTAL}$. Performance of a rule is then gauged by the number of net detections, or $ND$:

$$ND = DET_{TOTAL} - FA_{TOTAL}$$

Having an exact number that represents how well a single rule is behaving allows us to directly examine the effect of $n$, $r$ and $w$ on performance. Since we are interested in the difference in performance between INCR and OFF, discrepancy between values ($\delta$) is used:

$$\delta_X = \frac{X_{INCR} - X_{OFF}}{X_{OFF}}$$

where $X$ is either $n$, $r$, $w$ or $ND$. $\delta$ is normalized in order to bring the performance of all datasets onto a level playing field.

Discrepancy in which rule statistic is more responsible for discrepancy in performance? **Table 4** shows two avg. $|\delta_{ND}|$ for each statistic, one for underestimated statistic and one for overestimated, for each dataset (followed by st.dev). For example, the average performance error for rules with underestimated $n$ is shown in the $\delta_n < 0$ column. Some data is N/A because all rules with comparable $w$ are overestimated in UNM and underestimated in FIT/UTK. Furthermore, it is impossible for INCR to overestimate $r$. Overall, $\delta_n$ is most responsible for $\delta_{ND}$ and $\delta_w$ is least. This supports our choice of $ScalingFactor$ to help statistics on pg. 3, since it mostly helped $n$ values. Does over or under estimation in rule statistics cause more performance discrepancy? For $n$, overestimation usually produces more error, $w$ is opposite.

Does INCR over or under estimate rule statistics? ***Table 5*** lists average $\delta \pm$ st.dev. for rule statistics, independent of $\delta_{ND}$. The left half (white) contains the average $\delta$, which indicates if the rule statistic is over or underestimated (discrepancy direction). $n$, $r$, $w$ tend to be underestimated on average, except in two datasets, $w$ tends to be overestimated on average. The right half (gray) shows the avg. of absolute $\delta$, indicating the amount of over/underestimation (discrepancy magnitude). There is least error in $n$, followed by $w$ (except in UNM dataset) and $r$ has the most error. This suggests that our improvements did help $n$ and $w$ and the next large performance gain lies in improving $r$.

## Conclusions

We introduced an incremental version of the LERAD algorithm, which generates rules before all of training data is available, improving them as more data is analyzed. The incremental nature of our algorithm does not affect performance. Experiments show that after processing the same amount of data, the difference in accuracy of incremental vs offline algorithms is statistically insignificant. The incremental algorithm also generates fewer rules, leading to lower detection overhead. Our algorithm can be applied to datasets that were previously out of reach for offline methods.

Our approach to calculating statistics is most beneficial for $n$, somewhat good for $w$ and not relevant for $r$. To improve r, the distribution of consequent values would need to be modeled. Also, we currently do not have a method for setting the pruning threshold $B$; we plan to investigate the selection of $B$ based on the performance of different $B$ values on the validation sets.

***Table 4***: *Average $|\delta_{ND}|$ for n r w*

| | $\delta_n < 0$ | $\delta_r < 0$ | $\delta_w < 0$ | $\delta_n > 0$ | $\delta_r > 0$ | $\delta_w > 0$ |
|---|---|---|---|---|---|---|
| UNIV | 0.03±0.05 | 0.01±0.02 | 0.01±0.01 | 0.05±0.05 | | 0.00±0.00 |
| UNM | 0.31±2.19 | 0.45±3.32 | N/A | 0.77±4.73 | | 0.11±0.33 |
| LL TCP | 0.07±0.11 | 0.06±0.08 | 0.07±0.08 | 0.09±0.11 | N/A | 0.06±0.05 |
| FITUTK | 0.18±0.18 | 0.20±0.20 | 0.21±0.21 | 0.19±0.13 | | N/A |
| BSM | 0.25±0.50 | 0.16±0.39 | 0.09±0.12 | 0.10±0.13 | | 0.21±0.46 |

***Table 5***: *Average δ (discrepancies) in n r w*

| | Discrepancy Direction | | | Discrepancy Magnitude | | |
|---|---|---|---|---|---|---|
| | $\delta_n$ | $\delta_r$ | $\delta_w$ | $\lvert\delta_n\rvert$ | $\lvert\delta_r\rvert$ | $\lvert\delta_w\rvert$ |
| UNIV | -0.00±0.04 | -0.06±0.11 | -0.07±0.10 | 0.02±0.03 | 0.06±0.11 | 0.08±0.09 |
| UNM | -0.03±0.06 | -0.09±0.07 | 1.60±0.31 | 0.05±0.05 | 0.09±0.07 | 1.60±0.31 |
| LL TCP | -0.03±0.06 | -0.07±0.14 | -0.09±0.07 | 0.04±0.05 | 0.07±0.14 | 0.09±0.07 |
| FITUTK | -0.01±0.07 | -0.01±0.02 | -0.03±0.03 | 0.04±0.06 | 0.01±0.02 | 0.03±0.03 |
| BSM | -0.01±0.02 | -0.12±0.16 | 0.01±0.11 | 0.02±0.02 | 0.12±0.16 | 0.07±0.09 |

## References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. very Large Data Bases, (VLDB),* 487-499.

Das, K., & Schneider, J. (2007). Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 220-229.

Feng, H., Kolesnikov, O., Fogla, P., Lee, W., & Gong, W. (2003). Anomaly detection using call stack information. In *Proceedings of Symposium on Security and Privacy.* 62-75.

Forrest, S., Hofmeyr, S., Somayaji, A., & Longstaff, T. (1996). A sense of self for unix processes. In Proceedings of *1996 IEEE Symposium on Security and Privacy.* 120-128.

Kruegel, C., & Vigna, G. (2003). Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM Conference on Computer and Communications Security,* 251-261.

Lippmann, R., Haines, J. W., Fried, D. J., Korba, J., & Das, K. (2000). The 1999 DARPA off-line intrusion detection evaluation. Computer Networks, 34(4), 579-595.

Mahoney, M. V., & Chan, P. K. (2003a). Learning rules for anomaly detection of hostile network traffic. In *Proc. of International Conference on Data Mining (ICDM),* 601-604.

Mahoney, M. V., & Chan, P. K. (2003b). Learning Rules for Anomaly Detection of Hostile Network Traffic. *Technical Report CS-2003-16, Florida Tech.*

Mazeroff, G., De Cerqueira, V., Gregor, J., & Thomason, M. G. (2003). Probabilistic trees and automata for application behavior modeling. Paper presented at the *41st ACM Southeast Regional Conference Proceedings,* 435-440.

Tandon, G. & Chan, P. (2007). Weighting versus Pruning in Rule Validation for Detecting Network and Host Anomalies. In *Proc. ACM Intl. Conf. on Knowledge Discovery and Data Mining (KDD).* 697-706.

Wang, K., Cretu, G., & Stolfo, S. J. (2005). Anomalous payload-based worm detection and signature generation. In *Proceedings of RAID*, *Lecture Notes in Computer Science, 3858,* 227-246.

Wong, W. K., Moore, A., Cooper, G., & Wagner, M. (2005). What's strange about recent events (WSARE): An algorithm for the early detection of disease outbreaks. *Journal of Machine Learning Research, Vol 6*, 1961-1998.