International Journal For Technological Research In Engineering
Volume 1, Issue 8, April-2014 *ISSN (Online): 2347 - 4718*

DYNAMICALLY INTRUSION DETECTION SYSTEM MODEL

Md Nurul Hasan[1], Mr. Ar. Arunachalam[2]
[1]Student, [2]M. Tech Guide
Department of Computer Science
Bharath University, Chennai
Tamil Nadu, India.

Abstract: An intrusion detection system (IDS) is a security layer used to detect ongoing intrusive activities in information systems. Traditionally, intrusion detection relies on extensive knowledge of security experts, in particular, on their familiarity with the computer system to be protected. The system is evaluated using the KDDCup'99 intrusion detection dataset. Experimental results show that the system achieves up to 35% improvement in terms of misclassification cost when compared with a system lacking the tuning feature. If only 10% false predictions are used to tune the model, the system still achieves about 30% improvement. Moreover, when tuning is not delayed too long, the system can achieve about 20% improvement, with only 1.3% of the false predictions used to tune the model..
Keywords: Attack detection model, classification, data mining, Intrusion detection, learning algorithm, data stream algorithm.

1. INTRODUCTION

Intrusion detection relies on the extensive knowledge of security experts, in particular, on their familiarity with the computer system to be protected. To reduce this dependence, various data-mining and machine learning techniques have been used in research projects: Audit Data Analysis and Mining (ADAM) [1] combined the mining of association rules and classification to discover attacks from network traffic data. The Information and Systems Assurance Laboratory (ISA) intrusion detection system (IDS) employed multiple statistics-based analysis techniques, including chi-square [2] and exponentially weighted moving averages based on statistical process control [3], Mining Audit Data for Automated Models for Intrusion Detection (MAMAD ID) [5] applied association rules and a frequent episodes program. The Minnesota Intrusion Detection System (MINDS) [6] included a density-based outlier detection module and an association-pattern analysis module to summarize network connections. The quality of training data has a large effect on the learned model. In intrusion detection, however, it is difficult to collect high-quality training data. New attacks leveraging newly discovered security weaknesses emerge quickly and frequently. It is impossible to collect all related data on those new attacks to train a detection model before those attacks are detected and understood. In addition, due to the new hardware and software deployed in the system, system and user behaviors will keep on changing, which causes degradation in the performance of detection models. As a consequence, a fixed detection model is not suitable for an IDS. Instead, after an IDS is deployed, its detection model has to be tuned continually. For commercial products (mainly signature/misuse-based IDS), the main tuning method has been to filter out signatures to avoid generating noise [8] and add new signatures. In data-mining-based intrusion detection, system parameters are adjusted to balance the detection and false rates. Such tuning is coarse, and the procedure must be performed manually by the system operator. Other methods that have been proposed rely on "plugging in" a special purpose sub model [9] or superseding the current model by dynamically mined new models [10]–[12]. Training a special-purpose model forces the user to collect and construct high-quality training data. Mining a new model in real time from unverified data incurs the risk that the model could be trained by an experienced intruder to accept abnormal data. In this paper, we present tuning IDS by some algorithm the quality of training data has a large effect on the learned model. In intrusion detection, however, it is difficult to collect high-quality training data. New attacks leveraging newly discovered security weaknesses emerge quickly and frequently. It is impossible to collect all related data on those new attacks to train a detection model before those attacks are detected and understood. In addition, due to the new hardware and software deployed in the system, system and user behaviors will keep on changing, which causes degradation in the performance of detection models. As a consequence, a fixed detection model is not suitable for an IDS. Instead, after an IDS is deployed, its detection model has to be tuned continually. For commercial products (mainly signature/misuse-based IDS), the main tuning method has been to filter out signatures to avoid generating noise [8] and add new signatures. In data-mining-based intrusion detection, system parameters are adjusted to balance the detection and false rates. Such tuning is coarse, and the procedure must be performed manually by the system operator. Other methods that have been proposed rely on "plugging in" a special purpose sub model [9] or superseding the current model by dynamically mined new models [10]–[12]. Training a special-purpose model forces the user to collect and construct high-quality training data. Mining a new model in real time from unverified data incurs the risk that the model could be trained by an experienced intruder to accept abnormal data. In this paper, we present. Our system takes advantage of the analysis of alarms by the system operators: the detection model is tuned on-the-fly with the verified data, yet the burden on the system operator

is minimized. Experimental results show that the system achieves up to 35% improvement in terms of misclassification cost compared with the performance of a system lacking the model tuning procedure. If only 10% false predictions are used to tune the model, the system still achieves roughly 30% improvement. When tuning is delayed only a short time, the system achieves about 20% improvement with only 1.3% false predictions used to tune the model. Selective verification on predictions with lo Experimental results show that the system achieves up to 35% improvement in terms of misclassification cost compared with the performance of a system lacking the model tuning procedure. If only 10% false predictions are used to tune the model, the system still achieves roughly 30% improvement. When tuning is delayed only a short time, the system achieves about 20% improvement with only 1.3% false predictions used to tune the model. Selective verification on predictions with low.

2. RELATED WORK

Most existing IDS are optimized to detect attacks with high accuracy. However, they still have various disadvantages that have been outlined in a number of publications and a lot of work has been done to analyze IDS in order to direct future research (cf. [5], for instance). Besides others, one drawback is the large amount of alerts produced. Recent research focuses on the correlation of alerts from (possibly multiple) IDS. If not stated otherwise, all approaches outlined in the following present either online algorithms or—as we see it—can easily be extended to an online version. Probably, the most comprehensive approach to alert correlation is introduced in [6]. One step in the presented correlation approach is attack thread reconstruction, which can be seen as a kind of attack instance recognition. No clustering algorithm is used, but a strict sorting of alerts within a temporal window of fixed length according to the source, destination, and attack classification (attack type). In [7], a similar approach is used to eliminate duplicates, i.e., alerts that share the same quadruple of source and destination address as well as source and destination port. In addition, alerts are aggregated (online) into predefined clusters (so-called situations) in order to provide a more condensed view of the current attack situation. The definition of such situations is also used in [8] to cluster alerts. In [9], alert clustering is used to group alerts that belong to the same attack occurrence. Even though called clustering, there is no clustering algorithm in a classic sense. The alerts from one (or possibly several) IDS are stored in a relational database and a similarity relation—which is based on expert rules—is used to group similar alerts together. Two alerts are defined to be similar, for instance, if both occur within a fixed time window and their source and target match exactly. As already mentioned, these approaches are likely to fail under real-life conditions with imperfect classifiers (i.e., low-level IDS) with false alerts or wrongly adjusted time windows. Another approach to alert correlation is presented in [10]. A weighted, attribute-wise similarity operator is used to decide whether to fuse two alerts or not. However, as already stated in [11] and [12], this approach suffers from the high number of parameters that need to be set. The similarity operator presented in [13] has the same disadvantage— there are lots of parameters that must be set by the user and there is no or only little guidance in order to find good values. In [14], another clustering algorithm that is based on attribute-wise similarity measures with user defined parameters is presented. However, a closer look at the parameter setting reveals that the similarity measure, in fact, degenerates to a strict sorting according to the source and destination IP addresses and ports of the alerts. The drawbacks that arise thereof are the same as those mentioned above. In [15], three different approaches are presented to fuse alerts. The first, quite simple one groups alerts according to their source IP address only. The other two approaches are based on different supervised learning techniques. Besides a basic least-squares error approach, multi-layer perceptions, radial basis function networks, and decision trees are used to decide whether to fuse a new alert with an already existing meta-alert (called scenario) or not. Due to the supervised nature, labeled training data need to be generated which could be quite difficult in case of various attack instances. The same or quite similar techniques as described so far are also applied in many other approaches to alert correlation, especially in the field of intrusion scenario detection. Prominent research in scenario detection is described in [16],[17], [18], for example. More details can be found in [19].In [20], an offline clustering solution based on the CURE algorithm is presented. The solution is restricted to numerical attributes. In addition, the number of clusters must beset manually. This is problematic, as in fact it assumes that the security expert has knowledge about the actual number of ongoing attack instances. The alert clustering solution described in [11] is more related to ours. A link-based clustering approach is used to repeatedly fuse alerts into more generalized ones. The intention is to discover the reasons for the existence of the majority of alerts, the so-called root causes, and to eliminate them subsequently. An attack instance in our sense can also be seen as a kind of root cause, but in [11] root causes are regarded as "generally persistent" that does not hold for attack instances that occur only within a limited time window. Furthermore, only root causes that are responsible for a majority of alerts are of interest and the attribute-oriented induction algorithm is forced "to find large clusters" as the alert load can thus be reduced at most. Attack instances that result in a small number of alerts (such as PHF or FFB) are likely to be ignored completely. The main difference to our approach is that the algorithm can only be used in an offline setting and is intended to analyze historical alert logs. In contrast, we use an online approach to model the current attack situation. The alert clustering approach described in [12] is based on [11] but aims at reducing the false positive rate. The created cluster structure is used as a filter to reduce the amount of created alerts. Those alerts that are similar to already known false positives are kept back, whereas alerts that are considered to be legitimate (i.e., dissimilar to all known false positives) are reported and not

further aggregated. The same idea—but based on a different offline clustering algorithm—is presented in [21]. A completely different clustering approach is presented in [22]. There, the reconstruction error of an autoassociator neural network (AA-NN) is used to distinguish different types of alerts. Alerts that yield the same (or a similar) reconstruction error are put into the same cluster. The approach can be applied online, but an offline training phase and training data are needed to train the AA-NN and also to manually adjust intervals for the reconstruction error that determine which alerts are clustered together. In addition, it turned out that due to the dimensionality reduction by the AA-NN, alerts of different types can have the same reconstruction error which leads to erroneous clustering. In our prior work, we applied the well-known c-means clustering algorithm in order to identify attack instances [23]. However, this algorithm also works in a purely offline manner.

3. PROPOSED SYSTEM

A. Prediction Model and Learning Algorithm

Different model representations have been used in detection Models presented in the literature, among them are rules (Signatures) [1], [5], decision trees [13], neural networks [14], statistical models [2], [3], or Petri nets [15]. In order to allow tuning parts of the model easily and precisely without affecting the rest of the model, we choose rules to represent the prediction model. In an earlier study, this model has demonstrated a good performance [16]. Our model consists of a set of binary classifiers learned from the training dataset by the simple learner with iterative pruning to produce error reduction (SLIPPER) [17], a binary learning algorithm. The initial creation of the detection model is shown in the block diagram in Fig. 1. The preprocessor prepares all binary training datasets from the original training dataset. The algorithm capturing this

Architecture diagram

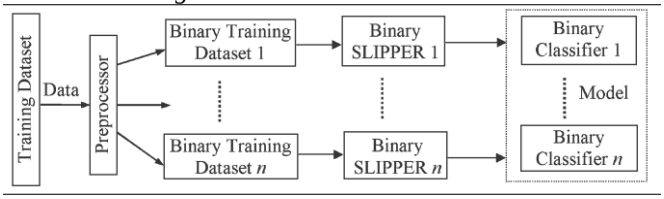

Fig. 1. Creation of initial model for didsm.

Preprocessor and the details of creating the prediction model have been described in [16]. The binary SLIPPER learning algorithm proposed by Cohen and Singer [17] is a general-purpose rule-learning system based on confidence-rate boosting [18]. A weak learner is boosted to find a single weak hypothesis (an IF–THEN rule), and then, the training data are reweighted for the next round of boosting. Unlike other conventional rule learners, data covered by learned rules are not removed from the training set. Such data are given lower weights in subsequent boosting rounds. All weak hypotheses from each round of boosting are compressed and simplified, and then combined into a strong hypothesis, constituting a binary classifier. An example of a binary classifier. This example is part of a binary classifier of the initial model in our system described below. Each rule starts with a predictive label. Followed by two parameters used to calculate the confidence in predictions made by this rule. The keyword "IF" introduces the conditions of the rule. These conditions are used to check whether the rule covers a data sample.

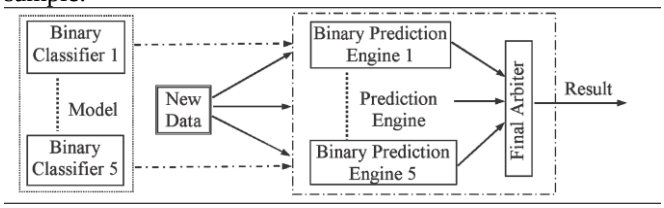

Fig. 2. Prediction on new data in didsm

In SLIPPER, an objective function such as (6) from [17] is used to search for a good rule with positive confidence during each round of boosting. The selected rule with positive confidence is compared with a default rule with negative confidence to determine the result of boosting. A default rule covers all data records and, thus, does not have conditions; all default rules are compressed into a single final default rule. For example, SLIPPER is a time-efficient learning algorithm. For example, it took 2 h to learn a model from roughly half million training records on a Pentium IV system with a 512-MB RAM running at 2.6 GHz.

4. ILLUSTRATION OF FEEDBACK SESSIONS

A. Prediction Engine

Binary learning algorithms can only build binary classifiers. For intrusion detection, the minimal requirement is to alarm in case intrusive activity is detected. Beyond alarms, operators expect that the IDS will report more details regarding possible attacks, at least the attack type. We group attacks into categories such as denial-of-service (dos), probing (probe), remote-to local (r2l), and user-to-root (u2r). Correspondingly, we constructed five binary classifiers from the training dataset. One binary classifier ("BC-Normal") predicts whether the input data record is normal. The other four binary classifiers ("BC-Probe," "BC-Dos," "BC-U2r," and "BC-R2l") predict whether the input data record constitutes a particular attack. For example, the binary classifier "BC-Probe" predicts whether the input data record is a probing attack. The prediction engine in our system consists of five binary prediction engines together with a final arbiter, as shown in Fig. 2. We refer to this multiclassifier version of SLIPPER as MC-SLIPPER. The training procedure used to construct the initial model for MC-SLIPPER is described in detail in [16]. Each binary prediction engine outputs a prediction result on the input data according to its binary classifier, and the final arbiter determines and reports the result to the system operator. The binary prediction engine is the same as the final hypothesis in SLIPPER [17],

B. Sensor Networks

The extensive number of research work in this area has appeared in the literature. Due to the limited energy budget available at sensor nodes, the primary issue is how to develop energy-efficient techniques to reduce communication and energy costs in the networks. Approximate-based data aggregation techniques have also been proposed. The idea is to tradeoff some data quality for improved energy efficiency. Silberstein et al. develop a sampling-based approach to evaluate approximate top-k queries in wireless sensor networks. Based on statistical modeling techniques, a model-driven approach was proposed in to balance the confidence of the query answer against the communication cost in the network. Moreover, continuous top-k queries for sensor networks have been studied in and. In addition, a distributed threshold join algorithm has been developed for top-k queries. These studies, considering no uncertain data, have a different focus from our study.

C. Data pruning

The cluster heads are responsible for generating uncertain data tuples from the collected raw sensor readings within their clusters. To answer a query, it's natural for the cluster heads to prune redundant uncertain data tuples before delivery to the base station in order to reduce communication and energy cost. The key issue here is how to derive a compact set of tuples essential for the base station to answer the probabilistic top-k queries.

D. Performance evaluation

In this section, we will examine how to assess the performance of an IDS and how to improve the system based on the experimental data. We will rely on the KDDCup'99 dataset provided by Defense Advanced Research Projects Agency (DARPA) as this dataset contains several weeks of attack data and has been used to assess the performance of a number of IDS. While this dataset contained labeled data, in order to mitigate the burden of manually labeling training data in real-life situations, we developed a supporting tool. We will use the total misclassification cost (TMC) as the primary indicator of system performance. In order to be able to improve our system based on the experimental results, we also develop a methodology of studying the performance of individual rules.

E. Dataset

A proper dataset must be obtained to facilitate experimentation. In our experimental environment, it was difficult to obtain real-life datasets due to limitations of network size and limited external access. Unfortunately, usable datasets are rarely published as these involve sensitive information such as the network architecture, security mechanisms, and so on. Thus, in this paper, we rely on the publicly available KDDCup'99 intrusion detection dataset. This dataset was collected from a network simulating a typical U.S. Air Force LAN and also reflects dynamic change within the network. The KDDCup'99 intrusion detection dataset was developed based on the 1998 DARPA intrusion detection evaluation program, prepared and managed by the MIT Lincoln Laboratories. The objective of this program was to survey and evaluate intrusion detection research. Lincoln Laboratories set up an environment to acquire nine weeks of raw TCP data for a local area network (LAN) simulating a typical U.S. Air Force LAN. This LAN was operated as if it is a true Air Force environment, and it was subjected to multiple attacks. The raw training data dump was about 4 GB of compressed binary TCP data from the first seven weeks of network traffic alone. The data dump was processed into roughly five million connection records. The test data were constructed from the network traffic in the last two weeks, which yielded around two million connection records. In the KDDCup'99 dataset, each record represents a TCP/IP network connection with a total of 41 features. Domain experts derived some of the features related to content [5]. Statistical features were generated using a 2-s time window. Five classes of connections were identified, including normal network connections. he four classes of abnormal connections (attacks) are dos, probing (probe), r2l, and u2r. Each attack class is further divided into subclasses. For example, class dos includes subclass smurf, neptune, back, teardrop, and so on, representing

5. ASSOCIATED WORK

In recent years, many works have been done to Here; we review representative work in the areas of 1) top-k Query processing in wireless sensor networks, and 2) top-k query processing on uncertain data. Top-k query processing in sensor networks. An extensive number of research works in this area has appeared in the literature [21], [24], [25], [26]). Due to the limited energy budget available at sensor nodes, the primary issue is how to develop energy-efficient techniques to reduce communication and energy costs in the networks. TAG [21] is one of the first studies in this area. By exploring the semantics of aggregate operators (e.g., sum, avg, and top-k), in-network processing approach is adopted to suppress redundant data transmissions in wireless sensor networks. Approximate-based data aggregation techniques have also been proposed [27], [25].

The idea is to tradeoff some data quality for improved energy efficiency. Silberstein et al. develop a sampling-based approach to evaluate approximate on statistical modeling techniques, a model-driven approach was proposed in [5] to balance the confidence of the query answer against the communication cost in the network. Moreover, continuous top-k queries for sensor networks have been studied in [28] and [29]. In addition, a distributed threshold join algorithm has been developed for top-k queries [24]. These studies, considering no uncertain data, have a different focus from our study. Top-k query processing on uncertain data. While research works on conventional top-k queries are mostly based on some deterministic scoring functions, the new factor of tuple membership probability in uncertain databases makes evaluation of probabilistic top-k queries very complicated since the top-k answer set depends not only on the ranking scores of candidate tuples but also their probabilities [8]. For uncertain databases, two interesting

top-k definitions (i.e., U-Topk and U-kRanks) and A_-like algorithms are proposed [17]. U-Topk returns a list of k tuples that has the highest probability to be in the top-k list over all possible worlds. U-kRanks returns a list of k tuples such that the ith record has the highest probability to be the ith best record in all possible worlds. In [13], PT-Topk query, which returns the set of tuples with a probability of at least p to be in the top-k lists in the possible worlds, is studied. Inspired by the concept of dominate set in the top-k query, an algorithm which avoids unfolding all possible worlds is given. Besides, a sampling method is developed to quickly compute an approximation with quality guarantee to the answer set by drawing a small sample of the uncertain data. In [19], the expected rank of each tuple across all possible worlds serves as the ranking function for finding the final answer. In [30], U-Topk and U-kRank queries are improved by exploiting their stop conditions. In [31], all existing top-k semantics have been unified by using generating functions. Recently, a study on processing top-k queries over a distributed uncertain database is reported in [14] and [23]. Li et al. [14] only support top-k queries with the expected ranking semantic. On the contrary, our proposal is a general approach which is applicable to probabilistic top-k queries with any semantic. Furthermore, instead of repeatedly requesting data which may last for several rounds, our protocols are guaranteed to be completed within no more than two rounds.

These differences uniquely differentiate our effort from [14]. Our previous work [23] as the initial attempt only includes the concept of sufficient set. In this paper, besides of sufficient set, we propose another important concept of necessary set. With the aid of these two concepts, we further develop a suite of algorithms, which show much better performance than the one in [23]. Probabilistic ranked queries based on uncertainty at the attribute level are studied in [32], [33], and [19]. A unique study that ranks tuples by their probabilities satisfying the query is presented in [12]. Finally, uncertain top-k query is studied under the setting of streaming databases where a compact data set is exploited to support efficient slide window top-k queries [18]. We will apply sufficient set and necessary set to sensor networks with tree topology, to further improve query processing performance by facilitating sophisticated in-network filtering at the intermediate nodes along the routing path to the root

6. EXPERIMENTAL SETUP AND RESULT

Table 1
Data distribution in the kddcup'99 dataset

	normal	probe	dos	u2r	r2l
Training Data	19.69%	0.83%	79.24%	0.01%	0.23%
Test Data	19.48%	1.34%	73.90%	0.07%	5.21%

Popular types of dos attacks. Two training datasets from the first seven weeks of network traffic are available. The full dataset includes about five million records and a smaller subset containing only 10% of the data records but with the same distribution as the full dataset. We used the 10% subset to train our binary SLIPPER classifiers. The labeled test dataset includes 311 029 records with a different distribution of attacks, then the training dataset (see Table I). Only 22 out of the 39 attack subclasses in the test data were present in the training data. The different distributions between the training and test datasets and the new types of attacks in the test dataset reflect the dynamic change of the nature of attacks common in real-life systems. Sabhnani and Serpen analyzed the dissimilarity between the training dataset and the test dataset [20].

A. System Performance

We evaluated our system on the KDDCup'99 dataset and compared it with other systems, including a system built from PNrules [22], the KDDCup'99 winner [23] and runner-up [24], a multiple-classifier system [25], and a system based on repeated incremental pruning to produce error reduction (RIPPER) [26]. More details on these systems can be found in Section V. An IDS generates alarms whenever it detects an attack while ignoring normal behavior. That is, any classification of network connection into an attack class will generate an alarm, while a classification as normal will not. Security officers will pay.

Table 2
Performance comparison of various ids

System Name	TMC	Overall Accuracy
MC-SLIPPER by BP Network [16]	68,490	92.70%
MC-SLIPPER by Confidence Ratio [16]	70,177	92.59%
Multiple Classifier System [25]	71,096	93.04%
MC-SLIPPER by Confidence [16]	72,494	92.06%
KDDCup'99 Winner [23]	72,500	92.71%
KDDCup'99 Runner-up [24]	73,287	92.92%
Simple RIPPER [26]	73,622	92.66%
PNrule [22]	74,058	92.59%

Table 3
Statistical data for mc-slipper rules on test dataset

Rule	P#	$l = 1$	$l = 2$	$l \geq 3$	$l \geq 100$	$l \geq 1000$	N#	T#	FPR
BC-Normal_29	17,335	21.80%	10.55%	67.66%	31.22%	22.43%	8	53,862	24.36%
BC-Normal_43	16,457	25.91%	15.59%	58.50%	3.01%	0.00%	41	45,999	26.40%
BC-Probe_17	1,471	6.19%	3.54%	90.28%	58.74%	0.00%	6	3,762	28.19%
BC-Probe_3	1,221	4.59%	4.10%	91.32%	70.84%	0.00%	22	2,845	30.41%
BC-Normal_50	2,384	13.76%	7.30%	78.94%	22.73%	0.00%	46	4143	36.97%
BC-Normal_22	10,141	17.00%	14.48%	68.52%	1.38%	0.00%	0	15,868	38.99%
BC-Normal_48	15,497	16.64%	11.74%	71.61%	17.22%	0.00%	13	23,965	39.29%
BC-Normal_49	13,805	12.85%	9.71%	77.44%	42.99%	38.22%	20	16,738	45.23%
BC-Probe_4	10,909	10.63%	8.78%	80.58%	34.11%	0.00%	0	12,218	47.17%
BC-Normal_28	9,179	7.31%	6.69%	86.00%	6.58%	0.00%	2	10,072	47.69%
BC-Normal_33	12,280	10.04%	9.90%	80.06%	2.96%	0.00%	1	12,223	50.12%
BC-Normal_45	5,018	2.21%	1.95%	95.83%	38.36%	0.00%	6	4,313	53.81%
BC-Normal_32	4,824	0.15%	0.00%	99.85%	99.85%	99.85%	3	3,164	60.41%
BC-Normal_11	4,875	0.04%	0.00%	99.96%	99.96%	99.96%	0	1,894	72.02%
BC-Probe_12	11,109	1.52%	1.73%	96.75%	59.84%	0.00%	0	2,787	79.94%
BC-Probe_16	40,917	0.08%	0.04%	99.88%	99.19%	87.17%	108	3,032	93.12%
BC-Probe_5	40,519	0.06%	0.03%	99.91%	99.53%	94.86%	0	2,368	94.48%
BC-Probe_7	40,705	0.02%	0.00%	99.97%	99.90%	99.19%	0	2,117	95.06%
BC-Normal_27	3,647	0.05%	0.11%	99.84%	96.22%	69.07%	0	22	99.40%

different levels of attention to different types of alarms. For example, probe alarms will typically be ignored, but an r2l alarm will be investigated and will result in counter measures, should it be determined to be true. To reflect the different levels of seriousness of alarms, a misclassification cost matrix defining the cost of each type of misclassification is used to evaluate the results of the KDDCup'99 competition. The TMC, which is the sum of the misclassification costs for all test data records, is the key measurement differentiating the performance of each system. Table II compares the performance of each evaluated system. We examine three systems based on MCSLIPPER, considering the three arbitration strategies [16]. The TMC of our systems is 72 494, 70 177, and 68 490, respectively. All three are better than the KDDCup'99 contest winner, whose TMC is 72 500 [27]. MC-SLIPPER using BP neural network arbitration shows the best performance among the compared systems.

As shown in Table III, the MC-SLIPPER systems achieved excellent TMC. However, TMC does not tell us how well each rule performs on the test dataset. To assess system performance along this dimension, we first extract the sequence of prediction results for each rule from the experimental data. Table III shows the statistical data for individual rules from different binary classifiers. For space reasons, we only show data for those 19 rules whose false prediction rates are greater than 20% and which cover more than 1% of all test data. These 19 rules contribute to 262 193 out of the 299 471 false positive predictions (87.55%) and have the biggest negative impact on the overall performance of our MC-SLIPPER system when tested on the KDDCup'99 test dataset to which a rule contributes. Similarly, columns "N#" and "T#" show the false negative predictions and true predictions for each rule, respectively.

As can be seen from this table, the number of false negative predictions is small compared to the number of false positive predictions. The last column titled "FPR" is the overall false prediction rate, computed by (P# + N#)/(P# + N# + T#). The *l* in the column titles refers to the number of successive false positive predictions. When *l* is equal to one, this false prediction is an isolated false prediction. We particularly examine the situations where long sequences of false positive predictions occur. of rules in MC-SLIPPER. *Property 1:* Isolated false predictions exist for most rules and can amount to about 25% of all false predictions (see column titled "$l = 1$"). *Property 2:* Often, false predictions come in long successive prediction sequences (see the columns titled "$l \geq 3$," "$l \geq 100$," and "$l \geq 1000$").

Long successive false prediction sequences provide an opportunity for our system to benefit from model tuning. Pattern where the length of successive false positive prediction sequences *l* is greater than or equal to three. Therefore, we cannot apply (9) to evaluate the benefit of tuning. Compared to the large number of false positive predictions shown in column "P#," the small number of false negative predictions shown in column "N#" can safely be ignored. To estimate the benefit of tuning from the data in Table III.

```
LOOP
  INPUT Testdata (InputData):
    FinalResult = Predict (InputData);
    Feedback = Verify_Predition (InputData, FinalResult);
    IF ( is_false_ prediction (Feedback) AND should_tune_model () )
      Tune_Model (Feedback, FinalResult);
END
```

Pseudo code for DIDSM with full and instant tuning.

7. RELATED WORK

Sabhnani and Serpen [25] built a multiclassifier system using Multi-layer perceptions, K-means clustering, and a Gaussian classifier after evaluating the performance of a comprehensive set of pattern recognition and machine learning algorithms on the KDDCup'99 dataset. The TMC of this multi classifier system is 71 096, and the cost per example is 0.2285. However, the significant drawback of their system is that the multiclassifier model was built based on the performance of different sub classifiers on the test dataset. Giacinto *et al.* [28] proposed a multiclassifier system for intrusion detection based on distinct feature representations: content, intrinsic, and traffic features were used to train three different classifiers, and a decision fusion function was used to generate the final prediction. The cost per example is 0.2254. No confusion matrix of the prediction is reported in their study. Kumar [26] applied RIPPER to the KDDCup'99 dataset. RIPPER is an optimized version of incremental reduced error pruning (IREP), which is a rule-learning algorithm optimized to reduce errors on large datasets. The TMC is 73 622, and the cost per example is 0.2367. Agawam and Joshi [22] proposed an improved two stage general-to specific framework (PNrule) for learning a rule-based model. PNrule balances support and accuracy when inferring rules from its training dataset to overcome the problem of small disjoints. For multiclass classification, a cost-sensitive scoring algorithm was developed to resolve conflicts between multiple classifiers using a misclassification cost matrix, and the final prediction was determined according to Bayes optimality rule. The TMC is 74 058, and the cost per example is 0.2381 when tested on KDDCup'99 dataset. Pfahringer constructed an ensemble of 50 × 10 C5 decision trees as a final predictor using a cost-sensitive bagged boosting algorithm [23]. The final prediction was made according to minimal conditional risk, which is a sum of error cost by class probabilities. This predictor won the KDDCup'99 contest. The TMC is 72 500, and the cost per example is 0.2331. Levin's kernel miner [24] is based on building the optimal decision forest. A global optimization criterion was used to minimize the value of the multiple estimators, including the TMCs. The tool placed second in the KDDCup'99 contest. The TMC is 73 287 and the cost per example is 0.2356.There are two approaches in updating the detection model: Add a sub model or supersede the current model. Lee *et al.* [9] proposed a "plug-in" method as a temporary solution. When

new intrusion emerges, a simple special-purpose classifier is trained to detect only the new attack. The new classifier is plugged into the existing IDS to enable detection of the new attack. The main existing detection models remain unchanged. When a better model or a single model that can detect the new intrusion as well as the old intrusions is available later, the temporal model can be replaced. This method takes advantage of the fact that training a new specific classifier is significantly faster than retraining a monolithic model from all data, and thus, it enables detection of new attacks as soon as possible. However, before the new classifier can be trained, high-quality training data should be collected. For a very new attack, it is not an easy task to collect the appropriate training data. Training then becomes the job of the system operators who usually lack the knowledge to train a model. Having the newly mined model supersede the current detection model was presented in various systems [10]–[12].

The study in [10] and [11] proposed architecture to implement adaptive model generation. In this architecture, different detection model generation algorithms have been developed to mine the new model on real-time data. The new model can supersede the current model on-the-fly. The study in [12] deployed incremental mining to develop new models on real-time data and update the detection profile in adaptive IDS. The profile (model) for the activity during an overlapping sliding window is incrementally mined, and the similarity between the recent and base profiles is evaluated. If the similarity stays above a threshold level, the base profile is taken to be a correct reflection of the current activities. When the similarity falls below the threshold, the rate of change is examined. If the change is abrupt, it is interpreted as an intrusion. The base profile will not be updated. Otherwise, it is treated as a normal change, and the base profile will be updated. However, this system cannot deal with situations where both intrusive and normal behavior changes occur within the sliding window. Because those models are mined on real-time data, an experienced attacker could train the model gradually to accept intrusive activity as normal.

8. CONCLUSION

Because computer networks are continuously changing, it is difficult to collect high-quality training data to build intrusion detection models. In this paper, rather than focusing on building a highly effective initial detection model, we propose to improve a detection model dynamically after the model is deployed when it is exposed to new data. In our approach, the detection performance is fed back into the detection model, and the model is adaptively tuned. To simplify the tuning procedure, we represent the detection model in the form of rule sets, which are easily understood and controlled; tuning amounts to adjusting confidence values associated with each rule. This approach is simple yet effective. Our experimental results show that the TMC of DIDSM with full and instant tuning drops about35% from the cost of the MC-SLIPPER system with a fixed detection model. If only 10% false predictions are used to tune the model, the system still achieves about 30% performance improvement. When tuning is delayed by only a short time, the system achieves 20% improvement when only 1.3% false predictions are used to tune the model. ATIDS imposes a relatively small burden on the system operator: operators need to mark the false alarms after they identify them.

REFERENCES

[1] D. Barbara, J. Couto, S. Jajodia, L. Popyack, and N.Wu, "ADAM: Detecting intrusions by data mining," in *Proc. IEEE Workshop Inf. Assurance and Security*, Jun. 2001, pp. 11–16.

[2] N. Ye, S. Emran, X. Li, and Q. Chen, "Statistical process control for computer intrusion detection," in *Proc. DISCEX II*, Jun. 2001, vol. 1, pp. 3–14.

[3] N. Ye, S. Vilbert, and Q. Chen, "Computer intrusion detection through EWMA for auto correlated and uncorrelated data," *IEEE Trans. Rel.*, 52, no. 1, pp. 75–82, Mar. 2003.

[4] N. Ye, S. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis audit trails for host-based intrusion detection," *IEEE Trans. Comput.*, vol. 51, no. 7, pp. 810–820, Jul. 2002.

[5] W. Lee and S. Stolfo, "A framework for constructing features and models for intrusion detection systems," *ACMTrans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 227–261, Nov. 2000.

[6] L. Ertoz, E. Eilertson, A. Lazarevic, P. Tan, J. Srivastava, V. Kumar, and P. Dokas, *The MINDS—Minnesota Intrusion Detection System: Next Generation Data Mining*. Cambridge, MA: MIT Press, 2004.

[7] K. Julish, "Data mining for intrusion detection: A critical review," IBM, Kluwer, Boston, MA, Res. Rep. RZ 3398, Feb. 2002. No. 93450.

[8] I. Dubrawsky and R. Saville, *SAFE: IDS Deployment, Tuning, and Logging in Depth*, CISCO SAFE White Paper.[Online]. Available: http://www.cisco.com/go/safe

[9] W. Lee, S. Stolfo, and P. Chan, "Real time data mining-based intrusion detection," in *Proc. DISCEX II*, Jun. 2001, pp. 89–100.

[10] E. Eskin, M. Miller, Z. Zhong, G. Yi, W. Lee, and S. Stolfo, "Adaptive model generation for intrusion detection systems," in *Proc. 7th ACM Conf. Comput. Security Workshop Intrusion Detection and Prevention*, Nov. 2000. [Online]. Available: http://www1.cs.columbia.edu/ids/ publications/adaptive-ccsids00.pdf

[11] A. Honig, A. Howard, E. Eskin, and S. Stolfo, "Adaptive model generation: An architecture for the deployment of data mining-based intrusion detection systems," in *Data Mining for Security Applications*. Norwell, MA: Kluwer, 2002.

[12] M. Hossian and S. Bridges, "A framework for an adaptive intrusion detection system with data mining," in *Proc. 13th Annu. CITSS*, Jun. 2001. [Online]. Available: http://www.cs.msstate.edu/~bridges/papers/citss-

2001.pdf

[13] X. Li and N. Ye, "Decision tree classifiers for computer intrusion detection," *J. Parallel Distrib. Comput. Prac.* vol. 4, no. 2, pp. 179–180, 2003.
[14] J. Ryan, M. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," in *Proc. Advances NIPS 10*, Denver, CO, 1997, pp. 943–949.
[15] S. Kumar and E. Spafford, "A pattern matching model for misuse intrusion detection," in *Proc. 17th Nat. Comput. Security Conf.*, 1994, pp. 11–21.
[16] Z. Yu and J. Tsai, "A multi-class SLIPPER system for intrusion detection," in *Proc. 28th IEEE Annu. Int. COMPSAC*, Sep. 2004, pp. 212–217.
[17] W. Cohen and Y. Singer, "A simple, fast, and effective rule learner," in *Proc. Annu. Conf. Amer. Assoc. Artif. Intell.*, 1999, pp. 335–342.
[18] S. Robert and S. Yoram, "Improved boosting algorithms using confidence rated predictions," *Mach. Learn.*, vol. 37, no. 3, pp. 297–336, Dec. 1999.
[19] L. Faussett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1994

[20] M. Sabhnani and G. Serpen, "Why machine learning algorithms fail in misuse detection on KDD intrusion detection data set," *Intel. Data Anal.*, vol. 8, no. 4, pp. 403–415, 2004.
[21] T. Kohonen, *Self-Organizing Maps*. New York: Springer Verlag, 1997.
[22] R. Agarwal and M. Joshi, "PNrule: A new framework for learning classifier models in data mining (a case-study in network intrusion detection)," in Proc. 1st SIAM Conf. Data Mining, Apr. 2001. [Online]. Available: http://www.siam.org/meetings/sdm01/pdf/sdm01_30.pdf
[23] B. Pfahringer, "Winning the KDD99 classification cup: Bagged boosting," *ACM SIGKDD Explore.*, vol. 1, no. 2, pp. 65–66, 1999.
[24] I. Levin, "KDD-99 classifier learning contest LLSoft's results overview," *ACM SIGKDD Explore.*, vol. 1, no. 2, pp. 67–75, 1999.
[25] M. Sabhnani and G. Serpen, "Application of machine learning algorithms

