ADWICE - Anomaly Detection with Real-time Incremental Clustering*

Kalle Burbeck and Simin Nadjm-Tehrani

Department of Computer and Information Science
Linköpings universitet SE–581 83 Linköping, Sweden
{kalbu, simin}@ida.liu.se

Abstract. Anomaly detection, detection of deviations from what is considered normal, is an important complement to misuse detection based on attack signatures. Anomaly detection in real-time places hard requirements on the algorithms used, making many proposed data mining techniques less suitable. ADWICE (Anomaly Detection With fast Incremental Clustering) uses the first phase of the existing BIRCH clustering framework to implement fast, scalable and adaptive anomaly detection. We extend the original clustering algorithm and apply the resulting detection mechanism for analysis of data from IP networks. The performance is demonstrated on the KDD data set as well as on data from a test network at a telecom company. Our experiments show a good detection quality (95 %) and acceptable false positives rate (2.8 %) considering the online, real-time characteristics of the algorithm. The number of alarms is then further reduced by application of the aggregation techniques implemented in the Safeguard architecture.

1 Introduction

The threats to computer-based systems on which we are all dependent are ever increasing, thereby increasing the need for technology to handle those threats. One study[1] estimates that the number of intrusion attempts over the entire Internet is in the order of 25 billion each day and increasing. McHugh[2] claims that the attacks are getting more and more sophisticated while they get more automated and thus the skills needed to launch them are reduced. Intrusion Detection Systems (IDS) attempt to respond to this trend by applying knowledge-based techniques (typically realised as signature-based misuse detection), or behaviour-based techniques (e.g. by applying machine learning for detection of anomalies).

Due to increasing complexity of the intrusion detection task, use of many IDS sensors to increase coverage and the need for improved usability of intrusion

* This work was supported by the European project Safeguard IST–2001–32685. We would like to thank Thomas Dagonnier, Tomas Lingvall and Stefan Burschka at Swisscom for fruitful discussions and their many months of work with the test network. The Safeguard agent architecture has been developed with the input from all the research nodes of the project, the cooperation of whom is gratefully acknowledged.

detection, a recent trend is alert or event correlation[3]. Correlation combines information from multiple sources to improve information quality. By correlation the strength of different types of detection schemes may be combined, and weaknesses compensated for.

The main detection scheme of most commercial intrusion detection systems is misuse detection, where known bad behaviour (attacks) are encoded into signatures. Misuse detection can only detect attacks that are well known and for which signatures have been written. In anomaly detection normal (good) behaviour of users or the protected system is modelled, often using machine learning or data mining techniques. During detection new data is matched against the normality model, and deviations are marked as anomalies. Since no knowledge of attacks is needed to train the normality model, anomaly detection may detect previously unknown attacks.

Anomaly detection still faces many challenges, where one of the most important is the relatively high rate of false alarms (false positives). We argue that the usefulness of anomaly detection is increased if combined with further aggregation, correlation and analysis of alarms[4], thereby minimizing the number of false alarms propagated to the administrator and helping to further diagnose the anomaly. In this paper we explain the role of anomaly detection in a distributed architecture for agents that has been developed within the European Safeguard project[5].

We apply clustering as the technique for training of the normality model, where similar data points are grouped together into clusters using a distance function. Clustering is suitable for anomaly detection, since no knowledge of the attack classes is needed whilst training. Contrast this to other learning approaches, e.g. classification, where the classification algorithm needs to be presented with both normal and known attack data to be able to separate those classes during detection. Our approach to anomaly detection, ADWICE (Anomaly Detection With fast Incremental Clustering), is an adaptive scheme based on the BIRCH clustering algorithm[6]. BIRCH has previously been used in applications such as web mining of user sessions on web-pages[7] but to our knowledge there has previously been no extensions of the algorithm for intrusion detection. We proceed by comparing our work with related research and point out the advantages of ADWICE. In section 3 we present the Safeguard agent architecture where the clustering based anomaly detection fits in. In section 4 we describe the anomaly detection algorithm. Evaluation of the algorithms is presented in section 5 followed by a concluding discussion in section 6.

2 Motivation

2.1 IDS Data Problems and Dependencies

One fundamental problem of intrusion detection research is the limited availability of good data to be used for evaluation. Producing intrusion detection data is a labour intensive and complex task involving generation of normal system

data as well as attacks, and labelling the data to make evaluation possible. If a real network is used, the problem of producing good normal data is reduced, but then the data may be too sensitive to be released in public.

For learning based methods, good data is not only necessary for evaluation and testing, but also for training. Thus applying a learning based method in the real world, puts even harder requirements on the data. The data used for training need to be representative to the network where the learning based method will be applied, possibly requiring generation of new data for each deployment. *Classification based methods*[8, 9], or supervised learning, require training data that contains normal data as well as good representatives of those attacks that should be detected to be able to separate attacks from normality. Complete coverage of even known and recent attacks would be a daunting task indeed due to the abundance of attacks encountered globally. Even worse, the attacks in the training data set need to be labelled with the attack class or classes.

Clustering, or unsupervised learning, has attracted some interest[10–13] in the context of intrusion detection. The interesting feature of clustering is the possibility to learn without knowledge of attack classes, thereby reducing training data requirement, and possibly making clustering based techniques more viable than classification-based techniques in a real world setting. There exist at least two approaches.

When doing *unsupervised anomaly detection* a model based on clusters of data is trained using unlabelled data, normal as well as attacks. The assumption is that the relative amount of attacks in the training data is very small compared to normal data, a reasonable assumption that may or may not hold in the real world context for which it is applied. If this assumption holds, anomalies and attacks may be detected based on cluster sizes. Large clusters correspond to normal data, and small clusters possibly correspond to attacks. A number of unsupervised detection schemes have been evaluated on the KDD data set with varying success[10–12]. The accuracy is however relatively low which reduces the direct applicability in a real network.

In the second approach, which we denote simply *(pure) anomaly detection* in this paper, training data is assumed to consist only of normal data. Munson and Wimer[13] used a cluster based model (Watcher) to protect a real web server, proving anomaly detection based on clustering to be useful in real life.

Acceptable accuracy of the unsupervised anomaly detection scheme may be very hard to obtain, even though the idea is very attractive. Pure anomaly detection, with more knowledge of data used for training, may be able provide better accuracy than the unsupervised approach. Pure anomaly detection also avoids the coverage problem of classification techniques, and requires no labelling of training data similar to unsupervised anomaly detection. Generating training data in a highly controlled network now simply consists of generating normal data. This is the approach adopted in this paper, and the normality of the training data in our case is ensured by access to a large test network build specifically for experimental purposes in the Safeguard project.

In a real live network with connection to Internet, data can never be assumed to be free of attacks. Pure anomaly detection also works when some attacks are included in the training data, but those attacks will during detection be considered normal and therefore not detected. To increase detection coverage, attacks should be removed to as large an extent as possible, making coverage a trade-off with data cleaning effort. An efficient approach should be to use existing misuse detectors with updated rule-bases in the preparatory phase, to reduce costly human effort. Updated signature based systems should with high probability detect many of the currently known attacks, simplifying removal of most attacks in training data. A possibly complementary approach is to train temporary models on different data sets and let them vote on normality to decide what data to use for the final normality model.

Certain attacks, such as Denial of Service and scanning can produce large amounts of attack data. On the other hand, some normal types of system activities might produce limited amounts of data, but still be desirable to incorporate into the detection model. Those two cases falsify the assumption of unsupervised anomaly detection and need to be handled separately. Pure anomaly detection such as ADWICE does not have those problems since detection is not based on cluster sizes.

2.2 IDS Management Effort

One of the inherent problems of anomaly detection is the false positives rate. In most settings normality is not easy to capture. Normality changes constantly, due to changing user behaviour as well as hardware or software changes. An algorithm that can perfectly capture normality of static test data, will therefore not necessarily work well in a real life setting with changing normality. The anomaly detection model needs to be adaptable. When possible, and if security policy allows, it should be autonomously adaptive to minimize the human effort. In other cases an administrator needs to be able to update the anomaly model with simple means, without destroying what is already learnt. And the effort spent updating the model should be minimal compared to the effort of training the initial model. ADWICE is incremental, supporting easy adaptation and extension of the normality model.

2.3 Scalability and Performance Issues

For critical infrastructures or valuable company computer based assets it is important that intrusions are detected in real-time with minimal time-to-detection to minimize the consequences of the intrusion. An intrusion detection system in a real-time environment needs to be fast enough to cope with the information flow, have explicit limits on resource usage and also needs to adapt to changes in the protected network in real-time.

Many proposed clustering techniques require quadratic time for training[14], making real-time adaptation of a cluster-based model hard. This implies that most clustering-based approaches would require time consuming off-line training

to update the model. They may also not be scalable, requiring all training data to be kept in main memory during training, limiting the size of the trained model.

- ADWICE is scalable, since only compact summeries of clusters are kept im memory rather then the complete data set.
- ADWICE has good performance due to local clustering and an integrated tree index for searching the model.

3 The Safeguard Context

Safeguard is a European research project aiming to enhance survivability of critical infrastructures by using agent technology. The Safeguard agent architecture is presented in Fig. 1. This architecture is evaluated in the context of telecommunication and energy distribution networks. The agents should improve survivability of those large complex critical infrastructures (LCCI:s), by detecting and handling intrusions as well as faults in the protected systems. The key to a generic solution applicable in many infrastructures is in the definition of roles for various agents. There may be several instances of each agent in each LCCI. The agents run on a platform (middleware) that provides generic services such as discovery and messaging services. These are believed to be common for the defence of many infrastructures, but should be instantiated to more specific roles in each domain. The generic roles can be described as follows:

- *Wrapper agents* wrap standard INFOSEC devices and existing LCCI diagnosis mechanisms, and provide their outputs after some filtering and normalisation for use by other agents.
- *Topology agents* gather dynamic network topology information, e.g. host types, operating system types, services provided, known vulnerabilities.
- *Hybrid detector agents* utilise domain knowledge for a given infrastructure, but combine it with behavioural detection mechanisms (e.g. anomaly detection with white lists).
- *Correlation agents* identify problems that are difficult to diagnose with one source of information in the network, by using several sources of information from wrapper, topology, or hybrid detector agents. They use the data sources to order, filter and focus on certain alarms, or predict reduced availability of network critical services. One type of correlation agent performs adaptive filtering and aggregation to further reduce the alarm rates.
- *Action agents* enable automatic and semi automatic responses when evaluation of a problem is finished.
- *Negotiation agents* communicate with agents in other LCCI:s to request services and pass on information about major security alarms.
- *HMI (Human-Machine Interface) agents* provide an appropriate interface, including overview, for one or many system operators.
- *Actuator agents* are wrappers for interacting with lower layer software and hardware (e.g. changing firewall rules).

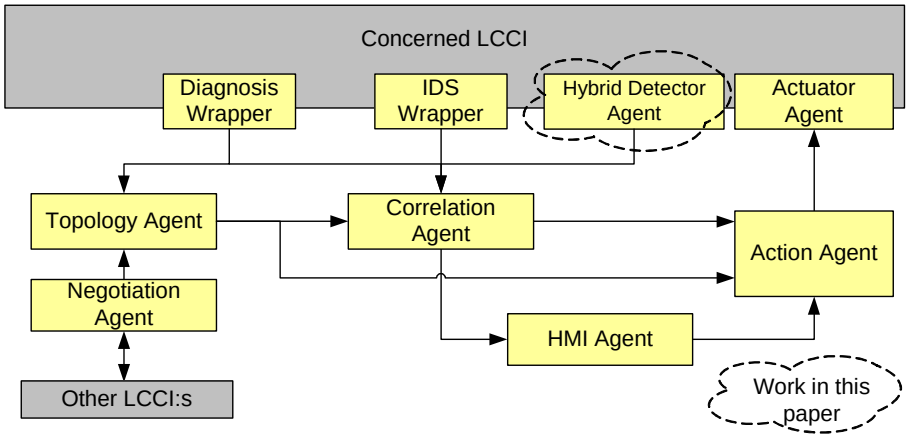

Fig. 1. The Safeguard agent architecture

In the context of a management network for telecom service providers we have identified the following needs:

- Reducing information overload ([4] and section 5.5 on aggregation)
- Increasing coverage by providing new sources of information (this paper)
- Increasing information quality by reducing false positives[4]
- Collating information, such as correlating alarms[4] and combining with topology information
- Presenting a global view of a network (in Safeguard Demonstrator)

In this paper we describe the anomaly detection engine for an instance of the Hybrid detection agent. The agent combines a clustering based anomaly detection engine (ADWICE) with a white-list engine. The white-list engine implements simple specification based intrusion detection[15] where data known to be normal are described by manually constructed signatures. In our case hosts and services known to produce abnormal behaviour (e.g. DNS server port 53)

are filtered away, but rules for arbitrary features can be used. Data considered normal by the white-list engine are not fed into ADWICE. This reduces the size of the normality model without decreasing detection coverage.

4 The Anomaly Detection Algorithm

This section describes how ADWICE handles training and detection. The basis of ADWICE, the BIRCH clustering algorithm, requires data to be numeric. Non-numeric data is therefore assumed to be transformed into numeric format by pre-processing.

4.1 Basic Concepts

The basic concepts are presented in the original BIRCH paper[6] and the relevant parts are summarized here.

Given n d-dimensional data vectors $\boldsymbol{v_i}$ in a cluster $CFj = \{\boldsymbol{v_i}|i = 1 \ldots n\}$ the centroid $\boldsymbol{v_0}$ and radius $R(CF_j)$ are defined as:

$$\boldsymbol{v_0} = \frac{\sum_{i=1}^{n} \boldsymbol{v_i}}{n} \qquad R\left(CF_j\right) = \sqrt{\frac{\sum_{i=1}^{n} \left(\boldsymbol{v_i} - \boldsymbol{v_0}\right)^2}{n}} \tag{1}$$

R is the average distance from member points in the cluster to the centroid and is a measure of the tightness of the cluster around the centroid.

A fundamental idea of BIRCH is to store only condensed information, denoted *cluster feature*, instead of all data points of a cluster. A cluster feature is a triple $CF = (n, \boldsymbol{S}, SS)$ where n is the number of data points in the cluster, $\boldsymbol{S}$ is the linear sum of the n data points and SS is the square sum of all data points. Given the CF for one cluster, centroid $\boldsymbol{v_0}$ and radius R may be computed. The distance between a data point $\boldsymbol{v_i}$ and a cluster CF_j is the Euclidian distance between $\boldsymbol{v_i}$ and the centroid, denoted $D(\boldsymbol{v_i}, CF_j)$ while the distance between two clusters CF_i and CF_j is the Euclidian distance between their centroids, denoted $D(CF_i, CF_j)$. If two clusters $CF_i = (n_i, \boldsymbol{S_i}, SS_i)$ and $CF_j = (n_j, \boldsymbol{S_j}, SS_j)$ are merged, the CF of the resulting cluster may be computed as $(n_i + n_j, \boldsymbol{S_i} + \boldsymbol{S_j}, SS_i + SS_j)$. This also holds if one of the CF:s is only one data point making incremental update of CF:s possible.

A CF tree is a height balanced tree with three parameters, branching factor (B), threshold (T), and maximum number of clusters (M). A leaf node contains at most B entries, each of the form (CF_i) where $i \in \{1, \ldots, B\}$. Each CF_i of the leaf node must satisfy a threshold requirement (TR) with respect to the threshold value T. Two different threshold requirements have been evaluated with ADWICE. The first threshold requirement where $R(CF_i) \leq T$ corresponds to a threshold requirement suggested in the original paper and is therefore used as base line in this work (ADWICE–TRR). A large cluster may absorb a small group of data points located relatively far from the cluster centre. This small group of data points may be better represented by their own cluster since detection is based on distances. A second threshold requirement was therefore

evaluated where $D(\boldsymbol{v_i}, CF_i) \leq T$ was used as decision criteria ($\boldsymbol{v_i}$ is the new data point to be incorporated into the cluster). This version of the algorithm will be referred to as ADWICE–TRD.

Each non-leaf node contains at most B entries of the form (CF_i, child$_i$), where $i \in \{1, \ldots, B\}$ and child$_i$ is a pointer to the node's i-th child. Each CF at non-leaf level summarises all child CF:s in the level below.

4.2 Training

The CF tree is the normality model of the anomaly detection algorithm. During training, each data vector $\boldsymbol{v}$ is inserted into the CF-tree incrementally following the steps described below:

1. *Search for closest leaf*: Recursively descend from the root to the closest leaf, by in each step choosing child i such that $D(\boldsymbol{v}, CF_i) < D(\boldsymbol{v}, CF_j)$ for every other child j.
2. *Update the leaf*: Find closest CF_i by computing $D(\boldsymbol{v}, CF_j)$ for all CF_j in the leaf. If CF_i may absorb $\boldsymbol{v}$ without violating the threshold requirement (TR) update CF_i to include $\boldsymbol{v}$. If TR is violated, create a new CF_k entry in the leaf out of $\boldsymbol{v}$. If the number of CF:s including CF_k is below B we are done. Otherwise the leaf needs to be split in two. During splitting, the two farthest CF:s of the leaf are selected as seeds and all other CF_k from the old leaf are distributed between the two new leafs. Each CF_k is merged with the leaf with the closest seed.
3. *Modify the path to the leaf*: After an insert, the tree needs to be updated. In the absence of split, the CF:s along the paths to the updated leaf need to be recomputed to include $\boldsymbol{v}$ by incrementally updating the CF:s. If a split occurred, we need to insert a new non-leaf entry in the parent node of the two new leafs and re-compute the CF summary for the new leafs. If there is free space in the parent node (i.e. the number of children is below B) the new non-leaf CF is inserted. Otherwise the parent is split in turn. Splitting may proceed all the way up to the root in which case the depth of the tree increases when a new root is inserted.

If the size of the tree increases so that the number of nodes is larger than M, the tree needs to be rebuilt. The threshold T is increased, all CF:s at leaf level are collected and inserted anew into the tree. Now it is not single data points that are inserted but rather CF:s. Since T has been increased, old clusters may be merged thereby reducing the size of the tree. If the increase of T is too small, a new rebuild of the tree may be needed to reduce the size below M again. A heuristic described in the original BIRCH paper may be used for increasing the threshold to minimize the number of rebuilds, but in this work we use a simple constant to increase T conservatively (to avoid influencing the result by the heuristic).

Of the three parameters T, B and M the threshold T is the simplest to set, as it may be initialised to zero. The branching factor B influences the training and

detection time but may also influence detection accuracy. The original paper suggests using a branching factor of 15, but of course they do not consider anomaly detection accuracy since the original algorithm is not used for this purpose.

The M parameter needs to be decided using experiments. Since it is only an upper bound of the number of clusters produced by the algorithm it is easier to set than an exact number of clusters as required by other clustering algorithms. As M limits the size of the CF-tree it is an upper bound on the memory usage of ADWICE. Note that in general M needs to be set much lower than the number of data represented by the normality model to avoid over-fitting (i.e. training a model which is very good when tested with the training data but fails to produce good results when tested with data that differs from the training data). M also needs to be set high enough so that the number of clusters is enough for representing normality.

4.3 Detection

When a normality model is trained, it may be used to detect anomalies in unknown data. When a new data point $\boldsymbol{v}$ arrives detection starts with a top down search from the root to find the closest cluster feature CF_i. This search is performed in the same way as during training. When the search is done, the distance $D(\boldsymbol{v}, CF_i)$ from the centroid of the cluster to the new data point $\boldsymbol{v}$ is computed. Informally, if D is small, i.e. lower than a limit, $\boldsymbol{v}$ is similar to data included in the normality model and $\boldsymbol{v}$ should therefore be considered normal. If D is large, $\boldsymbol{v}$ is an anomaly.

Let the threshold T be the limit (L) used for detection. Using two parameters E_1 and E_2, $MaxL = E_1 * L$ and $MinL = E_2 * L$ may be computed. Then we compute the belief that $\boldsymbol{v}$ is anomalous using the formula below:

$$belief = \begin{cases} 0 & \text{if } D \leq MinL \\ 1 & \text{if } D \geq MaxL \\ \frac{D-MinL}{MaxL-MinL} & \text{if } MinL < D < MaxL \end{cases} \quad (2)$$

A belief threshold (BT) is then used to make the final decision. If we consider $\boldsymbol{v}$ anomalous and raise an alarm. The belief threshold may be used by the administrator to change the sensitivity of the anomaly detection. For the rest of the paper to simplify the evaluation we set $E_1 = E_2 = E$ so that $\boldsymbol{v}$ is anomalous if and only if $D > MaxL$. Note that clusters are spherical but the area used for detection of multiple clusters may overlap, implying that the clusters may be used to represent also non-spherical regions of normality. Time complexity of testing as well as of training is in ordo $N \log C$ where N is the number of processed data and C is the number if clusters in the model.

4.4 Adaptation of the Normality Model

As described earlier, agents need to be adaptable in order to cope with varying LCCI conditions including changing normality. Here we describe two scenarios

in which it is very useful to have an incremental algorithm in order to adapt to changing normality.

In some settings, it may be useful to let the normality model relearn autonomously. If normality drifts slowly, an incremental clustering algorithm may handle this in real-time during detection by incorporating every test data classified as normal with a certain confidence into the normality model. If slower drift of normality is required, a random subset of encountered data based on sampling could be incorporated into the normality model.

Even if autonomous relearning is not allowed in a specific network setting there is need for model adaptation. Imagine that the ADWICE normality model has been trained, and is producing good results for a specific network during detection. At this point in time the administrator recognizes that normality has changed and a new class of data needs to be included as normal. Otherwise this new normality class produces false positives. Due to the incremental property, the administrator can incorporate this new class without the need to relearn the existing working normality model. Note that there is no need for retraining the complete model or to take the model off-line. The administrator may interleave incremental training of data from the new normality class with detection.

5 Evaluation

In all following experiments ADWICE-TRD is used unless otherwise stated.

5.1 Data set

Performing attacks in real networks to evaluate on-line anomaly detection is not realistic and our work therefore shares the weaknesses of evaluation in somewhat "unrealistic" settings with other published research work in the area. Our approach for dealing this somewhat synthetic situation is as follows. We use KDDCUP99 data set[16] to test the real-time properties of the algorithm. Having a large number of attack types and a large number of features to consider can thus work as a proof of concept for the distinguishing attributes of the algorithm (unknown attacks, fast on-line, incremental model building). We then go on to evaluate the algorithm in a test network that has been specifically built with the aim of emulating a realistic telecom management network. While large number of future tests with different criteria are still possible on this test network, this initial set of tests illustrates the sort of problems that are detected by ADWICE and not covered by current commercial INFOSEC devices deployed on the emulated network.

Despite the shortcomings of the DARPA related datasets[16] (see also section 6) they have been used in at least twenty research papers and are unfortunately currently the only openly available data set commonly used data for comparison purposes. The original KDD training data set consists of almost five millions session records, where each session record consists of 41 fields (e.g. IP flags set, service, content based features, traffic statistics) summarizing a TCP session or

UDP connection. Since ADWICE assumes all training data is normal, attack data are removed from the KDD training data set and only the resulting normal data (972 781 records) are used for training. All 41 fields of the normal data are considered by ADWICE to build the model.

The testing data set consists of 311 029 session records of which 60 593 is normal and the other 250 436 records belong to 37 different attack types ranging from IP sweeps to buffer overflow attacks. The use of the almost one million data records for training and more than 300 000 data for testing in the evaluation presented below illustrates the scalability of ADWICE.

Some features of KDD data are not numeric (e.g. service). Non-numeric features ranging over n values are made numeric by distributing the distinct values over the interval [0, 1]. However, two distinct values of the service feature (e.g. http, ftp) for example, should be considered equally close, regardless of where in the [0, 1] interval they are placed. This intuition cannot be captured without extending the present ADWICE algorithm. Instead the non-numeric values with $n > 2$ distinct values are scaled with a weight w. In the KDD dataset $n_{protocol} = 3$, $n_{flag} = 11$ and $n_{service} = 70$. If $w/n > 1$ this forces the algorithm to place two sessions that differ in such non-numeric multi-valued attributes in different clusters. That is, assuming the threshold condition requiring distance to be less than 1 to insert data into a cluster ($M \gg$ distinct number of combinations of multi-valued non-numeric attributes). This should be enforced since numerical values are scaled to [0, 1]. Otherwise a large difference in numerical attributes will anyway cause data to end up in the same cluster, making the model too general. If multi-valued attributes are equal, naturally the difference in the numerical attributes decides wether two data items end up in the same cluster.

5.2 Determining Parameters

If the maximum number of clusters M is set too low, the normality model will be too general leading to lower detection rate and lower accuracy of the algorithm. This was confirmed in experiments where M was increased from 2 000 to 25 000 in steps of 1 000. When setting M above 10 000 clusters the accuracy reaches a stable level meaning that setting M at least in this range should be large enough to represent the one million normal data points in the training set. In the forthcoming experiment M is therefore set to 12 000.

Experiments where the branching factor was increased from 2 to 2 048 in small steps showed that the chance of finding the correct cluster increases with the branching factor (i.e. decreasing false positives rate). However, increasing the branching factor also increases the training and testing time. The extreme setting $B = M$ would flatten out the tree completely, making the algorithm linear as opposed to logarithmic in time. The experiments showed that the false positives rate stabilized when the branching factor is increased above 16. In the forthcoming experiments the branching factor is therefore set to 20. Experiments where the branching factor was changed from 20 to 10 improved testing time by roughly 16 % illustrating the time-quality trade-off

5.3 Detection Rate vs. False Positives Rate

Figure 2 shows the trade-off between detection rate and false positive rate on an ROC diagram[17]. To highlight the result we also compare our algorithm ADWICE–TRD with ADWICE–TRR which is closer to the original BIRCH algorithm. The trade-off in this experiment is realized by changing the E-parameter from 5 (left-most part of the diagram) to 1 (right-most part of the diagram) increasing the detection space of the clusters, and therefore obtaining better detection rate while false positives rate also increases.

The result confirms that ADWICE is useful for anomaly detection. With a false positives rate of 2.8 % the detection rate is 95 % when E = 2. While not conclusive evidence, due to short-comings of KDD data, this false positives rate is comparable to alternative approaches with clustering of unlabeled data evaluated on small subsets of KDD data that produce false positives rate of 1,3–2,5 %. The detection rate of ADWICE is significantly better compared to those approaches (40–82 %) considering that training on unlabeled data is a harder problem.

Since the KDDCUP data initially was created to compare classification schemes, many different classification schemes have been applied to the KDDCUP data set. Classification implies that the algorithms were trained using both normal and attack data contrasted to ADWICE which is only trained on the normal training data. The attack knowledge makes differentiating of attack and normal classes an easier problem, and it was expected that the results[9] of the winning entry (C5 decision trees) should be superior to ADWICE. This was also the case[1] regarding false positives (0,54 %), however detection rate was slightly lower, 91,8 %. Due to the importance of low false positives rate we indeed consider this result superior to that of ADWICE. We think the other advantages of ADWICE (section 2) make up for this. Also, we recall that ADWICE is one element in a larger scheme of other Safeguard agents for enhancing survivability.

The result shows that for values of E above 4.0 and values of E below 1.75 the false positives rate and detection rate respectively improve very slowly for ADWICE-TRD. The comparison with the base-line shows that using R in the threshold requirement (ADWICE–TRR) implies higher false positives rate. Section 5.5 describes further reduction of false positives in ADWICE–TRD by aggregation.

5.4 Attack Class Results

The attacks in the test data can be divided into four categories:

– *Probe* - distinct attack types (e.g. IP sweep, vulnerability scanning) with 4 166 number of session records in total.

[1] In the original KDDCUP performance was measured using a confusion matrix where the result for each class is visible. Since ADWICE does not discern different attack classes, we could not compute our own matrix. Therefore overall false positives rates and detection rates of the classification scheme was computed out of the result for the individual classes.

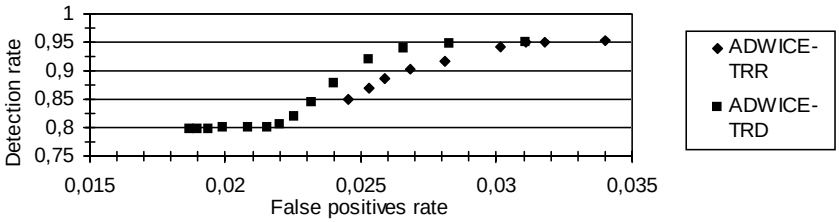

Fig. 2. Detection rate versus false positives when changing E from 5 to 1

- *Denial of Service (DOS)* - 10 distinct attack types (e.g. mail bomb, UDP storm) with 229 853 number of session records in total.
- *User-to-root (U2R)* - 8 distinct attack types (e.g. buffer overflow attacks, root kits) with 228 number of session records in total.
- *Remote-to-local (R2L)* - 14 distinct attack types (e.g. password guessing, worm attack) with 16 189 number of session records in total.

Since the number of data in the probe and DOS classes is much larger than U2R and R2L, a detection strategy may produce very good overall detection quality without handling the U2R and R2L classes that well. Therefore it is interesting to study the attack classes separately. Note that since ADWICE is an anomaly detector and has no knowledge of attack types, it will give the same classification for every attack type unlike a classification scheme.

Figure 3 shows the four attack classes Probe, DOS, U2R and R2L as well as the normal class (leftmost column) for completeness.

The results for Probe, DOS and U2R are very good, with accuracy from 88 % (U2R) to 99 % (DOS). However, the fourth attack class R2L produces in comparison a very bad result with an accuracy of only 31 %. It should be noted that the U2R and R2L classes are in general less visible in data and a lower accuracy should therefore be expected. The best entries of the original KDD-cup competition had a low detection rate for U2R and R2L attacks, therefore also a low accuracy for those classes.

5.5 Aggregation for Decreasing Alarm Rate

While 2–3 percent false positives rate produced by ADWICE may appear to be a low false positive rate in other applications, in practice this is not acceptable for

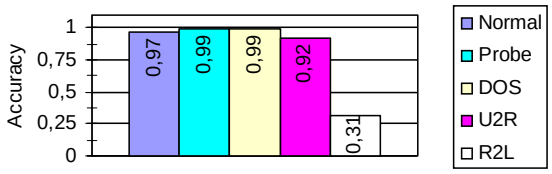

Fig. 3. The accuracy of ADWICE for individual attack classes and the normal class

network security[17]. Most realistic network data is normal, and if a detection scheme with a small percent of false positives is applied to millions of data records a day, the number of false alarms will be overwhelming. In this section we show how the total number of alarms can be further reduced through aggregation.

An anomaly detector often produces many similar alarms. This is true for new normal data that is not yet part of the normality model as well as for attacks types like DOS and network scans. Many similar alarms may be aggregated to one alarm, where the number of alarms is represented by a counter. In the Safeguard agent architecture aggregation is one important task of the Alert reduction correlation agent[4]. By aggregating similar alarms the information passed on to higher-level agents or human administrators becomes more compact and manageable. Here we evaluate how aggregation would affect the alarm rate produced from the KDD data set.

The KDD test data does not contain any notion of time. To illustrate the effect of aggregation we make the simplifying assumptions that one test data is presented to the anomaly detector each time unit. All alarms in which service, flag and protocol features are equal are aggregated during a time window of size 0 to 100. Of course aggregation of a subset of features also implies information loss. However, an aggregated alarm, referring to a certain service at a certain time facilitates the decision for narrowing down to individual alarms for further details (IP-address should have been included if present among KDD features). The result is shown in Fig. 4.

When a new alarm arrives, it is sent at once, to avoid increasing time to detection. When new alarms with the same signature arrive within the same time window, the first alarm is updated with a counter to represent the number of aggregated alarms. Without aggregation ADWICE produces 239 104 alarms during the 311 029 time units. Using a short time window of 10 time units, the number of aggregated alarms becomes 28 950. Increasing the time window to 100 will reduce the original number of alarms to 5 561, an impressive reduction

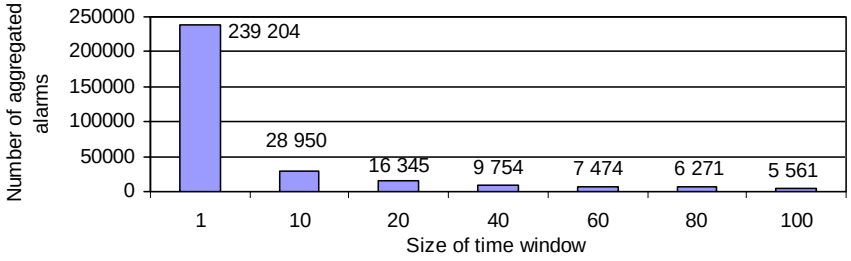

Fig. 4. The number of aggregated alarms for different time windows

of 97,7 %. The explanation is that many attacks (probes, DOS) lead to a large amount of similar alarms. Note that aggregation also reduces false positives, since normal sessions belonging to a certain subclass of normality may be very similar. While it might seem that aggregation only makes the alarms less visible (does not remove them) it is in fact a pragmatic solution that was appreciated by our industrial partners, since it significantly reduces the time/effort at higher (human-intensive) levels of investigation. The simple time slot based aggregation provides a flexible system in which time slots can be adaptively chosen as flexible 'knobs' in response to different requirements.

5.6 Usefulness of Incremental Training

To evaluate the incremental training of ADWICE we treat an arbitrary abnormal class as normal and pretend that the normality model for the KDD data should be updated with this class. Without loss of generality we choose the IP sweep attack type and call it 'normal-new'; thus, considering it a new normal class detected by the administrator. The model only trained on the original normal data will detect the normal-new class as attack, since it is not included in the model. This produces 'false positives'. The old incomplete normality model is then incrementally trained with the normal-new training data producing a new normality model that incorporates the normal-new class. Our evaluation showed that (without aggregation) the old model produced 300 false positives, whereas the new retrained model only three.

5.7 Evaluation in the Safeguard Test Network

One of the main efforts of the Safeguard project is the construction of the Safeguard telecom management test network, used for data generation for off-line use as well as full-scale on-line tests with the Safeguard agent architecture. At the time of evaluation of this work the network consisted of 50 machines (at present time about 100 machines) in multiple sub networks. Normal data can be generated by isolating the network from Internet and not running any internally generated attacks on the network.

Evaluation using data from the Safeguard test network is ongoing work. Here we present only some initial results from tests performed over a total time period of 36 hours. The ADWICE model was trained using data from a period known to contain only normal data. To keep parsing and feature computation time low to make real-time detection possible, features were only based on IP-packet headers, not on packet content (e.g. source and destination IP and ports, time, session length). This means of course that we at this stage can not detect events that are only visible by analyzing packet content. The purpose of this instance of the hybrid detection agent is to detect anomalies, outputting alarms that can be analysed by high level agents to identify time and place of attacks as well as failures or misconfigurations.

In Scenario 1 an attacker with physical access to the test network plugged in a new computer at time 15:33 and uploaded new scripts. In Scenario 2 those scripts are activated a few minutes later by the malicious user. The scripts are in this case harmless. They use HTTP on port 80 to browse Internet, but could just as well have been used for a distributed attack (e.g. Denial of Service) on an arbitrary port. The scripts are then active until midnight the first day, producing traffic considered anomalous for their respective hosts. During the night they stay passive. The following morning the scripts becomes active and execute until the test ends at 12:00 the second day.

Figure 5 illustrates the usefulness of the output of the hybrid detection agent. The 36 hours of testing was divided in periods of one minute and the number of alarms for each time period is counted.

For Scenario 1, all alarms relating to the new host (IP x.x.202.234) is shown. For Scenario 2 all alarms with source or destination port 80 are shown. The figure shows clearly how the malicious user connects at interval nr 902 (corresponding to 15:34), when the scripts executes, waits during the night, and then executes again. Some false alarms can also be noted, by the port 80 alarms occurring before the connection by the malicious user. This is possible since HTTP traffic was already present in the network before the malicious user connected.

6 Discussion and Future Work

The DARPA related data sets have been widely used but also criticized[16]. The normal traffic regularity as well as distribution of attacks compared to distribution of normality does not exactly correspond to network data in a real network.

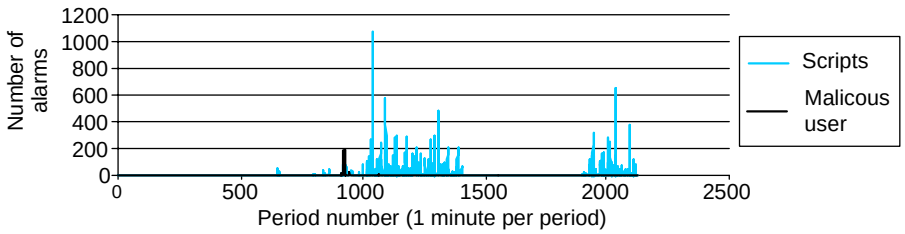

Fig. 5. Anomalous behaviour detected by ADWICE in the telecom management network. The figure shows the number of alarms for a certain time period and IP/port

Generation of a new public reference data set for IDS evaluation without the identified weaknesses of the DARPA data remains therefore as an important task of the research community. With this in mind, our DARPA/KDD based evaluation still shows feasibility of ADWICE given the assumptions that relevant features are used and that those features separate normal data from attacks. Local clustering was, according to our knowledge, used for the first time in the intrusion detection setting, whereby an optimal global clustering of data is not necessary. The incremental property of ADWICE is important to provide flexible adaptation of the model. Future work includes further evaluation of the algorithm in the context of the Safeguard test network. If made available also other public data sets will be considered. Unfortunately the GCP data provided by DARPA's Cyber Panel program[3] is not currently released to researchers outside USA.

Current work includes using the incremental feature of ADWICE for autonomous normality adapation. Evaluation of such adapation, may require long periods of data to study the effect of adaptation over time.

Our experience with ADWICE indicates, as hinted in the original BIRCH paper, that the index is not perfect. Our on-going work includes full evaluation of an alternative grid-based index with initial indications of improvement of the false positive rate by 0,5–1 % at a similar detection rate.

References

1. Yegneswaran, V., Barford, P., Ullrich, J.: Internet intrusions: global characteristics and prevalence. In: Proceedings of the 2003 ACM SIGMETRICS international

conference on Measurement and modeling of computer systems, San Diego, CA, USA, ACM Press (2003) 138–147
2. McHugh, J.: Intrusion and intrusion detection. International Journal of Information Security **1** (2001) 14–35
3. Haines, J., Kewley Ryder, D., Tinnel, L., Taylor, S.: Validation of sensor alert correlators. IEEE Security and Privacy **1** (2003) 46–56
4. Chyssler, T., Nadjm-Tehrani, S., Burschka, S., Burbeck, K.: Alarm reduction and correlation in defence of ip networks. In: Proceedings of International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE04), Modena, Italy (2004)
5. Safeguard: The safeguard project (2003) http://www.ist-safeguard.org/ Acc. May 2004.
6. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. SIGMOD Record 1996 ACM SIGMOD International Conference on Management of Data **25** (1996) 103–14
7. Fu, Y., Sandhu, K., Shih, M.Y.: A generalization-based approach to clustering of web usage sessions. In: Proceedings of the Web Usage Analysis and User Profiling. International WEBKDD'99 Workshop. Volume 1836 of Lecture Notes in Artificial Intelligence., San Diego, CA, USA, Springer-Verlag (2000) 21–38
8. Mukkamala, S., Janoski, G., Sung, A.: Intrusion detection using neural networks and support vector machines. In: Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN '02), Honolulu, HI, Institute of Electrical and Electronics Engineers Inc. (2002) 1702–1707
9. Elkan, C.: Results of the kdd'99 classifier learning. ACM SIGKDD Explorations **1** (2000) 63 – 64
10. Portnoy, L., Eskin, E., Stolfo, S.: Intrusion detection with unlabeled data using clustering. In: ACM Workshop on Data Mining Applied to Security. (2001)
11. Sequeira, K., Zaki, M.: Admit: Anomaly-based data mining for intrusions. In: Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton, Alberta, Canada, ACM Press (2002) 386–395
12. Guan, Y., Ghorbani, A.A., Belacel, N.: Y-means: A clustering method for intrusion detection. In: Proceedings of the IEEE Canadian Conference on Electrical and Computer Engineering, Montreal, Canada (2003)
13. Munson, J., Wimer, S.: Watcher: the missing piece of the security puzzle. In: Proceedings of the 17th Annual Computer Security Applications Conference, New Orleans, LA, USA, IEEE Comput. Soc (2001) 230–9
14. Han, J., Kamber, M.: Data Mining - Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
15. Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., Zhou, S.: Specification based anomaly detection: A new approach for detecting network intrusions. In: Proceedings of the ACM Conference on Computer and Communications Security, Washington, DC, USA, ACM Press (2002) 265–274
16. Mahoney, M.V., Chan, P.K.: An analysis of the 1999 darpa/lincoln laboratory evaluation data for network anomaly detection. In: Recent Advances in Intrusion Detection. Volume 2820 of Lecture Notes in Computer Science., Pittsburgh, PA, USA, Springer (2003) 220–237
17. Axelsson, S.: The base-rate fallacy and the difficulty of intrusion detection. ACM Transactions on Information and Systems Security **3** (2000) 186–205