available at www.sciencedirect.com

www.compseconline.com/publications/prodinf.htm

Information Security Technical Report

Adaptive real-time anomaly detection with incremental clustering

*Kalle Burbeck, Simin Nadjm-Tehrani**

Department of Computer and Information Science, Linköping University, Sweden

ARTICLE INFO

Published online 7 March 2007

ABSTRACT

Anomaly detection in information (IP) networks, detection of deviations from what is considered normal, is an important complement to misuse detection based on known attack descriptions. Performing anomaly detection in real-time places hard requirements on the algorithms used. First, to deal with the massive data volumes one needs to have efficient data structures and indexing mechanisms. Secondly, the dynamic nature of today's information networks makes the characterisation of normal requests and services difficult. What is considered as normal during some time interval may be classified as abnormal in a new context, and vice versa. These factors make many proposed data mining techniques less suitable for real-time intrusion detection. In this paper we present ADWICE, Anomaly Detection With fast Incremental Clustering, and propose a new grid index that is shown to improve detection performance while preserving efficiency in search. Moreover, we propose two mechanisms for adaptive evolution of the normality model: incremental extension with new elements of normal behaviour, and a new feature that enables forgetting of outdated elements of normal behaviour. These address the needs of a dynamic network environment such as a telecom management network. We evaluate the technique for network-based intrusion detection, using the KDD data set as well as on data from a telecom IP test network. The experiments show good detection quality and act as proof of concept for adaptation of normality.

1. Introduction

The threats to computer-based systems on which several critical infrastructures depend are alarmingly increasing, thereby increasing the need for technology to handle those threats (Yegneswaran et al., 2003). Increasing use of software components brings the benefit of rapid deployment and flexibility, together with the vulnerability to accidental and targeted misuse, making service availability the key challenge for many businesses and government agencies. The cycle from detection of some vulnerability in a software product, and the creation and application of a patch in all critical systems is unfortunately too long. It is long enough for malicious actors developing an attack tool, releasing it to a wider group, and launching massive attacks to take place *before* the patches are in place. One example is the Zotob-A worm and its variants (F-secure, 2005). It took only four days from the release of the patch by Microsoft (9 August 2005) before the worms exploiting the vulnerability were spreading on the Internet (13 August 2005).

Rapid patching is important but not enough (Lippmann et al., 2002). For a production system continuous patching may not be viable due to system complexity and diversity as well as compatibility requirements. For critical systems,

* *Corresponding author.*
E-mail address: simin@ida.liu.se (S. Nadjm-Tehrani).
1363-4127/$ – see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.istr.2007.02.004

defence in depth is needed incorporating many different security technologies (Venter and Eloff, 2003). This includes firewalls at network boundaries, and on individual hosts, removal of unused software and services, virus scanners, and so on; but it is increasingly evident that intrusion detection systems should be part of the hardened defence.

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). Also, due to increasing complexity of the intrusion detection task, the use of many IDS sensors to increase coverage, and the need for improved usability of intrusion detection, a recent trend is alert or event correlation (Morin and Hervé, 2003; Haines et al., 2003; Chyssler et al., 2004). 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 behaviours (attacks) are encoded into signatures. This type of misuse detection faces problems unless the signature database is kept up to date, and even then can only detect attacks that are well known and for which signatures have been written.

An alternative approach is *anomaly detection* in which normal behaviour of users or the protected system is modelled, often using machine learning or data mining techniques. During detection new data are 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. If an attack tool is published before a patch is applied and before the attack signatures are developed or installed, the anomaly detection system may be the only remaining defence. Some attack types, including a subset of denial of service and scanning attacks, alter the statistical distribution of the system data when present. This implies that anomaly detection may be a general and perhaps the most viable approach to detect such attacks.

Anomaly detection still faces many challenges, where one of the most important is the relatively high rate of false alarms (false positives). The problem of capturing a complex normality makes the high rate of false positives intrinsic to anomaly detection. We argue that the usefulness of anomaly detection is increased if combined with further aggregation, correlation and analysis of alarms, thereby minimizing the number of false alarms propagated to the administrator (or automated response system) that further diagnoses the scenario.

The fact that normality changes constantly makes the false alarm problem even worse. A model with acceptable rates of false alarms may rapidly deteriorate in quality when normality changes over time. To minimize this problem and the resulting additional false alarms, the anomaly detection model needs to be adaptable.

The training of the normality model for anomaly detection may be performed by a variety of different techniques and many approaches have been evaluated. One important technique is *clustering*, where similar data points are grouped together into clusters using a distance function. As a data mining technique, clustering fits very well for anomaly detection, since no knowledge of the attack classes is needed whilst training. Contrast this to 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.

In this paper we present Anomaly Detection With fast Incremental Clustering (ADWICE), a novel adaptive anomaly detection scheme, inspired by the BIRCH clustering algorithm (Zhang et al., 1996), and extended with new capabilities. It is based on an earlier presented version of the algorithm (Burbeck and Nadjm-Tehrani, 2004) but complements the original technique with a novel search index that increases detection quality, and provides new means to handle adaptation (forgetting). We show the application of the new mechanisms in two settings: (1) an emulated test network at a major Telecom operator in Europe (Swisscom) for evaluating the scalability and timeliness of the algorithm, and (2) the comparative analysis of the detection quality of the algorithm based on the only common (open) data source available – the KDD99 attack data. Data mining approaches to anomaly detection are a recent area of exploration, and more work needs to be done to clarify their strengths by studies in several application domains. Therefore, this paper does not attempt to justify the appropriateness of clustering as a whole or compare performance with alternative machine learning work. The interested reader is referred to an earlier report (Burbeck, 2006) that covers some of this ground.

The paper is organised as follows. In Section 2 the motivation of this work and perspective with respect to related works is provided. Next we present the context of ADWICE within the Safeguard agent architecture. Section 4 describes the technique used, and evaluation is presented in Section 5. The final section discusses the results and presents some remaining challenges for future work.

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 for public research comparison.

For learning based methods, good data are 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 (Elkan, 2000; Mukkamala et al., 2002), or supervised learning, require training data that contain 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. This is in contrast with clustering-based methods that require no labelled training data set containing attacks (Portnoy et al., 2001; Sequeira and Zaki, 2002; Guan et al., 2003), thereby reducing training data requirement. 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 volume 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 (including the above named works). The accuracy is, however, relatively low which reduces the direct applicability in a real network.

In the second approach, which we simply denote (*pure*) *anomaly detection* in this paper, training data are assumed to consist *only of* normal data. Munson and Wimer (2001) 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 to provide better accuracy than the unsupervised approach. Pure anomaly detection, similar to unsupervised anomaly detection, avoids the coverage problem of classification techniques, and requires no labelling of training data. 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 100-node test network build specifically for experimental purposes in the European Safeguard project.[1]

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 be considered normal during detection 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 possible complementary approach is to train temporary models on different data sets and let them vote on normality during a pre-training phase to decide what data to use for the final normality model.

Certain attacks, such as Denial of Service (DoS) 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. Otherwise, if automatic updates are undesirable, an administrator should to be able to update the anomaly model with simple means, without destroying what is already learnt. Also the effort spent updating the model should be minimal compared to the effort of training the initial model. ADWICE is fully incremental, supporting easy adaptation and extension of the normality model. By fully incremental, we mean that instead of extensive periodic retraining sessions on stored off-line data ADWICE has on-line training capabilities without destroying what is already learnt. Also, when subsets of the model are no longer useful, those clusters can be forgotten.

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 (Han and Kamber, 2001), making real-time adaptation of a cluster-based model hard. Also they may not be scalable, requiring all training data to be kept in main memory during training, which limits the size of the trained model. We argue that it is important to consider scalability and performance in parallel to detection quality when evaluating algorithms for intrusion detection. Most work on applications of data mining to intrusion detection considers those issues to a very limited degree or not at all. ADWICE performance is linear in the number of input data thereby reducing training time compared to other algorithms. Training time as well as detection time is further reduced by using an integrated search index.

[1] The Safeguard project was an IST FP5 European project running 2001–2004.

To sum up, when compared with similar approaches ADWICE:

- Is scalable, since it avoids keeping data in memory, representing clusters by compact summaries.
- Has good performance since it uses local clustering and an integrated tree index for searching the model.

3. The Safeguard context

Safeguard (2001–2004) was a European research project aiming to enhance survivability of critical infrastructures by using agent technology. The Safeguard agent architecture is presented in Fig. 1. The agents should improve survivability of large complex critical infrastructures (LCCIs), by detecting and handling intrusions as well as faults in the protected systems. Much of the diagnosis and detection, however, builds upon existing sensors and (infosec) devices that are already in place in many such networks. The main contributions of Safeguard is the conceptualisation of the needed functions, development of new techniques for anomaly detection, and the addition of correlation engines and automatic reaction mechanisms for recovery and self-healing. The parallel activities of these agents relieve the human operators, while at a same time allow dynamic adaptation to new needs in terms of situation awareness of operators.

The key to a generic solution applicable in many infrastructures is in the definition of roles for various agents. These are believed to be common for the defence of many infrastructures, but should be instantiated to more specific roles in each domain. The Safeguard project demonstrated the instantiation of this agent architecture to electricity and telecom management domains. ADWICE was developed as the anomaly detector engine for one instance of a Safeguard agent and demonstrated in the telecom domain. In recent years, the algorithm has been improved and further developed. Before presenting the details of ADWICE, it is relevant to describe the background against which it was developed. The Safeguard 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. Examples are Snort, Syslog, Samhain and other host or network sensors.
– **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 with behavioural intrusion detection mechanisms.
– **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. Use the data sources to order, filter and focus on certain alarms, or predict reduced availability of network critical services. An example is a correlation agent that performs adaptive filtering and aggregation of the alarms to reduce the (false) alarm rates.
– **Action agents** enable automatic and semi-automatic responses when a problem is definitely (or with high probability) identified.
– **Negotiation agents** communicate with agents in other LCCIs 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. An example element of such an interface is a security dashboard indicating the (application-dependent) health of the network.
– **Actuators** are wrappers for interacting with lower layer software and hardware (e.g. changing firewall rules, adjusting configurations of detection agents/sensors).

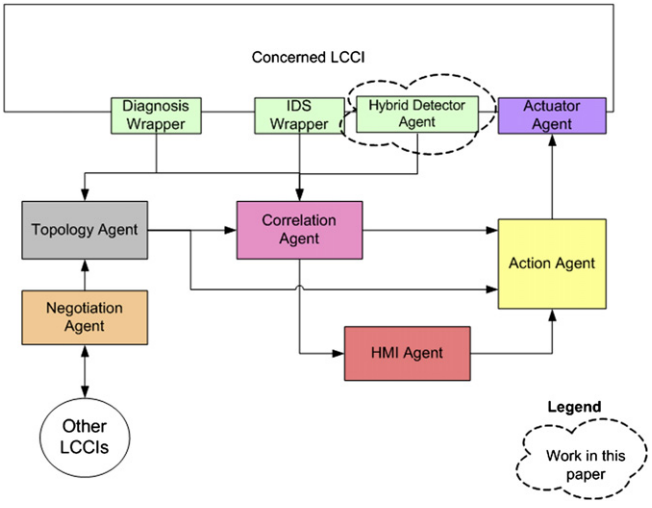

Fig. 1 – The Safeguard agent architecture.

Note that several instances of each agent type may be present to provide resilience and the benefits of distribution.

In the context of a management network for telecom service providers the following needs were identified and addressed:

- Reducing information overload (Chyssler et al., 2004).
- Increasing coverage by providing new sources of information using e.g. anomaly detection ((Burbeck and Nadjm-Tehrani, 2004) and the extensions in this paper (Bigham et al., 2003; Gamez et al., 2005)).
- Increasing information quality by reducing false positives (Chyssler et al., 2004).
- Collating information, such as correlating alarms (Chyssler et al., 2004) and combine with topology information (demonstrated in the final Safeguard demo).
- Presenting a global view of a network (demonstrated in the final Safeguard demo).

This paper presents one instance of Hybrid detector agent. The notion of *hybrid* comes from combining ADWICE together with a white list, i.e. a simple specification based element (see Sekar et al., 2002), deployed to detect anomalies.

4. The anomaly detection algorithm ADWICE

This section describes how ADWICE handles training and detection. The present implementation of ADWICE requires data

to be numeric. Non-numeric data are, therefore, assumed to be transformed into numeric format by pre-processing.

4.1. *Basic concepts*

An ADWICE model consists of a number of clusters, a number of parameters, and a tree index in which the leaves contain the clusters. We reuse the central idea of BIRCH, that is, to store only condensed information (cluster feature) instead of all data points of a cluster. A *cluster feature* is a triple $CF = (n, \vec{S}, SS)$ where n is the number of data points in the cluster, $\vec{S}$ is the linear sum of the n data points and SS is the square sum of all data points. Given n d-dimensional data vectors v_i and a cluster CF representing $\{v_i\ i = 1,\ldots, n\}$, the centroid v_0 and radius R(CF) are defined as:

$$v_0 = \sum_{i=1}^{n} v_i/n \tag{1}$$

$$R(\mathrm{CF}) = \sqrt{\sum_{i=1}^{n}(v_i - v_0)^2/n} \tag{2}$$

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.

Given the CF of a cluster, the centroid v_0 and radius R may be computed. The distance between a data point v_i and a cluster CF is the Euclidian distance between v_i and the centroid, denoted $D(v_i, \mathrm{CF})$ while the distance between two clusters CF_i and CF_j is the Euclidian distance between their centroids, denoted $D(\mathrm{CF}_i, \mathrm{CF}_j)$. If two clusters $\mathrm{CF}_i = (n_i, \vec{S}_i, \mathrm{SS}_i)$ and $\mathrm{CF}_j = (n_j, \vec{S}_j, \mathrm{SS}_j)$ are merged, the CF of the resulting cluster may be computed as $(n_i + n_j, \vec{S}_i + \vec{S}_j, \mathrm{SS}_i + \mathrm{SS}_j)$. This also holds if one of the CFs is only based on one data point making incremental update of CFs possible.

A *leaf node* contains at most LS (leaf space) entries, each of the form (CF_i) where $i \in \{1,\ldots, \mathrm{LS}\}$. Each CF_i of the leaf node must satisfy a threshold requirement (TR) with respect to the threshold value T which is evaluated to see if a cluster may absorb more data points. Two different threshold requirements have been evaluated with ADWICE. The first threshold requirement where $R(\mathrm{CF}_i) <= T$ corresponds to a threshold requirement suggested in the original BIRCH 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 developed where $D(v_i, \mathrm{CF}_i) <= T$ needs to hold to allow the cluster to absorb the new data point v_i. This version was evaluated as ADWICE-TRD.

4.2. *Training*

The algorithm for training works with a parameter M (maximum size of the model, denoted by total number of clusters, thus restricting memory requirements), and a parameter LS (leaf size, denoting maximum number of clusters in each leaf). Some additional parameters are specific to the type of index tree used, and those are explained together with the indices later.

Given a model and a new data vector v, a search for the closest cluster is performed. If the threshold requirement is fulfilled, the new data point may be merged with the closest cluster, otherwise a new cluster needs to be inserted. If the size (number of clusters) of the model has reached the maximum M, the threshold T is increased, the model rebuilt, and then v is inserted in the new model.

Below is an algorithmic description of the training phase of ADWICE, in which only the main points of the algorithm are presented and some simplifications made to facilitate presentation. Note that we have abstracted away the index, and present the specific workings of the original BIRCH index as well a grid based index in the next sections.

```
train(v, model)
  closestCF = findClosestCF(v, model)
    IF thresholdRequirementOK(v,closestCF) THEN
      merge(v,closestCF)
    ELSE
      IF size(model) >= M THEN
        increaseThreshold()
        model = rebuild(model)
        train(v, model)
      ELSE
        leaf = getLeaf(v,model)
        IF spaceInLeaf(leaf) THEN
          insert(newCF(v), leaf)
        ELSE
          splitLeaf(leaf, newCF(v))
```

Rebuilding the model requires much less effort than the initial insertion of data since only clusters rather than individual data points are inserted, and the number of clusters is significantly less than the number of data points. 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).

If it is possible to add clusters to the model (the size is still below M), we find the leaf where the new cluster should be included and insert the cluster if there is still space in the leaf. Otherwise we need to split the leaf, and insert the new cluster in the most suitable of the new leaves.

4.2.1. *Using the original BIRCH index*

The original BIRCH index consists of a CF *tree*, which is a height-balanced tree with four parameters: branching factor (B), threshold (T), maximum number of clusters (M), and leaf size (LS). Each non-leaf node contains at most B entries of the form $(\mathrm{CF}_i, \mathrm{child}_i)$, 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 CFs in the level below.

- Finding the closest cluster is done by recursively descending from the root to the closest leaf, and in each step choosing child i such that $D(v, \mathrm{CF}_i) < D(v, \mathrm{CF}_j)$ for every other child j.
- When inserting new data into the tree all nodes along the path to the root need to be updated. In the absence of a split,

the CFs along the paths to the updated leaf need to be recomputed to include v by incrementally updating the CFs. If a split occurred, we need to insert a new non-leaf entry in the parent node of the two new leaves and recompute the CF summary for the new leaves. 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.

- When splitting a leaf, the two farthest CFs of the leaf are selected as seeds and all other CF_j from the old leaf are distributed between the two new leaves. Each CF_j is merged with the leaf with the closest seed.

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. To our knowledge this is the first use of the CF based approach to anomaly detection.

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 at the training data but fails to produce good results for testing data that differs more or less from the training data). M also needs to be set high enough so that the number of clusters is enough for representing normality.

4.2.2. *Problems of original BIRCH index*

The original BIRCH index is not perfect. An intrinsic property of the basic search mechanism results in a sub-optimal search. That is, the search for the closest cluster sometimes selects the wrong path, and ends up in the wrong end of the tree. This is due to the very condensed information available to guide the search in each non-leaf. Index errors may influence the detection quality (and evaluation) of an algorithm. Though an initial evaluation of the algorithm showed interesting results (Burbeck and Nadjm-Tehrani, 2004) we are interested in pinpointing the extent of improvement possible if index errors are completely removed. Table 1 shows how index error influences anomaly detection results in general when a testing data vector is compared to the model.

Table 1 – Consequences of index errors for anomaly detection

Data is	Model covers data	Expected evaluation	Evaluation with index error
Normal	*Yes*	*True negative*	*False positive*
Normal	No	False positive	False positive
Attack	*Yes*	*False negative*	*True negative*
Attack	No	True positive	True positive

If the type of data is not included in the trained normality model an anomaly detection algorithm with index-errors returns the correct result, since there is no close cluster to find anyway. However, there are two cases where index errors may produce erroneous results. If the new data point is normal, and the model includes a close cluster, an index error results in a false positive instead of a true negative, thereby decreasing detection quality. On the other hand, if the new data are anomalous and such data have previously been (erroneously) included into the model, or if the attack data are very similar to normal data, the index error may result in a true positive, improving detection quality. In other words, index errors make the behaviour of the detection scheme unpredictable, since quality may both increase and decrease. The index errors do not completely invalidate evaluation, since if training data are to a large extent normal, the second case of error (causing improvement) is improbable. Still the result may be improved using an index not causing index errors. In this paper we address this issue by introducing a new index.

4.2.3. *The Grid-index*

Our third implementation of ADWICE aims to improve the index errors caused by the original BIRCH index, but maintain its incremental training capabilities.

There are two possibilities for handling the adaptability requirement. The index may be updated together with the clusters whenever model change occurs (which is the BIRCH approach) or the index may be independent of changes to the model. ADWICE-Grid follows the second principle.

A *subspace* of d-dimensional space is defined by two vectors, Max and Min for each dimension, specifying for each dimension an interval or *slice*. A *grid* is a division of space into subspaces. In two dimensions this results in a space divided into rectangles. The idea of the new grid index is to use a *grid tree* where each node specifies a subspace of a certain depth corresponding to the depth of the node. Leaves contain the clusters, and are the maximum depth of subspaces for each part of the tree. Fig. 2 shows a two-dimensional grid divided into subspaces together with the corresponding grid tree (the tree is further explained below). Note that not all subspaces contained by leaves need to be at the same level and that empty subspaces have no corresponding leaf. Our intuition and experience tells us that such a grid is sparsely populated. This means that suitable primitives such as hash tables (rather than lists) should be used to avoid empty subspaces take up space in the index tree.

Before starting with the implementation we tested the performance of the primitive operations used in an index tree. In case of the BIRCH index, the primitive operation is the distance function computing the Euclidian distance in multi-dimensional space. A performance evaluation shown in Fig. 3 revealed that a hash function call is 6–15 times faster than one call to the distance function depending on the number of dimensions (60–20). Since at each node of the CF tree of BIRCH, up to B (normally set to 10–20) branches may exist, using a hash table could be 100 times faster when the distance function is applied multiple times during linear search of

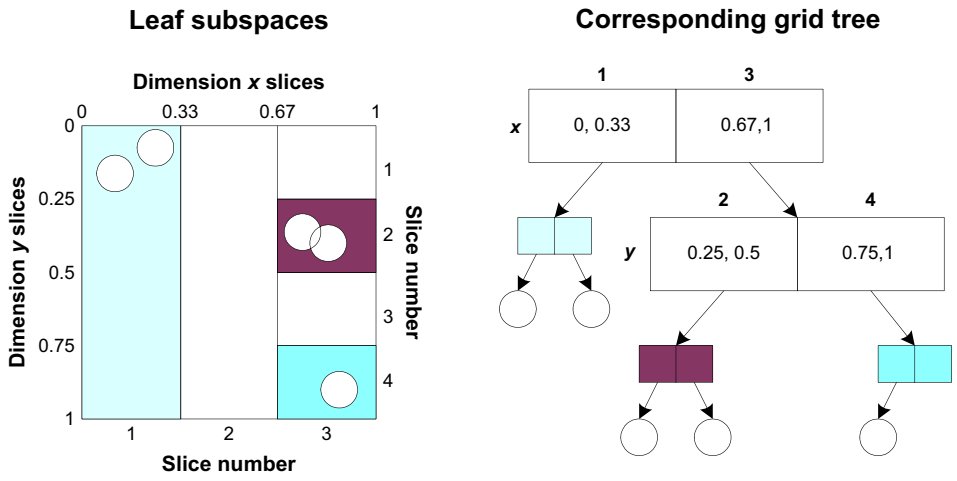

Fig. 2 – Illustration of the basic notions of the grid and grid tree.

a non-leaf node to find the correct child. This means that there is room for additional processing with the new index.

Our grid index consists of a grid tree, which is a sparse, possibly unbalanced tree with three parameters, threshold (T), leaf size (LS) and maximum number of clusters (M). Each dimension i of the d-dimensional space is assumed to have a maximum (Max_i) and a minimum (Min_i). These are in practice realized during feature extraction for unbounded domains, and lead to the division of the current dimension i in a certain number of intervals/slices (NumSlices_i) with a certain width (SliceWidth_i). A function getSliceDimension(depth) is devised which maps a node depth to one dimension. Each non-leaf node of depth j contains at most NumSlices_i children where each child of a non-leaf node is an entry in a hash table. The hash table is a mapping from interval number to a child node.

- To find the closest cluster of a new data item v in a node with depth j (starting with the root), we first compute the slice dimension $i = \text{getSliceDimension}(j)$. In the current node we then only consider dimension i of v. Given the value of v in dimension i ($v[i]$), the number of the interval into which v fits is computed. This interval number is mapped to a child using the hash table. In this way we find our way down to a leaf, where the data are located. In this leaf we may then do linear search among the clusters to find the closest.

Unfortunately there is a complication that makes the index more complex. There is no guarantee that the closest cluster actually is located in the same leaf as the data point itself. Merging of a data point with the closest cluster CF_i may be performed if the $D(v, \text{CF}_i) < T$. This means that the closest CF_i may be located inside any of the subspaces reachable within a distance T of the data point v. Accordingly we possibly need to search multiple paths at each node in the tree, depending on the value of T. At each non-leaf node at depth i we compute a *search width* $[v[i] - T, v[i] + T]$. All slices that overlap the search width are searched to find the closest cluster. Since space is sparse, many slices are not occupied. Since only children for occupied intervals exist as hash table entries, empty intervals do not need to be searched.

During training, there are two cases when the nodes of the grid tree need to be updated:

- If no cluster is close enough to absorb the data point, v is inserted into the model as a new cluster. If there does not exist a leaf subspace where the new cluster fits, a new leaf is created. However, there is no need for any additional updates of the tree, since nodes higher up do not contain any summary of data below.
- When the closest cluster absorbs v, its centroid is updated accordingly. This may cause the cluster to move in space. A cluster may potentially move outside its current subspace. In this case, the cluster is removed from its current leaf and inserted anew in the tree from the root, since the path all the way up to the root may have changed. If the cluster was the only one of the original leaf, the leaf itself is removed to keep unused subspaces without leaf representations.

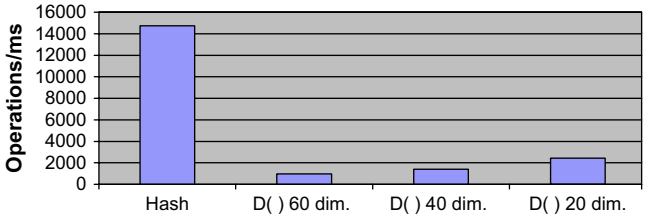

Fig. 3 – Performance of primitive index operations: original distance compared to a hash function.

Compared to the continuously updated original BIRCH index, the need for grid index updates are very small, since the first case above requires only insertion of one new leaf, and the second case occurs infrequently.

In case of a split of a leaf, the original leaf is transformed to a non-leaf node. The new node computes its split dimension according to its depth in the tree, and the clusters of the original leaf are inserted in the new node resulting in creation of leaves as children to the node.

In most cases not all dimensions need to be used for slicing (thus limiting the height of the tree), assuming that the function getSliceDimension(depth) is selected appropriately. However, if the situation arises that all dimensions have been used

for slicing, and still the number of clusters to be inserted does not fit in one leaf due to the limited leaf size (LS), then the LS parameter for that leaf can be increased, affecting only that leaf locally. This is the approach that our current implementation adopts. An alternative or complementary approach to handle this situation is to rebuild the tree using a smaller width of the intervals for each dimension.

The getSplitDimension(depth) should be defined manually or automatically according to properties of the input data to avoid a tree with a large depth. For example, if all data have the same or almost the same value in a certain dimension, this dimension is not useful for slicing, since the depth of the tree will increase without distributing the clusters into multiple children.

4.3. *Detection*

The detection procedure is the same for any choice of index. During the detection the model is searched for the closest cluster CF_i (using the index). Then the distance $D(v,CF_i)$ from the centroid of the cluster to the new data point v is computed. Informally, if D is small, i.e. lower than a limit, v is similar to data included in the normality model and v should therefore be considered normal. If D is large, v is an anomaly.

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

$$\mathrm{Belief} = \begin{cases} 0 & \text{if } D \leq \mathrm{MinL} \\ 1 & \text{if } D \geq \mathrm{MaxL} \\ \frac{D-\mathrm{MinL}}{\mathrm{MaxL}-\mathrm{MinL}} & \text{if } \mathrm{MinL} < D < \mathrm{MaxL} \end{cases} \quad (3)$$

A belief threshold (BT) is then used to make the final decision. If belief $\geq$ BT, we consider 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 v is anomalous if and only if $D > \mathrm{MaxL}$.

For the grid tree the initial search for the closest cluster differs slightly from the search during training. When deciding search width now MaxL rather than T needs to be used. Since the BIRCH index uses no notion of search width, search during detection is exactly the same as during training.

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.

Scenario 1: New cases of normality require the model to adapt incrementally. 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 subset of those 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 detection results for a specific network during some time interval. However, in an extension of the interval 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 relearning the working fragment of the existing normality model. The administrator may interleave incremental training with detection completely eliminating the need for downtime required by non-incremental approaches.

Scenario 2: The opposite scenario is when the model of normality needs to shrink. That is, something that was considered normal earlier is now considered as undesirable. In our approach this situation is adapted to by forgetting the segment of normality that is no longer considered as normal. This too can be performed autonomously or manually. The current autonomous forgetting process checks the model periodically (CheckPeriod) to decide what clusters can be forgotten. If the difference between current time and time of last use is larger then a forgetting threshold (RememberPeriod), the cluster is removed from the model. The rest of the model is not influenced.

Each time a cluster is used, either during training or detection, forgetting of the cluster is postponed, similar to the positive influence of repetition in case of human learning.

5. Evaluation

In all the following experiments ADWICE with a grid index is used unless otherwise stated. To make all data numeric, 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 such a feature (e.g. http, ftp), 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. If $w/n > 1$ this forces the algorithm to place two data points that differ in such non-numeric multi-valued attributes in different clusters. This builds on the assumption that the threshold should be significantly less than one ($M \gg$ 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 those multi-valued attributes are equal, naturally the difference in the numerical attributes decides whether two data items end up in the same cluster.

5.1. *Data sets*

Performing attacks in real networks to evaluate on-line anomaly detection is most often unrealistic. Our work deals

with this inherently synthetic situation as follows. We choose to start evaluation of detection quality using the KDDCUP99 intrusion detection data set (Hettich and Bay, 1999) to be able to compare the results with earlier evaluations, and test the scalability of ADWICE with respect to number of features and real-time properties. Despite the shortcomings of the DARPA related data sets (Mahoney and Chan, 2003; McHugh, 2000) 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. Since the KDD data set is already pre-processed into session records suitable for classification, it is an efficient way of evaluating a method over many attack types. The purpose of this evaluation is therefore a proof of concept for ADWICE, including the improved grid index. We then go on to evaluate the algorithm in the Safeguard test network built with the aim of emulating a realistic telecom management network.

The original KDD training data set consists of almost five million *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. Three features, protocol, flag and service, are multi-valued non-numeric features and are transformed accordingly. Since ADWICE assumes all training data to be 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 (41-dimensional) 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.

To illustrate forgetting and incremental training a data set generated at the Safeguard test network was used. A period of three days' (2004-03-08 00:00 until 2004-03-11 00:00) data are used for training an initial model, while the following seven days of data (2004-03-11 00:00 until 2004-03-19 11:00) are used for testing. The features include source and destination IP and port, time of day, connection length, bytes transferred and a flag indicating a parsing error. At this point no features are based on content, to avoid degrading performance of preprocessing (parsing tcpdump data and computing features) due to the real-time requirement of the Safeguard architecture. Fig. 4 shows how the Hybrid detection agent with ADWICE remotely accesses its data sources inside the Safeguard test network and after processing sends alarms to a correlation agent performing aggregation.

5.2. *Detection quality of ADWICE*

We have evaluated three different versions of ADWICE as shown in Fig. 5. The trade-off between detection rate and false positives rate are realised by changing the detection parameter *E* (from 5 on the leftmost measure to 1 on the rightmost).

The difference between ADWICE-TRR and TRD is the threshold requirement (see Section 4.1). Both use the original

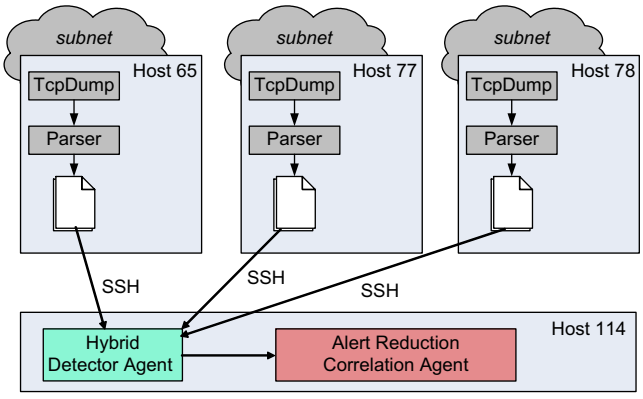

Fig. 4 – Remote data access in the Safeguard network.

CF-tree index. ADWICE-TRR is the variant that most closely resembles the BIRCH clustering algorithm. ADWICE-TRD improves detection quality by using distance not only for detection but also for the threshold requirement. ADWICE-Grid uses the distance based threshold requirement and the new grid index rather than the original BIRCH index, eliminating the index-misses and thereby further improving the result. Significant reduction of ADWICE false alarms using timestamp-based aggregation at correlation level is possible. ADWICE-TRD was successfully tested with this technique, the results of which can be found in earlier publication (Burbeck and Nadjm-Tehrani, 2004).

5.3. *Incremental training*

An important feature of this work is that the original model, known to reflect recent normality, does not need to be retrained as soon as new cases of normality are encountered. This is especially valuable in dynamic sectors like the telecom where additions of new services and operations are a norm. We evaluated this feature on three days of training data from the Safeguard test network for building an initial normality model. The model is then used for detection on the seven days of testing data. When certain types of traffic (new cases of normality) start producing false alarms the administrator tells ADWICE to incrementally learn the data causing those alarms to avoid similar false alarms in the future. Fig. 6 shows alarms for host x.x.202.183 in three

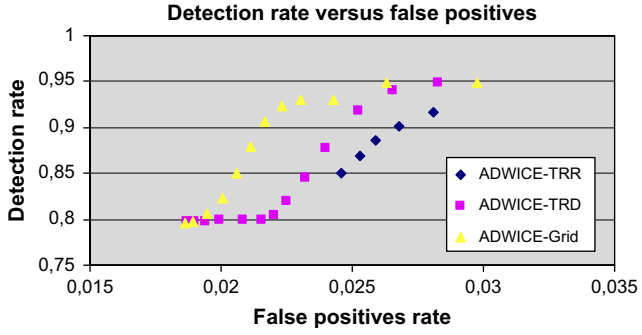

Fig. 5 – Detection quality of ADWICE on KDD data.

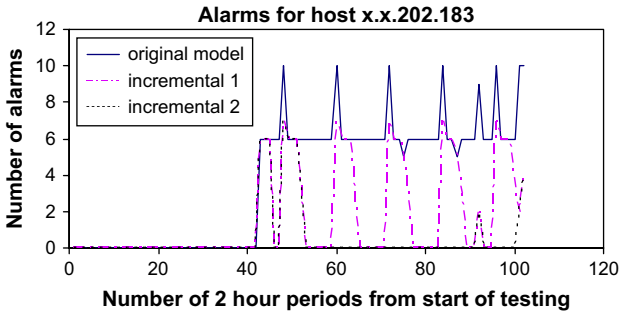

Fig. 6 – Adapting to changing normality with incremental training.

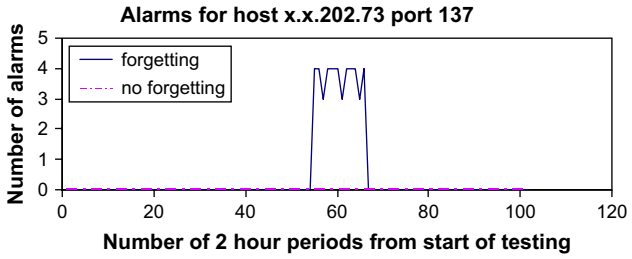

Fig. 7 – Adapting to changing normality using forgetting.

scenarios. Period number 1 starts at time 2004-03-11 00:00 (start of testing data) and each 2-h period presents the sum of alarms related to host x.x.202.183 during the corresponding time interval. At 2004-03-14 12:00, corresponding to period number 43, the host is connected to the network.

In the first scenario, no incremental training is used, and the testing is performed on the original model. This corresponds to the first curve of Fig. 6. We see that when the host connects, ADWICE starts to produce alarms and this continues until the testing ends at period 102.

In the second scenario the administrator recognizes the new class of alarms as false positives. She tells ADWICE to learn the data resulting in those false alarms at time 2004-03-14 17:55 (end of period 45). The second curve shows that many of the original false alarms are no longer produced. However, at regular intervals there are still many alarms. Those intervals correspond to non-working hours.

In the third scenario incremental training is done in two steps. After the first incremental training at 2004-03-14, a second incremental training is initiated at 2004-03-15 07:55 (end of period 52) when the administrator notices that false alarms related to host x.x.202.183 are still produced. Fig. 6 shows how almost all alarms now disappear after the second incremental training period.

The need for the two-step incremental learning arouse since the model differs between working hours and non-working hours. The alarms the administrator used for initial incremental training were all produced during working hours (2004-03-14 12:00 to 2004-03-14 17:55).

5.4. *Forgetting*

In this section we illustrate the use of forgetting. A model is trained on data from three days of data and is then used for detection with and without forgetting on the following seven days of data. Fig. 7 shows alarms for one instance of traffic (host x.x.202.73, port 137) that ceases to be present in the (normal) testing data, making that kind of traffic anomalous. With forgetting this fact is reflected in the normality model. In this experiment a CheckPeriod of 12 h and RememberPeriod of three days (72 h) are used.

When traffic from host x.x.202.73 on port 137 is again visible in data (periods 55–66) the traffic is detected as anomalous. Without forgetting these anomalies would go undetected.

5.5. *Timing and throughput studies*

During our experiments we concluded that the agent was able to process 1400 to 1900 session records per second. This was the case when data were read from a local file and using a model size of 12 000 clusters. The agent was executing on a host with Pentium 4 1.8 GHz processor with 512 MB memory.

In the safeguard test network multiple network sniffers were used as illustrated in Fig. 4. Since one agent needs information from multiple hosts, remote data access has to be used. We observed that if the hosts on which the sniffers executed became overloaded, the sniffers would start dropping packets and attacks could pass by without detection. An alternative was tested with the agent running at its own host, avoiding the danger of the sharing the processing resources with a sniffer. The down side was the additional time required to process data. When accessing multiple files remotely we used encryption of traffic by SSH. This decreased performance resulting in 500 session records processed per second.

6. Conclusions and future work

We have developed and evaluated an approach to fully adaptive anomaly detection and shown its feasibility and scalability with convincing results. Details of the implementation of a number of Safeguard agents including the agent that uses ADWICE, and details of selected IP packet features, and the Telecom test network can be found in a recent thesis (Burbeck, 2006). This also includes a detailed review of other works on agents for intrusion detection, clustering-based anomaly detection, and representatives from the non-clustering-based approaches, excluded here due to space limitations.

In clustering-based works, real-time detection and indexes for fast matching against the normality model are not part of the basic detection approach. We think however, that it is important to include the index in the detection scheme from the start, since the index may influence not only performance, but also other properties such as adaptiveness.

Our experience with incremental training indicates the need for new techniques to complement the anomaly detector. Julisch (2003) describes how clustering of alarms is used to identify root causes (e.g. misconfiguration of a system resulting in false alarms). This technique could be used to suggest to the administrator that large classes of similar alarms

may possibly be false positives that should be included into the model by incremental training.

Preliminary evaluations show that there may be a need for the administrator to tell the anomaly detector to learn not only the data producing the alarm, but also a generalisation of data. For example, the two-step incremental training in Section 5.3 would not have been necessary if the administrator could have told the system to learn that the data producing alarms was normal both during working and non-working hours. Those experiences call for intelligent tools to help the administrator maintain an anomaly detector in a real environment.

We have here evaluated two kinds of threshold requirements using radius (TRR) or distance (TRD). Using distance produces better detection quality but using the radius is more resistant to outliers of data raising the question whether a combination of these two could lead to the best of both worlds. Also forgetting may be used for handling outliers. Extensions of the work will consider cluster size and frequency of usage when deciding whether a cluster ought to be forgotten. Large clusters, corresponding to very common normality in the past, should be very resistant against forgetting. With this approach also outliers resulting in small clusters in the model could be handled by forgetting them. Another improvement would be to gradually decrease the influence of clusters over time, rather than forgetting them completely.

For full evaluation of forgetting and incremental training, a long period of data should be used. We see this work as a proof of concept. It is improbable that parts of normality should be forgotten already after a few days in a real network. Producing good public data for intrusion detection evaluation including anomaly detection and correlation is still an important task for the intrusion detection community and we think collaboration on this task is very important. Two approaches exist:

- A test network or simulator can be used for generation of data, thereby realising a fully controlled environment. Generation of publicly available data sets in the Safeguard test network is ongoing, but requires more resources, primarily to generate good normal data, but also to perform attacks.
- Data can be collected from real live networks. Here, normality is no longer a problem, but privacy is, and so is the issue of attack coverage. Emerging tools (Bishop, 2004) can be used to sanitize data with some limitations. Sanitizing data while keeping relevant properties of data so that intrusion detection analysis is still valid is not easy, especially for unstructured information, for example, network packet content. Attacks may have to be inserted into data off-line if enough real attacks are not present in data, since performing attacks in a real network is typically not a viable option.

We would like to explore both paths for evaluation of future work. Finally, it is worth highlighting that ADWICE is a general anomaly detector that can be applied to multi-dimensional data from other critical infrastructures. Application of the algorithm in new domains, and in particular, on sensor data from water management systems, is a topic for current study.

Acknowledgements

This work was supported by the European project Safeguard IST-2001-32685 and CENIIT (Centre for Industrial Information Technology at Linköping University). 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. The second author was partially supported by University of Luxembourg during preparation of the final version of this manuscript.

REFERENCES

Bigham J, Gamez D, Lu N. Safeguarding SCADA systems with anomaly detection. In: Proceedings of mathematical methods, models and architectures for computer network security. Lecture notes in computer science, vol. 2776. Springer Verlag; 2003. p. 171–82.

Bishop M. How to sanitize data. In: Proceedings of international workshops on enabling technologies: infrastructures for collaborative enterprises (WETICE04), IEEE Computer Society; June 2004.

Burbeck K. Adaptive real-time anomaly detection for safeguarding critical networks. Linköping University, ISBN 91-85497-23-1; February 2006 [Licentiate thesis, number 1231]. Available at: <http://www.diva-portal.org/liu/abstract.xsql?dbid=5973>.

Burbeck K, Nadjm-Tehrani S. ADWICE – anomaly detection with fast incremental clustering. In: Proceedings of the seventh international conference on security and cryptology (ICICS'04). Springer Verlag; December 2004.

Chyssler T, Nadjm-Tehrani S, Burschka S, Burbeck K. Alarm reduction and correlation in defence of IP networks. In: Proceedings of the international workshops on enabling technologies: infrastructures for collaborative enterprises (WETICE04), IEEE Computer Society; June 2004.

Elkan C. Results of the KDD'99 classifier learning. ACM SIGKDD Explorations January 2000;1(2):63–4.

F-secure. Available at: <http://www.f-secure.com> [accessed August 2005].

Gamez D, Nadjm-Tehrani S, Bigham J, Balducelli C, Chyssler T, Burbeck K. Safeguarding critical infrastructures. In: Diab HB, Zomaya AZ, editors. Dependable computer systems: paradigms, performance issues and applications. John Wiley and Sons; 2005 [chapter 18].

Guan Y, Ghorbani AA, Belacel N. Y-means: a clustering method for intrusion detection. In: Proceedings of the IEEE Canadian conference on electrical and computer engineering; May 2003.

Haines J, Kewley Ryder D, Tinnel L, Taylor S. Validation of sensor alert correlators. IEEE Security & Privacy January 2003; 1(1):46–56.

Han J, Kamber M. Data mining – concepts and techniques. Morgan Kaufmann Publishers Inc.; 2001.

Hettich S, Bay SD. The UCI KDD archive, <http://kdd.ics.uci.edu>; 1999 [accessed 10th July 2006].

Julisch K. Clustering intrusion detection alarms to support root cause analysis. ACM Transactions on Information and System Security (TISSEC) November 2003; 6(4):443–71.

Lippmann R, Webster SE, Stetson D. The effect of identifying vulnerabilities and patching software on the utility of network intrusion detection. In: Proceedings of recent advances in intrusion detection (RAID'02). Lecture notes in computer science, vol. 2516. Springer Verlag; 2002. p. 307–26.

Mahoney MV, Chan PK. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In: Proceedings of the recent advances in intrusion detection (RAID'03). Lecture notes in computer science, vol. 2822. Springer Verlag; September 2003. p. 220–37.

McHugh J. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. ACM Transactions on Information and System Security 2000;3(4):262–94.

Morin B, Hervé H. Correlation of intrusion symptoms: an application of chronicles. In: Proceedings of the recent advances in intrusion detection (RAID'03). Lecture notes in computer science, vol. 2822. Springer Verlag; September 2003. p. 220–37.

Mukkamala S, Janoski G, Sung A. Intrusion detection using neural networks and support vector machines. In: Proceedings of international joint conference on neural networks (IJCNN '02); May 2002. p. 1702–7.

Munson JC, Wimer S. Watcher: the missing piece of the security puzzle. In: Proceedings of the 17th annual computer security applications conference; December 2001. p. 230–9. Available at: <http://www.acsac.org>.

Portnoy L, Eskin E, Stolfo S. Intrusion detection with unlabeled data using clustering. In: ACM workshop on data mining applied to security; November 2001.

Sekar R, Gupta A, Frullo J, Shanbhag T, Tiwari A, Yang H, et al. Specification based anomaly detection: a new approach for detecting network intrusions. In: Proceedings of the ACM conference on computer and communications security; 25–29 October 2002. p. 265–74.

Sequeira K, Zaki M. ADMIT: anomaly-based data mining for intrusions. In: Proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining; 23–26 July 2002. p. 386–95.

Venter HS, Eloff JHP. A taxonomy for information security technologies. Computers and Security 2003;22(4):299–307.

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, 9–14 June 2003; p. 138–47.

Zhang T, Ramakrishnan R, Livny M. BIRCH: an efficient data clustering method for very large databases. In: SIGMOD record 1996 ACM SIGMOD international conference on management of data, vol. 25(2); 4–6 June 1996. p. 103–14.