Knowledge-Based Systems 70 (2014) 103–117

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks

Wei Wang [a], Thomas Guyet [b], René Quiniou [b], Marie-Odile Cordier [b], Florent Masseglia [c], Xiangliang Zhang [d,*]

[a] School of Computer and Information Technology, Beijing Jiaotong University, China
[b] Research Team DREAM, INRIA Rennes/IRISA, France
[c] Research Team Zenith, INRIA, France
[d] Division of Computer, Electrical and Mathematical Sciences & Engineering, King Abdullah University of Science and Technology (KAUST), Saudi Arabia

ARTICLE INFO

Article history:
Received 24 November 2013
Received in revised form 28 April 2014
Accepted 11 June 2014
Available online 22 June 2014

Keywords:
Anomaly detection
Intrusion detection
Adaptive system
Network security
Web security
Autonomic computing
Clustering

ABSTRACT

In this work, we propose a novel framework of autonomic intrusion detection that fulfills online and adaptive intrusion detection over unlabeled HTTP traffic streams in computer networks. The framework holds potential for self-managing: self-labeling, self-updating and self-adapting. Our framework employs the Affinity Propagation (AP) algorithm to learn a subject's behaviors through dynamical clustering of the streaming data. It automatically labels the data and adapts to normal behavior changes while identifies anomalies. Two large real HTTP traffic streams collected in our institute as well as a set of benchmark KDD'99 data are used to validate the framework and the method. The test results show that the autonomic model achieves better results in terms of effectiveness and efficiency compared to adaptive Sequential Karhunen–Loeve method and static AP as well as three other static anomaly detection methods, namely, *k*-NN, PCA and SVM.

1. Introduction

Network-borne attacks are currently major threats to information security. As an important technique in the defense-in-depth network security framework, intrusion detection has become a widely studied topic in computer networks in recent years. In general, the techniques for intrusion detection can be categorized as signature-based detection and anomaly detection. Signature-based detection (e.g., Snort [1]) relies on a database of signatures from known malicious threats. This is similar to the way most anti-virus software detects malware or other anomalous behavior. Clearly signature-based detection can only detect known attacks. Anomaly detection, on the other hand, defines a profile of a subject's normal activities and attempts to identify any unacceptable deviation as a potential attack. Any observable behavior of a system, e.g., a network's traffic [2,3], a computer host's operating system [4,5] or a mobile application [6,7], can be used as the subject information.

While signature-based detection can only recognize known attacks, anomaly detection holds great potential for detecting unforeseen intrusion attempts. As new attacks appear very frequently and signature-based detection methods may be overwhelmed by polymorphic attacks, using anomaly detection sensors to discover zero-day attacks has become a necessity rather than an option [8].

1.1. Problem statement

An ideal learning-based anomaly IDS (Intrusion Detection System) can be considered as a black box. It learns normal activities of a subject in an automated fashion. Security officers and system administrators thus have no need to manually analyze a large amount of applications. Once enough normal training data has been provided, analyzed and the profiles has been constructed, the IDS switches to detection stage. Although the efforts on anomaly detection have resulted in impressive progress, anomaly IDSs have several difficulties in practice.

First, a large amount of precisely labeled data for the training is very difficult to obtain in practice. Precise labeling is a very hard task and the data is often forbidden for labeling due to privacy policies. As Gates and Taylor [9] and Cretu et al. [8] pointed out,

* Corresponding author.
E-mail address: xiangliang.zhang@kaust.edu.sa (X. Zhang).
URL: http://mine.kaust.edu.sa (X. Zhang).

http://dx.doi.org/10.1016/j.knosys.2014.06.018
0950-7051/© 2014 Elsevier B.V. All rights reserved.

the problem of unavailable labeled training data sets is a key roadblock to the construction of an effective anomaly IDS.

Second, data for intrusion detection is typically streaming and dynamic. Normal behavior of a subject has become more and more polymorphic and it is thus impossible to collect a large amount of representative training data sets at once to train a detection model. A practical solution is to keep the detection models always updated by incorporating new incoming labeled data as soon as it is classified during the detection. This process of online learning guarantees that the most recent behavior of a subject can be incorporated into the models. However, many existing anomaly detection methods involve off-line labeling and off-line learning, where data is collected, manually labeled and then fed to a learning method to construct static detection models.

Third, many current anomaly detection approaches assume that the data distribution is stationary and the model is static accordingly. The static detection models have no ability to adapt to normal behavioral changes. In practice, however, data involved in current network environments evolves continuously and normal behavior of a subject may change over time. An effective anomaly detection method, therefore, should be able to track such changes and to deal with the problem of "concept drift" [10]. In other words, the models should be automatically re-built to adapt to normal behaviors when there is a change detected.

1.2. Solution outline

This paper tries to deal with the above three issues in anomaly intrusion detection. We propose a framework to fulfil autonomic intrusion detection that detects intrusions in an online and adaptive fashion through dynamic clustering of audit data streams. Specifically we mainly detect web attacks with HTTP traffic Streams in this paper. A benchmark data set (i.e., KDD'99 [11]) is also used to further evaluate our method.

Autonomic computing is proposed by IBM in 2001 [12,13]. It refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users [13]. Similar to autonomic computing, autonomic intrusion detection also works in a fashion of self-managing. For detecting attacks in unlabeled audit data streams, an autonomic IDS has abilities of:

- self-labeling: automatically identifying the anomalies in unlabeled massive audit data streams (deal with the first issue);
- self-updating: continuously updating the detection model by incorporating incoming labeled normal data in order to maintain an accurate detection model over time (deal with the second issue);
- self-adapting: adapting the detection model to behavioral changes by re-building the model as soon as a change is detected in data streams (deal with the third issue).

The self-updating ability consists in **updating the detection model** to take into account the normal variability of the data. On the opposite, self-adapting consists in **rebuilding the detection model** in case of behavioral changes.

Our framework is under **an assumption of rareness of anomalous behavior in a long run of a network or a system**: normal data is usually very common while abnormal data is relatively rare in practical detection environments. For example, we only found 3 attacks in a HTTP traffic stream collected on the main HTTP server of our institute during 10 days. Normal data items thus form relatively condensed clusters (i.e., the mean distance between all the data items and the cluster center is small) while abnormal data forms sparse clusters (i.e., small number of data items or data items sparsely distributed in a cluster). Hence, we iteratively "capture" the anomalies in audit data streams with clustering techniques.

In our framework, the **detection model** is a set of clusters of normal audit data items. A data item is an object defined to detect whether it is normal or anomalous. A network connection or a HTTP request is an example of a data item. Any incoming data item that deviates much from the current detection model is considered as a suspicious item and suspected to be an attack. A suspicious item can be a variant of normal data items due to concept drift or an attack. To refine our diagnosis, we thus define three states of a data item: *normal*, *suspicious* and *anomalous*. If a *suspicious* item is identified, it is then put into a **reservoir**. Otherwise, the detection model is updated with the current incoming data. The detection model is rebuilt upon a behavioral change is detected. A *suspicious* item is considered as real *anomalous* if it represents a suspicious cluster center or it belongs to a suspicious cluster after the model is rebuilt. This rule can also be interpreted as that "a *suspicious* item is considered as real *anomalous* if it is marked as *suspicious* again after the model is rebuilt".

1.3. Contributions

Our contributions are summarized in the following four aspects.

- We propose a novel framework for autonomic intrusion detection that works in complex and dynamic network environments. To the best of our knowledge, this is the first time a framework of autonomic intrusion detection is introduced. It "captures" anomalous behaviors by refining suspicious items in data streams with cluster techniques. Our autonomic IDS has the ability of self-managing: self-labeling, self-updating and self-adapting. It provides potential solutions to the three key issues of traditional anomaly detection methods.
- We employ a relatively new clustering algorithm (Affinity Propagation) AP [14] and its extension [15] in streaming environments to dynamically detect anomalous behavior based on our framework. The detection models consists of a smaller set of exemplars. The efficiency of anomaly detection is thus improved because of the compressed model. We use an effective forget mechanism to add weights to the recent behavior while reducing weights to the past so that the model is always tracking the current factual circumstances for effective detection. We also design several effective mechanisms for behavioral change detection.
- While most existing anomaly detection methods classify events as either *normal* or *anomalous*, as a mechanism for autonomic detection, we define the third status of events as *suspicious* which is between *normal* and *anomalous*. We thus further refine the *suspicious* events in which real *anomalous* can be identified with higher accuracy to reduce false positives while real *normal* can also be recognized to update the normal detection models.
- We test the framework of autonomic detection on two large real HTTP traffic streams as well as on a benchmark KDD'99 data. One HTTP stream contains rare attacks and the other contains bursty attacks. The test results show that by dynamic clustering the HTTP log streams and the network connections in KDD'99 data, our autonomic method is better than three other traditional static anomaly detection methods in terms of effectiveness and efficiency, for detecting rare attacks as well as bursty attacks.

The remainder of this paper is organized as follows. Section 2 introduces related work. Section 3 describes the framework and the method of autonomic intrusion detection. Section 4 describes the experiments. Discussion and analysis are presented in Section 5 and conclusion follows in Section 6.

2. Related work

Anomaly intrusion detection has been an active research area. The related work of this paper mainly falls into two aspects. One is anomaly intrusion detection and web attack detection and the other is adaptive or online anomaly intrusion detection.

2.1. Anomaly intrusion detection in general and web attack detection

Early studies [16] on anomaly detection mainly focused on modeling system or user behavior from monitored system log or accounting log data, including CPU usage, time of login, and names of files accessed. Schonlau and Theus [17] detected masquerades by building normal user behavior using truncated command sequences. System call sequences can also be used as a data source for the detection of anomalous program behavior. In 1996, Forrest et al. [4] introduced an anomaly detection method called time-delay embedding (tide), based on monitoring system calls invoked by active and privileged processes. In our previous work, we used Non-negative Matrix Factorization (NMF) [18], Principal Component Analysis (PCA) [19] as well as Self Organizing Maps (SOM) [5] to detect intrusions based on system calls or command sequences.

Network traffic has been widely used for anomaly intrusion detection. Wang and Stolfo [20] used payload of network traffic to detect anomalies. Lee et al. [2] extracted 41 attributes for each network connection and formed a well known KDD Cup 1999 data [11]. Many research groups [2,19] have used the KDD'99 data to validate their detection methods. Gan et al. [21] proposed a combined algorithm based on Partial Least Square (PLS) feature extraction and Core Vector Machine (CVM) algorithm for intrusion detection with KDD'99 data.

Krüegel and Vigna [22] are the first who used HTTP traffic (web server log files) to detect attacks against web servers and web-based applications. They investigated client queries with the parameters contained in these queries to detect potential attacks by six methods (e.g., length and structure of the parameters). Ingham and Inoue [23] collected a set of HTTP attacks and introduced a framework to compare different anomaly detection techniques for HTTP. Song et al. [24] used a mixture-of-markov-chains model for anomaly detection in web traffic. They used n-gram method to characterize the distribution of content and structure present within web-layer script input strings. Ariu et al. [25] proposed HMMPayl based method for detecting web-based attacks. HMMPayl used payload represented by a sequence of bytes as data input and used Hidden Markov Models (HMM) for the analysis. Lee and Kim [26] proposed a WarningBird system to detect suspicious URL for Twitter. The system considered correlated redirect chains of URLs included in a number of tweets. Razzaq et al. [27] proposed an ontology based approach that specifies web application attacks using the context of consequence, semantic rules and specifications of application protocols by analyzing the specified portion of user requests.

In order to countermeasure attacks at application layer, Chan et al. proposed a policy-enhanced Adaptive Neuro-Fuzzy Inference System (ANFIS) model [28] as well as Fuzzy Association Rule Model (FARM) [29] to detect the XML and SOAP related attacks at application layer. Suriadi et al. [30] attempted to defend web services from DoS attacks using client puzzles that is a cryptographic countermeasure providing a form of gradual authentication by requiring the client to solve some computationally difficult problems before access is granted. Sangeetha et al. [31] developed a Fuzzy Rule-Base Based Intrusion Detection System on Application Layer which works in the application layer of the network stack.

All the above detection models are static (i.e., no update process and no adaptivity to behavioral changes) and need labeled data for training (i.e., no ability of self-labeling). Clustering algorithms have been used for anomaly intrusion detection over unlabeled data. As unsupervised learning algorithms, clustering methods are able to discover rare events which can be considered as "anomalies" assuming that normal behavior is very common while anomalies usually lie in sparse regions of the feature space in practical detection environments [32–34]. Eskin et al. [32] used several unsupervised methods, namely, cluster based estimation, k-NN and one class Support Vector Machines (SVM) for network intrusion detection. Portnoy et al. [33] employed single-linkage clustering method to detect intrusions over unlabeled data. Leung and Leckie [34] introduced a density-based and grid-based clustering algorithm for unsupervised anomaly detection. Though these clustering methods detected anomalies over unlabeled data, all the detect models are static. The goal of this paper is to develop a dynamic detection model working in a fashion of autonomic computing.

In order to obtain a large amount of clean training data, Cretu et al. [8] tried to sanitize training data by combining the "micro-model" in a voting scheme with which some attacks in the training data can be discovered and filtered out. While this technique can be considered for automatically labeling the data, it did not provide an adaptive detection model.

2.2. Adaptive and online anomaly intrusion detection

There are some related work on adaptive or online intrusion detection. Cretu et al. [35] proposed a method that tries to allow the detection sensor to adapt to behaviors of the protected host during the training phase. Rehák et al. [36] developed a framework for online monitoring and optimization of IDS based on the insertion of network traffic. Rassam et al. [37] proposed two anomaly detection models, namely, Principal Component Classifier-based Anomaly Detection (PCCAD) and adaptive PCCAD for static and dynamic environments. Both models utilize Principal Component Classifier (PCC) to measure the dissimilarity between sensor measurements in the feature space. The adaptive PCCAD model incorporates an incremental learning method that tracks the dynamic normal changes of data streams collected by real sensor network projects in the monitored environment. Our framework is different from Rehák et al.'s work as our framework has the ability of data self-labeling. Instead of building micro-models for voting to sanitize the data in [35], our method automatically labels the data by dynamic clustering. The main difference between our work and Rassam's work [37] lies in that our framework is always kept updated while their models only provide adaptation.

Yu et al. [38] proposed an adaptive tuning IDS that can be tuned based on the detection performance. The detection model is represented by sets of rules and tuning amounts to adjusting confidence values associated with each rule. While this work is somewhat related to ours in that the detection model is adaptively tunable, it is tuned according to the feedback provided by the operators (administrators) when false alarms are identified. In contrast, our model is autonomically kept adapted and continuously kept updated. Also, our detection model is represented by compressed clusters of normal data other than the rules. While Yu et al.'s system needs a large amount of well-labeled data, our system has ability of self-labeling that is an important merit for an automated IDS.

Maggi et al. [39] proposed a technique to discriminate between legitimate changes and anomalous behaviors in web applications. Responses in HTTP traffic are analyzed to find new and previously unmodeled parameters. This information is then used to update request and session models. Though this system shares the element of updating normal profiles with our work, the method needs a large amount of well-labeled data for initial training. In contrast, our model is designed for autonomic use and has the ability of labeling the data.

A recent effort on dealing with unavailability of large training set for anomaly detection has been proposed in [40]. In this work, the values of the parameters extracted from HTTP requests are categorized according to their type (e.g., integer, string). Parameters of similar type are then used to induce similar models of normal behavior so as to construct a knowledge base of well-trained models in case of undertraining. Though this work is related to our model in that both work is to address deficiencies of well-labeled training data, the method has no ability of labeling the data. While the method is assumed to use in a static environment where data distribution is stationary, our model deals with the problem of "concept drift" in a dynamic environment by self-updating and self-adapting.

3. The framework of autonomic intrusion detection

3.1. The general framework

The framework works through an unsupervised clustering algorithm over data streams [41]. With a clustering method, similar data items are grouped together, forming a cluster. In the remainder of this paper, we use term *exemplar* to represent a cluster center. Note that an *exemplar* is a real data item in the cluster (other than a mean cluster center as in k-means).

Our method adaptively detects anomalies with the following three steps (see the pseudo code shown in Algorithm 1 and in Fig. 1). The main notation and terminology used in the framework are listed in Table 1, in which the first 4 notations are for the detection models and others are parameters for the detection process.

- **Step 1. Building the initial model with an online clustering algorithm.** The first bunch of data stream is clustered and the *exemplars* as well as their associated items are thus obtained with the clustering algorithm. Suspicious items are identified by some criteria (see Section 3.3) and then put into a reservoir.
- **Step 2. Identifying suspicious items and updating the model in data streaming environments.** As the stream flows in, each incoming data item is compared to all the exemplars. If the item is largely far from the nearest exemplar, the item will be immediately flagged as anomalous. If the item is far (but not largely far) from the nearest exemplar, it is marked as *suspicious* and then put into the reservoir. Otherwise the item is regarded as *normal* and the model is updated accordingly with the normal item.
- **Step 3. Rebuilding the model and identifying attacks.** Upon a rebuilding criterion is triggered, the detection model is rebuilt with the current exemplars and the suspicious items in the reservoir, using the clustering algorithm again. *Anomalous* is identified if a *suspicious* item remains as *suspicious* after the model is rebuilt.

Algorithm 1. Pseudo code of the framework

```
Audit data stream $e_1, \ldots, e_t, \ldots$; fit threshold $N_{size}, \varepsilon, \lambda$
Clustering $(e_1, \ldots, e_T)$ with a clustering algorithm//
  initialization
Reservoir = {} //initialization
if $n_i < N_{size}$ or $\mu_i \geqslant \varepsilon$ then
    Reservoir $\leftarrow$ all items associated to $e_i$ //suspicious items
end if
for $t > T$ do
  //process each new arriving items $e_t$
  find $e_i$ which is the nearest exemplar to item $e_t$
  if $d(e_i, e_t) > \lambda$ then
    $e_t$ is identified as an anomaly//$e_t$ largely deviates from the
    normal model
  else if $d(e_i, e_t) < \varepsilon$ then
    Update the model//$e_t$ has been considered as a normal
    item
  else
    Reservoir $\leftarrow e_t$ //$e_t$ is considered as suspicious
  end if
  if change detected then
    //rebuilding process is triggered
    Rebuild the model (Re-clustering)
    Consider each exemplar $e_j$ in Reservoir
    if $e_j$ appears at least twice in Reservoir then
        all the items associated to $e_j$ are flagged as anomalies
    end if
  end if
end for
```

While our framework could be based on any *data stream* clustering algorithms, in this paper we use a novel clustering algorithm, Affinity Propagation (AP) [14] and its extension [15,42] in streaming environments. Clustering large-scale *streaming data* is currently a new research topic [43,44] and developing an effective and efficient *streaming data clustering algorithms* is still an important issue [43]. AP clustering has no need to define the number of clusters beforehand and this is an important advantage for autonomic intrusion detection because it is very difficult to have *a priori* knowledge for the data, especially for a very large amount of streaming data that always evolves over time.

3.2. Building initial detection models with AP

Affinity Propagation (AP) [14] is employed to build initial detection models. Let $\mathcal{E} = \{e_1, \ldots, e_N\}$ be a set of data items, and let $d(e_i, e_j)$ denote the distance (e.g., an Euclidean distance) between items e_i and e_j:

$$d(e_i, e_j) = \|e_i - e_j\| \tag{1}$$

The fitness function is defined by:

$$\mathbf{E}(c) = \sum_{i=1}^{N} S(e_i, e_{c(i)}) \tag{2}$$

where $c(i)$ is the index of the *exemplar* representing the item e_i in a cluster; $S(e_i, e_j)$ is set to $-d(e_i, e_j)^2$ if $i \neq = j$ and is set to a small constant $-s^*$ ($s^* \geqslant 0$) otherwise. $-s^*$ represents a preference that e_i itself be chosen as an exemplar. AP finds the mapping **c** maximizing the fitness function $\mathbf{E}(c)$ defined by (2) to cluster the data items. The resolution of this optimization problem is achieved by a message passing algorithm [14].

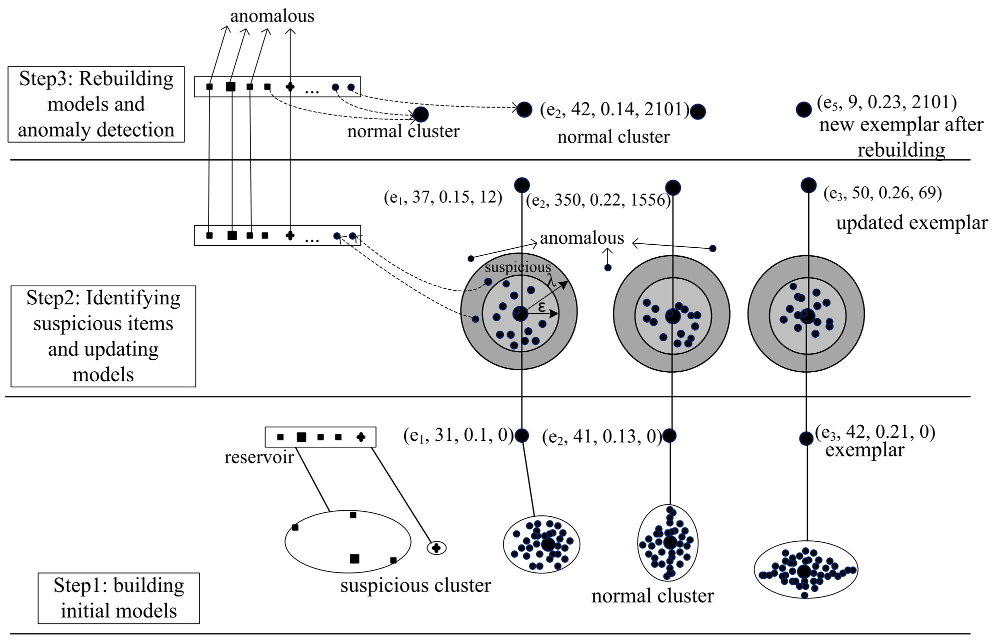

Fig. 1. Three steps of the framework of our autonomic detection model. After initial clustering, the detection models are represented by a set of clusters with 4-tuples. For example, a detection model is denoted by $(e_2, 41, 0.14, 0)$ where e_2 is the exemplar of a normal cluster, 41 is the number of items belonging to the cluster, 0.13 is the mean distance between e_3 and its associated items, 0 is the timestamp the cluster is lastly updated (here it's 0 because of the initial clustering). In step 2, the model is updated when a new *normal* item is assigned to the model (i.e., the distance between the new item and e_2 is smaller than ε). In this example, a large number (309) of date items have been assigned to this cluster when the timestamp is 1556. The models will be rebuilt in Step 3 once a behavioral change is detected when the timestamp is 2101. The number of items that belong to exemplar e_2 has become smaller again (42), and this indicates that the clustering of streaming data is dynamic. Suspicious items will be identified as either real anomalous or normal after the models are rebuilt.

Table 1
Main notations and terminologies used in the framework.

e_i	The exemplar of the cluster i
n_i	The number of items associated to exemplar i (i.e., the number of items in the cluster i)
μ_i	The mean distance between exemplar e_i and all its associated items
t_i	The last timestamp when an item was assigned to e_i (i.e., the timestamp when a cluster has lastly been updated)
N_{size}	Threshold for identification of suspicious items: minimum number of items for forming a normal cluster
ε	Threshold for identification of suspicious items: maximum distance between a normal item and its nearest exemplar or maximum mean distance between a normal exemplar and all its associated items
λ	Threshold for immediate anomaly identification
$N_{reservoir}$	Parameter for rebuilding criterion: the number of suspicious items in the reservoir
r	Parameter for rebuilding criterion: the percentage of suspicious items since last clustering
δ	Parameter for rebuilding criterion: time window length
Δ	Parameter for forgetting mechanism: time window length in which no item is newly assigned to an exemplar

In practical use, there may be some items that are exactly the same in the audit data. In our study, we use Weighted AP (WAP) that adds the multiple-appear items more weights to let one of them have more probability to be an exemplar [15]. Let data set $\mathcal{E}' = \{(e_i, n_i)\}$ involve n_i copies of item e_i, for $i = 1, \ldots, L$. WAP considers the similarity matrix defined as:

$$S(e_i, e_j) = \begin{cases} -n_i d(e_i, e_j)^2 & \text{if } i \neq j \\ -s^* & \text{otherwise} \end{cases}$$

Unlike K-means or K-centers, AP has no need to pre-define how many exemplars or clusters exist in the data. Instead, AP specifies the penalty s^* for allowing an item to become an exemplar. Note that for $s^* = 0$, the best solution is the trivial one, selecting every item as an exemplar.

Definition 1. The **detection model** is a set of clusters after initial clustering is finished. Each cluster is represented by a 4-tuple (e_i, n_i, μ_i, t_i), where the parameters are all defined in Table 1. μ_i is calculated as

$$\mu_i = \frac{1}{n_i}\sum_{j=1}^{n_i} d(e_i, e_j) \tag{3}$$

where e_j ranges over all items associated to exemplar e_i.

3.3. Identifying anomalies as well as suspicious items and updating the models

After the first bunch of data stream is clustered, we identify the suspicious items by looking at the size and the sparseness of each cluster. If the size of a cluster is very small (i.e., few items are associated to an exemplar ($n_i < N_{size}$), or the cluster is very sparse (i.e., the mean distance between an exemplar and all its corresponding items is very big ($\mu_i \geqslant \varepsilon$)), all the items in the cluster are marked as *suspicious*. N_{size} and ε are two pre-defined thresholds defined in Table 1. The suspicious items are then put into a reservoir for further investigation. A suspicious item may be changed as *normal* or as *anomalous* later.

For each new incoming item e_t at time t, its nearest exemplar e_i is found. If $d(e_t, e_i)$ is very large, say larger than a predefined threshold λ (i.e., $d(e_t, e_i) \geqslant \lambda$), e_t will be immediately flagged as anomalous. If $d(e_t, e_i)$ is large but not very large (i.e., $\varepsilon < d(e_t, e_i) < \lambda$), e_t is identified as suspicious and then put into the reservoir. Otherwise, e_t is considered as normal and is assigned to the i-th cluster and the model is **updated** as follows:

$$e_i = e_i \quad \text{(the exemplar remains the same)}$$
$$n_i = n_i \times \left(\frac{\triangle}{\triangle + (t - t_i)} + \frac{1}{n_i + 1}\right) \quad \text{(the number of items increases)}$$
$$\mu_i = \frac{\mu_i \times n_i + d(e, e_i)}{n_i + 1} \times \left(\frac{\triangle}{\triangle + (t - t_i)}\right) \quad \text{(update the mean distance)}$$
$$t_i = t \quad \text{(update the time when the model is last updated)}$$

where $\frac{\triangle}{\triangle+(t-t_i)}$ is a **forgetting factor**. As the environment is always evolving, in order to keep updated the exact current behavior, we need to put more weights on the recent models while reducing weights on the past. The above update rules with the forgetting factor enforce the stability of the model. The proof of this proposition is given in Appendix A. In order to avoid the impact of outdated normal behavior, if an exemplar e_i has never been assigned with a single item in a time window $\triangle$, the exemplar is simply reset as a common item:

$$e_i = e_i; \quad n_i = 1; \quad \mu_i = 0$$

Note that $\triangle$ or $(t - t_i)$ is not a real time interval. It denotes the number of items flowing in during the period between the two timestamps.

3.4. Rebuilding the model and identifying attacks

The detection model should be rebuilt if a behavioral change is detected (i.e., a rebuilding criterion is met). In our framework, we use three criteria to trigger the rebuilding process. The model is rebuilt if any of the following three criteria is met:

- the number of incoming suspicious items exceeds a pre-defined threshold $N_{reservoir}$,
- the time window length exceeds threshold δ after the latest clustering,
- the percentage of suspicious items since the latest clustering exceeds threshold r.

The first two criteria indicate gradual change of the detection models while the third describes a sudden change. Large percentage of suspicious items means that there are many suspicious items in a short time and this indicates a sudden change in the audit data.

Upon a rebuilding criterion is triggered, the model will be rebuilt with the current detection model and all the suspicious items in the reservoir. Given a suspicious item, WAP is to cluster $\{(e_i, n_i)\} \cup \left\{\left(e'_j, 1\right)\right\}$, where (e_i, n_i) denotes an exemplar of the current detection model and e'_j is a suspicious item in the reservoir. The adaptation process for model rebuilding is defined as [42]:

$$S(e_i, e_i) = -s^* + \Sigma_i d(e, e_i) \quad S(e_i, e_j) = -n_i d(e_i, e_j)^2$$
$$S\left(e_i, e'_j\right) = -n_i d\left(e_i, e'_j\right)^2 \quad S\left(e'_j, e_i\right) = -d\left(e_i, e'_j\right)^2$$
$$S\left(e'_j, e'_j\right) = -s^*$$

We investigate the suspicious items in the reservoir again after rebuilding the cluster. If an exemplar in the reservoir appears at least twice, the status of the exemplar as well as all the items associated with the exemplar is changed from *suspicious* to *anomalous* and they are thus identified as anomalies.

3.4.1. Discussion

While most existing work identifies a data item as either *normal* or *anomalous*, in this work, we define its third status as *suspicious*. It is seen from Fig. 1 (concentric circles in Step 2) that suspicious is a buffer zone located between the *normal* and *anomalous*. Instead of identifying suspicious items directly as normal or anomalous, our model refines the diagnosis. Our model differentiates real anomalous from drifted normal behavior. Normal behavior of a subject may vary a lot. Suspicious that occasionally occurs during a short interval may be a normal event with some variance. This often happens when normal behavior is drifting. The suspicious items in this circumstance should not be immediately identified as anomalous. However, if a suspicious item remains during a relatively long period (e.g., at the time the model is rebuilt), it is anomalous with a high probability.

4. Experiments and results

In order to evaluate our autonomic model, we compare our dynamic AP model to another online adaptive method called adaptive Sequential Karhunen–Loeve (SKL) [45], to static AP method as well as to other three static methods:

- *k*-NN (*k*-Nearest Neighbor) model [46,47],
- PCA (Principal Component Analysis) model [19],
- one class SVM model [48].

The detection mechanism of static AP is the same as that of the unsupervised clustering methods reported in [32,33]. Static AP is the special case of our dynamic AP model if all the data streams are clustered by AP once only.

ROC (Receiver Operating Characteristic) curves are used to evaluate the detection performance of different intrusion detection methods. A ROC curve is the plot of True Positive Rate (TPR), calculated as the percentage of attacks detected, against False Positive Rates (FPR), calculated as the percentage of normal items falsely classified as anomalous. There is a tradeoff between the TPR and FPR. The ROC curves are obtained by setting different thresholds (e.g., ε_1 for *k*-NN, ε_2 for PCA). Points nearer to the upper left corner of the ROC curve are the most desirable.

For sake of completeness, we briefly introduce the static methods in Section 4.1. We describe the adaptive SKL method in Section 4.2. Then, in Section 4.3 we present the two real HTTP log streams used in the experiments, the preprocessing step and the experimental results. We describe the experimental results on KDD'99 data in Section 4.4.

4.1. Static models

4.1.1. k-NN based static anomaly intrusion detection

For all the static detection methods, normal data is used as the training set and it is assumed that the normal behaviors are embedded in the normal data sets and anomalous behaviors are different from normal behaviors.

For a given k, k-NN ranks the distances between a test data item and its neighbors among the normal training samples [46]. Euclidean distance is usually used as the distance metric.

Given a test item, the Euclidean distance between the test item and each item in the training data set is calculated. The distance scores are sorted and the k nearest neighbors are chosen to determine whether the test item is normal or not. In anomaly intrusion detection, we average the k closest distance scores as the *anomaly index*. If the *anomaly index* of a test item is above a threshold ε_1 , the test item is then classified as abnormal. Otherwise it is considered as normal.

It is clear that the k-NN method is the special case of AP method if all the items for clustering with AP are set as exemplars (i.e., $s^* = 0$). The difference is that k-NN is static while our method with AP is dynamic. It is thus meaningful to compare these two intrusion detection methods based on the same audit data streams.

4.1.2. PCA based static anomaly intrusion detection

PCA [49] is a widely used dimensionality reduction techniques. Given a set of observations $X_1, \ldots, X_n$, suppose each observation is represented by a row vector of length m, the data set is thus represented by a matrix $X_{n\times m}$. The average observation is defined as $\Psi = \frac{1}{n}\sum_{i=1}^{n} X_i$. Observation deviation from the average is defined as $\Phi_i = X_i - \Psi$. The sample covariance matrix of the data set is defined as $C = \frac{1}{n}\sum_{i=1}^{n}(X_i - \Psi)(X_i - \Psi)^T$.

Suppose $(\lambda_1, u_1), (\lambda_2, u_2), \ldots, (\lambda_m, u_m)$ are m eigenvalue–eigenvector pairs of the sample covariance matrix C. We choose k eigenvectors having the largest eigenvalues. Often there will be just a few large eigenvalues, and this implies that k is the inherent dimensionality of the subspace governing the "signal" while the remaining $(m - k)$ dimensions generally contain noise [50]. The dimensionality of the subspace k can be determined by $\frac{\sum_{i=1}^{k}\lambda_i}{\sum_{i=1}^{m}\lambda_i} \geqslant \alpha$, where α is the ratio of variation in the subspace to the total variation in the original space. We form a $(m \times k)$ (usually $k \ll m$ for data reduction) matrix U whose columns consist of the k eigenvectors. The new representation of the data is obtained by projecting it onto the k-dimensional subspace according to the rules $Y_i = (X_i - \Psi)U = \Phi_i U$.

If $X_{n\times m}$ describes normal behaviors, the k-dimensional subspace spanned by k Principal Components is capturing the normal principal patterns. When reconstructing mean-adjusted X_i by $X_i' = Y_i U$, the distance between X_i and its reconstruction X_i' would be small [50]. On this property the intrusion detection model is based [19]. As PCA seeks a projection that best represents the data in a least-square sense, we use the squared Euclidean distance in the experiments to measure the distance between the two data items:

$$\varepsilon = \|X_i - X_i'\|^2 \tag{4}$$

ε is characterized as the *anomaly index*. For a given test item, if ε is below a predefined threshold ε_2, the test item is identified as normal. Otherwise it is identified as anomalous.

4.1.3. One class SVM based static anomaly intrusion detection

We use one class SVM that was proposed by Schölkopf et al. [48] for static anomaly detection. One class SVM algorithm is to map the data into a feature space $\mathcal{H}$ by a mapping function $\Phi(X)$, such that the dot product in $\mathcal{H}$ can be computed using a kernel $k(X_i, X_j) = \Phi(X_i)\cdot\Phi(X_j)$. The mapped data in $\mathcal{H}$ are separated from the origin with maximum margin using a hyperplane $\mathrm{w}\cdot\Phi(X) - \rho = 0$ while w is a weight vector and ρ is offset parameterizing the hyperplane in $\mathcal{H}$. Small $\|\mathrm{w}\|$ coresponds to a large margin of separation from the origin.

Given training vectors $X_1, X_2, \ldots, X_n$ belonging to normal class, the primal form of optimization function is

$$\min_{\mathrm{w},\xi,\rho} \quad \frac{1}{2}\|\mathrm{w}\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho,$$
$$\text{subject to} \quad \mathrm{w}\cdot\Phi(X_i) \geqslant \rho - \xi_i \quad \text{and} \quad \xi_i \geqslant 0,$$

where $\nu \in (0, 1]$ is an upper bound on the fraction of data that may be outliers.

Solving the optimization problem, the decision function is defined as

$$f(X) = sgn(\mathrm{w}\cdot\Phi(X) - \rho) \tag{5}$$

This function returns the value $+1$ in a "small" region capturing the training data, and -1 elsewhere.

In anomaly detection, we use the normal data to learn the function $f(X)$. If the decision function (5) gives a positive value for a test data item, it is classified as *normal*. Otherwise, it is considered as *anomalous*.

4.2. Adaptive sequential Karhunen–Loeve method

The Karhunen–Loeve transform is a method for approximating a set of data items in high dimension by a low dimensional subspace. It is closely related to PCA [49]. In this paper, we modified the Sequential Karhunen–Loeve (SKL) algorithm in [45] so that it suits for anomaly intrusion detection in our experiments, as the original SKL [45] is proposed for only updating the models (image database) and has no intention for anomaly detection. We call the modified algorithm as Adaptive SKL and employ it in the experiments for results comparison.

Adaptive SKL starts by calculating the Singular Value Decomposition (SVD) of the first bunch of data (we use normal data for the initial training like k-NN, PCA and SVM). The SVD decomposes a $N \times M$ matrix X represents it as a product of three matrices, $X = UDV^T$, where U is the $N \times R$ columns orthogonal matrix of the (left) singular vectors, D is $R \times R$ diagonal matrix whose diagonal elements are the singular values of X and V is an $M \times R$ orthonormal matrix whose columns are the right singular vectors, where R is the rank of matrix X. The detection model is the KL basis spanned by the k most significant left singular vectors.

In this paper, we modified the SKL [45] in the following two aspects: (1) we update the detection model with only normal data items that have been identified by the model; and (2) we adaptively rebuild the model as soon as a change is detected. To facilitate the comparison with other methods, we define the same three criteria explained in Table 1 and in Section 3.4 for change detection. Like PCA, we measure the distance between a mean-adjusted test data item with its re-construction on the subspace spanned by the KL basis. The main task of adaptive SKL is to adaptively update the KL basis by rebuilding the models in order to track the evolving normal behaviors. We obtain the detection results by adjusting the threshold ε_3 that measures the distance.

4.3. Experiments on two real HTTP traffic streams

4.3.1. Data description

Web-based security flaws represent a substantial portion of the total number of vulnerabilities [22]. It was indicated [51] that web-based vulnerabilities account for more than 25% of the total number of security flaws. Web-based attack detection is thus an important issue in computer network security. Different from many other traditional intrusion detection methods using separated data sets (e.g., data has been divided into small pieces of data sets in which time series information may be lost), in our experiments, we detect web attacks with real HTTP traffic streams. In some ways our method is analogous to an anomaly detection version of the Snort sensor [1], focusing on port 80.

In the experiments, we collected two large HTTP traffic streams on the main Apache server of our institutes. The two real traffic streams represent two attack scenarios in the networks. The first stream contains rare attacks while the second contains bursty attacks. The two log streams have been well labeled. In the first traffic stream, we only found 3 attacks distributed in the data set. We therefore collected real attack data and then randomly inserted them into the data set. The second stream is real data containing bursty attacks.

The attacks in the first traffic mainly include JS XSS attacks, input validation error, URL decoding error, SQL-injection attacks, PHP remote file inclusion attacks and DoS attacks while the second traffic streams mainly include JS XSS attacks, SQL-injection attacks and PHP remote file inclusion attacks. Fig. 2 shows examples of normal requests as well as of attacks in the traffic streams.

The data is a high speed streaming flow. For example, in less than 11 days we have collected more than 1.5 GB data including more than 5 million HTTP requests in the first traffic stream.[1] In detail, the first traffic contains 36 attack requests distributed in more than 5.7 million requests while the second contains 239 attacks occurring in a very short interval (between request 7923th and 9743th after filtering, see Fig. 3) in more than 1.4 million requests.

The two traffic streams are described in Tables 2 and 3.

Based on the first 800 normal (*attack free*) HTTP training requests, we use k-NN (set $k = 10$) to compute the average k closest distances between an incoming request and all the 800 normal requests. Fig. 3 shows the distance distribution of the requests after 800th in the second HTTP traffic stream. It is seen that the attacks are occurring in a short interval.

4.3.2. Data preprocessing

Our method examines individual HTTP requests (like Fig. 2) and models the content of script inputs. In order to reduce noise contained in the data streams, we filtered out most static requests (e.g., .html, .wav, .txt, .pdf, .swf) as well as widely known search engine robots (e.g., googlebot, Msnbot,) before the detection, because static requests cannot be attacks to the server. Note that we only filter out widely known free-attack static requests to guarantee no attack is removed from the data. The data are largely reduced after the filtering. In the first traffic stream, for example, only 4.66% of the original requests remain after the filtering (see Table 2).

In the experiments, we used character distribution of each path source in HTTP requests as the feature. Character distribution was first introduced for anomaly detection by Krüegel and Vigna [22] who computed them in a coarse way. They first sorted the 256 ASCII characters by frequency and aggregate them into 6 groups: 0, 1–3, 4–6, 7–11, 12–15, and 16–255, and compute one single uniform distribution model of these 6 segments for all attributes. Also, they only modeled the query attributes. In contrast, we model the full byte distribution of each path source (including queries) in HTTP requests. The test results in [22] showed that even the coarse character distribution features can detect most web attacks. Wang and Stolfo [20] used full character distribution of payload of network traffic for anomaly network detection and the results also showed its effectiveness.

There are 256 types of ASCII code in total but only 95 types of ASCII code (between 33 and 127) appear in the HTTP requests (unprintable characters are not allowed). The character distribution is computed as the frequency of each ASCII code in the path source of HTTP requests. For example, one HTTP request is shown as:

```
60.50.99.87 - - [02/Jan/2007:07:47:03 + 0100] ''GET
/acacia/project/edccaeteras/wakka.php?wiki=ActionOr-
phanedPages/referrers
HTTP/1.0'' 200 10753 ''http://a.js1.bosja.com/index1.htm''
''-''
```

The character distribution of printable ASCII codes of the path source "/acacia/project/edccaeteras/wakka.php? wiki=ActionOrphanedPages/referrers" is computed as:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0.014 0.069 0 0 0 0 0 0 0 0 0 0 0 0 0
0.014 0 0.014 0 0.014 0 0 0 0 0
0 0 0 0 0 0 0 0 0.014 0.014 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.125
0 0.083 0.028 0.125 0.014 0.014
0.028 0.056 0.014 0.042 0 0 0.028 0.028 0.056 0 0.0976 0.042
0.042 0 0 0.028 0 0 0 0 0 0 0 0 0 0 0
```

[1] All the preprocessed data sets as well as the programs used in this paper are available upon request to the first author.

As a consequence, each HTTP request (data item) is represented by a vector of 95 dimensions. The intrusion detection aims at identifying whether an item is normal or anomalous. There are some other different features that can be extracted from the traffic streams [22,39,40]. The character distribution can be regarded as a 1-gram method (with respect to n-gram [24]) to consider the content of the HTTP request.

4.3.3. Parameters setting

We define three parameters for the rebuilding criteria to guarantee the tolerance and effectiveness of our model: if one parameter is not well-set, another parameter can take the role. We have performed extensive experiments. Our experiments show that $N_{reservoir}$ is not sensitive to the results and it is set 300. One can set Δ (the parameter for forgetting mechanism; it is the length of a time window in which no item is newly assigned to an exemplar, see Table 1) as a large number, e.g., 10,000 or 5000, so that the weights are slowly reduced.

In most cases the users only need to define parameters r and δ. r is the parameter to identify a sudden change of a subject's behavior. We set r as 60% in the experiments based on the testing results with different tries. Besides adaptation to behavior changes, another important purpose of rebuilding the model is the detection of anomalous behaviors. Our goal is to identify the attacks as early as possible by checking the status of *suspicious* items after rebuilding the model. In our detection model, a side parameter δ is defined to trigger the rebuilding process of the model for anomaly detection even if there is no obvious behavioral change. The criterion of setting δ can be based on our assumption of rareness of abnormal data. For example, we can set N_{size} to be two orders of magnitude larger than δ, so that the occurrence probability of an attack is about 1%. As a result, we do not need to rigorously set a single parameter as we have used three parameters for rebuilding the model to control the detection process. In the experiments, we set δ as 2000 and all the parameters used in the experiments are summarized in Table 4.

Another three parameters are threshold N_{size}, λ and ε that are defined for the detection of anomalous or suspicious items. λ is set a large value so that real anomalies are correctly detected immediately without false positives. We set λ as 1.2 and set N_{size} as 4. The results presented in this paper are mainly obtained based on the adjustment of threshold ε.

Although we need to set 6 parameters in the models, as discussed above, one parameter ($N_{reservoir}$) is not sensitive to the results and the other 4 parameters (δ, Δ, N_{size} and λ) are not easy to determine. In practical use, one can also visualize the detection process (see Figs. 5 and 6) and determine the parameters accordingly.

In static models, for static AP, to facilitate comparison, we set the same parameters as used in our dynamic model. Since static AP clusters all the data once only, we only need to set λ and N_{size} and the results are also mainly obtained based on the adjustment of threshold ε.

For k-NN, we set $k = 10$ as the results are not sensitive to k based on our previous experimental results [47]. For Adaptively SKL, we define δ as 10,000 for the model rebuilding. For Adaptive SKL, PCA and k-NN, we obtain the different test results by adjusting the thresholds of *anomaly index*, defined as the distance between a test data item and the normal detection model, $\varepsilon_1, \varepsilon_2$ and ε_3 respectively. For SVM, we use the kernel as the RBF and adjust a main parameter to obtain the different results. We made our own programs for k-NN, Adaptive SKL and PCA. We used LibSVM tools (Version 2.88) [52] for SVM.

In each experiment, the first 800 HTTP requests (one attack occurs in the first 800 requests in the first traffic stream) are used to perform the initial clustering with dynamic AP. For Adaptive

(a) http://www.aweb.com/MsgPage.php?msgID=23
(b) http://www.aweb.com/ MsgPage.php?msgID=<script>Msg.location='http://www.anewweb.com/cgi-bin/cookie.cgi?'%20+document.cookie</script>
(c) http://www.aweb.com/MsgPage.php?msgID=/content/base/build/explorer/none.php?..:..:..:..:..:..:..:etc:passwd
(d) http://www.aweb.com/aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Fig. 2. Examples of normal requests and attack requests in the traffic streams. (a) A common request. (b) A JS XSS attack. It gathers the user's cookie and then send a request to anewweb.com with the cookie in the query. (c) Remote file inclusion attack. The attacker grabs the password file. (d) DoS attack.

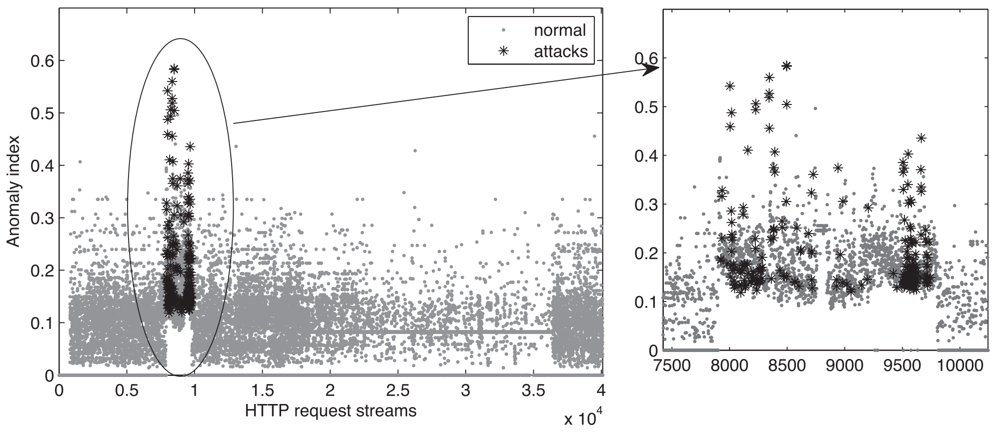

Fig. 3. Distance distribution in the second HTTP traffic stream: the average 10 closest distances between an incoming request and all the first 800 training normal requests.

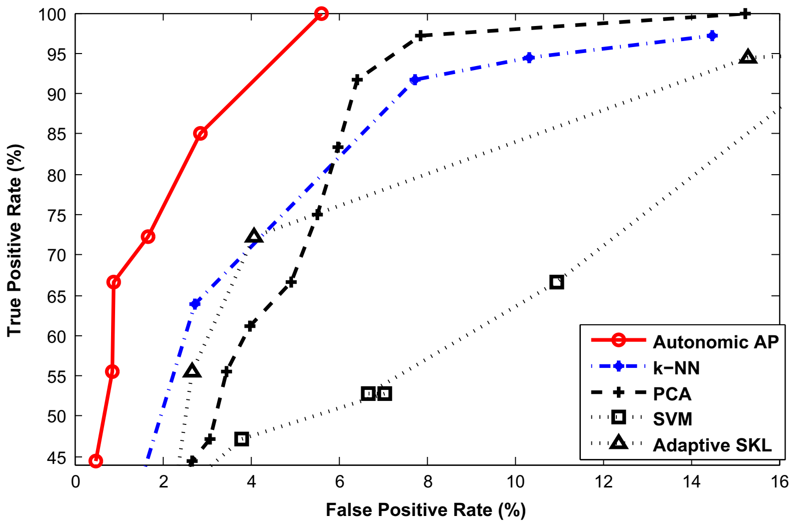

Fig. 4. ROC curves for rare attack detection with dynamic AP, adaptive SKL, *k*-NN, PCA and SVM.

Table 2
The first HTTP traffic containing rare attacks.

Before filtering		After filtering		# Normal requests	# Attack requests	Duration
File size	# Requests	File size	# Requests			
1,536 MB	5,700,949	53.5 MB	265,752	265,716	36	10 days 21 h

Table 3
The second HTTP traffic containing intensive attacks.

Before filtering		After Filtering		# Normal requests	# Attack requests	Duration
File size	# Requests	File size	# Requests			
561.2 MB	1,449,379	9.5 MB	40,095	39,856	239	3 days 3 h

Table 4
Parameters used in the experiments.

$N_{reservoir}$	r	δ	Δ	N_{size}	λ
300	60%	2000	10,000	4	1.2

SKL, k-NN, PCA as well as SVM, the first 800 **normal** (attack free) HTTP requests are used as the training data set. All the other HTTP requests are used during the detection process.

4.3.4. Test results

4.3.4.1. Rare attack detection. Table 5 and Fig. 4 compare the rare attack detection performances of our autonomic AP method with those of the static methods k-NN, PCA and SVM. Static AP is not available to detect the anomalies over the first HTTP traffic stream containing rare attacks, because the first traffic stream is too massive and AP (as well as most of clustering algorithms) suffers a lot of computations. The AP computational complexity is $N^2 \log(N)$; it involves the matrix S of pair distances, with quadratic complexity in the number N of items, severely hindering its use on large-scale datasets [53,42]. The experiments were running for several days with AP on the first HTTP traffic stream and the results were not available. The experiments with AP on the second HTTP traffic stream are reported in next Section.

In order to evaluate the real time performance, in Table 8 we list the average CPU time required for detection of all the HTTP test requests. The programs were made in MATLAB and were run on a machine with a CPU Intel dual Core 2.66 GHz, RAM 2.0G, Linux kernel 2.6.17.

As stated, it is meaningful to compare k-NN method and the autonomic AP model because k-NN is its special case. From Table 5 and Fig. 4, it is seen that the autonomic model is more effective than k-NN. While our model detects 100% of attacks with a false positive rate of 5.62% (shown in bold in the Table), k-NN detects 97.22% attacks with a higher false positive rate of 14.48%. The autonomic model is also better than PCA, adaptive SKL and SVM in terms of detection accuracy. From Table 6, it is also observed that the autonomic model is more efficient than k-NN for the detection.

Although adaptive SKL works online and adaptively, it can learn abnormal behaviors if an attack has been falsely identified as normal. The adaptive SKL thus leads to a biased model that results in higher FPRs. Our autonomic model, however, consists of exemplars (clusters) that can be considered as micro-models. The detection

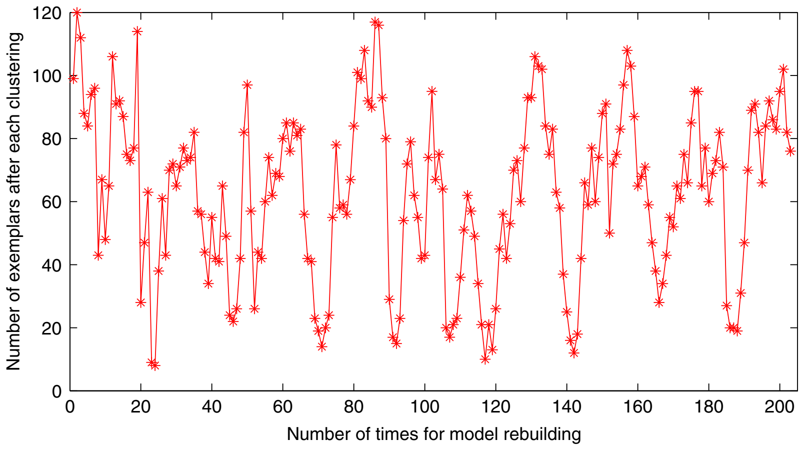

Fig. 5. The number of exemplars (clusters) after each clustering.

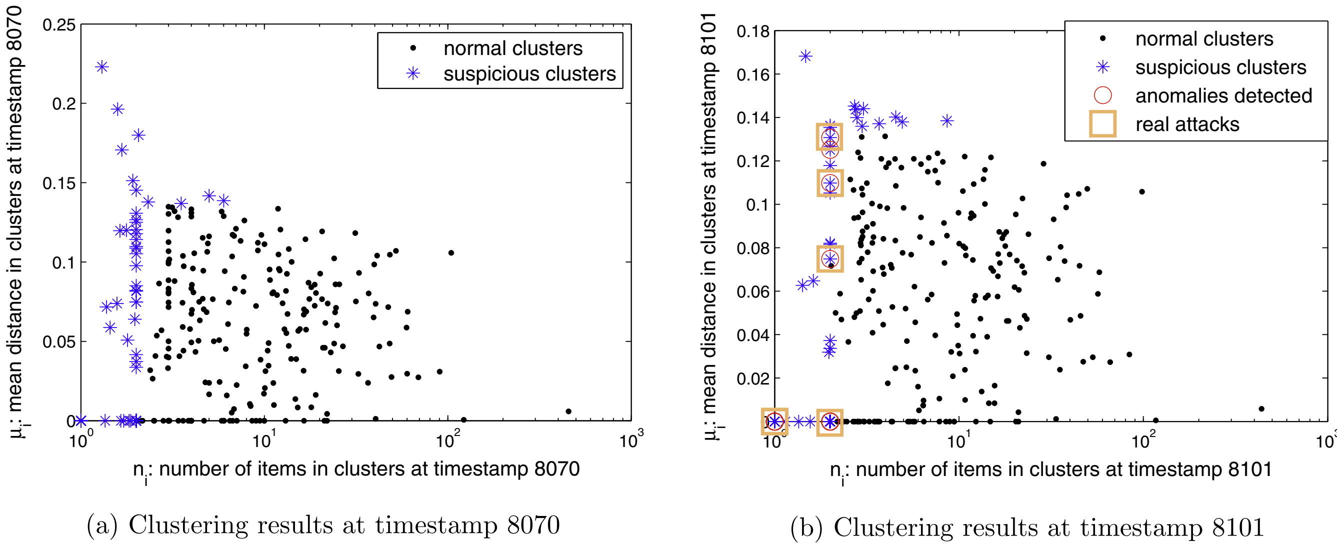

Fig. 6. Results of two adjacent clustering processes. Suspicious clusters (blue star) are identified in (a) since its size is small ($n_i < N_{size}$, here $N_{size} = 4$)) or it is sparse ($\mu_i \geqslant \varepsilon$). Suspicious clusters are identified as real anomalous (in read cycles) if they remain suspicious again after re-clustering. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 5
True positive rates and false positive rates with autonomic AP vs. *k*-NN, PCA, adaptive SKL and SVM.

Autonomic AP			*k*-NN		
ε	TPR (%)	FPR (%)	ε_1	TPR (%)	FPR (%)
0.150	**55.56**	**0.86**	0.200	38.89	1.32
0.129	**66.67**	**1.31**	0.175	55.56	2.15
0.093	**86.11**	**2.86**	0.155	86.11	4.33
0.070	**100**	**5.62**	0.118	97.22	14.48

PCA			Adaptive SKL			SVM	
ε_2	TPR (%)	FPR (%)	ε_3	TPR (%)	FPR (%)	TPR (%)	FPR (%)
0.0035	55.56	3.46	0.006	17.15	0.79	38.89	1.64
0.0015	66.67	4.92	0.003	20.92	0.17	41.67	2.62
0.00004	83.33	5.99	0.001	46.44	8.64	52.78	7.06
0.000026	100	15.23	0.0002	68.62	22.25	66.67	10.96

Table 6
Average CPU time required for detection in the first traffic stream with autonomic AP and with *k*-NN.

	AP	*k*-NN
Testing (s)	537.6	6052.9

can be effective even few clusters are biased. Moreover, we define the third status of incoming data items as suspicious. The suspicious data is not directly used for updating the models. Our model thus is more effective for online anomaly detection.

Fig. 5 shows the number of exemplars after each clustering. The number of exemplars is always changing over time. This indicates that the behavior of the data is evolving and static methods hence may not be effective. Table 7 shows that the average number of exemplars is only 64 during the detection. A large number of data items are clustered and the total number of exemplars in the data is thus much less than the number of original data items (e.g., 800 training items for static methods). Each incoming test item only needs to compare less points (e.g., 64 vs. 800) with autonomic model and a lot of computation is thus reduced.

4.3.4.2. Bursty attack detection. It is a challenge for our autonomic model to detect bursty attacks because this situation does not match well the assumption of rareness of abnormal data. There are many sudden changes when bursty attacks occur in the traffic stream as the percentage of suspicious items quickly increases in a short time. As a consequence, the parameter r is active in detecting bursty attacks while it is not in detecting rare attacks.

In the experiments, we used the same parameters (shown in Table 4) that have been used for the first traffic stream in order to test the robustness of our model in detection of the second traffic stream. To investigate the mechanism of autonomic detection, we visualize the detection process in Fig. 6. Fig. 6(a) shows the clustering results at timestamp 8070 when bursty attacks are occurring (see Fig. 3). At timestamp 8101, as the percentage (r) of suspicious items during time 8070–8101 is bigger than 60%, the rebuilding process is triggered and the re-clustering results

Table 7
Number of times of rebuilding and average number of exemplars during detection in the first traffic stream.

# Times for rebuilding	Average # exemplars
203	64

are shown in Fig. 6(b). It is seen that our model detects 11 real attacks (note that there are 7 overlapped attacks on the bottom left ($n_i = 1$ and $\mu_i = 0$)) while there is a false positive.

The ROC Curves (see Fig. 7) are obtained by adjusting threshold ε. Matlab has no capacity to run static AP on all the 40,095 HTTP requests. We then use C programs in the experiments for static AP and they work as the second HTTP traffic stream is not too large. It is observed that the autonomic model is more effective than adaptive SKL, static AP, *k*-NN, PCA and SVM for detection of most bursty attacks over the HTTP traffic streams. Table 8 shows that autonomic AP method is much more efficient than static AP for bursty attack detection. Static AP is very time consuming, making it not be suitable for the practical use in intrusion detection. Our dynamic AP is more efficient than *k*-NN as well. It's shown in Table 9 that the models have been rebuilt for 63 times for detection of all the attacks. Similarly, the average number of exemplars during the detection is only 91 and this largely reduces detection computation as an incoming data item only needs to compare itself with less data items in the normal models.

4.4. Experiments on KDD'99 data

4.4.1. Data description

In order to further evaluate our method, we use another benchmark data, KDD'99 data [11], for network intrusion detection. Although the process of data collection has been criticized [54] with some flaws, the KDD'99 data is so far probably the only large-size, available and well labeled network data source in public. We used the data in our experiments based on two reasons. First, the data has been widely used for evaluating various intrusion detection methods and our detection results can be compared with others. Second, the data provides numerous types of anomalies.

The raw data contains traffic in a simulated military network that consists of hundreds of hosts. The data includes 7 weeks of training set and 2 weeks of test set. In our experiments, we only use part of the training set. The raw training set of the data was pre-processed into about 5 million connection records by Lee et al. [2] as part of the UCI KDD archive [11]. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows from a source IP address to a target IP address under some well defined protocols [11]. In the training set, each network connection is labeled as either normal, or as an exactly one specific kind of attack. The network connection data contains 41 features among which 34 are numeric and 7 are symbolic. Only the 34 numeric features were used in the experiments. Each connection in the data set is thus transformed into a 34-dimensional vector as data input for detection. There are 22 types of attacks in total in the subset and these attacks fall into one of 4 categories: DoS: denial-of-service (e.g., teardrop); R2L: unauthorized access from a remote machine (e.g., password guessing); U2R: unauthorized access to local superuser (root) privileges by a local unprivileged user (e.g., buffer overflow attacks) and PROBE: surveillance and other probing (e.g., port scanning).

In the experiments, we used all the 97,278 normal data connection in the data set. 1000 attacks are randomly selected from the attack data and then randomly inserted into the normal data to simulate the real environments, so that the number of attacks is approximately 1% of the total number of the connections.

In the experiments, we normalize the 34 continuous attributes before detection, so that some attributes do not dominate the others. In the experiments, we use statistical normalization that converts a data into standard Normal distribution with mean zero and unit variance, as statistical normalization has been shown to be most effective in our previous work [55].

The statistical normalization is defined as

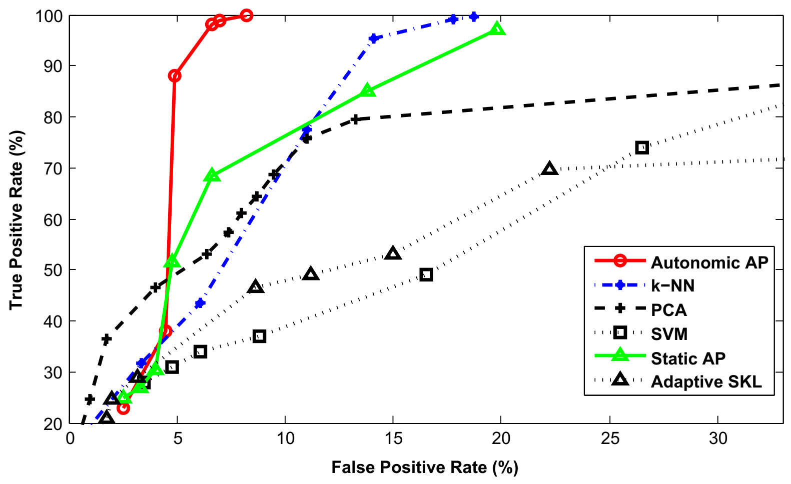

Fig. 7. ROC curves with autonomic AP model as well as with static AP, adaptive SKL, *k*-NN, PCA and SVM for bursty attack detection.

Table 8
Average CPU time required for detection in the second traffic stream with autonomic AP and with *k*-NN.

	Dynamic AP	Static AP	*k*-NN
Testing (s)	98.1	120,125	943.9

Table 9
Number of times of rebuilding and average number of exemplars in detection of the second traffic stream.

# Times for rebuilding	Average # exemplars
63	91

$$x_i = \frac{v_i - \mu}{\sigma} \tag{6}$$

where μ is mean value of a given attribute and σ is its stand deviation.

4.4.2. Testing results

In the experiments, we used the first 800 connections for initial AP clustering and used the first clean 800 normal connections as the training set for *k*-NN, adaptive SKL, PCA and one class SVM.

For the adaptive method AP, in order to test the robustness of our model, we set all the parameters as the same shown in Table 4, except the parameter λ. In the experiments, we set the λ as 13.16 and adjust parameters N_{size} and ε to obtain different detection results. The detection results and the ROC curves with the four methods are shown in Table 10 and in Fig. 8, respectively. The experiments are conducted on the same machine and the CPU time used for the detection with AP and with *k*-NN is shown in Table 11.

From Table 10 and Fig. 8, it is seen that adaptive AP obtains slightly better results than *k*-NN method. Part of the results for AP is the same as those of *k*-NN. This is because (1) the normal data of KDD'99 evolves very slowly and a static model does not lose much information contained in the data and the detection results keep good; and (2) our method uses the parameter λ and in some cases the detection mechanism is the same with *k*-NN because λ is set as a constant value. AP as well as *k*-NN is much better than adaptive SKL, PCA and SVM in terms of detection accuracy. Table 11 shows that AP method requires much less time than *k*-NN for testing all the test network connections.

Table 10
The testing results with AP as well as with *k*-NN, adaptive SKL, PCA and SVM on KDD'99 data.

AP		*k*-NN	
TPR (%)	FPR (%)	TPR (%)	FPR (%)
23.3	**0.28**	23.1	0.13
89.9	**0.57**	89.9	0.58
95.4	**0.67**	95.4	0.67
98.4	**1.01**	98.4	1.22

PCA		Adaptive SKL		SVM	
TPR (%)	FPR (%)	TPR (%)	FPR (%)	TPR (%)	FPR (%)
23.05	1.01	30.71	14.93	99.90	21.99
86.4	16.8	60.22	19.32	100	26.99
96.17	17.12	99.59	26.28		
99.30	19.59				

From Fig. 8 it is seen that the features of ROC curves produced by the five methods based on KDD'99 data are different from those on HTTP traffic streams. There are a large number of identical data items in KDD'99 data while few identical data items exist in HTTP traffic streams. Since our autonomic model as well as *k*-NN method use the nearest distances between the data items to detect anomalies, the *anomaly index* between identical items is 0, resulting in better ROC curves. In contrast, the other three methods do not directly use the distances between the data items for the detection. Since the *anomaly index* of identical data items are the same, increasing the thresholds leads to the increase of false positives while the true positive rates remain. The ROC curves of PCA, adaptive SKL and SVM, are thus first flat and then steep, as shown in Fig. 8.

5. Discussion

Detection environment always evolves over time and traditional static anomaly detection methods may not meet the practical use. For example, it is observed from Fig. 3 that the data distribution changes at around the 8000th request. Fig. 5 also shows "concept drift" of the data. Considering that in most cases audit data in anomaly intrusion detection problem is not static, we need adaptive models to adapt to behavioral changes for effective detection.

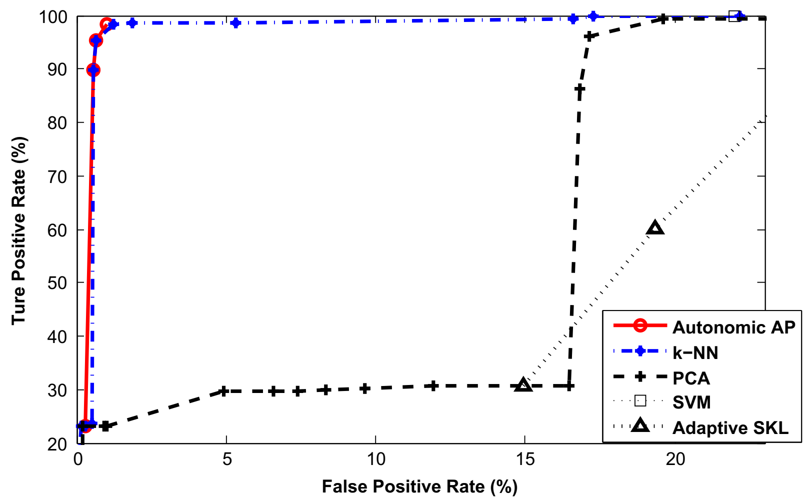

Fig. 8. ROC curves with autonomic AP as well as adaptive SKL, k-NN, PCA and SVM methods for network intrusion detection.

Table 11
The CPU time used for network intrusion detection on KDD'99 data with AP and with k-NN.

	AP	k-NN
Testing (s)	38.48	1138.6

Our autonomic intrusion detection framework holds potential to a use in complex and dynamic network environments. It detects intrusions in a fashion of self-managing: self-labeling, self-updating and self-adapting. First, it automatically labels the incoming data through dynamic unsupervised clustering on data streams. Second, it updates the detection models by incorporating new incoming clean normal data items as soon as they are determined as normal. Finally, it adapts to behavioral changes by rebuilding the models as soon as a "change" is detected.

Behind the framework there is only one assumption that normal activities are common while anomalous behavior is rare in a long run of a system or network. Eskin et al. [32] supposed that normal data outnumbers the intrusion data by a factor of 100:1. We found that in many cases this is true, although the factor may vary. For example, we only found 3 attacks in the first traffic stream collected on the main HTTP server of our institute during nearly two weeks. However, the assumption does not hold in some circumstances in networks (e.g., DoS attacks). In our model, we introduce a parameter r to trigger the rebuilding process of the model if bursty attacks are occurring. The test results show that our model is effective as well for detection of bursty attacks. As a solution, signature-based IDSs can also be incorporated to our framework as a complementary mechanism to detect and filter out large-scale intensive attacks.

Data for intrusion detection becomes increasingly massive. Building a lightweight intrusion detection model to achieve real-time detection therefore becomes an important challenge. Our dynamic AP model abstracts audit data by finding a small set of representative exemplars from the first bunch of data. It then dynamically and adaptively updates the model based on the exemplars with several parameters. Data can thus be processed in an efficient way.

There are possibilities that large-scale attacks could be falsely classified as normal by our model. Attack behaviors are thus incorporated into the normal model. However, our detection method can still detect the attacks embedded in the normal model, but with a time period. First, the attack cluster will be reset as a common item after the period Δ (see Section 3.3). It means that the large attack cluster loses its weights and becomes a single data item. Second, the attack will be then classified as anomalous if in a period Δ not many similar attacks are occurring (i.e., attack model will not be updated). The forget mechanism guarantees the self-correction of the detection model by eliminating real attacks that has been falsely learned so that the normal model is kept clean in a long run for the detection.

There are also limitations in our work. First, many real anomalies can only be detected after the detection models are rebuilt. This may cause a delay in detection. However, we have a parameter δ to trigger the model rebuilding process even there is no obvious change detected. Most anomalies can thus be detected within period δ. Another limitation is that the current detection accuracy still needs to be improved. One solution is to use more features of the data. We leave this to future work.

6. Conclusion

Online and adaptive anomaly intrusion detection is a difficult task because no *a priori* knowledge (e.g., data distribution as well as labeled information) can be provided to the learning methods. In this paper, we propose a novel framework of autonomic intrusion detection that detects intrusions in an online and adaptive

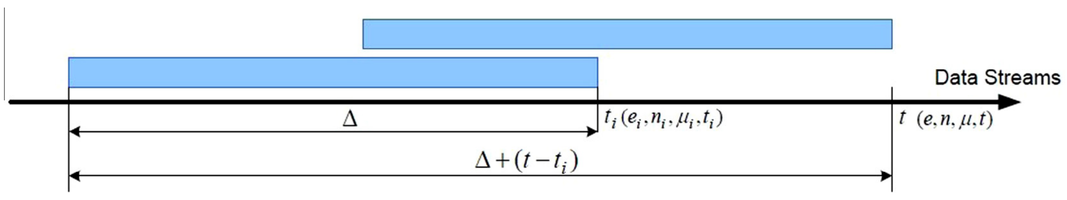

Fig. 9. Sketch figure for the detection model at t and at t_i.

fashion through dynamic clustering of audit data streams. The framework is self-managing in three aspects: self-labeling, self-updating and self-adapting. This is the first time the framework of autonomic intrusion detection is proposed. Two real HTTP traffic streams as well as a set of benchmark data (KDD'99) are used to evaluate the framework and the method. Experimental results in detecting rare attacks as well as in detecting bursty attacks in the two data sets show that the framework and the method are promising in terms of effectiveness and efficiency compared to adaptive SKL and static AP as well as other static methods, namely, *k*-NN, PCA and SVM.

Our future work is summarized as four aspects: (1) developing a mechanism that adaptively adjusts the parameters based on the detection performance; (2) incorporating signature-based detection methods or effective static methods into our framework to improve its ability of detecting large-scale intensive attacks; (3) using more features of the data to improve the detection performance; and (4) testing other data stream clustering methods based on the framework with other data sources.

Acknowledgements

The first author thanks European Research Consortium for Informatics and Mathematics (ERCIM) fellowship programme for the support. The work reported in this paper is supported by the Fundamental Research Funds for the Central Universities of China (No. K13JB00160), the Startup Funds for Talents of Beijing Jiaotong University (No. K12RC00050), the Ph.D. Programs Foundation of Ministry of Education of China (20120009120010) and the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry (No. K14C300020).

Appendix A. Proof of the proposition

Proposition 1. *The update rules in Section 3.3 with the forgetting factor enforce the stability of the model.*

Proof. Let the detection model at time t_i be represented as (e_i, n_i, μ_i, t_i) and at time t be as (e, n, μ, t). On one hand, from the point of the model at time t (see Fig. 9), the expected n (average) can be calculated as $\frac{n}{\Delta}(\Delta + (t - t_i))$. On the other hand, from the point of the model at time t_i, the expect n is calculated as $(n_i + \frac{\Delta+(t-t_i)}{n_i+1} \times (\frac{n_i}{\Delta}))$. We have

$$\frac{n}{\Delta}(\Delta + (t - t_i)) = n_i + \frac{\Delta + (t - t_i)}{n_i + 1} \times \left(\frac{n_i}{\Delta}\right) \tag{7}$$

We thus have $n = n_i \times (\frac{\Delta}{\Delta+(t-t_i)} + \frac{1}{n_i+1})$. Updating rules for μ can also be obtained based on the same average calculation. □

Appendix B. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.knosys.2014.06.018.

References

[1] Snort. Snort, 2014. <http://www.snort.org/> (retrieved February 2014).

[2] Wenke Lee, Salvatore J. Stolfo, Kui W. Mok, A data mining framework for building intrusion detection models, in: IEEE S&P, 1999, pp. 120–132.

[3] Shobha Venkataraman, David Brumley, Subhabrata Sen, Oliver Spatscheck, Automatically inferring the evolution of malicious activity on the internet, in: NDSS, 2013.

[4] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, Thomas A. Longstaff, A sense of self for unix processes, in: IEEE S&P, 1996, pp. 120–128.

[5] Wei Wang, Xiaohong Guan, Xiangliang Zhang, Liwei Yang, Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data, Comput. Secur. 25 (7) (2006) 539–550.

[6] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, Drebin: efficient and explainable detection of android malware in your pocket, in: NDSS, 2014.

[7] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, Michalis Faloutsos, Profiledroid: multi-layer profiling of android applications, in: MOBICOM, 2012, pp. 137–148.

[8] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, Angelos D. Keromytis, Casting out demons: sanitizing training data for anomaly sensors, in: IEEE S&P, 2008, pp. 81–95.

[9] Carrie Gates, Carol Taylor, Challenging the anomaly detection paradigm: a provocative discussion, in: NSPW, 2006, pp. 21–29.

[10] Terran Lane, Carla E. Brodley, Approaches to online learning and concept drift for user identification in computer security, in: KDD, 1998, pp. 259–263.

[11] KDD-Data, Kdd cup 1999 Data, 1999. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (retrieved February 2014).

[12] Irina Rish, Mark Brodie, Sheng Ma, Natalia Odintsova, Alina Beygelzimer, Genady Grabarnik, Karina Hernandez, Adaptive diagnosis in distributed systems, IEEE Trans. Neural Networks 16 (5) (2005) 1088–1109.

[13] IBM, Autonomic Computing, 2014. <http://www.ibm.com/autonomic> (retrieved February 2014).

[14] Brendan J. Frey, Delbert Dueck, Clustering by passing messages between data points, Science 315 (5814) (2007) 972–976.

[15] Xiangliang Zhang, Cyril Furtlehner, Michèle Sebag, Data streaming with affinity propagation, in: ECML/PKDD, 2008.

[16] S.E. Smaha. Haystack: an intrusion detection system, in: Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference, 1988.

[17] Matthias Schonlau, Martin Theus, Detecting masquerades in intrusion detection based on unpopular commands, Inf. Process. Lett. 76 (1-2) (2000) 33–38.

[18] Xiaohong Guan, Wei Wang, Xiangliang Zhang, Fast intrusion detection based on a non-negative matrix factorization model, J. Network Comput. Appl. 32 (1) (2009) 31–44.

[19] Wei Wang, Xiaohong Guan, Xiangliang Zhang, Processing of massive audit data streams for real-time anomaly intrusion detection, Comput. Commun. 31 (1) (2008) 58–72.

[20] Ke Wang, Salvatore J. Stolfo, Anomalous payload-based network intrusion detection, in: RAID, 2004, pp. 203–222.

[21] Xu sheng Gan, Jing shun Duanmu, Jia fu Wang, Wei Cong, Anomaly intrusion detection based on pls feature extraction and core vector machine, Knowl.-Based Syst. 40 (2013) 1–6.

[22] Christopher Krügel, Giovanni Vigna, Anomaly detection of web-based attacks, in: ACM CCS, 2003, pp. 251–261.

[23] Kenneth L. Ingham, Hajime Inoue, Comparing anomaly detection techniques for http, in: RAID, 2007, pp. 42–62.

[24] Yingbo Song, Angelos D. Keromytis, Salvatore J. Stolfo, Spectrogram: a mixture-of-markov-chains model for anomaly detection in web traffic, in: NDSS, 2009.

[25] Davide Ariu, Roberto Tronci, Giorgio Giacinto, Hmmpayl: an intrusion detection system based on hidden markov models, Comput. Secur. 30 (4) (2011) 221–241.

[26] Sangho Lee, Jong Kim, Warningbird: detecting suspicious urls in twitter stream, in: NDSS, 2012.

[27] Abdul Razzaq, Khalid Latif, H. Farooq Ahmad, Ali Hur, Zahid Anwar, Peter Charles Bloodsworth, Semantic security against web application attacks, Inf. Sci. 254 (1) (2014) 19–38.

[28] Gaik-Yee Chan, Chien-Sing Lee, Swee-Huay Heng, Policy-enhanced anfis model to counter soap-related attacks, Knowl.-Based Syst. 35 (2012) 64–76.

[29] Gaik-Yee Chan, Chien-Sing Lee, Swee-Huay Heng, Discovering fuzzy association rule patterns and increasing sensitivity analysis of xml-related attacks, J. Network Comput. Appl. 36 (2) (2013) 829–842.

[30] Suriadi Suriadi, Douglas Stebila, Andrew J. Clark, Hua Liu, Defending web services against denial of service attacks using client puzzles, in: ICWS, 2011, pp. 25–32.

[31] S. Sangeetha, S. Haripriya, S.G. Mohana Priya, V. Vaidehi, N. Srinivasan, Fuzzy rule-base based intrusion detection system on application layer, in: CNSA, 2010, pp. 27–36.

[32] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, Sal Stolfo, A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data, Appl. Data Mining Comput. Secur. (2002).

[33] L. Portnoy, E. Eskin, S. Stolfo, Intrusion Detection with Unlabeled Data Using Clustering, 2001.

[34] Kingsly Leung, Christopher Leckie, Unsupervised anomaly detection in network intrusion detection using clusters, in: Proc. 28th Australasian CS Conf., CRPITV, vol. 38, 2005, pp. 333–342.

[35] Gabriela F. Cretu, Angelos Stavrou, Michael E. Locasto, Salvatore J. Stolfo, Adaptive anomaly detection via self-calibration and dynamic updating, in: RAID, 2009, pp. 41–60.

[36] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pechoucek, Martin Grill, Jan Stiborek, Karel Bartos, Thomas Engel, Runtime monitoring and dynamic reconfiguration for intrusion detection systems, in: RAID, 2009, pp. 61–80.

[37] Murad A. Rassama, Anazida Zainala, Mohd. Aizaini Maarofaand, Adaptive and online data anomaly detection for wireless sensor systems, Knowl.-Based Syst. (2014) (available online first).
[38] Zhenwei Yu, Jeffrey J.P. Tsai, Thomas J. Weigert, An adaptive automatically tuning intrusion detection system, ACM Trans. Auton. Adapt. Syst. 3 (3) (2008).
[39] Federico Maggi, William K. Robertson, Christopher Krügel, Giovanni Vigna, Protecting a moving target: addressing web application concept drift, in: RAID, 2009, pp. 21–40.
[40] William K. Robertson, Federico Maggi, Christopher Kruegel, Giovanni Vigna, Effective anomaly detection with scarce training data, in: NDSS, 2010.
[41] Wei Wang, Florent Masseglia, Thomas Guyet, Rene Quiniou, Marie-Odile Cordier, A general framework for adaptive and online detection of web attacks, in: WWW, 2009, pp. 1141–1142.
[42] Xiangliang Zhang, Cyril Furtlehner, Cecile Germain-Renaud, Michele Sebag, Data stream clustering with affinity propagation, IEEE Trans. Knowl. Data Eng. (2014) (online first).
[43] Joao Gama, Knowledge Discovery from Data Streams, Chapman and Hall/CRC, 2010.
[44] Ed. Charu C. Aggarwal, Chandan K. Reddy, Data Clustering: Algorithms and Applications, Chapman and Hall/CRC, 2013.
[45] Avraham Levy, Michael Lindenbaum, Sequential Karhunen–Loeve basis extraction and its application to images, IEEE Trans. Image Process. 9 (2000) 1371–1374.
[46] Yihua Liao, V. Rao Vemuri, Using text categorization techniques for intrusion detection, in: USENIX Security Symposium, 2002, pp. 51–59.
[47] Wei Wang, Xiangliang Zhang, Sylvain Gombault, Constructing attribute weights from computer audit data for effective intrusion detection, J. Sys. Soft. 82 (12) (2009) 1974–1981.
[48] Bernhard Schölkopf, John C. Platt, John Shawe-Taylor, Alex J. Smola, Robert C. Williamson, Estimating the support of a high-dimensional distribution, Neural Comput. 13 (7) (2001) 1443–1471.
[49] I.T. Jolliffe, Principal Component Analysis, 2nd ed., Springer-Verlag, Berlin, 2002.
[50] R.O. Duda, P.E. Hart, D.G. Stork, Pattern Classification, China Machine Press, 2004.
[51] William K. Robertson, Giovanni Vigna, Christopher Krügel, Richard A. Kemmerer, Using generalization and characterization techniques in the anomaly-based detection of web attacks, in: NDSS, 2006.
[52] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm..
[53] Xiangliang Zhang, Cyril Furtlehner, Julien Perez, Cécile Germain-Renaud, Michèle Sebag, Toward autonomic grids: analyzing the job flow with affinity streaming, in: KDD, 2009, pp. 987–996.
[54] John McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, ACM Trans. Inf. Syst. Secur. 3 (4) (2000) 262–294.
[55] Wei Wang, Xiangliang Zhang, Sylvain Gombault, Svein J. Knapskog, Attribute normalization in network intrusion detection, in: ISPAN, 2009, pp. 448–453.