International Journal on Artificial Intelligence Tools
© World Scientific Publishing Company

EVOLVING INSIDER THREAT DETECTION STREAM MINING PERSPECTIVE

PALLABI PARVEEN, NATHAN MCDANIEL, ZACKARY WEGER, JONATHAN EVANS,
BHAVANI THURAISINGHAM, KEVIN HAMLEN, LATIFUR KHAN

*Department of Computer Science, University of Texas at Dallas, 800 W. Campbell Rd.
Richardson, TX 75080-3021, USA*
*{pxp013300, nate.mcdaniel, zrw100020, jonathan.evans,
bhavani.thuraisingham, hamlen, lkhan}@utdallas.edu*

Received 15 November 2012
Accepted 15 August 2013
Published 25 October 2013

Evidence of malicious insider activity is often buried within large data streams, such as system logs accumulated over months or years. Ensemble-based stream mining leverages multiple classification models to achieve highly accurate anomaly detection in such streams, even when the stream is unbounded, evolving, and unlabeled. This makes the approach effective for identifying insider threats who attempt to conceal their activities by varying their behaviors over time. This paper applies ensemble-based stream mining, supervised and unsupervised learning, and graph-based anomaly detection to the problem of insider threat detection. It demonstrates that the ensemble-based approach is significantly more effective than traditional single-model methods, supervised learning outperforms unsupervised learning, and increasing the cost of false negatives correlates to higher accuracy. Future work will consider a wider range of tunable parameters in an effort to further reduce false positives, include a more sophisticated polling algorithm for weighting better models, and implement parallelization to lower runtimes to more rapidly detect emerging insider threats.

Keywords: stream data mining; supervised and unsupervised learning; insider threat detection

1. Introduction

There is a growing consensus within the intelligence community that malicious insiders are perhaps the most potent threats to information assurance in many or most organizations.[1,31,24,35] One traditional approach to the *insider threat* detection problem is *supervised learning*, which builds data classification models from training data. Unfortunately, the training process for supervised learning methods tends to be time-consuming and expensive, and generally requires large amounts of well-balanced training data to be effective. In our experiments we observe that less than 3% of the data in realistic datasets for this problem are associated with insider threats (the minority class); over 97% of the data is associated with non-threats

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

(the majority class). Hence, traditional support vector machines (SVM)[36,37] trained from such imbalanced data are likely to perform poorly on test datasets.

One-class SVMs (OCSVM)[37] address the rare-class issue by building a model that considers only normal data (i.e., non-threat data). During the testing phase, test data is classified as normal or anomalous based on geometric deviations from the model. However, the approach is only applicable to bounded-length, static data streams. In contrast, insider threat-related data is typically continuous, and threat patterns evolve over time. In other words, the data is a stream of unbounded length. Hence, effective classification models must be adaptive (i.e., able to cope with evolving concepts) and highly efficient in order to build the model from large amounts of evolving data.

An alternative approach is *unsupervised learning*, which can be effectively applied to purely unlabeled data—i.e., data in which no points are explicitly identified as anomalous or non-anomalous. *Graph-based anomaly detection (GBAD)* is one important form of unsupervised learning,[2,3,4] but has traditionally been limited to static, finite-length datasets. This limits its application to streams related to insider threats, which tend to have unbounded length and threat patterns that evolve over time. Applying GBAD to the insider threat problem therefore requires that the models used be adaptive and efficient. Adding these qualities allow effective models to be built from vast amounts of evolving data.

In this paper we cast insider threat detection as a stream mining problem and propose two methods for efficiently detecting anomalies in stream data. To cope with concept-evolution, our supervised approach maintains an evolving ensemble of multiple OCSVM models. Our unsupervised approach combines multiple GBAD models in an *ensemble* of classifiers. The ensemble updating process is designed in both cases to keep the ensemble current as the stream evolves. This evolutionary capability improves the classifier's survival of *concept-drift* as the behavior of both legitimate and illegitimate agents varies over time. In experiments, we use test data that records system call data for a large, Unix-based, multiuser system.

The main contributions of this work can be summarized as follows.

- We show how stream mining can be effectively applied to detect insider threats.
- We propose a supervised learning solution that copes with evolving concepts using one-class SVMs.
- We increase the accuracy of the supervised approach by weighting the cost of false negatives.
- We propose an unsupervised learning algorithm that copes with changes based on GBAD.
- We effectively address the challenge of limited labeled training data (rare instance issues).
- We exploit the power of stream mining and graph-based mining by effectively combining the two in a unified manner. This is the first work to our knowledge to harness these two approaches for insider threat detection.

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

- We compare one and two class support vector machines on how well they handle stream insider threat problems.
- We compare supervised and unsupervised stream learning approaches and show which has superior effectiveness using real-world insider threat data.

The remainder of the paper is organized as follows. Section 2 presents related work. Section 3 presents our ensemble-based approaches to insider threat. Section 4 discusses the background of supervised and unsupervised learning methods. Section 5 describes our experiments and testing methodology. Section 6 presents our results and findings. Finally, Section 7 concludes with an assessment of the viability of ensemble-based mining for real-world insider threat detection.

2. Related Work

Insider threat detection work has applied ideas from both intrusion detection and external threat detection.[5,6,7,41,26] Supervised learning approaches collect system call trace logs containing records of normal and anomalous behavior,[8,9,25,29] extract n-gram features from the collected data, and use the extracted features to train classifiers. Text classification approaches treat each system call as a word in a bag-of-words model.[10]. Various attributes of system calls, including arguments, object path, return value, and error status, have been exploited as features in various supervised learning methods.[11,12]

Hybrid high-order Markov chain models detect anomalies by identifying a *signature behavior* for a particular user based on their command sequences.[40] The Probabilistic Anomaly Detection (PAD) algorithm[38] is a general purpose algorithm for anomaly detection (in the windows environment) that assumes anomalies or noise is a rare event in the training data. Masquerade detection is argued over by some individuals. A number of detection methods were applied to a data set of "truncated" UNIX shell commands for 70 users.[5] Commands were collected using the UNIX acct auditing mechanism. For each user a number of commands were gathered over a period of time (`http://www.schonlau.net`). The detection methods were supervised by a multistep Markovian model and a combination of Bayes and Markov approaches. It was argued that the data set was not appropriate for the masquerade detection task.[7,41] It was pointed out that the period of data gathering varied greatly from user to user (from several days to several months). Furthermore, commands were not logged in the order in which they were typed. Instead, they were coalesced when the application terminated the audit mechanism. This leads to the unfortunate consequence of possible faulty analysis of strict sequence data. Therefore, in this proposed work we have not considered this dataset.

These approaches differ from our supervised approach in that these learning approaches are static in nature and do not learn over evolving streams. In other words, stream characteristics of data are not explored further. Hence, static learning performance may degrade over time. On the other hand, our supervised approach will learn from evolving data streams. Our proposed work is based on supervised

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

learning and it can handle dynamic data or stream data well by learning from evolving streams.

In anomaly detection, one class SVM algorithm is used.[38] OCSVM builds a model by training on normal data and then classifies test data as benign or anomalous based on geometric deviations from that normal training data. For masquerade detection, one class SVM training is as affective as two class training.[38] Investigations have been made into SVMs using binary features and frequency based features. The one class SVM algorithm with binary features performed the best.

Recursive mining has been proposed to find frequent patterns.[42] One class SVM classifier were used for masquerade detection after the patterns were encoded with unique symbols and all sequences rewritten with this new coding.

To the best of our knowledge there is no work that extends this OCSVM in a stream domain. Although our approach relies on OCSVM, it is extended to the stream domain so that it can cope with changes.

Past works have also explored unsupervised learning for insider threat detection, but only to static streams to our knowledge.[13,27,28] Static GBAD approaches[2,3,4,32] represent threat and non-threat data as a graph and apply unsupervised learning to detect anomalies. The *minimum description length (MDL)* approach to GBAD has been applied to email, cell phone traffic, business processes, and cybercrime datasets.[14,15] Our work builds upon GBAD and MDL to support dynamic, evolving streams.

Stream mining[16] is a relatively new category of data mining research that applies to continuous data streams. In such settings, both supervised and unsupervised learning must be adaptive in order to cope with data whose characteristics change over time. There are two main approaches to adaptation: *incremental learning*[17,39] and *ensemble-based learning*[18,19,16]. Past work has demonstrated that ensemble-based approaches are the more effective of the two, motivating our approach.

Ensembles have been used in the past to bolster the effectiveness of positive/negative classification.[20,19] By maintaining an ensemble of K models that collectively vote on the final classification, the number of *false negatives (FN)* and *false positives (FP)* for a test set can be reduced. As better models are created, poorer models are discarded to maintain an ensemble of size exactly K. This helps the ensemble evolve with the changing characteristics of the stream and keeps the classification task tractable.

A comparison of the above related works is summarized in Table 1. A more complete survey is available in 34. Our previous work in this area includes both supervised[43] and unsupervised[44] methods. This is the first time we have combined the progress of both of our previously explored methods and included both supervised and unsupervised algorithms for the purpose of detecting insider threats. We have also extended our previous work in a few very significant ways. We have formally taken a look at the effects on detection efficiency for different values for q, the number of normative substructures considered in a model, and k, the ensemble size. This analysis aims to find the most effective parameters for which our algorithm

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

will run. We have also introduced a weighted False Negative cost to offset model selection and elimination. This weighted cost aims to punish the models that most adequately help our cause in detecting insider threats the least. Both of these new additions expound greatly on our previous work and have produced measurable results.

Table 1. Simulation Configuration

Approach	**(Un)Super-vised**	**concept-drift**	**insider threat**	**graph-based**
9	S	×	✓	×
27	S	✓	×	×
13, 28	U	×	✓	×
GBAD	U	×	✓	✓
stream 20, 16	S	✓	N/A	N/A
ensemble (ours)	S/U	✓	✓	✓

3. Ensemble-based Insider Threat Detection

Data relevant to insider threats is typically accumulated over many years of organization and system operations, and is therefore best characterized as an unbounded data stream. Such a stream can be partitioned into a sequence of discrete *chunks*; for example, each chunk might comprise a week's worth of data.

Figure 1 illustrates how a classifier's decision boundary changes when such a stream observes concept-drift. Each circle in the picture denotes a data point, with unfilled circles representing *true negatives (TN)* (i.e., non-anomalies) and solid circles representing *true positives (TP)* (i.e., anomalies). The solid line in each chunk represents the decision boundary for that chunk, while the dashed line represents the decision boundary for the previous chunk.

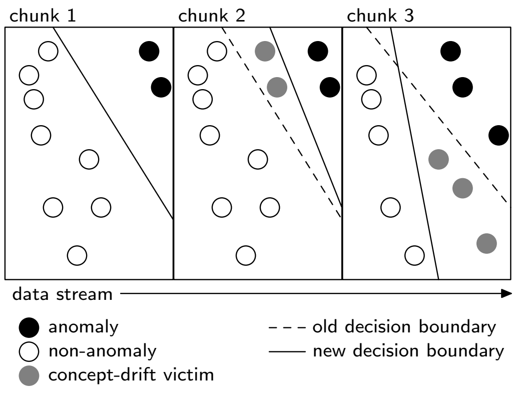

Fig. 1. Concept drift in stream data

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

Shaded circles are those that embody a new concept that has drifted relative to the previous chunk. In order to classify these properly, the decision boundary must be adjusted to account for the new concept. There are two possible varieties of *misapprehension* (false detection):

(1) The decision boundary of chunk 2 moves upward relative to chunk 1. As a result, some non-anomalous data is incorrectly classified as anomalous, causing the FP (false positive) rate to rise.
(2) The decision boundary of chunk 3 moves downward relative to chunk 2. As a result, some anomalous data is incorrectly classified as non-anomalous, causing the FN (false negative) rate to rise.

In general, the old and new decision boundaries can intersect, causing both of the above cases to occur simultaneously for the same chunk. Therefore, both FP and FN counts may increase.

These observations suggest that a model built from a single chunk or any finite prefix of chunks is inadequate to properly classify all data in the stream. This motivates the adoption of our ensemble approach, which classifies data using an evolving set of K models.

The ensemble classification procedure is illustrated in Figure 2. We first build a model using OCSVM (supervised approach) or GBAD (unsupervised approach) from an individual chunk. In the case of GBAD *normative substructures* are identified in the chunk, each represented as a subgraph. To identify an anomaly, a test substructure is compared against each model of the ensemble. A model will classify the test substructure as an anomaly based on how much the test differs from the model's normative substructure. Once all models cast their votes, weighted majority voting is applied to make a final classification decision.

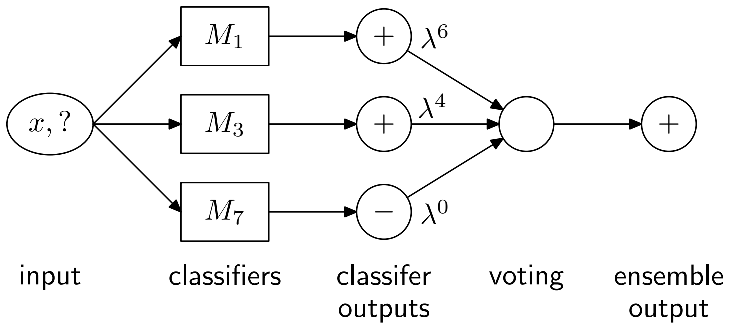

Fig. 2. Ensemble classification, where M_1 to M_7 are models within the ensemble that come to together to form a consensus.

Ensemble evolution is arranged so as to maintain a set of exactly K models at all times. As each new chunk arrives, a K+1st model is created from the new chunk and one victim model of these $K+1$ models is discarded. The discard victim can

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

Algorithm 1: Unsupervised Ensemble Classification and Updating

```
   Input: $E$ (ensemble), $t$ (test graph), and $S$ (chunk)
   Output: $A$ (anomalies), and $E'$ (updated ensemble)
1  $M' \leftarrow NewModel(S)$ ;                          // build new model
2  $E' \leftarrow E \cup \{M'\}$ ;                        // add model to ensemble
3  foreach $M \in E'$ do                                   // for each model
4      $c_M \leftarrow 0$;
5      foreach $q \in M$ do                                // find anomaly candidates
6          $A_1 \leftarrow GBAD_P(t, q)$;
7          $A_2 \leftarrow GBAD_{MDL}(t, q)$;
8          $A_3 \leftarrow GBAD_{MPS}(t, q)$;
9          $A_M \leftarrow ParseResults(A_1, A_2, A_3)$
       end
   end
10 foreach $a \in \bigcup_{M \in E'} A_M$ do               // for each candidate
11     if $round(WA(E', a)) = 1$ then                      // if anomaly
12         $A \leftarrow A \cup \{a\}$;
13         foreach $M \in E'$ do                           // approbate yes-voters
14             if $a \in A_M$ then $c_M \leftarrow c_M + 1$
           end
       end
15     else                                                // if non-anomaly
16         foreach $M \in E'$ do                           // approbate no-voters
17             if $a \notin A_M$ then $c_M \leftarrow c_M + 1$
           end
       end
   end
18 $E' \leftarrow E' - \{choose(\arg\min_M(c_M))\}$;       // drop worst model
```

be selected in a number of ways. One approach is to calculate the prediction error of each of the $K + 1$ models on the most recent chunk and discard the poorest predictor. This requires the *ground truth* to be immediately available for the most recent chunk so that prediction error can be accurately measured. If the ground truth is not available, we instead rely on majority voting; the model with least agreement with the majority decision is discarded. This results in an ensemble of the K models that best match the current concept.

Algorithm 1 summarizes the unsupervised classification and ensemble updating algorithm. Lines 1–2 build a new model from the most recent chunk and temporarily add it to the ensemble. Next, Lines 3–9 apply each model in the ensemble to test graph t for possible anomalies. We use three varieties of GBAD for each model (P, MDL, and MPS), each discussed in Section 4.2. Finally, Lines 10–18 update the

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

ensemble by discarding the model with the most disagreements from the weighted majority opinion. If multiple models have the most disagreements, an arbitrary poorest-performing one is discarded.

Weighted majority opinions are computed in Line 11 using the formula

$$WA(E,a) = \frac{\sum_{\{i \mid M_i \in E,\, a \in A_{M_i}\}} \lambda^{\ell-i}}{\sum_{\{i \mid M_i \in E\}} \lambda^{\ell-i}} \tag{1}$$

where $M_i \in E$ is a model in ensemble E that was trained from chunk i, A_{M_i} is the set of anomalies reported by model M_i, $\lambda \in [0,1]$ is a constant *fading factor*,[33] and ℓ is the index of the most recent chunk. Model M_i's vote therefore receives weight $\lambda^{\ell-i}$, with the most recently constructed model receiving weight $\lambda^0 = 1$, the model trained from the previous chunk receiving weight λ^1 (if it still exists in the ensemble), etc. This has the effect of weighting the votes of more recent models above those of potentially outdated ones when $\lambda < 1$. Weighted average $WA(E,a)$ is then rounded to the nearest integer (0 or 1) in Line 11 to obtain the weighted majority vote.

For example, in Figure 2, models M_1, M_3, and M_7 vote positive, positive, and negative, respectively, for input sample x. If $\ell = 7$ is the most recent chunk, these votes are weighted λ^6, λ^4, and 1, respectively. The weighted average is therefore $WA(E,x) = (\lambda^6+\lambda^4)/(\lambda^6+\lambda^4+1)$. If $\lambda \leq 0.86$, the negative majority opinion wins in this case; however, if $\lambda \geq 0.87$, the newer model's vote outweighs the two older dissenting opinions, and the result is a positive classification. Parameter λ can thus be tuned to balance the importance of large amounts of older information against smaller amounts of newer information.

Our approach uses the results from previous iterations of GBAD to identify anomalies in subsequent data chunks. That is, normative substructures found in previous GBAD iterations may persist in each model. This allows each model to consider all data since the model's introduction to the ensemble, not just that of the current chunk. When streams observe concept-drift, this can be a significant advantage because the ensemble can identify patterns that are normative over the entire data stream or a significant number of chunks but not in the current chunk. Thus, insiders whose malicious behavior is infrequent can still be detected.

Algorithm 2 shows the basic building blocks of our supervised algorithm. Here, we first present how we update the model. Input for algorithm 2 will be as follows: D_u is the most recently labeled data chunk (most recent training chunk) and A is the ensemble. Lines 1–2 calculate the prediction error of each model on D_u. Line 3 builds a new model using OCSVM on D_u. Line 4 produces $K+1$ models. Line 5 discards the model with the maximum prediction error, keeping the K best models.

Algorithm 3, focuses on ensemble testing. Ensemble A and the latest unlabeled chunk of instance D_u will be the input. Line 1 performs feature extraction and selection using the latest chunk of unlabeled data. Lines 2–6 will take each extracted feature from D_u and do an anomaly prediction. Lines 4–5 use each model to predict the anomaly status for a particular feature. Finally, Line 6 predicts anomalies based

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

Algorithm 2: Supervised Ensemble Classification Updating

```
  Input: $D_u$ (most recently labeled chunk), and $A$ (ensemble)
  Output: $A'$ (updated ensemble)
1 foreach $M \in A$ do                                  // for each model
2     $test(M, D_u)$ ;                             // compute expected error
  end
3 $M_n \leftarrow OCSVM(D_u)$;                    // newly trained classifier
4 $test(M_n, D_u)$ ;                               // compute expected error
5 $A' \leftarrow \{K : M_n \cup A\}$ ;        // eliminates highest expected error
```

Algorithm 3: Supervised Testing Algorithm

```
  Input: $D_u$ (most recent unlabeled chunk), and $A$ (ensemble)
  Output: $D'_u$ (labeled/predicted $D_u$)
1 $F_u \leftarrow ExtractandSelectFeatures(D_u)$;
2 foreach $x_j \in F_u$ do                            // for each feature
3     $R \leftarrow NULL$;                            // build new results
4     foreach $M \in A$ do                              // for each model
5         $R \leftarrow R \cup predict(x_j, M)$;        // predict if anomaly
      end
6     $anomalies \leftarrow MajorityVote(R)$
  end
```

on majority voting of the results.

Our ensemble method uses the results from previous iterations of OCSVM executions to identify anomalies in subsequent data chunks. This allows the consideration of more than just the current data being analyzed. Models found in previous OCSVM iterations are also analyzed, not just the models of the current dataset chunk. The ensemble handles the execution in this manner because patterns identified in previous chunks may be normative over the entire data stream or a significant number of chunks but not in the current execution chunk. Thus insiders whose malicious behavior is infrequent will be detected. It is important to note that we always keep our ensemble size fixed. Hence, an outdated model which is performing worst on the most recent chunks will be replaced by the new one.

It is important to note that the size of the ensemble remains fixed over time. Outdated models that are performing poorly are replaced by better-performing, newer models that are more suited to the current concept. This keeps each round of classification tractable even though the total amount of data in the stream is potentially unbounded.

4. Details of Learning Classes

This section will describe the different classes of learning techniques. It serves the purpose of providing more detail as to exactly how each method arrives at detecting insider threats and how ensemble models are built, modified and discarded. The first subsection goes over supervised learning in detail and the second subsection goes over unsupervised learning. Both contain the formulas necessary to understand the inner workings of each class of learning.

4.1. *Supervised Learning*

In a chunk, a model is built using OCSVM.[37] The OCSVM approach first maps training data into a high dimensional feature space (via a kernel). Next, the algorithm iteratively finds the maximal margin hyperplane which best separates the training data from the origin. The OCSVM may be considered as a regular two-class SVM. Here the first class entails all the training data, and the second class is the origin. Thus, the hyperplane (or linear decision boundary) corresponds to the classification rule:

$$f(x) = \langle w, x \rangle + b \tag{2}$$

where w is the normal vector and b is a bias term. The OCSVM solves an optimization problem to find the rule with maximal geometric margin. This classification rule will be used to assign a label to a test example x. If $f(x) < 0$, we label x as an anomaly, otherwise it is labeled normal. In reality there is a trade-off between maximizing the distance of the hyperplane from the origin and the number of training data points contained in the region separated from the origin by the hyperplane.

4.2. *Unsupervised Learning*

Algorithm 1 uses three varieties of GBAD to infer potential anomalies using each model. GBAD is a graph-based approach to finding anomalies in data by searching for three factors: modifications, insertions, and deletions of vertices and edges. Each unique factor runs its own algorithm that finds a normative substructure and attempts to find the substructures that are similar but not completely identical to the discovered normative substructure. A normative substructure is a recurring subgraph of vertices and edges that, when coalesced into a single vertex, most compresses the overall graph. The rectangle in Figure 3 identifies an example of normative substructure for the depicted graph.

Our implementation uses SUBDUE[21] to find normative substructures. The best normative substructure can be characterized as the one with minimal description length (MDL):

$$L(S, G) = DL(G \mid S) + DL(S) \tag{3}$$

where G is the entire graph, S is the substructure being analyzed, $DL(G \mid S)$ is the description length of G after being compressed by S, and $DL(S)$ is the description

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

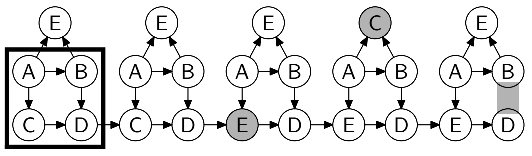

Fig. 3. A basic graph with a normative substructure (boxed) and anomalies (shaded). Nodes A through E are destinctively unique procedures. Shaded nodes and edges represent anomolies compared to that of the boxed subgraph.

length of the substructure being analyzed. Description length $DL(G)$ is the minimum number of bits necessary to describe graph G.[22]

Insider threats appear as small percentage differences from the normative substructures. This is because insider threats attempt to closely mimic legitimate system operations except for small variations embodied by illegitimate behavior. We apply three different approaches for identifying such anomalies, discussed below.

4.2.1. *GBAD-MDL*

Upon finding the best compressing normative substructure, GBAD-MDL searches for deviations from that normative substructure in subsequent substructures. By analyzing substructures of the same size as the normative one, differences in the edges and vertices' labels and in the direction or endpoints of edges are identified. The most anomalous of these are those substructures for which the fewest modifications are required to produce a substructure isomorphic to the normative one. In Figure 3, the shaded vertex labeled E is an anomaly discovered by GBAD-MDL.

4.2.2. *GBAD-P*

In contrast, GBAD-P searches for insertions that, if deleted, yield the normative substructure. Insertions made to a graph are viewed as extensions of the normative substructure. GBAD-P calculates the probability of each extension based on edge and vertex labels, and therefore exploits label information to discover anomalies. The probability is given by

$$P(A{=}v) = P(A{=}v \mid A)\, P(A) \tag{4}$$

where A represents an edge or vertex attribute and v represents its value. Probability $P(A{=}v \mid A)$ can be generated by a Gaussian distribution:

$$\rho(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{5}$$

where μ is the mean and σ is the standard deviation. Higher values of $\rho(x)$ correspond to more anomalous substructures.

Using GBAD-P therefore ensures that malicious insider behavior that is reflected by the actual data in the graph (rather than merely its structure) can be reliably

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

identified as anomalous by our algorithm. In Figure 3, the shaded vertex labeled C is an anomaly discovered by GBAD-P.

4.2.3. *GBAD-MPS*

Finally, GBAD-MPS considers deletions that, if re-inserted, yield the normative substructure. To discover these, GBAD-MPS examines the parent structure. Changes in size and orientation in the parent signify deletions amongst the subgraphs. The most anomalous substructures are those with the smallest transformation cost required to make the parent substructures identical. In Figure 3, the last substructure of A-B-C-D vertices is identified as anomalous by GBAD-MPS because of the missing edge between B and D marked by the shaded rectangle.

5. Experiments

We tested both of our algorithms on the 1998 Lincoln Laboratory Intrusion Detection dataset.[23] This dataset consists of daily system logs containing all system calls performed by all processes over a 7 week period. It was created using the Basic Security Mode (BSM) auditing program. Each log consists of tokens that represent system calls using the syntax exemplified in Figure 4.

```
header,129,2,execve(2),,Tue Jun 16 08:14:29 1998, +
        518925003 msec
path/op/local/bin/tcsh
attribute,100755,root,other,8388613,79914,0
exec_args,1,
-tcsh
subject,2142,2142,rjm,2142,rjm,401,400,24
        1 135.13.216.191
return,success,0
trailer,129
```

Fig. 4. A sample system call record from the MIT Lincoln dataset

```
Time, userID, machineIP, command, arg, path, return
1 1:29669 6:1 8:1 21:1 32:1 36:0
```

Fig. 5. Feature set extracted from Figure 4

The token arguments begin with a header line and end with a trailer line. The header line reports the size of the token in bytes, a version number, the system call, and the date and time of execution in milliseconds. The second line reports the full path name of the executing process. The optional `attribute` line identifies the user and group of the owner, the file system and node, and the device. The next line reports the number of arguments to the system call, followed by the arguments themselves on the following line. The `subject` line reports the audit ID, effective

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

user and group IDs, real user and group IDs, process ID, session ID, and terminal port and address, respectively. Finally, the `return` line reports the outcome and return value of the system call.

Since many system calls are the result of automatic processes not initiated by any particular user, they are therefore not pertinent to the detection of insider threat. We limit our attention to user-affiliated system calls. These include calls for `exec`, `execve`, `utime`, `login`, `logout`, `su`, `setegid`, `seteuid`, `setuid`, `rsh`, `rexecd`, `passwd`, `rexd`, and `ftp`. All of these correspond to logging in/out or file operations initiated by users, and are therefore relevant to insider threat detection. Restricting our attention to such operations helps to reduce extraneous noise in the dataset. Further, some tokens contain calls made by users from the outside, via web servers, and are not pertinent to the detection of *insider* threats. There are six such users in this data set and have been pulled out. Table 2 reports statistics for the dataset after all irrelevant tokens have been filtered out and the attribute data in Figure 6 has been extracted. Preprocessing extracted 62K tokens spanning 500K vertices. These reflected the activity of all users over 9 weeks.

Figure 5 shows the features extracted from the output data in Figure 4 for our supervised approach and Figure 6 depicts the subgraph structure yielded for our unsupervised approach.

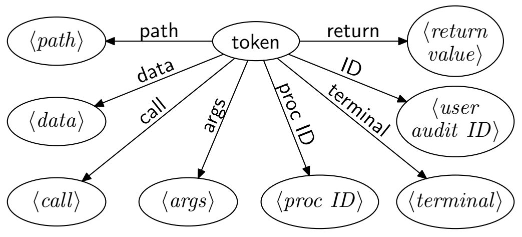

Fig. 6. A token subgraph

The first number in Figure 5 is the classification of the token as either anomalous (-1) or normal (1). The classification is used by 2-class SVM for training the model, but is unused (although required) for one class SVM. The rest of the line is a list of index-value pairs, which are separated by a colon (:). The index represent the dimension for use by SVM, and the value is the value of the token along that dimension. The value must be numeric. The list must be ascending by index. Indices that are missing are assumed to have a value of 0. Attributes which are categorical in nature (and can take the value of any one of N categories) are represented by N dimensions. In Figure 5, "1:29669" means that the time of day (in seconds) is 29669, "6:1" means that the user's ID (which is categorical) is 2142, "8:1" means that the machine IP address (also categorical) is 135.13.216.191, "21:1" means that the command (categorical) is execve, "32:1" means the path begins with `/opt`, and "36:0" means that the return value is 0. The mappings between the data values and

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

the indices were set internally by a configuration file.

All of these features are important for different reasons. The time of day could indicate that the user is making system calls during normal business hours, or, alternatively, is logging in late at night, which could be anomalous. The path could indicate the security level of the system call being made—for instance, a path beginning with `/sbin` could indicate use of important system files, while a path like `/bin/mail` could indicate something more benign, like sending mail. The user ID is important to distinguish events; what is anomalous for one user may not be anomalous for another. A programmer that normally works from 9am to 5pm would not be expected to login at midnight, but a maintenance technician (who performs maintenance on server equipment during off hours, at night), would. Frequent changes in machine IP address or changes that are not frequent enough could indicate something anomalous. Lastly, the system call itself could indicate an anomaly most users would be expected to login and logout, but only administrators would be expected to invoke super user privileges with a command such as `su`.

We used LIBSVM[36] to build our models and to generate predictions for our test cases in our supervised approach. First, we will give an overview of our use of SVM software, which is standard procedure and is well documented in LIBSVM's help files. We chose to use the RBF (radial-based function) kernel for the SVM. It was chosen because it gives good results for our data set. Parameters for the kernel (in the case of two-class SVM, C and γ, and in the case of one-class SVM, ν and γ) were chosen so that the F_1 measure was maximized. We chose to use the F_1 measure in this case (over other measures of accuracy) because, for the classifier to do well according to this metric, it must minimize false positives while also minimizing false negatives. Before training a model with our feature set, we used LIBSVM to scale the input data to the range $[0, 1]$. This was done to ensure that dimensions which takes on high values (like time) do not outweigh dimensions that take on low values (such as dimensions which represent categorical variables). The parameters that were used to scale the training data for the model are the same parameters that were used to scale that model's test data. Therefore, the model's test data will be in the vicinity of the range $[0, 1]$.

We conducted two experiments with the SVM. The first, as seen in Table 4, was designed to compare one-class SVM with two-class SVM for the purposes of insider threat detection, and the second, as seen in Table 5, was designed to compare a stream classification approach with a more traditional approach to classification. We will begin by describing our comparison of one-class and two-class SVM. For this experiment, we took the 7 weeks of data, and randomly divided it into halves. We deemed the first half training data and the other half testing data. We constructed a simple one-class and two-class model from the training data and recorded the accuracy of the model in predicting the test data.

For the insider threat detection approach we use an ensemble-based approach that is scored in real time. The ensemble maintains K models that use one-class SVM, each constructed from a single day and weighted according to the accuracy

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

of the models' previous decisions. For each test token, the ensemble reports the majority vote of its models.

The stream approach outlined above is more practical for detecting insider threats because insider threats are stream in nature and occur in real time. A situation like that in the first experiment above is not one that will occur in the real world. In the real world, insider threats must be detected as they occur, not after months of data have piled in. Therefore, it is reasonable to compare our updating stream ensemble with a simple one-class SVM model constructed once and tested (but not updated) as a stream of new data becomes available, see Table 5.

For our unsupervised approach, we needed to accurately depict the effects of two variables. Those variables are K, the number of ensembles maintained, and q, the number of normative substructures maintained for each model in the ensemble. We used a subset of data during this wide variety of experiments, as depicted in Table 3, in order to complete them in a manageable time. The decision to use the small subset of data was arrived at due to the exponential growth in cost for checking subgraph isomorphism.

Each ensemble iteration was run with q values between 1 and 8. Iterations were made with ensemble sizes of K values between 1 and 6.

Table 2. Dataset statistics after filtering and attribute extraction

Statistic	**Value**
# vertices	500,000
# tokens	62,000
# normative substructures	5
# users	all
duration	9 weeks

Table 3. Summary of data subset A

Statistic	**Dataset** A
user	donaldh
# vertices	269
# edges	556
week	2–8
weekday	Friday

6. Results

Performance and accuracy was measured in terms of total false positives (FP) and false negatives (FN) throughout 7 weeks of test data. The Lincoln Laboratory dataset was chosen for both its large size and because its set of anomalies is well known, facilitating an accurate performance assessment via misapprehension counts.

Table 4 shows the results for the first experiment using our supervised method. One-class SVM outperforms two-class SVM in the first experiment. Simply, two-class SVM is unable to detect any of the positive cases correctly. Although the two-class SVM does achieve a higher accuracy, it is at the cost of having a 100% false negative rate. By varying the parameters for the two-class SVM, we found it possible to increase the false positive rate (the SVM made an attempt to discriminate between anomaly and normal data), but in no case could the two-class SVM predict even one of the truly anomalous cases correctly. One-class SVM, on the

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

other hand, achieves a moderately low false negative rate (20%), while maintaining a high accuracy (87.40%). This demonstrates the superiority of one-class SVM over two-class SVM for insider threat detection.

Table 4. Experiment A: One-class vs Two-class SVM

	One-class SVM	**Two-class SVM**
False Positives	3706	0
True Negatives	25701	29407
False Negatives	1	5
True Positives	4	0
Accuracy	0.87	0.99
False Positive Rate	0.13	0.0
False Negative Rate	0.2	1.0

The superiority of one-class SVM over two-class SVM for insider threat detection further justifies our decision to use one-class SVM for our test of stream data. Table 5 gives a summary of our results for the second experiment using our supervised method. The updating stream achieves much higher accuracy than the non-updating stream, while maintaining an equivalent, and minimal, false negative rate (10%). The accuracy of the updating stream is 76%, while the accuracy of the non-updating stream is 58%.

Table 5. Experiment B: Updating vs Non-updating Stream Approach

	Updating Stream	**Non-updating Stream**
False Positives	13774	24426
True Negatives	44362	33710
False Negatives	1	1
True Positives	9	9
Accuracy	0.76	0.58
False Positive Rate	0.24	0.42
False Negative Rate	0.1	0.1

The superiority of updating stream over non updating stream for insider threat detection further justifies our decision to use updating stream for our test of stream data. By using labeled data, we establish a ground truth for our supervised learning algorithm. This ground truth allows us to place higher weights on false negatives or false positives. By weighing one more than the other, we punish a model more for producing that which we have increased the weight for. When detecting insider threats it is more important that we do not miss a threat (false negative) than identify a false threat (false positive). Therefore, we weigh false negative more heavily—i.e. we add a *FN cost*. Figure 7 and Figure 8 show the results of weighting the false negatives more heavily than false positives with this established ground truth. This is to say, that at a FN cost of 50, a false negative that is produced will count against a model 50 times more than a false positive will. Increasing the FN cost also increases the accuracy of our OCSVM, updating stream approach. We can see that that increasing the FN cost up to 30 only increases the total cost without affecting the accuracy, but after this, the accuracy climbs and the total cost comes down. Total cost, as calculated by Equation 6, represents the total number of false positives and false negative after they have been modified by the increased FN Cost. We see this trend peak at a FN cost of 80 where accuracy reaches nearly 56% and the total cost is at a low of 25229.

$$TotalCost = TotalFalsePositives + (TotalFalseNegatives * FNCost) \tag{6}$$

The false negatives are weighted by cost more heavily than false positives because it is more important to catch all insider threats. False positives are acceptable in some cases, but an insider threat detection system is useless if it does not catch all positive instances of insider threat activity. This is why models who fail to catch positive cases and produce these false negatives are punished, in our best case result, 80 times more heavily than those who produce false positives.

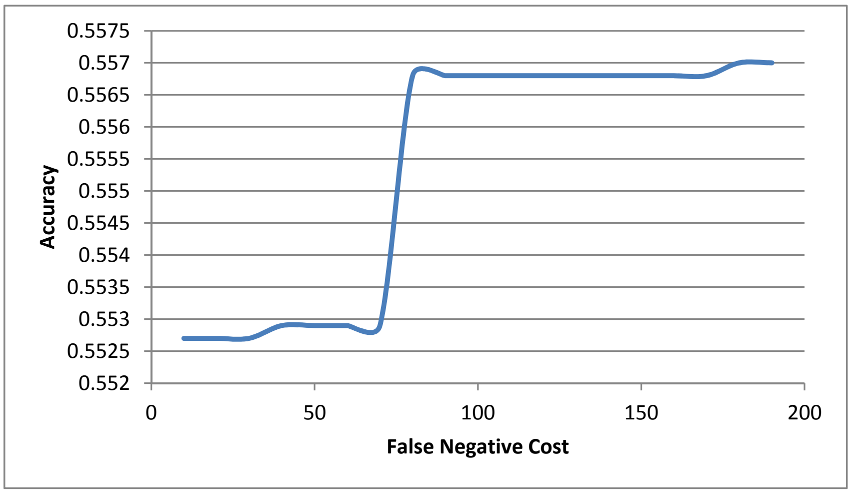

Fig. 7. Accuracy by FN Cost

Table 6 reinforces our decision to include *FN cost* during model elimination that heavily punishes models who produce false negatives over those that produce false positives. Including FN cost increases the accuracy of the ensemble and provides a better F_2 Measure.

We next investigate the impact of parameters K (the ensemble size) and q (the number of normative substructures per model) on the classification accuracy and running times for our unsupervised approach. To more easily perform the larger number of experiments necessary to chart these relationships, we employ the smaller

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

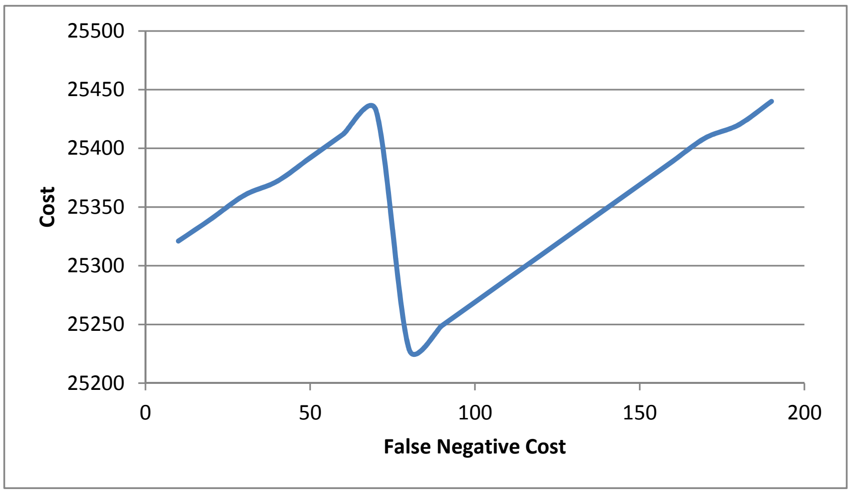

Fig. 8. Total Cost by FN Cost

Table 6. Impact of FN Cost

	Accuracy	F_2 Measure
w/ FN Cost	0.55682	0.00159
w/o FN Cost	0.45195	0.00141

datasets summarized in Table 3 for these experiments. Dataset A consists of activity associated with user `donaldh` during weeks 2–8. This user displays malicious insider activity during the respective time period. This dataset evince similar trends for all relationships discussed henceforth; therefore we report only the details for dataset A throughout the remainder of the section.

Figure 9 shows the relationship between the cutoff q for the number of normative substructures and the running time in dataset A. Times increase approximately linearly until $q = 5$ because there are only 4 normative structures in dataset A. The search for a 5th structure therefore fails (but contributes running time), and higher values of q have no further effect.

Figure 10 shows the impact of ensemble size K and runtimes for dataset A. As expected, runtimes increase approximately linearly with the number of models (2 seconds per model on average in this dataset).

Increasing q and K also tends to aid in the discovery of true positives (TP). Figures 11 and 12 illustrate by showing the positive relationships of q and K, respectively, to TP. Once $q = 4$ normative substructures are considered per model and $K = 4$ models are consulted per ensemble, the classifier reliably detects all 7 true positives in dataset A. These values of q and K therefore strike the best balance between coverage of all insider threats and the efficient runtimes necessary for high

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

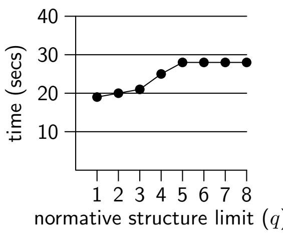

Fig. 9. The effect of q on runtimes for fixed $K = 6$ on dataset A

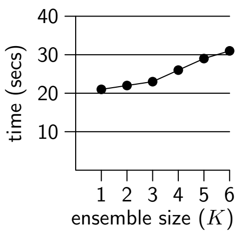

Fig. 10. The effect of K on runtimes for fixed $q = 4$ on dataset A

responsiveness.

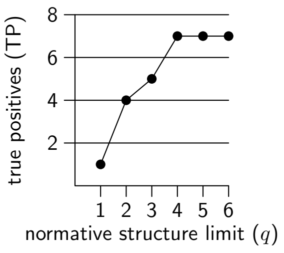

Fig. 11. The effect of q on TP rates for fixed $K = 6$ on dataset A

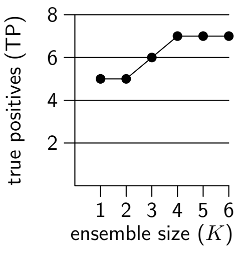

Fig. 12. The effect of K on TP rates for fixed $q = 4$ on dataset A

Increasing q to 4 does come at the price of raising more false alarms, however. Figure 13 shows that the false positive rate increases along with the true positive rate until $q = 4$. Dataset A has only 4 normative structures, so increasing q beyond this point has no effect. This is supported with $q = 4, 5, 6$ showing no increase in TP.

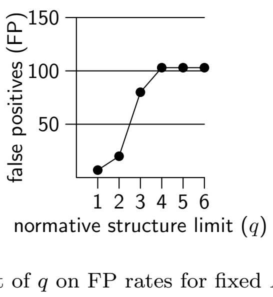

Fig. 13. The effect of q on FP rates for fixed $K = 6$ on dataset A

Table 7 considers the impact of weighted versus unweighted majority voting on the classification accuracy. The unweighted columns are those for $\lambda = 1$, and the weighted columns use fading factor $\lambda = 0.9$. The dataset consists of all tokens associated with user ID 2143. Weighted majority voting has no effect in these ex-

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

periments except when $K = 4$, where it reduces the FP rate from 124 (unweighted) to 85 (weighted) and increases the TN rate from 51 (unweighted) to 90 (weighted). However, since these results can be obtained for $K = 3$ without weighted voting, we conclude that weighted voting merely serves to mitigate a poor choice of K; weighted voting has little or no impact when K is chosen wisely.

Table 7. Impact of fading factor λ (weighted voting)

	$K = 2$		$K = 3$		$K = 4$	
	λ=1	λ=0.9	λ=1	λ=0.9	λ=1	λ=0.9
TP	10	10	10	10	14	14
FP	79	79	85	85	124	85
TN	96	96	90	90	51	90
FN	4	4	4	4	0	0

Table 8 gives a summary of our results comparing our supervised and unsupervised learning approaches. For example, on dataset A the supervised learning achieves much higher accuracy (71%) than the unsupervised learning (56%), while maintaining lower false positive rate (31%) and false negative rate (0%). On the other hand, unsupervised learning achieves 56% accuracy, 54% false positive rate and 42% false negative rate.

Table 8. Supervised vs Non Supervised Learning Approach

	Supervised	**Unsupervised**
False Positives	55	95
True Negatives	122	82
False Negatives	0	5
True Positives	12	7
Accuracy	0.71	0.56
False Positive Rate	0.31	0.54
False Negative Rate	0.0	0.42

7. Conclusions

The supervised learning approach to insider threat detection outperformed the unsupervised learning approach. The supervised method succeeded in identifying all 12 anomalies in the 1998 Lincoln Laboratory Intrusion Detection dataset with zero false negatives and a lower false positive rate than the unsupervised approach. The technique combines the power of one-class SVMS with the adaptiveness of stream mining to achieve effective, practical insider threat detection for unbounded, evolving data streams. Increasing the weighted cost of false negatives increased accuracy and ultimately allowed our approach to perform as well as it did. Though false positives could be further reduced through more parameter tuning, our supervised approach accomplished its goal, detecting all of the insider threats.

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

Acknowledgments

The authors would like to thank Dr. Robert Herklotz for his support. This work is supported by the Air Force Office of Scientific Research, under grant FA9550-08-1-0088.

Appendix A.

In this appendix, we explain the method for obtaining accuracy and F_2 Measure reported in some of our results. Because the data set used is well documented, the true positives are all well known and we can accurately measure false positives and false negatives. F_2 Measure is a specific case of F_β Measure that weighs recall more than precision

$$F_\beta = (1+\beta^2)\frac{(precision)(recall)}{\beta^2(precision)+(recall)}$$

Precision and recall are defined in terms of true/false positives and negatives by

$$precision = \frac{TP}{(TP+FP)}$$

$$recall = \frac{TP}{(TP+FN)}$$

Accuracy can also be measured in term of true/false positives and negatives by

$$accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

References

1. R. C. Brackney and R. H. Anderson, *Understanding the Insider Threat.* RAND Corporation, (March 2004).
2. D. J. Cook and L. B. Holder, *Mining Graph Data.* John Wiley & Sons, Inc. (Hoboken, New Jersey, 2007).
3. W. Eberle and L. B. Holder, Mining for Structural Anomalies in Graph-based Data, in *Proceedings of the International Conference on Data Mining (DMIN).* (2007) 376–389.
4. D. J. Cook and L. B. Holder, Graph-based Data Mining, in *IEEE Intelligent Systems.* **15:2**(2000) 32–41.
5. M. Schonlau and W. DuMouchel and W. Ju and A. F. Karr and M. Theus and Y. Vardi, Computer Intrusion: Detecting masquerades, in *Statistical Science.* **16:1**(2001) 1–17.
6. K. Wang and S. J. Stolfo, One-Class Training for Masquerade Detection, in *Proceedings of the ICDM Workshop on Data Mining for Computer Security (DMSEC).* (2003).
7. R. A. Maxion, Masquerade Detection Using Enriched Command Lines, in *Proceedings of the IEEE International Conference on Dependable Systems & Networks (DSN).* (2003) 5–14.
8. S. Forrest and S. A. Hofmeyr and A. Somayaji and T. A. Longstaff, A Sense of Self for Unix Processes, in *Proceedings of the IEEE Symposium on Computer Security and Privacy (S&P).* (1996) 120–128.

9. S. A. Hofmeyr and S. Forrest and A. Somayaji, Intrusion Detection using Sequences of System Calls, in *Journal of Computer Security.* **6:3**(1998) 151–180.
10. Y. Liao and V. R. Vemuri, Using Text Categorization Techniques for Intrusion Detection, in *Proceedings of the 11th USENIX Security Symposium.* (2002) 51–59.
11. C. Krügel and D. Mutz and F. Valeur and G. Vigna, On the Detection of Anomalous System Call Arguments, in *Proceedings of the 8th European Symposium on Research in Computer Security (ESORICS).* (2003) 326–343.
12. G. Tandon and P. Chan, Learning Rules from System Call Arguments and Sequences for Anomaly Detection, in *Proceedings of the ICDM Workshop on Data Mining for Computer Security (DMSEC).* (2003) 20–29.
13. A. Liu and C. Martin and T. Hetherington and S. Matzner, A Comparison of System Call Feature Representations for Insider Threat Detection, in *Proceedings of the IEEE Information Assurance Workshop (IAW).* (2005) 340–347.
14. S. Staniford-Chen and S. Cheung and R. Crawford and M. Dilger and J. Frank and J. Hoagland and K. Levitt and C. Wee and R. Yip and D. Zerkle, A Graph Based Intrusion Detection System for Large Networks, in *Proceedings of the 19th National Information Systems Security Conference.* (1996) 361–370.
15. E. Kowalski and T. Conway and S. Keverline and M. Williams and D. Cappelli and B. Willke and A. Moore, Insider Threat Study: Illicit Cyber Activity in the Government Sector, in *U.S. Department of Homeland Security, U.S. Secret Service, CERT, and the Software Engineering Institute (Carnegie Mellon University).* (January 2008).
16. W. Fan, Systematic Data Selection to Mine Concept-drifting Data Streams, in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD).* (2004) 128–137.
17. P. Domingos and G. Hulten, Mining High-speed Data Streams, in *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD).* (2000) 71–80.
18. M. M. Masud and Q. Chen and J. Gao and L. Khan and C. Aggarwal and J. Han and B. Thuraisingham, Addressing Concept-evolution in Concept-drifting Data Streams, in *Proceedings of the IEEE International Conference on Data Mining (ICDM).* (2010) 929–934.
19. M. M. Masud and J. Gao and L. Khan and J. Han and B. Thuraisingham, Classification and Novel Class Detection in Concept-drifting Data Streams under Time Constraints, in *IEEE Transactions on Knowledge and Data Engineering (TKDE).* **23:6**(2011) 859–874.
20. M. M. Masud and J. Gao and L. Khan and J. Han and B. Thuraisingham, A Practical Approach to Classify Evolving Data Streams: Training with Limited Amount of Labeled Data, in *Proceedings of the IEEE International Conference on Data Mining (ICDM).* (2008) 929–934.
21. N. S. Ketkar and L. B. Holder and D. J. Cook, Subdue: Compression-based Frequent Pattern Discovery in Graph Data, in *Proceedings of the ACM KDD Workshop on Open-Source Data Mining.* (2005).
22. W. Eberle and J. Graves and L. Holder, Insider Threat Detection Using a Graph-based Approach, in *Journal of Applied Security Research.* **6:1**(2011) 32–81.
23. K. Kendall, A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems, in *Massachusetts Institute of Technology.* (1998).
24. S. Matzner and T. Hetherington, Detecting Early Indications of a Malicious Insider, in *IA Newsletter.* **7:2**(2004) 42–45.
25. N. Nguyen and P. Reiher and G. H. Kuenning, Detecting Insider Threats by Monitoring System Call Activity, in *Proceedings of the IEEE Information Assurance Workshop*

Published in International Journal on Artificial Intelligence Tools, Vol. 22, No. 5 (1360013)

(IAW). (2003) 45–52.

26. E. E. Schultz, A Framework for Understanding and Predicting Insider Attacks, in *Computers and Security.* **21:6**(2002) 526–531.
27. E. Eskin and A. Arnold and M. Prerau and L. Portnoy and S. Stolfo, A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data, in *Applications of Data Mining in Computer Security.* (Springer 2002)
28. E. Eskin and M. Miller and Z. Zhong and G. Yi and W. Lee and S. Stolfo, Adaptive Model Generation for Intrusion Detection Systems, in *Proceedings of the ACM CCS Workshop on Intrusion Detection and Prevention (WIDP).* (2000).
29. D. Gao and M. K. Reiter and D. Song, On Gray-box Program Tracking for Anomaly Detection, in *Proceedings of the USENIX Security Symposium.* (2004) 103–118.
30. J. A. Swets and R. M. Pickett, Evaluation of Diagnostic Systems: Methods from Signal Detection Theory, in *Medical Physics.* **10:2**(1983) 266–267.
31. M. P. Hampton and M. Levi, Fast Spinning into Oblivion? Recent Developments in Money-laundering Policies and Offshore Finance Centres, in *Third World Quarterly.* **20:3**(1999) 645–656.
32. X. Yan and J. Han, gSpan: Graph-based Substructure Pattern Mining, in *Proceedings of the International Conference on Data Mining (ICDM).* (2002) 721–724.
33. L. Chen and S. Zhang and L. Tu, An Algorithm for Mining Frequent Items on Data Stream Using Fading Factor, in *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC).* (2009) 172–177.
34. M. B. Salem and S. Herkshkop and S. J. Stolfo, A Survey of Insider Attack Detection Research, in *Insider Attack and Cyber Security.* **39**(2008) 69–90.
35. M. B. Salem and S. J. Stolfo, Modeling User Search Behavior for Masquerade Detection, in *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID).* (2011).
36. C. Chang and C. Lin, LIBSVM: a library for support vector machines, in *ACM Transactions on Intelligent Systems and Technology.* (2011) 2:27:1–27:27. http://www.csie.ntu.edu.tw/ cjlin/libsvm.
37. L. M. Manevitz and M. Yousef, One-class SVMs for document classification, in *The Journal of Machine Learning Research.* (March 2002) 2.
38. S. J. Stolfo and F. Apap and E. Eskin and K. Heller and S. Hershkop and A. Honig and K. Svore, A Comparative Evaluation of Two Algorithms for Windows Registry Anomaly Detection, in *Journal of Computer Security.* **13:4** (Amsterdam, The Netherlands, July 2005) 659–693.
39. B. D. Davison and H. Hirsh, Predicting Sequences of User Actions. In Working Notes of the Joint Workshop on Predicting the Future: AI Approches to Time Series Analysis, in *Proceedings of the 15th National Conference on Artificial Intelligence and Machine.* (AAAI Press 1998) 5–12.
40. W. Ju and Y. Vardi, A Hybrid High-order Markov Chain Model for Computer Intrusion Detection, in *Journal of Computational and Graphical Statistics.* (June 2001).
41. R. A. Maxion and T. N. Townsend, Masquerade Detection Augmented with Error Analysis, in *IEEE Transactions on Reliability.* (2004) 53(1):124–147.
42. B. K. Szymanski and Y. Zhang, Recursive Data Mining for Masquerade Detection and Author Identification, in *Proceedings of the 13th Annual IEEE Information Assurance Workshop.* (IEEE Computer Society Press 2004).
43. P. Parveen and Z. Weger and B. Thuraisingham and K. Hamlen and L. Khan, Supervised Learning for Insider Threat Detection Using Stream Mining, in *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI).* (2011) 1032–1039.

44. P. Parveen and J. Evans and B. Thuraisingham and K. W. Hamlen and L. Khan, Insider Threat Detection Using Stream Mining and Graph Mining, in *Proceedings of the 3rd IEEE Conference on Privacy, Security, Risk and Trust (PASSAT)*. (2011) 1102–1110.