2012 ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust

Unsupervised Ensemble based Learning for Insider Threat Detection

Pallabi Parveen, Nate McDaniel, Varun S. Hariharan, Bhavani Thuraisingham and Latifur Khan
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas
Email: pxp013300, nate.mcdaniel, Varun.Hariharan, bhavani.thuraisingham, lkhan@utdallas.edu

***Abstract*—Insider threats are veritable needles within the hay stack. Their occurrence is rare and when they do occur, are usually masked well within normal operation. The detection of these threats requires identifying these rare anomalous needles in a contextualized setting where behaviors are constantly evolving over time. To this refined search, this paper proposes and tests an unsupervised, ensemble based learning algorithm that maintains a compressed dictionary of repetitive sequences found throughout dynamic data streams of unbounded length to identify anomalies. In unsupervised learning, compression-based techniques are used to model common behavior sequences. This results in a classifier exhibiting a substantial increase in classification accuracy for data streams containing insider threat anomalies. This ensemble of classifiers allows the unsupervised approach to outperform traditional static learning approaches and boosts the effectiveness over supervised learning approaches.**

I. Introduction

The modern computing infrastructure is under attack from a massive potential insider threat issue that is constantly growing [1]. This issue continues to spread in intensity and permeability as more diverse people gain access to sensitive information. Additional usage of this sensitive data is required in order to maintain a global competitive advantage and the detection of insider threats is particularly challenging. Insiders are often intimately familiar with the internal working of a system and conceal their actions by molding them very closely to legitimate tasks and activities carried on by the system. The extremely wide spread proliferation of web access and utility greatly exacerbates the problem. Even without access to a physical system, and insider can construct commands to take advantage of legitimate user privileges. By manipulating these privileges repeatedly an insider can, over time, gain access to programs and data that allow them read or write private data, system privileges, or malicious software. With these dire consequences on the horizon, there is great need for a system that can detect and terminate anomalous behavior in its early stages.

To the end of eliminating insider threats, one possible approach is a supervised learning method. The supervised method requires training data to build up its classification models before using them to make a decision on any possible anomalies. This training process tends to be very time-consuming and requires an unrealistic amount of balanced training data from both non-anomalous and anomalous classes. Finding such balanced training data for anomalous instances is often quite difficult to obtained. A well documented instance of this is an insider threat dataset (MIT Lincoln lab dataset) where only about 0.03% of the training data is associated with insider threats and 99.97% of the data is associated with non-threats. This is an immensely imbalanced pool of training data and traditional supervised learning algorithms (e.g., support vector machines (SVM)) [2] trained from such an imbalanced dataset are likely to perform poorly on test datasets.

Instead of using data that is labeled as anomalous or not, an unsupervised learning approach can be used. This alternative approach can be effectively applied to purely unlabeled data that is not explicitly identified at any point to be anomalous or non-anomalous. This learning approach forms its basis of non-anomalous data from regular user input. Over time, consistent user command input will form legitimate patterns that can be considered to be a users normal behavior. By establishing this natural baseline, future patterns can be captured and analyzed in an unsupervised manner. Every new pattern found in an evolving steam of user inputs can be compared to the frequent command patterns previously collected for anomalies.

Data that is associated with insider threat detection and classification is often continuous. The data is not static and is unbounded in length. This makes any strictly static approach, supervised or unsupervised, not applicable. In these systems, the patterns of average users and insider threats can gradually evolve over time. A novice programmer can develop his skills to become an expert programmer over time. An Insider threat can change his actions to more closely mimic legitimate user processes. In either case, the patterns at either end of these developments can look drastically different when compared directly to each other. These natural changes will not be treated as anomalies in our approach. Instead, we classify them as natural concept drift. The traditional static supervised and unsupervised methods raise unnecessary false alarms with these cases because they are unable to handle them when they arise in the system. These traditional methods encounter high false positive rates. Learning models must be adapt in coping with evolving concepts and highly efficient at building models from large amounts of data to rapidly detecting real threats.

For these reasons, the insider threat problem can be conceptualized as a stream mining problem that applies to continuous data streams. Whether using a supervised or unsupervised

978-0-7695-4848-7/12 $26.00 © 2012 IEEE
DOI 10.1109/SocialCom-PASSAT.2012.106

IEEE computer society

learning algorithm, the method chosen must be highly adaptive to correctly deal with concept drifts under these conditions. Incremental learning and Ensemble based learning [3]–[5] are two adaptive approaches in order to overcome this hindrance. An ensemble of K models that collectively vote on the final classification can reduce the false negatives (FN) and false positives (FP) for a test set. As new models are created and old ones updated to be more precise, the least accurate models are discarded to always maintain an ensemble of exactly K current models.

To cope with concept-evolution, our approach maintains an ensemble of multiple unsupervised stream based sequence learning (USSL). During the learning process, we store the repetitive sequence patterns from a users actions or commands in a model called a Quantized Dictionary. In particular, longer patterns with higher weights due to frequent appearances in the stream are considered in the dictionary. An ensemble in this case is a collection of K models of type Quantized Dictionary. When new data arrives or is gathered, we generate a new Quantized Dictionary model from this new dataset. We will take the majority voting of all models to find the anomalous pattern sequences within this new data set. We will update the ensemble if the new dictionary outperforms others in the ensemble and will discard the least accurate model from the ensemble. Therefore, the ensemble always keeps the models current as the stream evolves, preserving high detection accuracy as both legitimate and illegitimate behaviors evolve over time. Our test data consists of real-time recorded user command sequences for multiple users of varying experience levels and a concept drift framework to further exhibit the practicality of this approach.

Our primary contributions are as follows. First, our approach shows how ensemble based stream mining can be effectively used for detecting insider threat and cope with the evolving concept drifts. Second, we propose an unsupervised learning solution (USSL) that finds pattern sequences from successive user actions or commands using stream based sequence learning. Third, we effectively integrate multiple USSL models in an ensemble of classifiers to exploit the power of ensemble based stream mining and sequence mining. To the best of our knowledge, there is no work that extends USSL in an ensemble based stream mining. Finally, we compare our approach with the supervised model for stream mining and show the effectiveness of our approach in terms of TPR and FPR on a benchmark dataset.

The rest of the paper is organized as follows. Section II presents our proposed stream based insider threat detection approach. Section III discusses about experiments, datasets, and results. It explains about the concept drift framework that is manually introduced in our dataset. Section IV presents related work. Finally, Section V concludes and suggests future work.

II. Stream Based Insider Threat Detection

Insider threat detection related data is stream based in nature. Data may be gathered over time, maybe even years.

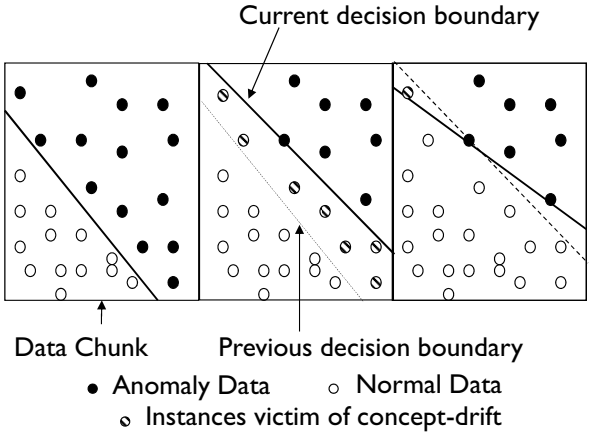

Fig. 1. Concept drift in stream data

In this case, we assume a continuous data stream will be converted into a number of chunks. For example, each chunk may represent a week and contain the data which arrived during that time period.

Figure 1 demonstrates how the classifier decision boundary changes over time (from one chunk to next chunk). Data points are associated with two classes (normal and anomalous). There are three contiguous data chunks as shown in the Figure 1. The dark straight line represents the decision boundary of its own chunk, whereas the dotted straight line represents the decision boundary of previous chunk. If there were no concept drift in the data stream, the decision boundary would be the same for both the current chunk and its previous chunk (the dotted and straight line). White dots represent the normal data (True Negative), blue dots represent anomaly data (True Positive), and striped dots represent the instances victim of concept drift.

We show the two different cases in the Figure 1 are as follows.

Case1: the decision boundary of the second chunk moves upwards compared to that of the first chunk. As a result, more normal data will be classified as anomalous by the decision boundary of the first chunk, thus FP will go up. Recall that a test point having true benign (normal) category classified as an anomalous by a classifier is known as a FP.

Case2: the decision boundary of the third chunk moves downwards compared to that of the first chunk. So, more anomalous data will be classified as normal data by the decision boundary of the first chunk, thus FN will go up. Recall that a test point having true malicious category classified as benign by a classifier is known as a FN.

In the more general case, the decision boundary of the current chunk can vary, which causes the decision boundary of the previous chunk to misclassify both normal and anomalous data. Therefore, both FP and FN may go up at the same time.

This suggests that a model built from a single chunk will not suffice. 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 Fig-

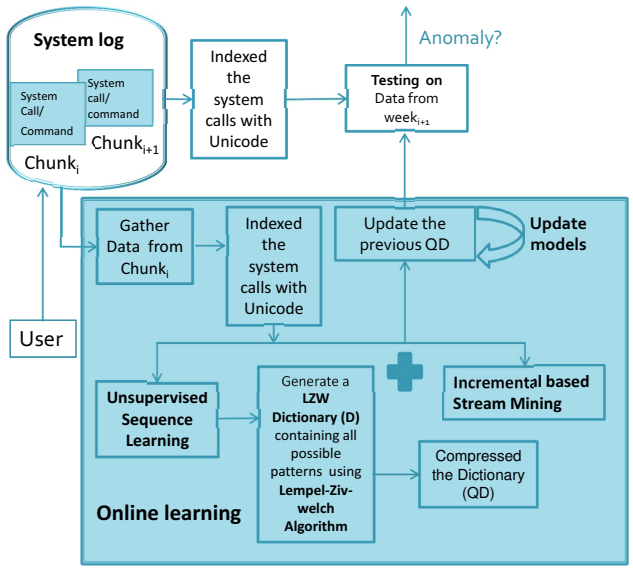

Fig. 2. Ensemble based unsupervised stream sequence learning

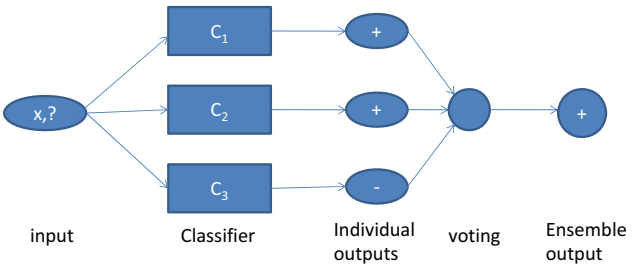

Fig. 3. Ensemble classification

ure 3. We first use static, unsupervised USSL to train models from an individual chunk. USSL identifies the normative patterns in the chunk and stores it in a quantized dictionary.

To identify an anomaly, a test point will be compared against each model of the ensemble. Recall that a model will declare the test data as an anomalous based on how much the test differs from the models normative patterns. Once all models cast their vote, we will apply majority voting to make the final decision as to whether the test point is an anomalous or not (as shown in Figure 3).

Model Update: We always keep an ensemble of fixed size models (K in that case). Hence, when a new chunk is processed (see Figure 3), we already have K models in the ensemble and the $K+1st$ model will be created from the current chunk. We need to update ensemble by replacing a victim model with this new model. Victim selection can be done in a number of ways. One approach is to calculate the prediction error of each model on the most recent chunk relative to the majority vote. Here, we assume *ground truth* on the most recent chunk is not available. If ground truth

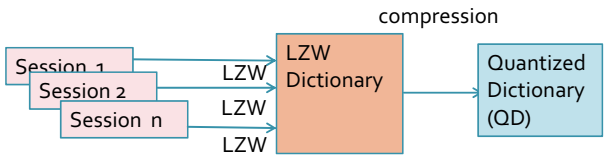

Fig. 4. Unsupervised Stream based Sequence Learning(USSL)

is available we can exploit this knowledge for training. The new model will replace the existing model from the ensemble which gives the maximum prediction error.

Unsupervised Stream based Sequence Learning (USSL): Normal user profiles are considered to be repetitive daily or weekly activities which are frequent sequences of commands, system calls etc. These repetitive command sequences are called normative patterns. These patterns reveal the regular, or normal, behavior of a user. When a user suddenly demonstrates unusual activities that indicate a significant excursion from normal behavior an alarm is raised for potential insider threat.

So, in order to identify an insider threats, first we need to find normal user behavior. For that we need to collect sequences of commands and find the potential normative patterns observed within these command sequences in an unsupervised fashion. This unsupervised approach also needs to identify normal user behavior in a single pass. One major challenge with these repetitive sequences is their variability in length. To combat this problem, we need to generate a dictionary which will contain any combination of possible normative patterns existing in the gathered data stream. Potential variations that could emerge within the data include the commencement of new events, the omission or modification of existing events, or the reordering of events in the sequence. *Eg., liftliftliftliftliftcomcomecomecomecome-come,* is a sequence of commands represented by the alphabets given in a data stream. . We will consider all patterns *li,if,ft,tl, lif, ift, ftl, lift, iftl etc.*, as our possible normative patterns. However, the huge size of the dictionary presents another significant challenge.

We have addressed the above two challenges in the following ways. First, we extract possible patterns from the current data chunk using single pass algorithm (e.g., LZW, Lempel-Ziv- Welch algorithm [6]) to prepare a dictionary. We called it LZW dictionary. LZW dictionary has a set of patterns and their corresponding weights according to

$$w_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \quad (1)$$

where w_i is the weight of a particular pattern p_i in the current chunk, f_i is the number of times the pattern p_i appears in the current chunk and n is the total number of distinct patterns found in that chunk.

Next, we compress the dictionary by keeping only the longest, frequent unique patterns according to their associated weight and length, while discarding other subsumed patterns. This technique is called compression method (CM) and the new dictionary is a Quantized dictionary (QD). The Quan-

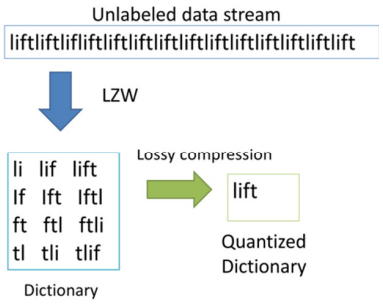

Fig. 5. Quantization of dictionary

tized dictionary has a set of patterns and their corresponding weights. Here, we use edit distance to find the longest pattern. Edit distance is a measure of similarity between pairs of strings [7]. It is the minimum number of actions required to transfer one string to another, where an action can be substitution, addition or deletion of a character into the string. As in case of the earlier example mentioned, the best normative pattern in the quantized dictionary would be lift, come etc.

This process is a lossy compression, but is sufficient enough to extract the meaningful normative patterns. The reason behind this is the patterns that we extract are the superset of the subsumed patterns. Moreover, as frequency is another control parameter in our experiment, the patterns which do not appear often cannot be regular user patterns.

Data relevant to insider threat is typically accumulated over many years of organization and system operations, and is therefore best characterized as an unbounded data stream. As our data is a continuous stream of data, we use ensemble based learning to continuously update our compressed dictionary. This continuous data stream is partitioned into a sequence of discrete chunks. For example, each chunk might be comprised of a day or weeks worth of data and may contain several user sessions. We generate our Quantized dictionary (QD) and their associated weight from each chunk. Weight is measured as the normalized frequency of a pattern within that chunk.

When a new chunk arrives we generate a new Quantized dictionary (QD) model and update the ensemble as mentioned earlier. Figure 2 shows the flow diagram of our *dynamic*, Ensemble based, Unsupervised Stream Sequence Learning method.

Algorithm 1 shows the basic building block for updating the Ensemble. Algorithm 1 takes the most recent data chunk S, ensemble E and test chunk T. Lines 1 to 2 generate a new Quantized Dictionary model from the most recent chunk S and temporarily add it to the ensemble E. Lines 3 to 6 test chunk T for anomalies for each model in the ensemble. Lines 7 to 14 find and label the anomalous patterns in test Chunk T according to the majority voting of the models in the ensemble. Finally, line 15 updates the ensemble by discarding the model with lowest accuracy. An arbitrary model is discarded in the case of multiple models having the same low performance.

A. Construct the LZW dictionary by selecting the patterns in the data stream

At the beginning, we consider that our data is not annotated (i.e., unsupervised). In other words, we dont know the possible sequence of future operations by the user. So, we use LZW algorithm [6] to extract the possible sequences that we can add to our dictionary. These can also be commands like *liftliftliftliftliftcomcomecomecomecomecome,* where each unique letter represents a unique system call or command. We have used Unicode to index each command. E.g, ls, cp, find are indexed as *l*, *c*, and *f*. The possible patterns or sequences are added to our dictionary would be *li, if,ft,tl, lif,ift, ftl, lift, iftl, ftli, tc, co, om, mc,com, come* and so on. When the sequence *li* is seen in the data stream for the second time, in order to avoid repetition it will not be included in the LZW dictionary. Instead, we increase the frequency by 1 and extend the pattern by concatenating it with the next character in the data stream, thus turning up a new pattern *lif*. We will continue the process until we reach the end of the current chunk. Figure 5 demonstrates how we generate an LZW dictionary from the data stream.

Algorithm 1: Update the Ensemble

Input: E (ensemble)
T (test chunk)
S (chunk)
Output: A (anomalies)
E' (updated ensemble)

1 $M' \leftarrow NewModel(S)$ // build new model // LZW algo [6] and algo 2
2 $E' \leftarrow E \cup \{M'\}$ // add model to ensemble
3 **foreach** $M \in E'$ **do** // for each model
4 $A_M \leftarrow T$
5 **foreach** $p \in M$ **do** // for each pattern
6 **if** $edit_{distance(x,p)} \leq \alpha = \frac{1}{3}$ **then** $A_M = A_M - x$
7 **foreach** $a \in \bigcup_{M \in E'} A_M$ **do** // for each candidate
8 **if** $round(WA(E', a)) = 1$ **then** // if anomaly
9 $A \leftarrow A \cup \{a\}$
10 **foreach** $M \in E'$ **do** // approbate yes-voters
11 **if** $a \in A_M$ **then** $c_M \leftarrow c_M + 1$
12 **else** // if non-anomaly
13 **foreach** $M \in E'$ **do** // approbate no-voters
14 **if** $a \notin A_M$ **then** $c_M \leftarrow c_M + 1$
15 $E' \leftarrow E' - \{choose(\arg\min_M(c_M))\}$ // drop worst model

Algorithm 2: Quantized Dictionary

Input: $D = \{P, W\}$ (LZW dictionary)
Output: QD (quantized dictionary)

while $D \neq 0$ **do**
$X \leftarrow D_1$ // first pattern
foreach $i \in D$ **do** // for each pattern
if $edit_{distance(X,D_i)} = 1$ **then** $P \leftarrow P \cup i$
$D \leftarrow D - X$
if $P \neq 0$ **then**
$X \leftarrow choose(\arg\max_i(P_i length(i)))$
$QD \leftarrow QD \cup X$
$D \leftarrow D - X$
$X \leftarrow D_1$ // next pattern

B. Constructing the quantized dictionary

Once we have our LZW dictionary, we keep the longest and most frequent patterns and discard all their subsumed patterns. Algorithm 2 shows step by step how a quantized dictionary is generated from LZW dictionary. Inputs of this algorithm are as follows: LZW dictionary D which contains a set of patterns P and their associated weight W. Line 2 picks a pattern (e.g., li). Lines 3 to 4 find all the closest patterns that are 1 edit distance away. Lines 6 to 9 keep the pattern which has the highest weight multiplied by its length and discard the other patterns. We repeat the steps (line 2 to 9) until we find the longest, frequent pattern (*lift*). After that, we start with a totally different pattern (*co*) and repeat the steps until we have explored all the patterns in the dictionary. Finally, we end up with a more compact dictionary which will contain many meaningful and useful sequences. We call this dictionary our quantized dictionary. Figure 5 demonstrates how we generate a quantized dictionary from the LZW dictionary.

Once, we identify different patterns *lift*, *come*, etc., any pattern with $X\%(\geq \%30$ in our implementation) deviation from all these patterns would be considered as anomaly. Here, we will use edit distance to identify the deviation.

C. Anomaly Detection (Testing)

Given a quantized dictionary, we need to find out the sequences in the data stream which may raise a potential threat. To formulate the problem, given the data stream S and the quantized dictionary $QD = qd1, qd2, ...qdm$, any pattern in the data stream is considered as an anomaly if it deviates from all the patterns in the quantized dictionary by more than $X\%(say > 30\%)$. In order to find the anomalies, we need to delete any pattern from the data stream S that is an exact match or α edit distance away from any pattern in QD. α can be half, one-third or one-fourth of the length of that particular pattern in QD.

In order to identify the non- matching patterns in the data stream S, we compute a distance matrix L which contains the edit distance between each pattern in QD and the data stream S. If we have a perfect match, i.e, edit distance 0 between a pattern qd_k and S, we can move backwards exactly the length of qd_k in order to find the starting point of that pattern in S and then delete it from the data stream. On the other hand, if there is an error in the match which is greater than 0 but less than α, in order to find the starting point of that pattern in the data stream, we need to traverse left, up or diagonal within the matrix according to which one among the mentioned value *(L[i,j-1], L[i-1,j-1], L[i-1,j])* gives the minimum respectively. Finally, once we find the starting point, we can delete that pattern from the data stream too. The remaining patterns in the data stream will be considered as anomalous.

III. Experiments and Results

A. Dataset

The data sets used for training and testing have been created from Trace Files got from the University of Calgary project [8]. As a part of that, 168 trace files were collected from 168 different users of Unix csh. There were 4 groups of people, namely novice-programmers, experienced-programmers, computer-scientists, non-programmers. The model that we have tried to construct is that of a novice-programmer who has been gaining experience over the weeks and is gradually using more and more command sequences similar to that of an experienced user. These gradual normal behavior changes will be known here as concept-drift. Anomaly detection in the presence of concept drift is difficult to achieve. Hence, our scenario is more realistic. This is a slow process that takes place over several weeks.

TABLE I
Description of Dataset

Description	Number
# of valid users	37
# of invalid users	131
# of valid commands per user	2400
# of anomalous commands in testing chunks	300

The Calgary dataset [8] as described above was modified by Maxion et al. [9] for masquerade detection. Here we followed the same guidelines to inject masquerades commands. From the given list of users, those users who have executed more than 2400 commands (that did not result in an error) were filtered out to form the valid-user pool. This list had 37 users. The remaining users were part of the invalid-user pool. Out of the list of invalid-users, 25 of them were chosen at random and a block of one hundred commands from the commands that they had executed were extracted and put together to form a list of 2500 (25 * 100) commands. The 2500 commands were brought together as 250 blocks of 10 commands each. Out of these 250 blocks, 30 blocks were chosen at random as the list of masquerade commands (300 commands). For each user in the valid users list, the total number of commands was truncated to 2400. These 2400 commands were split into 8 chunks of 300 commands each. The first chunk was kept aside as the training chunk (this contains no masquerade data). The other 7 chunks are the testing chunks. In the testing chunks, a number of masquerade data blocks (each block comprising of 10 commands) were inserted at random positions. As a result, for each user, we have one training chunk with 300 commands (no masquerade data) and three testing chunks which together have 2100 non-anomalous commands and 300 masquerade commands (see Table I).

B. Concept Drift in the training set

Here, we present a framework for concept drift with the goal of introducing artificial drift into data generators. First, we generate a number of chunks having normal and anomalous sequences as described above. Note that the above process does not take into account concept drift. Next, we take sequences of each chunk as input and generate a new sequence with a particular drift for that chunk. New sequences will have concept drift property. The framework processes each training example of user commands once and only once to produce variation predictions at any time. The framework is

passed the next available sequence of commands (say thousand commands for a chunk) from the user data. The framework then processes these commands, updating the data structures with distribution and bounded concept drift variance. The framework is then ready to process the next set of commands, and upon request can produce predicted variants based on the concept drift.

The data being passed into the framework is a set number of commands from different users in sequential order of execution by the user. Although commands are passed into the framework in large groups, each command is treated as an individual observation of that persons behavior. To calculate predicted variations the following drift formula is used

$$drift = \sqrt{\frac{\log \frac{1}{\delta}}{2dn}} \tag{2}$$

where δ is the variation constant, d is the current distribution of the command over the current number of individual observations, and n is the number of current observations made. A good value for the variation constant is $1*10^{-5}$. The variation constant shares an inverse relation to the overall drift values. The expected range in distribution among produced variations is calculated by adding and subtracting the calculated drift from the current distribution.

Upon the processing of a sample of user commands, predicted variations can be produced by the framework upon request. The request can be made of any designated size and the concept drift will provide new distributions that fall within the range of the calculated drift for a set of commands of this size. The produced set of commands or new ones can be used to update the concept drift and provide for a constantly evolving command distribution that represents an individual. Sudden changes that do not fit within a calculated concept drift can be flagged as suspicious and therefore, possibly representative of an insider threat.

Algorithm 3: Concept Drift in Sequence Stream

Input: F (file)
n (size of PV)
Output: PV (prediction variation)
foreach $C \in F$ **do** // for each command
  $D_C \leftarrow distribution(C)$
  $V_C \leftarrow variation(D_C)$
  $MaxDrift_C \leftarrow D_C + V_C$
  $MinDrift_C \leftarrow D_C - V_C$
$PV \leftarrow newPredictedVariation()$
foreach $C \in F$ **do** // for each command
  if $D_C < MinDrift_C$ **then** $PV \leftarrow PV \cup C$
  else if $D_C + 1 < MaxDrift_C$ **then**
    $flipCoin(PV \leftarrow PV \cup C)$
  else
    $discard(C)$

Algorithm 3 shows how the distribution is calculated for a set of commands and how the predicted variation is produced. As an example we take ten commands, instead of one thousand per chunk, such that we have $[C_1, C_2, C_1, C_2, C_3, C_1, C_4, C_5, C_1, C_1]$. The distributions will be $[C_1 = .5, C_2 = .2, C_3 = .1, C_4 = .1, C_5 = .1]$. With a value of $\delta = 1 * 10^{-5}$ the *Predicted Variance* for each command comes out as

$$C_1 = \sqrt{\frac{\log \frac{1}{1*10^{-5}}}{2 * .5 * 10}} \approx .7071 \# of occurance \tag{3}$$

We divide by the number of observation occurrences, in this case 10, because we want the concept drift per occurrence, not just for a sample of 10. Our PV values are $[C_1 \approx .07071, C_2 \approx .11180, C_3 \approx .15811, C_4 \approx .15811, C_5 \approx .15811]$. The adjusted Min/Max Drift comes out to be $[C_1 = .42929/.57071, C_2 = .08820/.31180, C_3 = 0/.25811, C_4 = 0/.25811, C_5 = 0/.25811]$.

From these drift values we can produce a requested predicted variation for another 10, or any number of, user command values. We look at the original sequence and assemble at least the minimum drift value worth of commands in the variation. In this example the first value is C_1 with a minimum of .42929 distribution so we add it, bringing its current distribution to 1/10 or .1. This is the case for the first 3 values resulting in $[C_1, C_2, C_1]$. The fourth value is C_2 who has met his minimum distribution drift. Adding him will not go over the maximum distribution so he is either randomly added or not. The new predicted variation could look like $[C_1, C_2, C_1, C_2, C_1, C_1, C_1, C_2, C_3, C_4]$ or $[C_1, C_2, C_1, C_3, C_1, C_4, C_1, C_5, C_3, C_1]$ of which both have command distributions similar to the original that fall within the new variance bounds.

The following results compare our approach, USSL, with a supervised method modified from the baseline approach suggested by Maxion [9], which uses Naive Bayes' classifier (NB). The modified version compared in this paper is more incremental in its model training, and thus will be referred to as *NB-INC*. All instances of both algorithms were tested using 8 chunks of data. At every test chunk the NB-INC method uses all previously seen chunks to build the model and train for the current test chunk. For test chunk 3, NB builds the classification model and trains on chunks 1 and 2. For test chunk 5 NB builds a model and collectively trains on chunks 1 through 4.

These USSL statistics are gathered in a "Grow as you Go" (GG) fashion. This is to say that, as the ensemble size being used grows larger the amount of test chunks to go through decreases. For an ensemble size of 1, models are built from chunk 1 and chunks 2 through 8 are considered testing chunks. For an ensemble size of 3, chunks 4 through 8 are considered to be testing chunks. Every new chunk, starting at 4 in this case, is used to update the models in each ensemble after a majority vote is reached and the test data is classified as an anomaly or not. To clarify, in the example of ensemble size of 3 after the new model is built, all models vote on the possible anomaly and contribute to deciding which model is considered the least accurate and thus discarded. However, only models that have survived an elimination round (i.e. deemed not to be least accurate at least once) are used to

TABLE II
NB-INC VS USSL-GG FOR VARIOUS DRIFT VALUES ON TPR AND FPR

Drift	TPR for NB-INC	TPR for USSL-GG	FPR for NB-INC	FPR USSL-GG
0.000001	0.34	**0.49**	0.12	**0.10**
0.00001	0.36	**0.58**	0.12	**0.09**
0.0001	0.37	**0.51**	0.11	**0.10**
0.001	0.38	**0.50**	0.11	**0.10**

measure the FP, TP, FN, and TN for an ensemble. Which is in this case models created from chunks 1 and 2 but not 3. The following data that requires calculation is done so using the four above mentioned values acquired in this fashion. The model updating process makes use of compression and model replacement. After models are updated the least accurate one is discarded to maintain the highest accuracy and maintain the status quo, since a new model is created before every update and anomaly classification step. There are other ways of updating models and selecting chunks for testing during the classification process not explored in this paper. For this purpose, the limited USSL method used for the shown results will be referred to as *USSL-GG*.

We have compared the NB and USSL-GG algorithms on the basis of true positive rate(TPR), false positive rate(FPR), execution time, accuracy, F_1 measure, and F_2 measure. TPR and FPR as measured by

$$TPR = \frac{true\ positives}{true\ positives + false\ negatives} \quad (4)$$

$$FPR = \frac{false\ positives}{false\ positives + true\ negatives} \quad (5)$$

are the rates at which actual insider threats are correctly and incorrectly identified by the algorithm. For these calculations a true positive(TP) is an identified anomaly that is actually an anomaly, a true negative(TN) is an identified piece of normal data that is not an anomaly, a false positive(FP) is an identified anomaly that is actually just normal data, and a false negative(FN) is an identified piece of normal data that is actually an anomaly.

Accuracy better measures the algorithms ability to pick out correct and incorrect insider threat instances on a whole. F_1 and F_2 measure are weighted variables calculated from the generic equation 7 that penalize for incorrectly identified insider threats (FP) and positive threats that were not identified (FN). The F_1 measure penalizes FP and FN equally while the F_2 measure penalizes FN much more. These values are calculated by

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \quad (6)$$

$$F_n = \frac{(1 + n^2)TP}{(1 + n^2)TP + (n^2)FN + FP} \quad (7)$$

and represent important parameters in catching insider threats.

Tables II, III and IV show the details of the value comparisons between NB-INC and USSL-GG for various drift values. USSL-GG has lower FPR and runtime values and higher

TABLE III
NB-INC VS USSL-GG FOR VARIOUS DRIFT VALUES ON ACCURACY AND RUNTIME

Drift	Acc for NB-INC	Acc for USSL-GG	time for NB-INC	time for USSL-GG
0.000001	0.80	**0.85**	52.0	**3.60**
0.00001	0.79	**0.87**	50.8	**3.54**
0.0001	0.82	**0.86**	51.0	**3.55**
0.001	0.81	**0.85**	53.4	**3.60**

TABLE IV
NB-INC VS USSL-GG FOR VARIOUS DRIFT VALUES ON F_1 AND F_2 MEASURE

Drift	F_1Msr for NB-INC	F_1Msr for USSL-GG	F_2Msr for NB-INC	F_2Msr for USSL-GG
0.000001	0.34	**0.44**	0.34	**0.47**
0.00001	0.36	**0.50**	0.36	**0.54**
0.0001	0.37	**0.45**	0.37	**0.49**
0.001	0.38	**0.44**	0.38	**0.47**

everything else than NB-INC across the board. USSL-GG runs faster with less false threat identifications while maintaining higher success rates at catching real threats than NB-INC. The USSL-GG data in tables II, III and IV are for its optimum results at ensemble size 3, which will be shown why this is optimal later.

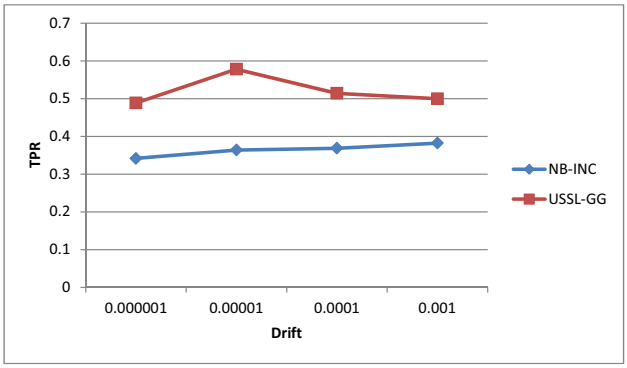

Fig. 6. Comparison between NB-INC vs our optimized model, USSL-GG in terms of TPR

With our optimization results for USSL-GG we make our final comparison with NB. Figures 6 and 7 show the TPR and FPR respectively. USSL-GG maintains higher TPR and lower FPR than NB-INC.

C. Choice of ensemble size

In Figure 8, we show the various concept drifts and ensemble sizes of USSL-GG compared in terms of TPR. As ensemble size increases, TPR decreases. This is not desired. In Figure 9 we show the same set of concept drifts and ensemble sizes compared in terms of FPR. We can see a steady decrease in TPR across all drifts as ensemble size increases. However, we see a much larger decrease in FPR from ensemble size 1 to 2 and from 2 to 3 than from 3 to 4 and 4 to 5 across all drift values. As ensemble size increases we achieve a desired lower FPR at the expense of also lowering TPR.

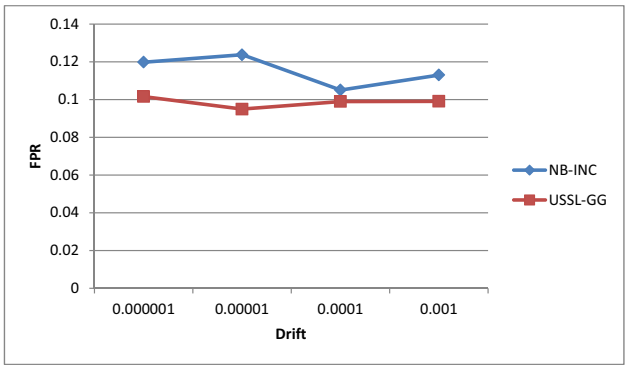

Fig. 7. Comparison between NB-INC vs our optimized model, USSL-GG in terms of FPR

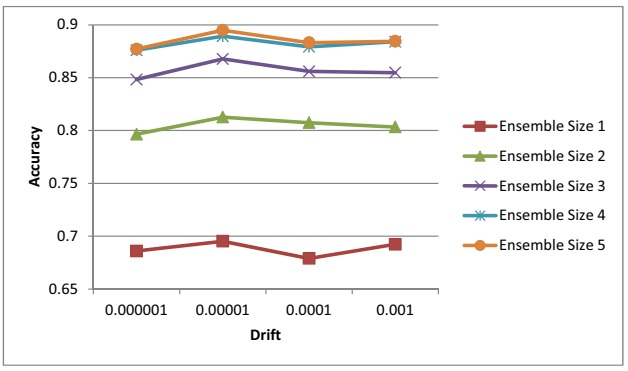

Fig. 10. Comparison of USSL-GG across multiple drifts and ensemble sizes in terms of Accuracy

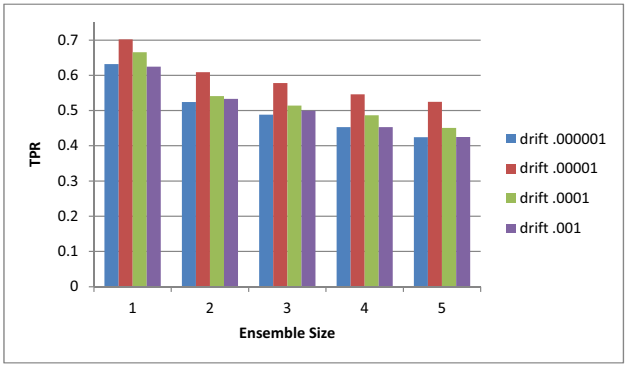

Fig. 8. Comparison of USSL-GG across multiple drifts and ensemble sizes in terms of TPR

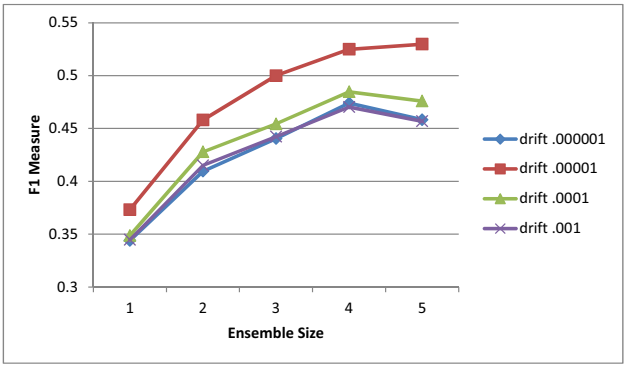

Fig. 11. Comparison of USSL-GG across multiple drifts and ensemble sizes in terms of F1 measure

In Figure 10, we show that when considering the rate at which positive and negative instances are correctly identified without punishing the algorithms for wrong classifications with the accuracy metric, USSL-GG performs better as ensemble size increases. Figure 11 shows F_1 Measure and Figure 12 shows F_2 Measure. With these two figures, we can see that USSL-GG performs better as ensemble size increases until

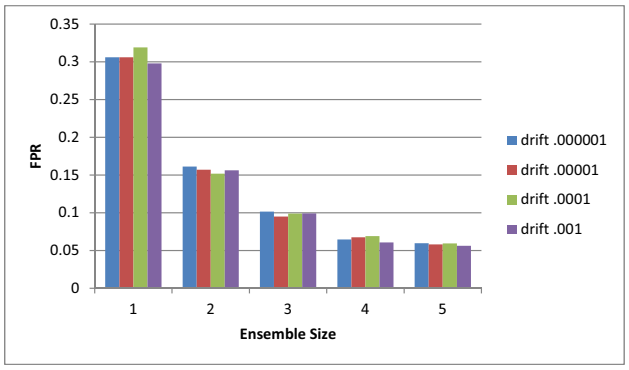

Fig. 9. Comparison of USSL-GG across multiple drifts and ensemble sizes in terms of FPR

size 4 or 5. This is due to F measure values penalizing missed insider threats and wrongly classifying harmless instances as threats. It does not give any direct bonuses for how many threats are correctly identified. This is important in insider threat detection because missing even one insider threat can be very detrimental to the system you are protecting. Wrongly classifying harmless instances are not as important, but they add the increased problem of having to double check detected instances. In Figure 12, we show a decrease in F_2 Measure after ensemble size 3. This makes USSL-GG most effective at ensemble size 3 because F_2 Measure penalizes false negatives more than the false positives and low FN is the most important part of insider threat detection as stated previously. This makes our final optimization of USSL-GG to include ensemble size 3. With an ensemble size of 3, it is also worthy noting that run times are slower than that of ensemble sizes 1 or 2. For ensemble sizes greater than 3 the run times grow exponentially. These greatly increased times are not desired and therefor ensemble sizes such as 4 or 5 are not optimal.

IV. Related Work

Insider threat detection work has utilized ideas from intrusion detection or external threat detection areas [2], [10]. For example, supervised learning has been applied to detect

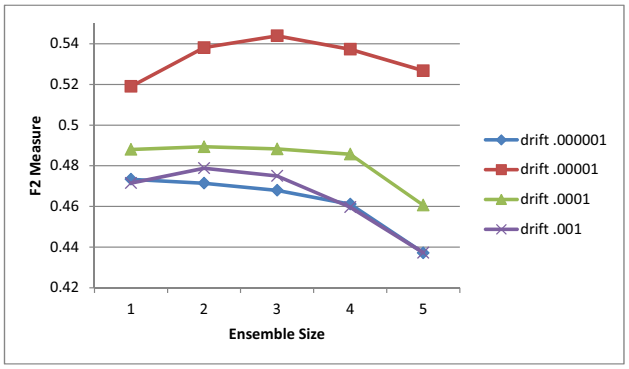

Fig. 12. Comparison of USSL-GG across multiple drifts and ensemble sizes in terms of F2 measure

insider threats. System call traces from normal activity and anomaly data are gathered [11]; features are extracted from this data using n-gram and finally, trained with classifiers. Authors [12] exploit text classification idea in insider threat domain where each system call is treated as a word in bag of words model. System call, and related attributes, arguments, object path, return value and error status of each system call are served as features in various supervised methods [13], [14]. A supervised model based on hybrid high-order Markov chain model was adopted by researchers [15]. A *signature behavior* for a particular user based on the command sequences that the user executed is identified and then anomaly is detected.

Schonlau et al. [10] applied a number of detection methods to a data set of "truncated" UNIX shell commands for 70 users. 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 are supervised based on multistep markovian model, and combination of Bayes and Markov approach. Maxion et al. [9] argued that Schonlau data set was not appropriate for the masquerade detection task and created new data set using Calgary dataset and apply static supervised model.

These approaches differ from our work in the following ways. These learning approaches are static in nature and do not learn over evolving stream. In other words, stream characteristics of data are not explored further. Hence, static learner performance may degrade over time. On the other hand, our approach will learn from evolving data stream. In this paper, we show that our approach is unsupervised and is as effective as supervised model (incremental).

Researchers have explored *unsupervised learning* [16] for insider threat detection. However, this learning algorithm is static in nature. Although, our approach is unsupervised, but at the same time learns from evolving stream over time and more data will be used for unsupervised learning.

In anomaly detection one class support vector machine (SVM) algorithm (OCSVM) is used. OCSVM builds a model from training on normal data and then classifies a test data as benign or anomaly based on geometric deviations from normal training data. Wang et al. [2] showed, for masquerade detection, one-class SVM training is as effective as two-class training. The authors have investigated SVMs using binary features and frequency based features. The one-class SVM algorithm with binary features performed the best. To find frequent patterns, Szymanski et al. [17] proposed recursive mining, encoded the patterns with unique symbols, and rewrote the sequence using this new coding. They used a one-class SVM classifier for masquerade detection. These learning approaches are static in nature and do not learn over evolving stream.

Stream mining is a new data mining area where data is continuous. In addition, characteristics of data may change over time (concept drift). Here, supervised and unsupervised learning need to be adaptive to cope with changes. There are two ways adaptive learning can be developed. One is incremental learning and the other one is ensemble-based learning. Incremental learning is used in user action prediction [18], but not for anomaly detection. Davidson et al. [19] introduced Incremental Probabilistic Action Modeling (IPAM), based on one-step command transition probabilities estimated from the training data. The probabilities were continuously updated with the arrival of a new command and modified with the usage of an exponential decay scheme. However, the algorithm is not designed for anomaly detection.

Therefore, to the best of our knowledge, there is almost no work from other researchers that handles insider threat detection in stream mining area. This is the first attempt to detect insider threat using stream mining.

Recently, unsupervised learning is applied to detect insider threat in a data stream [20]. This work does not consider sequence data for threat detection. Recall that sequence data is very common in insider threat scenario. Instead, it considers data as graph/vector and finds normative patterns and apply ensemble based technique to cope with changes. On the other hand, in our proposed approach, we consider user command sequences for anomaly detection and construct quantized dictionary for normal patterns. In addition, we have incorporated power of stream mining to cope with gradual changes. We have done experiments and shown that our USSL approach works well in the context of concept drift and anomaly detection.

In [21] an incremental approach is used. Ensemble based techniques are not incorporated, but the literature used shows that ensemble based techniques are more effective than those of the incremental variety for stream mining [4]. Therefore, this paper focuses on ensemble based techniques.

Refer to table V on which related approaches are unsupervised or supervised, and whether they focus on concept drift, detecting insider threat and sequenced data from stream mining.

V. Conclusions and Future Work

Our stream guided sequence learning performed well, with limited number of false positives as compared to static approaches. Our approach is a combination of compression and

TABLE V
CAPABILITIES AND FOCUSES OF VARIOUS RELATED APPROACHES

Approach	Un/Supervised	Drift	Insider Threat	Sequence
Ju [15]	S	N	Y	Y
Maxion [9]	S	N	Y	N
Liu [16]	U	N	Y	Y
Wang [2]	S	N	Y	N
Szymanski [17]	S	N	Y	Y
Masud [3]–[5]	S	Y	N	N
Parveen [20]	U	Y	Y	N
USSL-GG	U	Y	Y	Y

incremental learning. The approach adopts advantages from both compression & ensemble-based learning. In particular, compression offered unsupervised learning in a manageable manner and on the other hand ensemble based learning offered adaptive learning. The approach was tested on real command line dataset and shows effectiveness over static approaches in terms of TP and FP.

We could extend the work in the following directions. First, once a model is created in an unsupervised manner, we would like to update the model based on user feedback. Right now, once the model is created it remains unchanged. When ground truth is available over time, we would like to update the model further. Next, we will do additional experiment on more sophisticated scenarios, including different test chunk selection besides USSL-GG.

ACKNOWLEDGMENT

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 FA9550-08-1-0260.

REFERENCES

[1] M. B. Salem, S. Herkshkop, and S. J. Stolfo, "A survey of insider attack detection research," *Insider Attack and Cyber Security*, vol. 39, pp. 69–90, 2008.

[2] K. Wang and S. J. Stolfo, "One-class training for masquerade detection," in *Proc. ICDM Workshop on Data Mining for Computer Security (DMSEC)*, 2003.

[3] M. M. Masud, Q. Chen, J. Gao, L. Khan, C. Aggarwal, J. Han, and B. Thuraisingham, "Addressing concept-evolution in concept-drifting data streams," in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2010, pp. 929–934.

[4] M. M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *IEEE Trans. Knowl. Data Eng. (TKDE)*, vol. 23, no. 6, pp. 859–874, 2011.

[5] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham, "A practical approach to classify evolving data streams: Training with limited amount of labeled data," in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2008, pp. 929–934.

[6] T. A. Welch, "A technique for high-performance data compression," *IEEE Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[7] V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, no. 10, p. 707, 1966.

[8] S. Greenberg, "Using unix: Collected traces of 168 users," University of Calgary, Tech. Rep., 1988.

[9] R. A. Maxion, "Masquerade detection using enriched command lines," in *Proc. IEEE International Conference on Dependable Systems & Networks (DSN)*, 2003, pp. 5–14.

[10] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," *Statistical Science*, vol. 16, no. 1, pp. 1–17, 2001.

[11] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," *Journal of Computer Security*, vol. 6, no. 3, pp. 151–180, 1998.

[12] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection," in *Proc. 11th USENIX Security Symposium*, 2002, pp. 51–59.

[13] C. Krügel, D. Mutz, F. Valeur, and G. Vigna, "On the detection of anomalous system call arguments," in *Proc. 8th European Symposium on Research in Computer Security (ESORICS)*, 2003, pp. 326–343.

[14] G. Tandon and P. Chan, "Learning rules from system call arguments and sequences for anomaly detection," in *Proc. ICDM Workshop on Data Mining for Computer Security (DMSEC)*, 2003, pp. 20–29.

[15] W.-H. Ju and Y. Vardi, "A hybrid high-order markov chain model for computer intrusion detection," *Journal of Computational and Graphical Statistics*, June 2001.

[16] A. Liu, C. Martin, T. Hetherington, and S. Matzner, "A comparison of system call feature representations for insider threat detection," in *Proc. IEEE Information Assurance Workshop (IAW)*, 2005, pp. 340–347.

[17] B. K. Szymanski and Y. Zhang, "Recursive data mining for masquerade detection and author identification," in *13th Annual IEEE Information Assurance Workshop*. IEEE Computer Society Press, 2004.

[18] S.-L. C., S. M., and H. W. Guesgen, "Unsupervised learning of patterns in data streams using compression and edit distance," in *Twenty-Second International Joint Conference on Artificial Intelligence*, July 2011, pp. 16–22.

[19] 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 *15th National Conference on Artificial Intelligence and Machine*. AAAI Press, 1998, pp. 5–12.

[20] P. Parveen, J. Evans, B. Thuraisingham, K. Hamlen, and L. Khan, "Insider threat detection using stream mining and graph mining," in *Third IEEE International Conference on Privacy, Security, Risk and Trust*, Oct 2011.

[21] P. Parveen and B. Thuraisingham, "Unsupervised incremental sequence learning for insider threat detection," in *Proc. IEEE International Conference on Intelligence and Security (ISI)*, June 2012.