Unsupervised Ensemble Based Insider Threat Detection Using Hadoop

Siddhesh A. Bhure
Department of Computer Engineering
and Information Technology,
Veermata Jijabai Technological Institue,
Mumbai

Sowmiyaraksha R. Naik
Department of Computer Engineering
and Information Technology,
Veermata Jijabai Technological Institue,
Mumbai

***Abstract*—Insider Threat are often hard to find as their traces are buried within large data and logs they occurs rarely and when they occurs they hide well behind the normal operation of user.The detection of these requires identifying these suspicious patterns in a contextualized setting where behaviors are constantly upgrading over time.To do this this paper proposes Ensemble based detection approach which maintains the collective ensemble of the user's repetitive sequential patterns found in dynamic data stream.It applies ensemble stream mining approach and unsupervised learning approach to the problem of insider threat detection.This makes the approach suitable for identifying insiders who attempt to hide their activities by varying their behaviors over time.In unsupervised learning approach, compression-based techniques are used to model normative behavioural sequence which results in a classifier having high classification accuracy for data-streams containing traces of suspicious activities.This Ensemble-classifiers outperforms the static approaches for learning and improve efficiency over supervised approach.**

Keywords:- anomaly detection, insider threat, ensemble

I. INTRODUCTION

Recent news articles have reported that the cell phones of prominent Greek legislators were found to be bugged [1]. Rogue software was injected into the operational systems of the Greek cell phone provider, Vodafone Greece, which controlled a tap for incoming and outgoing calls on selected phones. The phone used by the prime minister and other high ranking officials were apparently targeted. This act was eventually traced to a malicious insider who had hacked the Vodafone system sometime in 2004 and installed the equivalent of a rootkit on an internal Ericsson phone s witch. The complexity of the attack could only be attributed to someone with intimate knowledge of entire switch operating software.

The inside attacker has been defined in many different contexts but there are many examples of insider attack known to most people.

For our purpose here, we define a malicious insider to be two classes of maleficent users; traitors and masqueraders. A traitor is a legitimate user present in an organization who has been granted access to systems and resources, but whose actions are violating the policy, and whose goal is to affect confidentially, integrity, or availability of some asset [1]. The traitor uses his/her authorized credentials when perpetrating their malicious actions, such as in the Greek cell phone provider case mentioned above. The most common example of an insider is a masquerader. masquerader is an attacker who succeeds in stealing a authorized user's identity and pretend to be another user for malicious purposes. Credit card fraud are the best example of masqueraders.

We may distinguish traitors and masqueraders based upon the amount information each has. A traitor of course has full knowledge of the systems . The masquerader may have far less knowledge than the traitor. Furthermore, an insider attack may be due to an innocent mistake by a legitimate user. Hence, insider attack may also be distinguished by intent of the user's actions. Traitors and masqueraders are two views of what we consider to be the insider threat.

The modern computing infrastructure is under attack from a massive potential insider threat issue that is constantly growing [2]. 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.

Instead of using data that is labeled as anomalous or not, an unsupervised learning approach can be used.[3] 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.

Continuous data is often associated with insider threat detection and classification. The Continuous Data is Dynamic in nature and unbounded in length which outperforms any static approach supervised or unsupervised. In these System pattern of any Normal User and suspicious activity threat can evolve gradually over time like a inexperience programmer can developer his skills to become a expert one over time.to closely mimic legitimate users behaviour an insider can change his action pattern at either end of this evolution can look drastically different when compared with each other.those natural changes won't be treated as anomalous behaviour instead we call them as natural concept drift traditional static approach will raise unnecessary false alarms in such cases because they are unable to handle them these methods encounter high false positive rates. Learning models must be highly adaptive in nature with evolving natural concept drift and efficient at developing models from large amount of data to rapidly check for anomalous activity.

Because of these reasons,the insider threat detection problem can be visualize as a stream mining problem of continuous data.The method which is used for detection whether it is unsupervised or supervised it must be highly adaptive to correctly adjust with these natural concept drift under these condition Ensemble based learning and Incremental Learning [4]-[6]are the two adaptive approaches in order to achive this goal.we get the collective vote on the final classification from an Ensemble of K-models that reduces the false negative and false positive fot the data set.In this process old models are updated and new ones are created,to be more precise , the least accurate model is rejected to always maintain an ensemble of exactly K size

To adjust with natural concept evolution an Ensemble of multiple unsupervised stream based learning is maintain.During learning repetitive sequence patterns from users action or commands is stored in a model called Quantized Dictionary.longer patterns having higher weights due to frequent appearances in data stream are considered in dictionary. A collection of k-models of type Quantized Dictionary is an Ensemble in this case. A new Quantized Dictionary model is generated when new data arrives or gathers. To find the anomalous pattern sequences within this new data set we will take the majority voting of all models.We will update the ensemble and will discard the least accurate model therefore, the Ensemble always keeps the latest model as stream evolves and preserves high detection accuracy as both legitimate and illegitimate behaviours evolve over time.

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 Ensemble based Stream learning solution (UESL) that finds pattern sequences from successive user actions or commands using stream based sequence learning. Third, we effectively integrate multiple UESL 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 Unsupervised stream learning in an ensemble based stream mining.

The rest of the paper is organized as follows. Section II presents Related Work Section III presents proposed stream based insider threat detection approach. Section IV discusses about scalability using Hadoop, MapReduce Section V discusses about the Experiments and Results Finally Section VI concludes.

II. RELATED WORK

Insider threat detection has utilized the ideas from intrusion detection or external threat detection areas [2], [10].,Supervised learning is used to detect insider threats. System call traces from normal user activity and anomalous data are gathered [11]; features are extracted from this using n-gram and finally, trained with classifiers. Authors [12] makes use of 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 ,object path and error status of each system call are used as features in various supervised methods [13], [14]. A supervised learning model based on hybrid high-order Markov chain model was used by researchers [15].To Detect the anomalous behaviour a signature behavior for a particular user based on the command sequences that the user executed is identified.

a number of detection methods were applied by Schonlau et al. [10] 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 The detection methods are supervised based on multistage markovian model, and the combination of Bayes and Markov approach.

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.Our approach On the other hand, will learn from evolving data stream. In this paper, we show that our approach is unsupervised and is as effective as supervised model (incremental).

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

Researchers have explored Unsupervised learning [16] for insider threat detection. However, this learning algorithm is static. Although, our approach is unsupervised, but at the same

time learns from evolving stream over time and more data as compared to previous one will be used for unsupervised learning.

In anomalous behaviour detection traditionally one class support vector machine (SVM) algorithm (OCSVM) is used. The authors have investigated that SVMs using binary features and frequency based features. The one-class SVM algorithm along with binary features performed the best. To find frequent patterns, Szymanski et al. [17].

A. Stream Mining

A new data mining area Stream mining where data is continuous. In addition,attributes of data may change over time (concept drift). Here, Unsupervised and supervised learning need to be adaptive to cope with changes. There are two ways by which adaptive learning can be developed. One is ensemble-based learning and the other one is incremental learning. Incremental learning is used in user action prediction [18], but not for anomaly detection. Davidson [19] introduced Incremental Probabilistic Action Modeling (IPAM), based on one-step command transition probabilities estimated from the training data. These 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 used for anomaly detection.

The unsupervised learning is applied to detect insider threat in a data stream [20]. This work does not consider sequence data for detection. The 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 adjust 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.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.

III. STREAM BASED INSIDER THREAT DETECTION

Insider threat detection related data is stream based in nature. Data may be gathered over time, maybe even years.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 or session and contains the data which arrived during that period of time

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 dotted straight line represents the decision boundary of previous chunk.The dark straight line represents the decision boundary of its own 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), black dots represent anomaly data (True Positive), and striped dots represent the instances victim of concept drift.

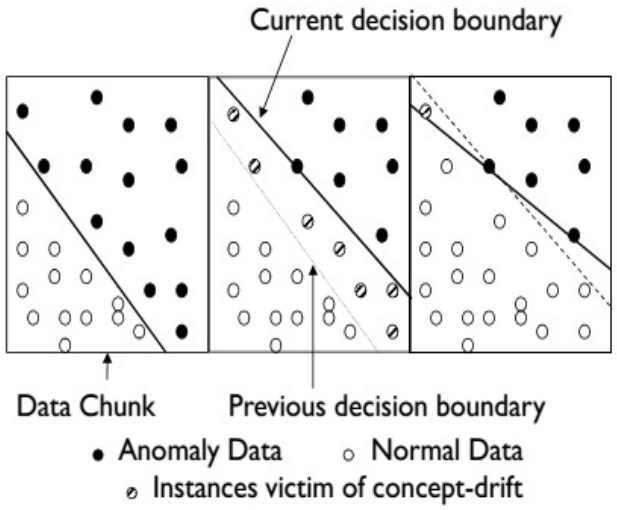

Fig. 1. Concept Drift In Stream Data

There are two different cases in the Figure 1 as follows.

Case 1 :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.

Case 2 :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 miss-classify both normal and anomalous data. Therefore, both FP (False Positive) and FN (False Negative) may go up at the same time.

The ensemble classification procedure is as follows We first use static, unsupervised UESL to train models from an individual chunk. UESL 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

Model Update: We always keep an ensemble of fixed size models (K in that case). Hence, when a new chunk is processed, 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 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: 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., $liftliftliftliftliftcomcomecomecomecomecome$, 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 [7]) to prepare a dictionary. We called it LZW dictionary. LZW dictionary has a set of patterns and their corresponding weights according to

$$w_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{1}$$

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 Quantized dictionary has a set of patterns and their corresponding weights. Here, we use edit distance to find the longest pattern [8].Edit distance is a measure of similarity between pairs of strings 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.

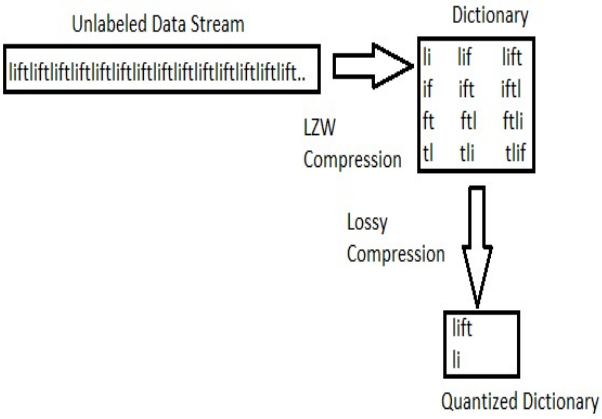

Fig. 2. Dictionary construction

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

Algorithm 1 shows the basic building block for Ensemble Updation. Algorithm 1 takes the most recent data chunk S, ensemble E and test chunk T. Lines 3 to 4 generate a new Quantized Dictionary from the most recent chunk S Lines 5 to 9 test chunk T for anomalies for each model in the ensemble.Lines 13 to 24 find and label the anomalous patterns in test Chunk T according to the majority voting of the other models in the ensemble. Finally, line 29 updates the ensemble by discarding the model with lowest accuracy. An arbitrary model is discarded in the case of two or more models having the same low performance.

Algorithm 1 Update the Ensemble

```
Input: E (ensemble), T (test chunk), S (chunk)
Output: E' (updated ensemble)
M' ← NewModel(S)
E' ← E ∪ {M'}
for each model M in ensemble E do
    A_M ← T
    for each pattern p in M do
        if EditDistance(x, p) ≤ α = 1/3 of length then
            A_M = A_M − x
        end if
    end for
end for
for each candidate a in ⋃_{M∈E'} A_M do
    if round(WeightedAverage(E', a)) = 1 then
        A ← A ∪ {a}
        for each model M in ensemble E' do
            if a ∈ A_M then
                c_M ← c_M + 1
            end if
        end for
    else
        for each model M in ensemble E' do
            if a ∉ A_M then
                c_M ← c_M + 1
            end if
        end for
    end if
end for
E' ← E' − {choose(arg min_M(c_M))}
```

Algorithm 2 Quantized Dictionary

```
1:  Input: D = {Pattern, Weight} (LZW Dictionary)
2:  Output: QD (Quantized Dictionary)
3:  Visited ← 0
4:  while D ≠ 0 do
5:      X ← D_j | j ∉ Visited, D_j ∈ D
6:      Visited ← Visited ∪ j
7:      for each pattern i in D do
8:          if EditDistance(X, D_i) = 1 then
9:              P ← P ∪ i
10:         end if
11:     end for
12:     D ← D − X
13:     if P ≠ 0 then
14:         X ← choose(arg max_i(w_i × l_i)) |
            l_i = Length(P_i), w_i = Weight(P_i), P_i ∈ P
15:         QD ← QD ∪ X
16:         D ← D − P
17:     end if
18:     X ← D_j | j ∉ Visited, D_j ∈ D
19:     Visited ← Visited ∪ j
20: end while
```

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 don't know the possible sequence of future operations by the user. So, we use LZW algorithm [7] 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 2 demonstrates how we generate an LZW dictionary from the data stream.

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 5 picks a pattern (e.g., li). Lines 7 to 9 find all the closest patterns that are 1 edit distance away. Lines 13 to 16 keep the pattern which has the highest weight multiplied by its length and discard the other patterns. We repeat the steps (line 5 to 16) 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.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 $X\%(say \geq 30\%)$.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

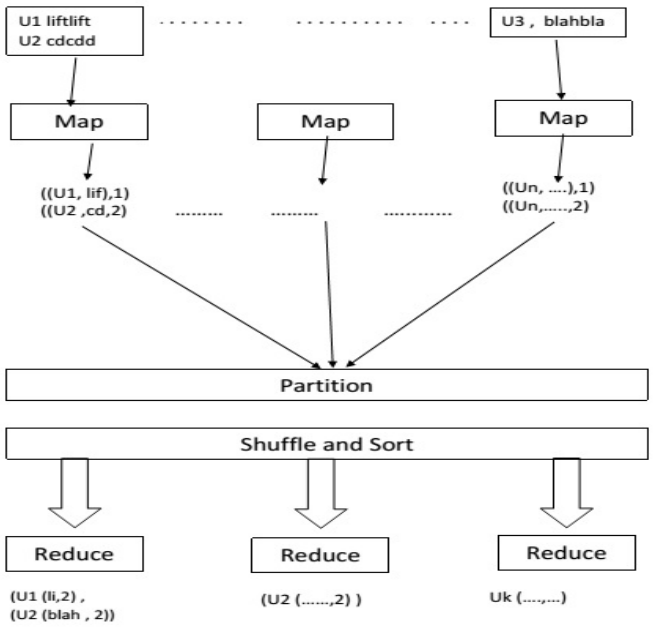

Fig. 3. MapReduce LZW Dictionary Construction

Algorithm 3 LZW Dictionary Construction using Map-Reduce(2MRJ)

1: Input: $gname : groupname$, $cseq : commandsequences$
2: Output: $Key : (gname, commandpattern(css))$, $count$

3: $map(stringgname, stringcseq)$
4: $start \leftarrow 1$, $end \leftarrow 2$
5: $css = (cs_{start} \cdots cs_{end})$
6: **if** $css \notin dictionary$ **then**
7: $\quad$ *Add css to thedictionary*
8: $\quad emit(pairs(gname, css), integer1)$
9: $\quad start \leftarrow start + 1$
10: $\quad end \leftarrow end + 1$
11: **else**
12: $\quad emit(pairs(gname, css), integer1)$
13: $\quad end \leftarrow end + 1$
14: **end if**

15: $reduce\ (pair(gname, css), (cnt1, cnt2, \cdots))$
16: $Sum \leftarrow 0$
17: **for** all $cnt \in (cnt1, cnt2, \cdots)$ **do**
18: $\quad sum \leftarrow sum + cnt$
19: **end for**
20: $emit(pair(gname, css), integersum)$

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.

IV. SCALABILIY USING HADOOP AND MAP-REDUCE

Construction of LZW dictionary and quantized dictionary is time consuming. We would like to address scalability issues of these algorithms. One possible solution is to adopt parallel/distributed computing. Here, we would like to exploit cloud computing based on commodity hardware. Cloud computing is a distributed parallel solution. For our approach, we utilize Hadoop and MapReduce based framework to facilitate parallel computing.

A. Scalable LZW and Quantized Dictionary Construction using Map Reduce Job

Our proposed approach exploits two map-reduce jobs The first MR job is dedicated for LZW dictionary construction in (Figure 3) and the second MR job is dedicated for quantized dictionary construction in (Figure 4).

In the first MR job, Mapper takes userid along with command sequence as an input to generate intermediate (key, values) pair having the form ((userid, css), 1). Note that css is a pattern which is a command subsequnce. In Reduce phase intermediate key (userid,css) will be the input.Here, keys are group ed together and values for the same key (pattern count) are added. For example, a particular user 1, has command sequences "liftlift".Map phase emits ((u1, li), 1) ((u1, lif), 1) value as intermediate key value pairs (see middle portion of Figure 3). Recall that the same intermediate key will go to a particular reducer. Hence, a particular user id along with pattern/css, i.e., key will arrive to the same reducer. Here, reducer will emit (user id, css) as key and value will be how many times (aggregated one) pattern appears in the command sequence for that user

Algorithm 3 presents pseudo code for LZW dictionary construction using a map reduce job. In Algorithm 3, input file consists of line by line input. Each line has entries namely, gname (userid) and command sequences (cseq). Next, mapp er will take gname (userid) as key and values will be command sequences for that user. In mapper we will look for patterns having length 2, 3, etc. Here, we will check whether patterns exist in the dictionary (line 6). If the pattern does not exist in the dictionary, we simply add that in the dictionary(line 7), and emit intermediate key value pairs (line 8). Here, keys will be composite having gname and pattern. Value will be frequency count 1. At line 9 and 10, we increment pointer so that we can look for patterns in new command sequences (cseq). If the pattern is in the dictionary, we simply emit at line 12 and cseq's end pointer is incremented. By not incrementing cseq's start pointer we will look for super-set patterns.

Reducer will run from line 15 to line 20 in Algorithm 3. In combiner we aggregate them at line 18. Finally, in line 20, we emit composite key (gname and pattern) and aggregated frequency count.

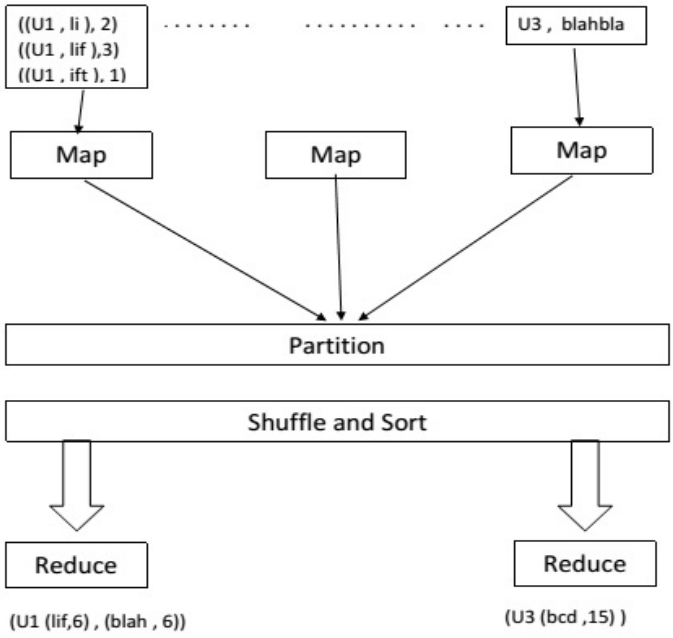

Fig. 4. MapReduce Dictionary Quantization

Algorithm 4 Compression/Quantization using Map-Reduce(2MRJ)

1: Input: $line : gname, commandsequence(css), frequencycount(cnt)$
2: Output: $Key : (gname, commandpattern(css))$

3: $map(stringline, string)$
4: $gname \leftarrow Spilt(line)$
5: $css \leftarrow Spilt(line)$
6: $cnt \leftarrow Spilt(line)$
7: $emit(gname, pair(css, cnt))$

8: $reduce(gname, (pair(css_1, cnt_1), pair(css_2, cnt_2), \cdots))$
9: $Sum \leftarrow 0$
10: **for** all $pair(css_i, cnt_i) \in ((pair(css_1, cnt_1), pair(css_2, cnt_2) \cdots)$ **do**
11: $H \leftarrow H + pair(css_i, cnt_i)$
12: **end for**
13: $QD = QuantizedDictionary(H)$
14: $emit(gname, pair(css, integer\ sum))$
15: **for** all $css_i \in QD$ **do**
16: $emit(gname, pair(css_i, count(css_i)))$
17: **end for**

In the second map reduce job quantization of the dictionary will be carried out (see Figure 4).Mapper will carry simple transformation by generating key based on user id, and value based on pattern frequency. In Algorithm 4, mapper will take each line as an input from input file produced by 1st map reduce job. Here, mapper will take input (userid,pattern) as key and frequency as value. Mapper emits intermediate key value pair where key will be user id and value will be concatenation of pattern and frequency.All patterns of a particular user and corresponding frequency will arrive at the same reducer. The reducer will conduct all pairwise edit distance calculation among all patterns for a particular user. Finally, user id as key and longest frequent patterns as value will be emitted by the reducer

At the reducer, each user (gname) will be input and list of values will be patterns and their frequency count. Here, compression of patterns will be carried out for that user. Recall that some patterns will be pruned using Edit distance.For a user, each pattern will b e stored into Hashmap, H. Each new entry in the H will be pattern as key and value as frequency count 1. For existing pattern in the dictionary, we will simply up date frequency count (line 11). At line 13 dictionary will be quantized and H will be up dated accordingly. Now, from quantized dictionary(QD), patterns along frequency count will be emitted as values and key will be gname (at line 14).

V. Experiments and Result

The following results compare our approach, Unsupervised Ensemble based insider threat detection, with a supervised method modified from the baseline approach suggested by Maxion [9], it uses Naive Bayes' classifier (NB). The modified version compared in this paper is more incremental in its model training, and thus we will referred it 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.

In our approach statistics are gathered using in following ways;The amount of test chunks to go through decreases as the ensemble size which is being used grows For an ensemble size of 1, models are built from chunk 1 and chunks 2 through 8 are considered testing chunks.Chunks 4 through 8 are considered to be testing chunks For an ensemble size of 3.Every new Chunk is used to update the models in each ensemble (in this case chunk 4) after a majority vote is reached and the test data is classified as an anomaly or not.In the example of ensemble size of 3 after the new model is built, all models vote on the possible anomaly this contribution decides which model is considered the least accurate and thus need to discard. The model that have survived the elimination round are used FP TP FN and TN.The data that requires calculation is done using four above mentioned values.Compression and model replacement is used by model updating process.The least accurate model is discarded to maintain the highest accuracy After other models are updated.

We have compared the NB-Incremental and UESL algorithms on the basis of true positive rate (TPR), false positive rate (FPR),execution time, accuracy. TPR and FPR is measured

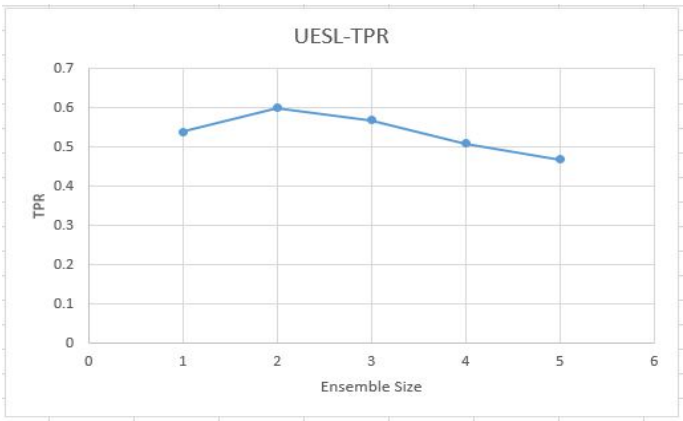

Fig. 5. Comparison of True positive rate with Ensemble-Size

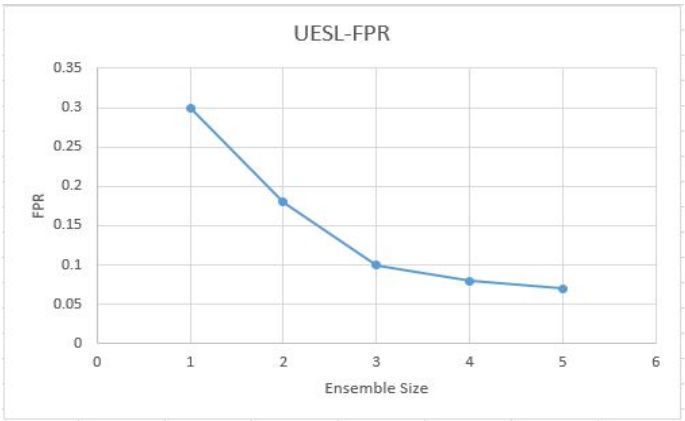

Fig. 6. Comparison of False Positive Rate with Ensemble- Size

by

$$TPR = \frac{TruePositives}{TruePositive + FalseNegatives} \tag{2}$$

$$FPR = \frac{FalsePositive}{FalsePositive + TrueNegative} \tag{3}$$

These are the rates at which actual insider threat incorrectly or correctly identified.A true negative(TN) is an identified normal data that is not an anomaly whereas a true positive(TP) is an identified anomalous behaviour that is actually an anomaly.A false positive(FP) is an identified anomalous behaviour but that is actually just normal data, and a false negative(FN) is an identified normal data that is actually an anomaly.

Accuracy defines the algorithms ability to pick out correct and incorrect insider threat instances as a whole

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

A. Choice of Ensemble Size

Here we show the Analysis of various parameter with respect to Ensemble Size. we can see the as the Ensemble Size increases TPR decreases.In Figure 5 we can see the steady decrease in TPR with respect to Ensemble size however we can see the larger decrease in FPR between Ensemble size 1 to 2 than in Ensemble size 3 to 4 and 4 to 5 (Figure 6). Thus we have observed that as the Ensemble size decreases we can achieve lower FPR but at the expense of also lowering TPR

In Figure 7 we can find that the rate at which positive and negative instances without punishing the algorithm for wrong classification using accuracy metrics UESL performs better as Ensemble size increases.

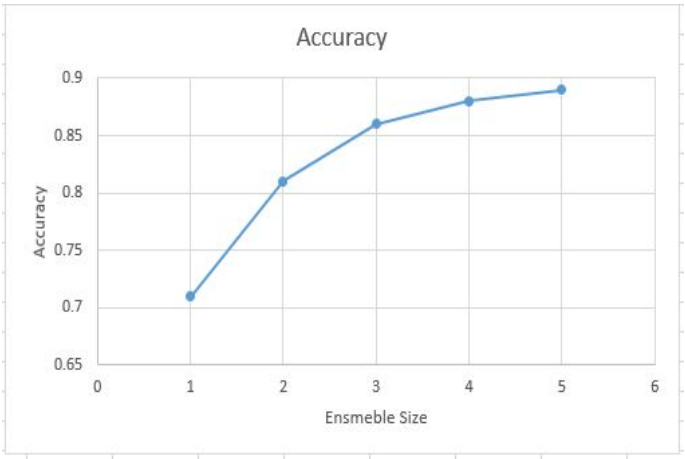

Fig. 7. Accuracy Measures

VI. CONCLUSION

Insider threat detection is a very important problem requiring critical attention. This paper presents approach to detect insider threats through augmented unsupervised and supervised learning techniques on evolving stream. Here, we have considered sequence and non sequence stream data.

We examine the problem of insider threat detection in the context of command sequences and propose an unsupervised ensemble based learning approach that can take into account concept drift. The approach adopts advantages of both compression and incremental learning. A classifier is typically built and trained using large amount of legitimate data. However,training a classifier is very expensive, and furthermore, it has problems when the baseline changes, as is the case in real life networks. We acknowledge this continuously changing feature of legitimate actions, and introduce the notion of concept drift to address the changes. The proposed unsupervised learning system adapts directly to the changes in command sequence data. In addition, to improve accuracy, we use an ensemble of K classifiers, instead of a single one. Voting is used, and a models of classifiers is used because classifiers with more recent data gradually replace those that are outdated. We address an important problem and propose a novel approach.

For sequence data, our stream guided sequence learning performed well, with limited number of false positives as compared to static approaches. This is because the approach adopts advantages from both compression and ensemble-based learn-

ing. In particular, compression offered unsupervised learning in a manageable manner and on the other hand ensemble based learning offered adaptive learning.

Compressed/quantized dictionary construction is computationally expensive. It does not scale well with a number of users. Hence, we look for distributed solution with parallel computing with commodity hardware. For this, all users quantized dictionary is constructed using a MapReduce framework on Hadoop.

REFERENCES

[1] Vassilis Prevelakis and Diomidis Spinellis. The athens affair. IEEESpectrum, 44:7:2633, 2007.

[2] M. B. Salem, S. Herkshkop, and S. J. Stolfo, "A survey of insider attack detection research," Insider Attack and Cyber Security

[3] K. Wang and S. J. Stolfo, "One-class training for masquerade detection," in Proc. ICDM Workshop on Data Mining for Computer Security

[4] 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

[5] 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.

[6] 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,"

[7] T. A. Welch, "A technique for high-performance data compression,"IEEE Computer,

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

[9] R. A. Maxion, "Masquerade detection using enriched command lines," in Proc. IEEE International Conference on Dependable Systems and Networks (DSN)

[10] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," Statistical Science

[11] S. A. Hofmeyr, S. Forrest, and A. Somayaji, "Intrusion detection using sequences of system calls," Journal of Computer Security

[12] Y. Liao and V. R. Vemuri, "Using text categorization techniques for intrusion detection,"

[13] C. Krugel, 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),

[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),

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

[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,

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

[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

[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"

[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.