A Unite and Conquer Based Ensemble learning Method for User Behavior Modeling

Abdoulaye Diop, Nahid Emad, Thierry Winter

A Unite and Conquer Based Ensemble learning Method for User Behavior Modeling

Abdoulaye Diop, Nahid Emad, Thierry Winter

Abstract—**IT companies use tools to analyze user and entity behavior to protect their information assets from insider threats. Although supervised machine learning methods seem to be the ideal solution for solving this problem, situations in which new employee activity data is labeled and balanced, are not so common. Besides, the data can have different origins, structures, and can be substantial. Therefore, it's difficult for a specific detection model to deal with and identify insiders in all cases effectively. To provide a solution to this problem, we are faced with methodological, algorithmic, and technological challenges. In this article, we try to meet these challenges by proposing a new approach based on ensemble learning methods to improve their performances from the point of view of accuracy and computation efficiency. With the detection of behavioral anomalies as a case study, we show the interest of this approach for its improvement of the prediction results and its efficacy on a high-performance computing system.**

Keywords—**Bagging, Boosting, High performance computing, Unite and conquer, Insider threat, User behavior modeling**

I. Introduction

In the cybersecurity domain, user and entity behavior analysis (UEBA) software are the tools used to stop insiders threats. In a company environment, insiders are mostly employees who misuse their access rights, or hackers that exploit flaws of the authentication systems with malicious intent. Companies use UEBA tools to determine if employee behavior is normal or abnormal. Employee behavior is hard to classify because it's nature diverges depending on the job role, the situation, and the organization structures. It can also evolve over time. They mainly use classification and anomaly detection techniques to detect insiders. Most of the techniques used to identify malicious activities are based on supervised machine learning methods. Other techniques use semi-supervised, unsupervised machine learning, and graph feature analysis. Their goal is to detect a divergence from an employee behavior profile or to find outliers in the all-around company activity data. The proposed solutions are diverse, but they are mostly facing the same problem of high false-positive/negative (FP/FN) and a lack of versatility. Depending on the company studied, a method initially performing well to detect an attack scenario can present an unstable detection accuracy. It makes sense to optimize a specific model for a particular company, but the cost of the software development and maintenance of these tools can represent a drawback. The data volume can also pose a challenge to implement a detection model. Machines limited in their computation power struggle to treat the massive amount of behavior data. A solution to this issue would be to build a model somehow adaptive, able to manage different and extensive data input. This model would have to consider detection accuracy, detection time, and maintainability constraints.

In this work, we propose an approach to counter this issue based on the use of unite and conquer and ensemble learning principles. Our solution uses the collaboration of multiple machine learning methods to build individual post-login activity profile. These profiles are used to classify a new activity record as regular or abnormal. The base methods of the ensemble of learners belong to the family of the unsupervised, supervised machine learning, and graph-based methods. We show that this approach makes it possible to obtain significant gains in accuracy relative to the base-methods, which constitute the global method proposed. To exploit the potential parallelism of the proposed solution, we implement it with high-performance parallel computing techniques and show its efficiency also in terms of execution speed. The rest of this paper is organized as follows. Section 2 presents some related works. In section 3, we define our approach and issued detection models. Section 4 presents the algorithm and parallel programming models according to which we implemented the proposed approach. Section 5 presents the results of a selection of our experiments and their analysis. Finally, Section 6 concludes this article and gives indications about some perspective of this work.

II. Related Work

Due to the nature of insider attack, using unsupervised learning and anomaly detection seems to be a natural decision to handle this type of issue [1], [2]. Haidar et al. [2] proposed a semi-supervised detection method using an ensemble-based scheme with two base classifiers; a one-class support vector machine (OcSVM) and an isolation forest (IForest). Moreover, they added a progressive update method using false positive oversampled FP chunks to refine their pre-generated models. A human domain expert labeled the FP results. Employee behavior can be considered as the dominant normal activity class and the insider action as an anomaly. However, an individual's behavior is a difficult phenomenon to classify. A behavior model is susceptible to be perturbated by new, unexpected or unusual good action and should evolve over time, since the studied individuals might change their habits. These specificities might not be capture by behavior profiles based on anomaly detection methods. Hence they always suffered from high FP/FN rates. The solution proposes by [2] gives insight on how we can boost the OcSVM and the IForest algorithm to handle the behavioral exception, but not

This project is funded by ATOS/EVIDIAN/ANRT, and the University of Paris Saclay/UVSQ (UPS/UVSQ). Abdoulaye Diop (e-mail:mamadou-abdoulaye.diop@atos.net) and Nahid Emad (e-mail: nahid.emad@uvsq.fr) are members of the Li-PaRAD and Maison de la simulation laboratories of UPS/UVSQ, Thierry Winter (e-mail: thierry.winter@atos.net) and Abdoulaye Diop are funded by ATOS/EVIDIAN compagny.

the change of behavior over time. Using a periodic training scheme combined with a smart windowing of the training data can be a solution to this problem. However, anomaly detection methods cannot characterize the source of the problem, which means that they can spot unusual or un-popular data samples, but cannot give insight into their causes.

Graph-based methods are another approach to tackle the problem of insiders. Based on the graph features analysis, the detection mechanism is to spot anomalous subgraphs, nodes or links. They share the same issue as anomaly detection approaches. They are also sensitive to exceptions, and they do not give information on the nature of the anomaly. Gamachchi et al. [3] proposed to use a graphical processing unit to extract graph-based features from a graph built with multidimensional data from the CERT of Carnegie Melon university data. They fed the features to an IForest algorithm to detect outliers without profiling normal behavior. This technique focuses on the study of local graph anomalies as a potential indicator of deviant behavior. In this work, the behavioral study parameters are chosen in a certain way by a human operator who is an expert in the field.

In the overall literature, supervised learning approaches, especially the ones based on deep learning, showcase better performance than anomaly detection approaches. Particularly approaches based on neural networks. However, they need substantial and balanced data. Since the activity dataset naturally contains more normal activity sample than insiders (i.e class imbalances), this method can struggle to perform well and can be susceptible to bias and variance issues. Tuor et al. in [4] proposed unsupervised online approaches based on a deep neural network (DNN) and a recurrent neural network (RNN) (i.e. using an LSTM architecture) as a prospective filter for a human data analyst. Their best model obtained an anomaly score of in 95.53 percentile for the insider activity. Their goal was to diminish the workload of a security analyst.

In the past, researchers use a combination of bagging and boosting to manage the bias-variance tradeoff. Kotsiantis et al. [5] proposed a bagging and boosting combination with sum rules voting. They constituted a global ensemble learning method with separated sub-ensemble, respectively, using bagging or boosting. The hypothesis of each sub-ensemble is combined using a sum rule to get the final prediction. They tested their solution on 36 well-known datasets from the UCI repository of machine learning, and used decision tree (DT) C4.5, decision stump (DS), naive bayes network (NBN), and a rule learner (OneR) as base classifiers. Overall they obtain better results using their combination of bagging boosting compared to individual bagging and boosting with this base-classifier. In the domain of power management, [6] proposed a parallel combination of bagging and boosting for a regression problem. They used artificial neural network (ANN) as base-classifier, for short term electricity load forecasting. They independently boosted ANN on a different bag of the training dataset. Here the sub-ensembles are all boosting methods. The result of their forecasting is obtained by averaging the result of each boosted ANN. They compared their effect against ANN, only Bagged ANN, and only boosted ANN. They had the best results with their bagged-boosted combination. Fauvel et al. proposed in [7] a hybrid ensemble method called local cascade ensemble by combining bagging, boosting and mixture of expert (ME) [8] methods, in a decision tree scheme. They use their approach for estrus detection (i.e. in the milk production industries, estrus is the only period where a cow is susceptible to pregnancy). The ME method is a method base on a divide and conquers strategy. This method divides the problem space between classifiers, supervised by a gating network (i.e. a weighted average scheme). Each classifier is trained with a different part of the dataset. They use this combination of bagging and boosting to handle the bias-variance tradeoff, and use the diversification properties of ME to learn the specificities of different parts of the dataset. Their approach showed better results when compared with other classifiers and commercial solutions for estrus detection.

These are a couple of examples of the use of the combination of bagging and boosting. To our knowledge, in the domain of insider threat detection, there is no work proposing to use a bagging and boosting combination to solve this. However, [9] studied the effect of boosting on classifiers trained to detect insiders' attack. They create a meta-learner by aggregating the boosted classifier using a probability vote. They tested their method by comparing the base-classifiers firstly with their boosted version. They boosted artificial neural network (ANN), naive bayes network (NBN), support vector classifier (SVC), decision tree (DT), and logistic regression methods. They obtained mixed results by comparing the accuracy and the AUC-score of the base classifiers before and after boosting. This process improved NBN, SVC, DT, and LR slightly, not ANN and RF, which showed a slight decrease in accuracy and AUC-score. Using their meta-learner, they didn't get better efficiency. However, they have a better area under the ROC curve.

A. *Detailed contribution*

Therefore, we propose a new and customized approach to combine bagging and boosting inspired by the previous work in the insider threat detection domain and the mitigation of high bias and high variance problem. We opt for the same strategy as the ME method. However, we diversify the distribution of the training data set using bagging and boosting sampling techniques to handle the bias-variance tradeoff. Contrary to a divide and conquer strategy like ME, we propose a restarting strategy based on the unite and conquer approach, mixing individual classifier feedback to improve the training set. Hence, in this work, we propose:

- A new iterative boosting and bagging combination relying on a restarting strategy inspired by the unite and conquer method, particularly well suited for insider detection problems.
- A fault-tolerant implementation strategy, to maximize the contribution of the best base-methods or their combination.
- An implementation scheme combining anomaly detection and supervised learning methods depending on the available data.

- A custom scalable parallel implementation model that can deal with high data load, and that take advantage of the high-performance machine architectures.

III. Proposed approach

Insiders threat are reported in different types of companies' work structure (e.g. companies in information and technologies, finance, healthcare). This heterogeneous nature of employees' activity could appear even inside of a given organization. The insider attack scenarios are also diverse since they can target different types of assets of a company. As a consequence of this heterogeneity, a single detection method might not work to detect insiders in all cases [9]. Using an ensemble learning strategy can represent a solution to this issue. Most of the ensemble method uses a combination of classifiers to obtain a more accurate final prediction. For instance, we can compare the detection performance of the base methods composing the ensemble of learners. For the final prediction, one option is to select the base-method with the best accuracy. Another option can be to use a voting scheme to make all the base-methods contribute to the class prediction. In this article, we propose a way to detect insiders in multiple settings. Our approach combines bagging and boosting techniques by using a unite and conquer strategy.

This approach consists of making collaborate several boosted methods (i.e. called co-methods) in a bagging context to solve the problem of insider threat detection. These two ensemble learning techniques are specifically used to handle the high bias (i.e. underfitting) and high variance (i.e. overfitting) problems. A classifier suffers from high bias when it's unable to fit the structure of the training set. On the other end, a classifier suffers from high variance when it's too specialized on the training set. Bagging solves the problem of overfitting by diversifying the training set distribution stochastically and share it with multiple classifiers. Boosting can handle underfitting by specializing a classifier list to a training set, capturing its underlying structure. However, this operation can result in over-specialization on the training set, increasing the overfitting risk. Hence a combination of the two methods can help to manage the bias-variance tradeoff. Balancing this tradeoff is the ability for a classifier to generalize beyond its training set. This is an essential advantage for insider threat detection since this behavior analysis system is bound to analyze new activities records continuously. The intrinsic parallelism in bagging is an advantage of this method, mainly when most of today's applications deal with vast data quantities and need their processing and analysis on parallel and distributed architectures.

We call the proposed framework UCEL (i.e. for unite and conquer ensemble learning). Given several boosting methods able to learn employees' post-login behavior individually. The UCEL framework uses a combination in an extended manner of bagging and boosting methods to create behavior profiles. These profiles are built by learning usual employees' conduct and work patterns, from activity data recorded on their companies' information systems. These individual profiles can be used to analyze and classify the employee's recent or on-going activities as a normal activity, unusual or insider activities. We name this particular instance of UCEL, a behavior profiler model (see *Fig1*). We give more detail about this framework in the *Section 3.B*.

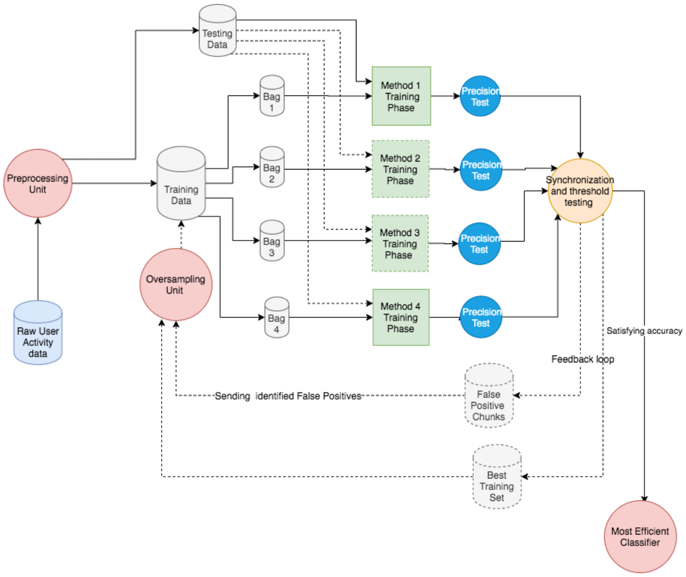


Fig. (1) Behavior profiler

A. *Unite and Conquer based method*

Unite and conquer is an approach initially used in linear algebra to solve large-size sparse linear systems and/or eigenvalue problems [10]. This approach consists of making collaborate several iterative methods (i.e. also called co-methods) to solve the same problem. This process accelerates the system's overall convergence by making use of intermediate results issued from an iteration of each co-method by all the others. This can be seen as a set of collaborative co-methods that share their restarting cycle parameters to choose the best of them and to reach convergence more quickly. Precisely, the aim of this sharing is to combine the intermediate results in order to define the best restarting information for each cycle of each of the co-methods allowing the global method to reach convergence faster. Let P be a large numerical problem to solve linear system, eigenvalue problem, etc. Let $L_1, L_2, \ldots, L_\ell$ be a set of iterative methods allowing to solve P, I_i^k be the initial condition (with $k = 0$) and restarting condition (with $k > 0$) of L_i (for $i = 1, \cdots, \ell$) and S_i^k be the approximated solution obtained by L_i at the end of its k-th iteration/cycle with I_i^k initial condition. The main steps of this approach to solve P are presented in algorithm 1.

B. *User behavior modeling with UCEL*

In a corporate environment, the vast majority of employees are not insiders; there is a natural class imbalance in their activity data. This means that there are many more samples of good activities recorded than there are of malicious activities. For example, if we focus on a single employee, insider action can be seen as a change in his/her usual work practices. These elements confirm that if we register the post-logging activities of companies' employees for an extended period, we will most likely have an imbalanced dataset. Since the flow of

Algorithm 1 Unite and Conquer Algorithm

```
1:  Start. Choose a starting matrix $[I_1^0, \cdots, I_\ell^0]$,
      let $k = 0$.
2:  Iterate. For $i = 1, \cdots, \ell$ do in parallel
3:    Compute $S_i^k$ by $L_i$ with initial condition $I_i^k$
4:    if (accuracy of $S_i^k$ is good enough) then STOP
5:    Share $S_i^k$ information with all other processes $j$
      ($j = 1, \cdots, \ell$ and $j \neq i$).
6:  Restart. Update initial condition $[I_1^{k+1}, \cdots, I_\ell^{k+1}]$
      for restarting by $f(S_1^k, \cdots, S_\ell^k)$ and go to 2.
```

activity data is continuous, labeling activity and balancing data before testing for insider threat might be more risky and costly for companies than the operation of characterizing anomalies when they are found. Hence, we propose to opt at first for the use of semi-supervised anomaly detection based UCEL method. We can then combine it with a supervised learning-based UCEL method when we have enough labeled and balanced data (i.e. using human operators to label the data or oversampling strategies [11] to deal with the imbalances). Hence the co-methods methods are chosen to be classic anomalies detection methods or supervised classification.

In a semi-supervised context [12], anomaly detection methods add samples with known labels to their data distribution to have extra information to their classification process. This action improves the decision boundary of classic unsupervised anomaly detection methods [2]. A behavior profiler with anomaly detection as co-methods can be alimented with a continuous feed of FP/FN samples, labeled by another security system, or human action. This profiler would work without a balanced dataset. It is important to note that individually none of base classification methods can't be considered as well adapted to all the situations. Each has unique advantages and disadvantages and can match the specificities of the companies activities. To enjoy the benefits of these methods, we propose their combination to the UCEL framework.

Before analyzing employee behavior by the proposed profilers, it is necessary to perform a data pre-processing step. In this step, we start by selecting samples from the raw activity data of a single employee and samples from an insider threat attack scenarios databases. We then perform classic feature engineering and finish by building a training, validation, and testing set from the cleaned data. The second step is to apply the main principle of unite and conquer to machine learning. The idea is to design our classification method using the same architecture of the unite and conquer methods for restarted iterative methods. We establish correspondence points between the two techniques. The system matrix corresponds to the original training data set. The subspace becomes a bag of data built with random sampling with replacement (RSR) and weighted random sampling (WRS). For instance, in the anomaly detection case, the co-methods or base-methods can be mainstream methods such as: (IForest), (OcSVM), robust covariance (Robcov) and local outliers factor (LOF). In the supervised learning case, we can also choose co-methods, such as: multilayer perceptron (MLP), gaussian naive bayes (GNB), KNN, and SVC. We can also build a solution using the same method but with different hyperparameters (i.e. changing the initial conditions). In other words, we can use the instances of the same base-method, which is equivalent to create the particular case of the UCEL methods called multiple base-method such as *multiple IForest* or *multiple Robcov*. However, the advantage of using co-methods differently is that it helps to build a heterogeneous consensus on the nature hypothesis of an analyzed behavior.

We start the first iteration by training the co-methods with bags generated with an RSR on the initial training data. We then test the co-methods with the validation set by computing the AUC-score of each co-methods. After that step, we then try to combine the strength of each co-method and build a weighted voting classifier (WVC) with their results. We then test its AUC-score against the score of the co-methods and the chosen detection threshold. The third step corresponds to the restarting step. We apply the principle of boosting on the previous training bag if the detection threshold is not reached. We start by gathering all the FP/FN of the co-methods and weigh them in the function of their popularity. The training data of the next iteration of a co-method is obtained by combining its most accurate training bag during the previous iterations, with the most popular FP/FN. Here, we use a RSR and a WRS again to build the new training bags. All the best bags correspondent to the co-method and WVC are stored, in order to use them in the following iterations. They are only updated when a new bag presents a better prediction score. We repeat the same process until the detection threshold, or the desired number of iteration is reached. The restarting strategy is a critical part of our model. It launches a new cycle of classifier training and testing. In theory, the new period has better starting conditions than the previous ones. For now, we used a simple restarting strategy. Still, the proposition of the ones with more effectiveness and sophistication will be the subject of our future research works (i.e. we can consider sending in addition to the best bag, the best hyperparameters in a multi-UCEL case).

Depending on input data, even with the boosting process, some co-methods might not be efficient do detect the insiders. Hence, they are not contributing to accelerate the convergence through the iteration. In those cases, UCEL still provides good results because the focus is always shifted to the best method and the best bags. This highlights the fault-tolerance capabilities of this approach. The bags are stochastically selected. So in the same conditions, the performance might drop if at a cycle the selection is unlucky. However, since we choose the best bag from the start, combined with the situation where the classifier is mistaken, the AUC-score is susceptible to oscillate and rise again. The stop condition for classic unite and conquer consists in reaching the chosen tolerance. The stop condition of the behavior profiler training is to have a base-classifier AUC-score reaching the chosen threshold. At that point, the trained co-method with or the weighted voting classifier the highest AUC since the first iteration is chosen to be the behavior profile.

IV. Parallel programming model for UCEL

One important aspect of the UCEL framework is its intrinsic multi-level parallelism. That means we can exploit coarse-grain inter co-methods as well as fine-grain intra co-method parallelism. Moreover, communications between co-methods can be synchronous or asynchronous. In this article, we focus on a synchronous implementation. Despite a loss of time due to the synchronization, the advantage of this model is the simplicity of its implementation and the existence of a certain determinism in the calculations. Another important aspect of UCEL framework is its heterogeneity. Indeed, the processes corresponding to base-classifiers could all be different. Thereby, it is possible to use an adapted hardware processor for each of them according to their natural parallelism. We assume that the targeted parallel architecture has a set of nodes; each of them could be itself a parallel processor. For the parallel implementation of UCEL, we consider a programming model where the nodes of underlying architecture act as a computing server (SN) or controller (CN). Under these hypotheses, each SN trains a classifier allowing predicting classes, selecting miss-classified data, and computing its AUC-score. Then, all of SN send the set of their results to the CN, which is in charge of determining which is the best set among them before sending this set to all the SN. This computation task of the CN constitutes a synchronization step and can be seen as a critical section. Consequently, its execution time will have a significant impact on the execution speed of the whole algorithm. According to the best information received, SN update starting conditions for a new cycle of their corresponding boosting classifier. In each cycle, if the detection accuracy threshold is reached by one of the co-methods or by the voting classifier, all the processes stop. Otherwise, a new cycle is launched where the best new training bag is created and transferred to SN for a new cycle.

Let T_S be the training set, V_S the validation set, and ℓ be the number of learners and bags of size m, $L^0 = [L_1^0, \cdots, L_\ell^0]$ be a set of initial learners, W_i^j be the set of miss-classified data issued from the jth cycle of the ith classifier and, B_{best}^j be the training set with the best AUC-score among $B_1^j, \cdots, B_\ell^j$. This bag is associated with L_{best} the most accurate learner and W_{best} the lightest FP/FN set. A classifier is considered as sufficiently trained if its AUC-score is larger than a precision threshold θ. The algorithm 2, called *behavior profiler*, depicts a parallel implementation of the UCEL framework according to the above programming model. Let a function V that creates a weighted voting classifier, and a function f that selects $(B_{best}^j, L_{best}^j, W_{best}^j)$ as the best results of each co-method received from all $i \in [1, \ell]$ processes.

Algorithm 2 Parallel behavior profiler
BP (in: T_S V_S, ℓ, q, θ; out: B_{best}, L_{best})

```
Start. Choose $\ell$, $m$, $L^0$ the $\ell$ learners, ...
Iterate. For $i = 1, \cdots, \ell$ do in parallel
  Iterate. For $j = 1, \cdots, q$
      Training and testing on SN$_i$
      Train $L_i^{j-1}$ on $B_i^j$, produce $L_i^j$, test $L_i^j$ on $V_S$ and select $W_i^j$
      Communication: send from SN$_i$ to CN
      Send ($B_i^j$, $L_i^j$, $W_i^j$, AUC-score($L_i^j$)) from CN$_i$ to SN
      Computation and stopping test on CN
      $WVC_j = V(L_i^j, AUC(L_i^j))$
      $B_{best}^j, L_{best}^j, W_{best}^j = f(L_i^j, B_i^j, W_i^j, WVC_j)$
      If (AUC-score($L_{best}^j$) $> \theta$) then STOP all processes
      Communication: send from CN to SN$_i$
      Send $(B_{best}^j, L_{best}^j, W_{best}^j)$ to all node $i$ for $i \in [1, \ell]$.
      Sampling on SN$_i$
      Set the bag $B_i^{j+1} = (1-\alpha) * W_{best}^j \cup (\alpha) * R_i^j$ where $R_i^j$ is the set of $(m_i - k_i^j)$ correctly predicted data in $B_{best}^j$ with $k_i^j = \mathbf{card}(W_{best}^j)$ and $\alpha$ is the updated weight given to miss-predicted data.
      Result.
      Set $L_{best}$ the best individual co-method or best weighted combination of co-methods during the iterations of all $\ell$ processes
```

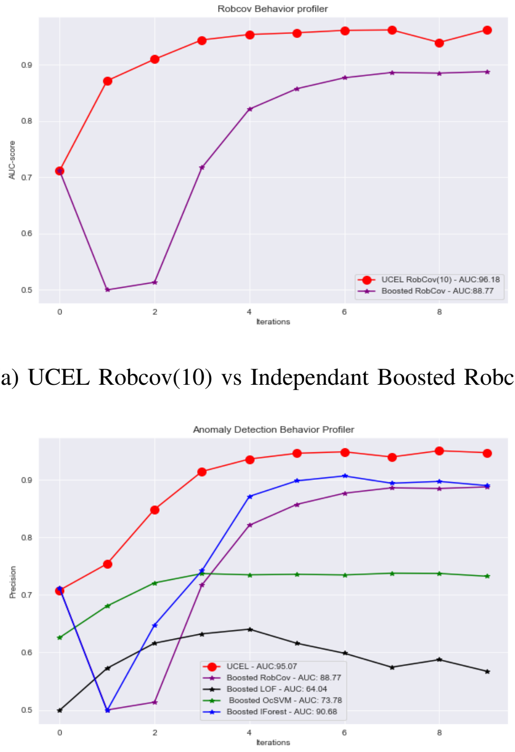
V. Experiments and performance evaluation

We implemented the behavior profiler *(BP)* with anomaly detection and supervised methods as co-methods. In order to evaluate the performances of this *BP*, we make use of two main performance metrics. The first one is the AUC-score allowing measurement of prediction performance and, consequently, (in)validating the approach. The second metric is speedup giving the gain of time due to the exploitation of parallelism in the *BP* algorithm. The dataset used for the experiments presented is the open-source *CERT Insider threat R4.2*. The *Computer Emergency Response Team* (CERT) dataset is an artificial insider threat dataset created by the CERT *National Center for Internal Threats* (NITC) division. It is a set of data composed of employees' normal post-connection activity in a synthetic context and insider attack scenario perpetrated by synthetic malicious actors. These scenarios are abnormal and suspicious activities that can be dangerous for businesses. To show the reliability of the *BP* algorithm, we presented the application of UCEL on three insider threat attack scenarios. However, we particularly focused on the following scenario: User begins surfing job websites and soliciting employment from a competitor. Before leaving the company, she/he uses a thumb drive (at markedly higher rates than their previous activity) to steal data. The other two scenarios description are available in the CERT dataset archive (i.e. in the *scenario.txt* file).

Method	TT Without UCEL	TT with UCEL
10 Robcov	0.61 - 0.53	0.96 - 0.96
4 AD	0.75 - 0.52	0.95 - 0.95
10 MLP	0.95 - 0.50	0.97 - 0.96
5 SM	0.98 - 0.98	0.99 - 0.99

TABLE (I) Train-Test AUC for BP with and without UCEL

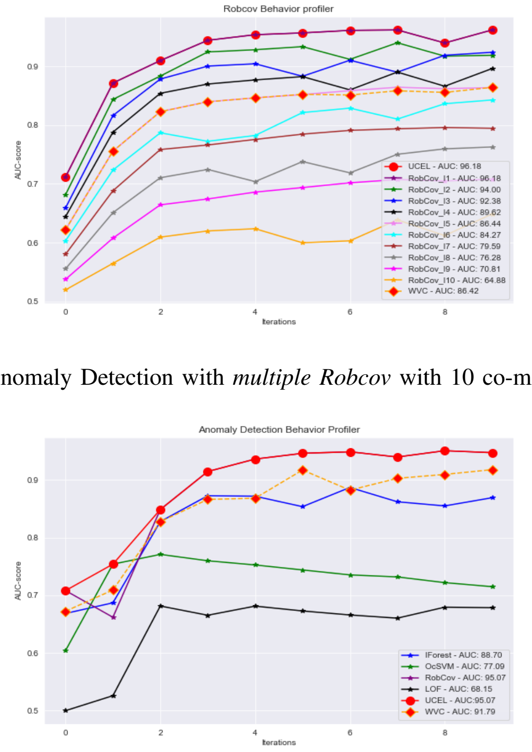
In order to validate our methods, we conducted three experimentations in a sequential execution context. First of all, we focus on the *scenario 2* to check the benefice of the UCEL approach. We compared the behavior profiler training result to the classic boosting technique applied iteratively to the base-



(a) UCEL Robcov(10) vs Independant Boosted Robcov

(b) 4AD vs independant Boosted 4AD method

Fig. (2) UCEL vs Boosting



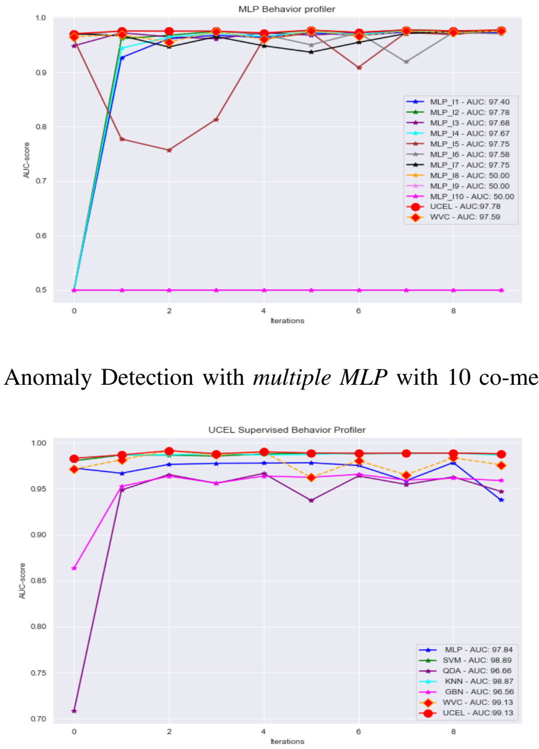
(a) Anomaly Detection with *multiple Robcov* with 10 co-methods

(b) Anomaly Detection Behavior Profiler with 4 co-methods

Fig. (3) AUC-score evolution throughout iterations in *BP* algorithm

methods. For the presented experiments the input values of the *Algorithm 2* are the following $l = 4$, $q = 10$ and $\theta = 1.0$. The sub-figure 2(a) presents the comparison of a behavior profiler built with 10 instances of the Robcov methods (i.e. UCEL(Robcov(10))) and an independent iteratively boosted Robcov(i.e. with its own FP/FN). For this experimentation, the UCEL(Robcov(10)) clearly outperforms the Robcov with the best hyperparameters. The figure 2(b) present the same type of comparison with a behavior profiler built with 4 different anomaly detection co-methods (4AD) trained using UCEL (i.e. UCEL(Robcov, LOF, IFOREST, OCSVM)), and the 4 boosted base-methods working independently. The result of this first test highlights that the behavior profiler with the UCEL approach presents a better AUC-score than the independently boosted base-methods. This approach uses the co-methods iteration's information to improve the classifiers' initial score to a greater extent. It leads to better final AUC-score and helps to reach a state of convergence faster.

For the second series of experiments, we tried to check the benefit of our UCEL approach to manage the bias and variance tradeoff. The sub-figure 3(a) and 3(b) showcase the AUC-score evolution of the co-methods during the training phase of the behavior profiler. These figures present the evolution of each co-method AUC-score when they receive the extra information gain of UCEL. The sub-figure 3(a) is an execution of 10 instances of the robust covariance classifiers. The different instances of Robcov are named *Robcov_I(k)*, with $k \in [1, 10]$. In the subfigure 3(b) 4 distinct anomaly detection classifiers are used as co-methods(i.e. L = [IForest, OcSVM, Robcov, LOF]). Since the score is pretty low after the first iteration, we can suspect that, individually, the co-methods suffer either from underfitting, overfitting, poor calibration of the hyperparameters, or don't have enough sample to establish a correct decision boundary. UCEL boosts their initial low AUC-score through the iterations. Some co-methods showcase a little drop of performance after reaching their peak or oscillating between low and high values from an iteration to another. For example, the OcSVM co-method doesn't seem to benefit from this boosting strategy after the second iteration. We suspect that it's also badly tuned. However, we need to do further investigation, particularly at the level of its objective function and the establishment of the decision boundary when we inject new elements. In the sub-figure 3(a) and 3(b) the precision threshold is never reached, but the AUC-score increases from 0.53 to 0.96 for (a), and 0.52 to 0.95 for (b). If we focus on the sub-figure(a), UCEL mostly improves the training accuracy from the methods with an initially low AUC-score. This is a direct consequence of the use of this particular combination of boosting and bagging, which improves weak learners' training errors by handling bias and variance tradeoff. However, the Robcov instances I9 and I10 don't seem to be improving a lot by UCEL. This is indicative of a poor selection of hyperparameters. This implies that tuning a classifier plays a non-negligible role in the performance of UCEL. Hence, except for the 9th and



(a) Anomaly Detection with *multiple MLP* with 10 co-methods

(b) Anomaly Detection with 4 different co-methods

Fig. (4) AUC-score evolution throughout iterations in *behavior profiler*

10th instances, the methods starting with low training AUC-score get improved by the restarting process injection of the wrongly classified element. This also indicates that even when two of the co-methods do not contribute to the classification performance, the UCEL approach still allows the other co-methods to get better classification results.

Table I shows the result of the training and testing AUC-score without and with the UCEL approach. Let *4AD* represents an execution UCEL with four anomaly detection methods and *5SM* a UCEL with five different supervised methods. The train-test(TT) results showcase high bias and high variance issues from the best method of the ten Robcov instances, and the four anomaly detection classifiers without UCEL. This points out that UCEL helps to manage bias and variance tradeoff to obtain better testing results. In the supervised learning case, the sub-figures 4(a) and 4(b) respectively present 10 MLP instances in (a) and 5 different supervised classifiers in (b). In this case, we also remark that UCEL only improves the co-methods with starting low AUC-score and doesn't improve the ones with an already high score. This a consequence of boosting and bagging working well only with weak learners as base-method. Strong learners can get improvement using this strategy, but not to the same extent than weak learners [9]. For instance, the KNN, GNB, and SVM classifiers are not improved by the UCEL process. Their AUC-score stays rather good and stable through the iterations. This is also indicative of well-tuned classifiers for this problem. However, in (b), the WVC of UCEL using all the co-methods produces better results than the individual classifiers. Hence the WVC was then chosen as the best model for the behavior profiler. In table I, we can observe that the best method of 10 MLP instances is still suffering from overfitting since the train test score varies from 0.95-0.50. The UCEL framework fixes this issue and helps to obtain a train-test score of 0.97-0.96. In the example with the five supervised learning methods, the AUC-score is 0.98, so pretty high for the individual co-methods. Despite that, UCEL adds an improvement of 1% to their scores. We can conclude that UCEL helps to improve the class prediction performance of the training and the testing error by managing the bias-variance tradeoff. Even if the classifiers are already performing well, UCEL might add a slight improvement with its weighted voting mechanism.

Our last experimentation is an overall study of our behavior profilers performance for the three scenarios of insider attacks. We obtained mostly satisfying test AUC-score and we present the results in Table II. In most of the scenarios, we tend to see better classification AUC-score for supervised methods than anomaly detection methods. This confirms the best strategy is to adopt the use of semi-supervised methods when the data is imbalanced, and then use supervised methods when the companies dispose of enough feedback. They're also the possibility to apply oversampling techniques to the imbalanced dataset before using supervised learning methods [11].

Scenario	10 Robcov	4AD	10 MLP	4 SM
1	0.95	0.95	0.95	0.96
2	0.96	0.95	0.96	0.99
3	0.82	0.64	0.98	0.93

TABLE (II) Test AUC for 3 insider attack scenarios

A. *Parallel performance analysis*

We highlighted that the UCEL approach gives reliable results to detect insiders. To study the performance of the parallel version of the behavior profiler, we ran an implementation of parallel algorithm 2 on the *GRID5000* platform. On the Lille cluster of GRID5000 we used 9 nodes with 4 cores each for our experimentation. The Python language and the *Multiprocessing*, *Multithreading*, and *mpi4py* APIs and libraries are used to express the algorithm' parallelism. The performance of the implemented *BP* is measured in terms of speedup that we can obtain when dataset size increases. We recall that speedup represents the ratio of the serial and parallel execution time of an implementation.

Figure 5 shows a significant speedup which reaches 4.3 when the *mpi4py* library is used. This is because the co-methods are working in parallel for their training and testing phase. This confirms that *mpi4py* is more fit to benefit from cluster hardware than the other one. Even though the execution is always faster for the parallel implementation, we can notice a limitation on the speedup gains when the number of data records increases past 500000. A small drop in the performance might occur due to the stochastic nature of the algorithm used. However, after 500000 records, the performance loss is continuous, and for all the parallelization libraries. This

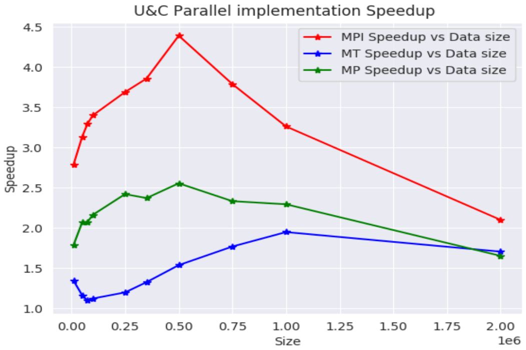


Fig. (5) Parallel *behavior profiler* on GRID'5K (8 co-methods run on 9 nodes)

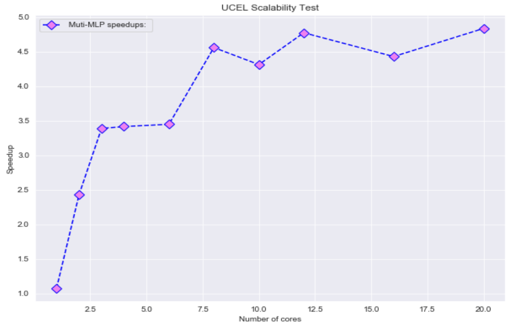


Fig. (6) Parallel *behavior profiler* strong scalability on GRID'5K (10 co-methods)

drop in performance is probably due to the multiplications of the communications at the synchronization steps. The co-methods have relatively different execution times. So the synchronization imposes the faster methods to wait for the slowest, before restarting a cycle. Clearly, we can conclude that a better execution time can be obtained when *behavior profiler* model is run in a high performance mode. However, the synchronizations in UCEL algorithms, limit those benefits. We will take into account the asynchronous communications in the future implementation of our *BP* model. Figure 6 presents the strong scalability test of supervised BP. We run this test using a UCEL implementation with 10 MLP as co-methods, and a fixed activity dataset size of 500000 entries. The result of this test highlights that the speedup of our parallel behavior profiler rises from 1 to approximately 4.5 when we increase the number of processing cores. This figure demonstrates the scalability character of the UCEL implementation.

VI. Conclusion

Due to the evolution of the threat landscape and continuous hacker innovation, user behavior analysis software has become an essential tool to counter insider threats. Based on the monitoring of user activities, the goal is to detect insiders in a company environment. In this article, we presented a new detection model based on the application of the unite and conquer approach to ensemble learning techniques, and addressed issues that insider detection software, face today. We highlighted that this framework, called UCEL, gives reliable results to detect insiders. Indeed, we show that UCEL increases the accuracy of all weak learners participating to behavior profiler. On the other hand, UCEL doesn't have a beneficial impact on strong learners. But the presence of these latter, as co-methods, has a positive impact on the results of the global method. By studying three attack scenarios, we also showed that the UCEL framework is reliable and gives good results. In addition, the study of these scenarios confirmed that the best strategy is to adopt the use of semi-supervised methods when the data is imbalanced, and then to use supervised methods when the companies dispose of enough feedback. To study the performance of a parallel version of our behavior profiler, we ran a kind of parallel client-server implementation of the algorithm 2 on the *GRID5000* platform. The presented results show that we can obtain up to 4.3 speedup. Nevertheless, due to a synchronization step in each boosting iteration, this speedup decreases when the dataset size increases. The solution to this issue is to address the asynchronous communication in the algorithm. Indeed, the asynchronous communication would allow overlapping communications with computation.

References

[1] L. Sun, S. Versteeg, S. Boztas, and A. Rao, "Detecting anomalous user behavior using an extended isolation forest algorithm: an enterprise case study," *In Computer Research Repository(CoRR)*, 2016.

[2] D. Haidar and M. M. Gaber, "Adaptive one-class ensemble-based anomaly detection: an application to insider threats," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–9, IEEE, 2018.

[3] A. Gamachchi, L. Sun, and S. Boztas, "A graph based framework for malicious insider threat detection," in *50th Hawaii International Conference on System Sciences*, 2017.

[4] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[5] S. Kotsiantis and P. Pintelas, "Combining bagging and boosting," *International Journal of Computational Intelligence*, vol. 1, no. 4, pp. 324–333, 2004.

[6] A. Khwaja, A. Anpalagan, M. Naeem, and B. Venkatesh, "Joint bagged-boosted artificial neural networks: Using ensemble machine learning to improve short-term electricity load forecasting," *Electric Power Systems Research*, vol. 179, p. 106080, 2020.

[7] K. Fauvel, V. Masson, E. Fromont, P. Faverdin, and A. Termier, "Towards sustainable dairy management-a machine learning enhanced method for estrus detection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3051–3059, 2019.

[8] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.

[9] A. J. Hall, N. Pitropakis, W. J. Buchanan, and N. Moradpoor, "Predicting malicious insider threat scenarios using organizational data and a heterogeneous stack-classifier," in *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5034–5039, IEEE, 2018.

[10] N. Emad and S. Petiton, "Unite and conquer approach for high scale numerical computing," *Journal of computational science*, vol. 14, pp. 5–14, 2016.

[11] N. M. Sheykhkanloo and A. Hall, "Insider threat detection using supervised machine learning algorithms on an extremely imbalanced dataset," *International Journal of Cyber Warfare and Terrorism (IJCWT)*, vol. 10, no. 2, pp. 1–26, 2020.

[12] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld, "Toward supervised anomaly detection," *Journal of Artificial Intelligence Research*, vol. 46, pp. 235–262, 2013.