Insider Threat Detection

Amira EssamEldin Ibrahim Elgebaly
University of Ottawa
aelge029@uottawa.ca

Amr Ashraf Mahmoud Elsherbiny
University of Ottawa
aelsh030@uottawa.ca

Aya Abd-Elnaser Muhammed Abd-Elgawad
University of Ottawa
aabde182@uottawa.ca

Hagar Hesham Kamel Ismail Negm
University of Ottawa
hnegm052@uottawa.ca

Mohamed Ibrahim Mohamed Abdelal
University of Ottawa
mabde149@uottawa.ca

***Abstract* — The threat of insiders received significant attention from a significant number of researchers, as a sensitive and decisive issue for most organizations in the digital world today. It is also a major source of information security and can cause further damage and financial loss in comparison any other threats. In this paper, we used feature engineering for features representing daily user activities. We tried different machine learning models such as random forests, xgboost and catboost. XGBoost outperformed other models by achieving the highest f1 score 86 %. Thus, our framework shows its ability to discover internal threats that leverage log files ingested by Kafka topics to predict any malicious activities carried out by employees in an organization. The detected malicious activities are then sent to a Kibana dashboard in real time for threat analysts to take action proactively.**

Keywords— Insider threats, Feature engineering, Machine learning, XGboost, Random forest, Catboost.

I. Introduction

When it comes to the security of an organization's assets - both physical and digital - serious measures are taken for their protection from outside threats. The focus is usually on outside threats since they are expected. Organizations are more prone to exploitation by malicious actors from the outside for various reasons. One example would be the theft of the confidential information they carry, by their competitors for instance. However, when it is an insider threat, it is usually unexpected and much more difficult to defend against. An insider is an individual entrusted by the organization, has access to its resources and has knowledge about its inner workings. When an insider performs a malicious act - whether intentionally or unintentionally - within the organization, they pose a huge threat. An insider threat can easily shatter the confidentiality, integrity, and availability of an organization's assets. Insider threats mostly occur out of spite by discontented employees who want to sabotage their organization's resources for revenge. Sometimes, it occurs due to carelessness or lack of awareness of the organization's security policies. Attacks involving insiders can be extremely challenging to detect and can cause huge losses. Restricting the privileges of insiders can prevent them from performing their required tasks. Nowadays, many surveys show how dangerous the threat is from within. For example, the 2021 Internal Threats Report from Cyber Security Insiders indicates that 98% of organizations feel vulnerable to internal attacks [1], and malicious insiders cause significant damage and losses to organizations. According to The Ponemon Institute reports [1], the total average cost of a threat increased by 31% between 2017 and 2019. That reaches on average: $ 8.76 million in 2017 and $11.45 million in 2019. It is therefore extremely important to identify internal threats with the lowest possible false positive rate. To do this we provided a solution that can discover the insider threat and detect malicious Users based on Day Activity by using machine learning and this is what we will be discussing in our paper.

II. Related Work

A. [2] proposed a method that can be broken down into two stages. The first stage is feature representation. The user behavior logs from five event sources are initially organized by user ID. Then, the activity of each user is ordered chronologically and divided into sessions that are marked by login and logoff activities. Features are extracted from the sessions of each user and transformed into feature sequences of the same length. TF-IDF representations are used. The second stage can be divided into two components that work together simultaneously – an insider threat detector and a user identifier that is based on behavioral modeling. To detect whether the session is either legitimate or malicious, the feature sequences are fed into an ensemble detector with an LSTM network as a base. An ablation study was carried out to evaluate the importance of each component used in the methodology. The results of this approach were also compared with previous results in the literature by evaluating the method on two different versions of the dataset – CERT4.2 and CERT6.2. This methodology achieved the highest areas under the curve (AUCs) on both datasets - 99.2% and 95.3% respectively.

B. The method proposed by [3] tackles the insider threat detection problem. Insider attackers cause powerful harm to the organization ranging from $100,000 to $500,000 as they can pass all security layers and steal sensitive data. Their proposed solution is using LSTM autoencoder. It was trained and evaluated on CMU CERT dataset v4.2 which contains 930 normal users and 70 malicious insiders and multiple instances for the attackers' scenarios. The model achieves accuracy (90.60%), precision (97%) and F1-Score (94%).

C. [4] propose a model composed of an ensemble of 4 different deep autoencoders, each trained on a specific type of audit data consisting of host-based and network-based logs. The host-based data consists of logon/logoff data, file operations, and USB operations whereas network-based data consists of http operations. A lot of information can be extracted from these audit logs, but this work focuses mainly on the frequency of events. For example, for http operations, information like amount of data and domains can be helpful but also computationally expensive to extract.

Therefore, only the number of occurrences of events like upload, download, and visit is considered. The autoencoders are trained in an unsupervised fashion to reduce the reliance on subject matter experts. To train the autoencoders, frequency based features are extracted from each set of logs, where the daily logs are divided into 1-hour long windows and events of each type are counted. After that, the data is fed to the deep autoencoders, which will then learn the normal behavior of employees through minimizing reconstruction error. Any abnormal event in the future, like an employee performing more file operations than they usually do, will have a larger reconstruction error, and can be classified as an anomaly. Finally, the data from the four autoencoders is fed to a 2-step top-N recommendation system. In the first step, top N% of each autoencoder output is marked as possibly anomalous, giving us 4 different vectors. Since a malicious insider is likely to behave abnormally in different ways, the 4 vectors are passed through another top-N recommendation system to combine their results and end up with a single vector to perform the necessary investigation.

D. [5] deal with a method that shows how one can build a psychological profile of a user based on both analysis of user sentiment for web browsing and email content, and thus internal threats can be predicted based on these analyzes and proactively detect malicious insiders with negative emotions. In this paper, an effective approach is proposed to proactively detect internal threats based on sentiment analysis from both web pages and emails. The system creates a sentiment profile for each informed person. This file includes the daily threat value and weekly threat value by both the sentiment analysis module and the malicious URL detection module. The CMU CERT dataset and the Enron Email dataset were used to evaluate the performance of this system. In fact, the result was as follows that this system can detect malicious insiders before doing any risky activities or any malicious activities and it can also detect internal threats as it achieved 100% sentiment rating and 96% accuracy for http content as well as email.

E. [6] designed architecture to solve imbalanced problem and detect the Insider Threat. "The dataset contains 32,770,222 event records generated by the1000 normal and anomalous users. 7323 of the generated activities are malicious insider instances that were manually injected by experts, representing 3 different scenarios of insider threat". The hardware for the environment is an Ubuntu 18.04.5 LTS operating system runs on a machine with an NVIDIA 1660Ti GPU on a 3.7GHz Intel Core i7-8700HQ, 16GB RAM. The authors try to detect the insider threat and solve imbalanced problem in the dataset. The authors give overview of previous work. They discussed several of papers and the main problem in the paper moreover, they tried to solve this one of this problem by add ADASYN method before the classification model to solve imbalanced data.

III. DATASET

In order to simulate the inner workings of an organization as realistically as possible without violating the privacy and confidentiality of real users from an actual organization, the dataset used for such a system needs to be synthetic data generated by models to cover different facets of digital organizational activity. For the implementation of this system, the CERT dataset [7] was used, specifically version 5.2 of the dataset. CERT is a collection of synthetically generated data depicting several insider threat scenarios in an organization. It includes several types of log files that simulate digital organizational activities for a period of 18 months.

A. Data Generation

For realistically generating the dataset, different aspects of an organization's topology need to be taken into consideration such as the social aspect, the behavioral aspect, the preferences of employees and the content they access, and their ownership of digital organizational assets.
The following models were used for this complex generation of synthetic data:

- **Relationship Graph Model:** which is a representation of the organizational relationships between employees. This portrays the social and functional structure of the organization.
- **Asset Graph Model:** which is a model that depicts the ownership of both physical and digital assets by the organization's employees, such as PCs, external devices and files.
- **Behavioral Model:** a model which depicts how each employee usually uses their assets.
- **Communication Model:** which portrays the relationship between employees through emails sent.
- **Topic Model:** depicting the user's interests in terms of web browsing tendencies and email content for instance.

B. Data Description

The dataset contains 5 different types of log files and their details figure 1:

1. **Logon Logs:** contains logs about user logon and log off activities.
2. **Device Logs:** contains logs about user usage of external thumb drives.
3. **File Logs:** contains logs about file operations performed by the user.
4. **HTTP Logs:** contains logs about web browsing activities of users.
5. **Email Logs:** contains logs about the emails sent and received by users.

The dataset also contains an LDAP directory which contains 18 files containing information about the employees, their supervisors, their functional units and departments within the organization over the 18 months of activity present in the dataset.

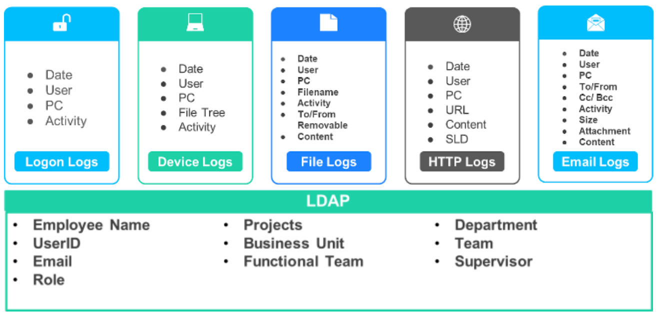

Figure 3.1. Dataset Description

C. *Scenarios*

Insiders have access to organizational assets, both physical and digital. They take advantage of this by carrying out malicious scenarios with the intention of theft of confidential information or sabotage when they feel disgruntled or are about to leave the organization. The following 4 scenarios are carried out by malicious users and their logs traces are present in the dataset:

1. Scenario one simulates an employee that is planning to leave the organization. This employee starts changing their usual behavior from being active only during working hours and not using removable drives to logging in after hours and using removable drives and uploading sensitive data to wikileaks.org.
2. Scenario two simulates an employee searching for a job in a competitor's organization. This can be seen in their web visits to employment websites. Right before leaving the company, their thumb drive usage drastically increases and they steal organizational data.
3. Scenario three simulates a disgruntled system administrator that is about to leave the company. This employee uses a thumb drive to transfer a keylogger to his supervisor's PC and steal his credentials. The system admin then uses the credentials to send mass emails to the whole organization causing panic.
4. Scenario four simulates an employee stealing interesting files from their colleagues' devices using thumb drives and sending these files to their home emails.

IV. METHODOLOGY

A. *Feature Extraction*

In this system, we ingest data from 5 different log domains, which are: Logon/logoff, device, file, email, and HTTP. We define working hours to be from 7 AM to 6 PM, and to identify each user's PC, we count each user's logons on different PCs, and choose the most frequent one to be the user's PC and the others can be either shared or belong to other users. This gives us a one-to-one mapping between users and PC which will be used later for feature extraction.

We then aggregate daily actions from each domain separately, and extract 2 different feature sets adapted from the literature, which we call fine-grained and coarse-grained feature sets. We then train models on each feature set separately, and choose the one that achieves the best performance. In the coarse-grained feature set [8], we count the number of events in each domain like number of logons, number of file operations, number of websites visited, but we do not go into too many details of what kind file was opened, or what kind of website was visited. This gives us 33 features in total, described in table 4.1.

Domain	**Features**
Logon	#Logons, #PCs logged on, #after hour logons, #logons on user's PC, #logons on other PC(s
Device	#device accesses, #PCs with device access, #after hour device accesses, #device accesses on user's PC, #device usage on other PC(s)
File	#file accesses, #PCs with file accesses, #distinct files, #after hour file accesses, #file access on user's PC, #file accesses on other PC(s)
HTTP	#web visits, #PCs with web visits, #URLs visited, #after hour web visits, #URLs visited from other PC(s)
Email Sent/Received	#emails, #distinct recipients, #internal emails, #internal

	recipients, #emails sent after hour, #emails with attachment(s), #emails sent from other PC(s)

Table 4.1. Coarse-grained feature set.

On the other hand, in the fine-grained feature set [9] we extract more information about each action performed by the users. This information is split into two parts, count features and statistical features. In count features, we compute detailed event frequencies like the number of file operations on different file types (doc, zip, etc.), their sizes, whether they are done locally or on a removable drive. In addition to that, we split websites by topic to have social websites, cloud services, hacktivist websites, and job search websites. For each of these events, we count their frequencies on the user's PC, on shared PCs, and whether they occurred during working hours or after working hours. This gives us 102 features.

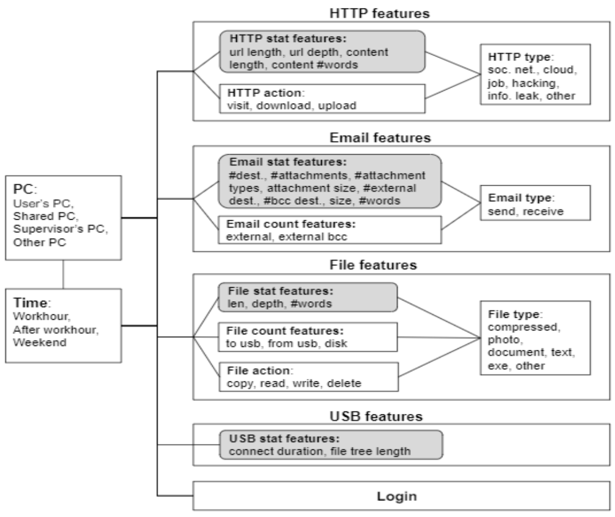
HTTP features
HTTP stat features: url length, url depth, content length, content #words
HTTP action: visit, download, upload
HTTP type: soc. net., cloud, job, hacking, info. leak, other
Email features
Email stat features: #dest., #attachments, #attachment types, attachment size, #external dest., #bcc dest., size, #words
Email count features: external, external bcc
Email type: send, receive
PC: User's PC, Shared PC, Supervisor's PC, Other PC
Time: Workhour, After workhour, Weekend
File features
File stat features: len, depth, #words
File count features: to usb, from usb, disk
File action: copy, read, write, delete
File type: compressed, photo, document, text, exe, other
USB features
USB stat features: connect duration, file tree length
Login

Figure 4.2. Fine-grained feature Set, adapted from [9].

B. Models

After extracting the features, we train machine learning models to detect malicious behavior based on daily logs. We choose to train 3 different models that have been known to perform well on such problems with class imbalance. The 3 chosen models are Random Forest, XGBoost, and CatBoost. Below, we provide a brief description of each:

- **Random Forest:** A Random Forest is an ensemble learning method that constructs multiple decision trees, where each of which is trained on a subset of the features and a subset of the training data sampled with replacement. After test time, each decision tree votes to the class it believes the test sample belongs to, and the votes are aggregated to output a single prediction.
- **XGBoost:** Extreme Gradient Boosting is an optimized open-source implementation of gradient boosting, which uses additive training to build the tree ensemble. This is similar to Random Forests in structure, but the learning process differs. In additive training, the model starts with a constant prediction, and incrementally adds trees that minimize a certain objective function. To approximate the objective function, it uses a second order taylor expansion of the function.
- **CatBoost:** CatBoost is another implementation of gradient boosting decision trees. It is famous for its ability to perform well with the default parameters, which save the time taken for hyperparameter tuning. In addition to that, it is able to handle both categorical and numerical data simultaneously.

C. Baseline Model

It is essential to choose a baseline model to compare our results with. Therefore, we choose the random forest model from [9], trained with daily granularity on idealistic training conditions. It achieves an instance-based f1-score of almost 62%.

V. Experimental Setting

A. Downsampling the data

As the dataset size is very large and would require a lot of computational power to process, we start by down sampling our data to decrease its size. Due to the imbalance in the dataset and the small number of malicious events compared to the benign ones, we sample all the malicious users, which are 100 users in our case, and then randomly sample another 100 benign users. The dataset is still imbalanced as the malicious users perform many more normal actions than malicious, but its size is manageable now for our computers to process.

B. Train/test Split

The first step in training our models is splitting the data into training and testing sets. We follow two different approaches to do so, which are instance-based splits and user-based splits. In the instance-based split, we sample 75% of all our instances (daily features of users) to be in the training set, and the rest is in the testing set. As different days of a malicious scenario can be in both training and testing sets, and assuming that the malicious actors' behavior is similar in both, this introduces a challenge for a possible data leak. To account for this, we also split our data based on users, where we sample 75% of the malicious and benign users to be in the training set, and the rest in the testing set. This guarantees that a malicious scenario is either completely in the training or in the testing set.

C. Feature Selection

As the number of features is very large, especially with the fine features, we perform feature selection by fitting a random forest to the data then ranking the features by importance. We vary the number of features from 5 to 80, and choose the number that maximizes the f1-score. We observe that by choosing 30 features achieves the maximum accuracy. Therefore, we fix our number of features to 30,

and then perform hyperparameter tuning of the 3 mentioned models.
Figures 4.2 and 4.3 show the feature importance of the fine-grained features and the coarse-grained features.

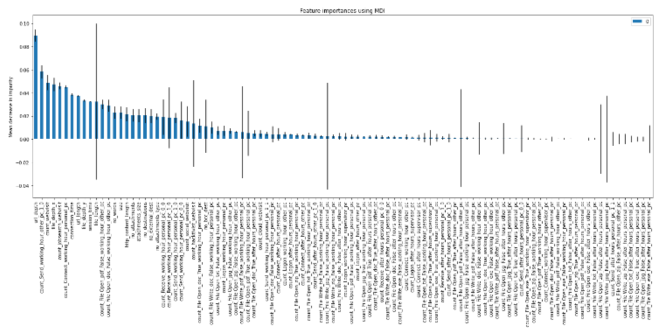

Figure 4.3. Fine-grained feature importance

D. *Hyperparameter Tuning*

After splitting our data, we perform hyperparameter tuning using randomized search on our different models. To tune the Random Forest, we tune the class weight to be either balanced or none, the maximum number of features to be either square root of all features or log2, and the maximum depth to be from 5 to 10 or none. For XGBoost, we tune the subsample ratio from 0.5 to 1, the maximum depth of trees from 3 to 25 and the number of gradient boosted trees from 50 to 300. As the CatBoost is known to perform well with the default parameters, we leave them as is.

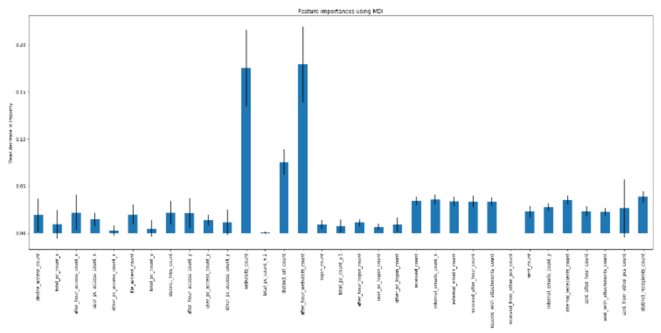

Figure 4.4. Coarse-grained feature importance

E. *Evaluation Metrics*

As the data is heavily imbalanced, we choose to employ the commonly used metrics of F1-score and false positive rate. Mathematically, the F1-score is defined as:

$$F1 = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

And the false positive rate is defined as:

$$F1 = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

We have 2 different strategies for evaluation, namely the instance-based and the user-based. For the instance-based, we simply evaluate the model on the daily correct/incorrect predictions it makes, whereas in the user-based we aggregate the predictions of a user and flag the user as malicious if any one of their instances is malicious. The assumption here is that if the first detection is early enough, the threat analyst will have time to act and therefore the later malicious actions will not take place in real life.

VI. Results and Discussion

As mentioned in the Methodology section, we have 2 types of features; coarse and fine features and 2 ways of splitting the data which are user-based splitting and instance-based splitting. We applied a random forest on the different combinations of settings to decide the best configuration based on F1-score. In the coming sections, the results will be shown.

A. *Coarse Features*

Instance-based split using coarse features using sklearn train_test_split with 0.25 for testing data results in F1_score:0.74, Recall: 0.6, Precision: 0.97, Accuracy: 99%. According to figure 6.1, we detect 142 malicious instances as benign and 7 benign instances as malicious.

User-based split using coarse features results in F1_score:0.71, Recall: 0.77, Precision: 0.66, Accuracy: 99%. According to figure 6.2, we detect 80 malicious instances as benign and 133 benign instances as malicious. Figure 6.3 represents users' evaluation confusion matrix that specifies that we detect all malicious users and 2 beginning users as malicious.

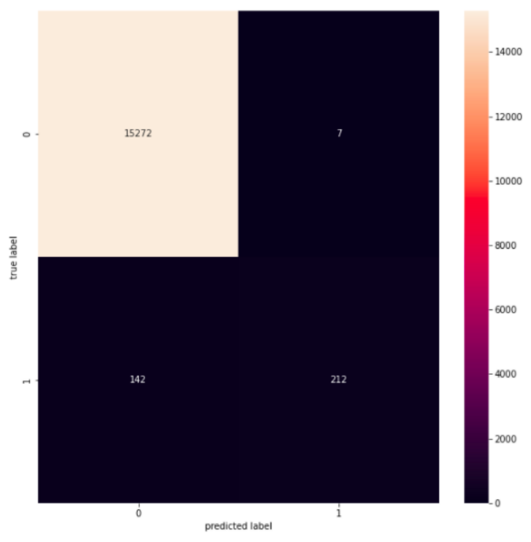

Figure 6.1. Instance-based splitting coarse features

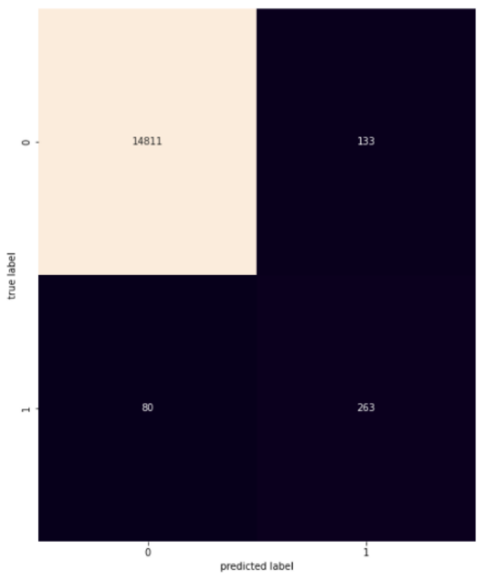

Figure 6.2. User-based split coarse features (Instance evaluation)

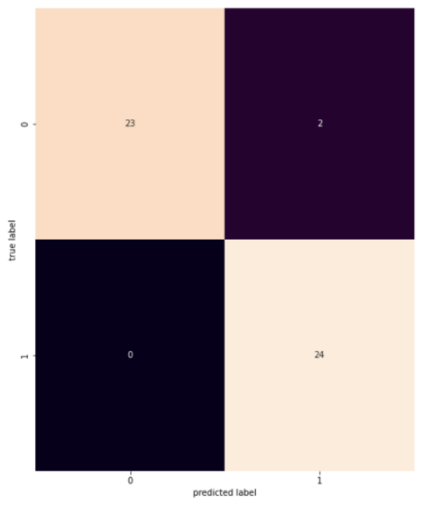

Figure 6.3. User-based split coarse features (user evaluation)

B. *Fine Features*

Instance-based split using Fine features using sklearn train_test_split with 0.25 for testing data results in F1_score:0.84. According to figure 6.4, we successfully detected 251 malicious instances out of 317 instances.

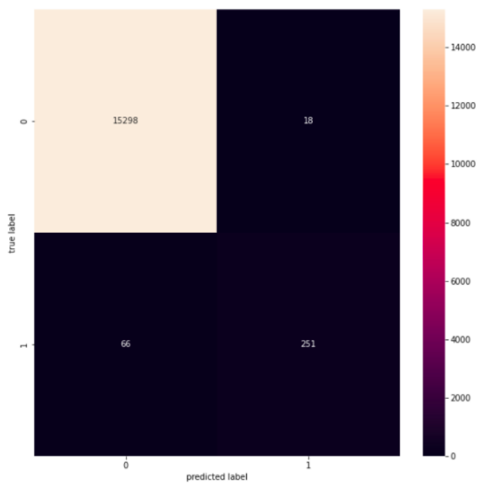

Figure 7.4. Instance-based split Fine features (instance evaluation).

User-based split using Fine features results in F1_score:0.83. According to Figure 6.5, we detect 74 malicious instances as benign and 43 benign instances as malicious. Figure 6.6 represents user evaluation confusion matrix that specify that successfully detect 23 malicious users out of 24 malicious users.

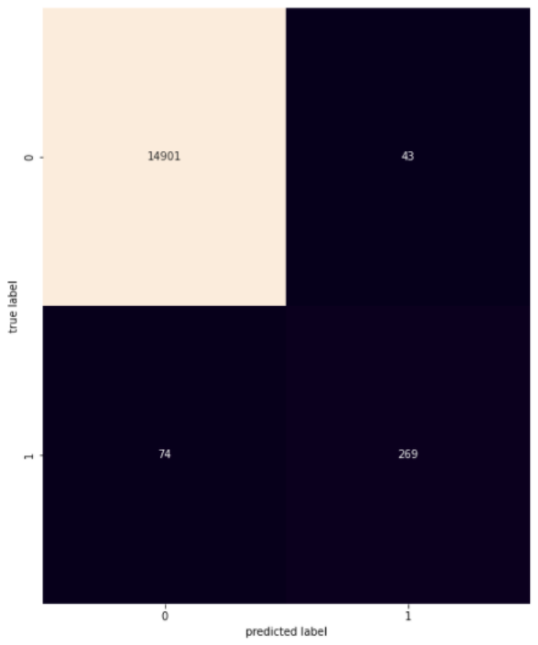

Figure 6.5. User-based split Fine features (Instance evaluation)

The summarization of the results is shown in Table 6.1. We notice that Fine Features over perform Coarse Features and Instance-based splitting over perform user-based splitting. But, we decided to use user-based splitting on Fine Features as it is more realistic because the company will train the model according to its current employees and after that the model will be used for different new employees. In addition, we may suffer from information leakage when using Instance-based splitting for longer attacks as some of the attack's malicious activities will appear in training data and others will be in testing data.

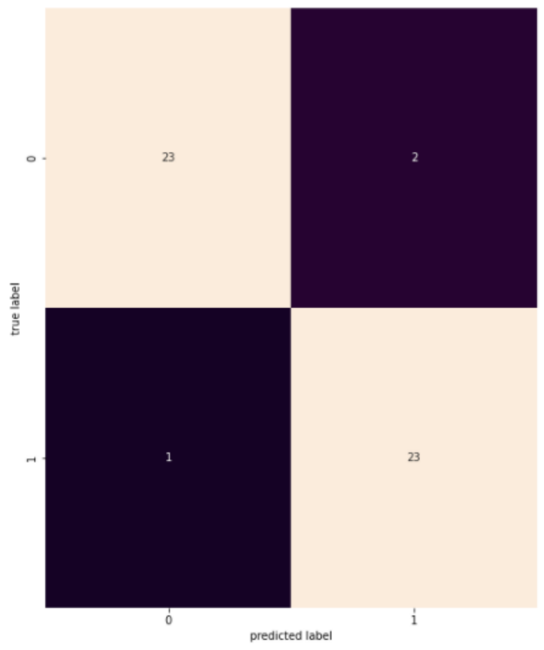

Figure 6.6. User-based split Fine features (user evaluation)

Splitting/Features		Coarse Features	Fine Features
Instance-based split	F1_score	0.74	0.84
	Instance-based Confusion matrix	FN:124 FP: 7	FN:66 FP: 18
User-based split	F1_score	0.71	**0.83**
	Instance-based Confusion matrix	FN:80 FP: 133	FN: 74 FP: 43
	User-based Confusion matrix	FN:0 FP: 2	FN: 1 FP: 2

Table 6.1. Summary of Results.

C. Feature Selection

We apply random forest on the user-based split using Fine Features and select features as discussed in the Methodology part. 30 features achieved the highest f1_Score: 0.79. The Top 30 features are illustrated in Table 6.2.

url_depth	count_Send_working_hour_other_pc_1_0	count_webvisit	file_depth_x	count_jobsearch_webvisit
count_Connect_working_hour_personal_pc	connection_time	url_length	file_depth_y	session_time
file_length	count_File Open_pdf_False_working_hour_other_pc	count_File Open_doc_False_working_hour_other_pc	no_words	size
http_content_length	no_attachments	attachments_size	no_destinations	no_external_dest
no_attachments_type	count_Receive_working_hour_personal_pc_0_0	count_Receive_working_hour_personal_pc_1_0	count_Send_working_hour_personal_pc_1_0	count_Send_working_hour_personal_pc_0_0
count_social_webvisit	count_hacktivist_webvisit	count_File Open_doc_True_working_hour_personal_pc	no_bcc_dest	count_File Write_doc_False_working_hour_personal_pc

Table 6.2. Top 30 Features.

D. Model Selection

Here we apply Random Forest, XGboost and Catboost on the top 30 features and tuning their parameters to select the champion model.

- **Random Forest:** We tuned Random Forest Model on the selected features and the best parameters were max_features='sqrt'.The results are f1_Score:0.79, precision: 0.97, recall: 0.67. According to figure 6.7, we successfully detected 181 malicious instances out of 269 malicious instances. According to figure 6.8, we don't have False Positives and we only have 3 False Negatives for user evaluation.

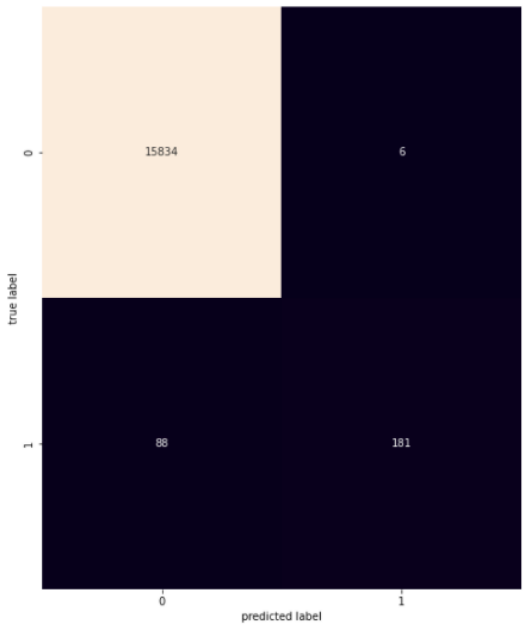

Figure 6.7. Random Forest (Instance evaluation)

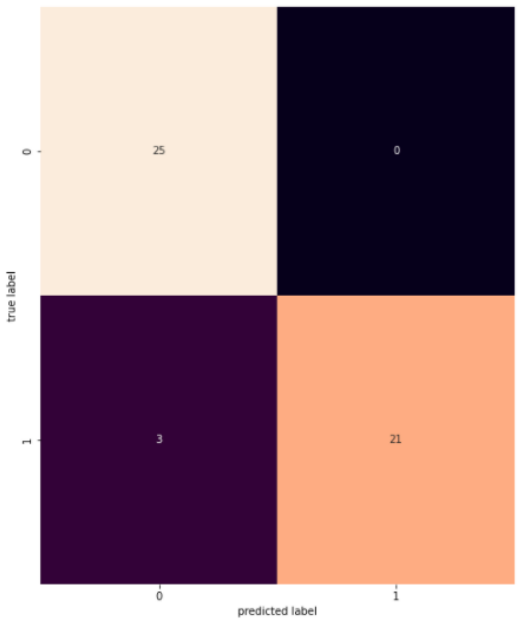

Figure 6.8. User-based split Fine features (user evaluation)

- **XGboost:** We tuned XGboost Model on the selected features and the best parameters were max_depth=7, n_estimators =112, subsample=0.8999999999999999.The results are f1_Score:0.86, precision: 0.96, recall: 0.77. According to figure 6.9, we predicted 61 malicious instances as benign and 8 benign instances as malicious. According to figure 6.10, we successfully detected 22 malicious users out of 24 but we also detected 1 benign user as malicious.

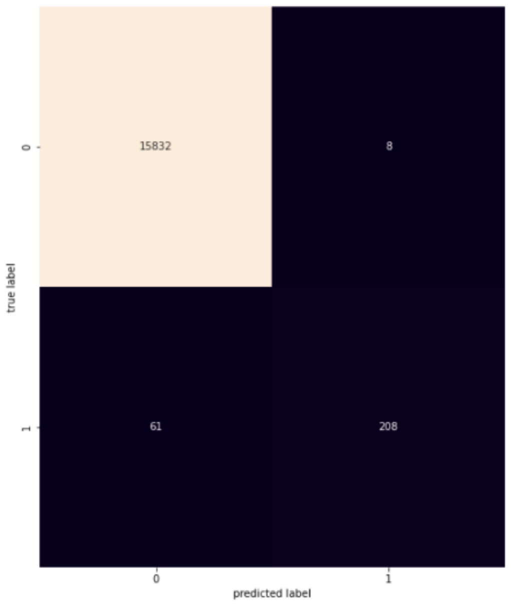

Figure 6.9. XGboost (Instance evaluation)

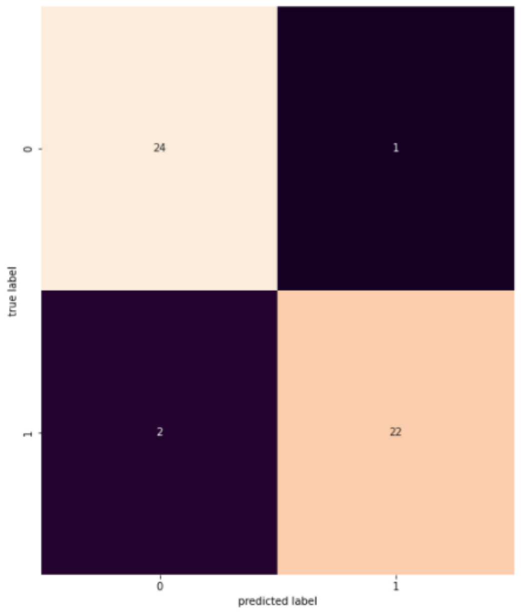

Figure 6.10. XGboost (user evaluation)

- **Catboost:** CatBoost achieves f1_Score:0.85, precision: 0.90, recall: 0.8. Figure 6.11 shows that the model detected 55 malicious instances as benign and 23 benign instances as malicious. Figure 6.12 represents the user evaluation and shows that the model failed to detect 1 malicious insider and succussedded to detect 23 malicious. But, it detects 5 benign users as malicious.

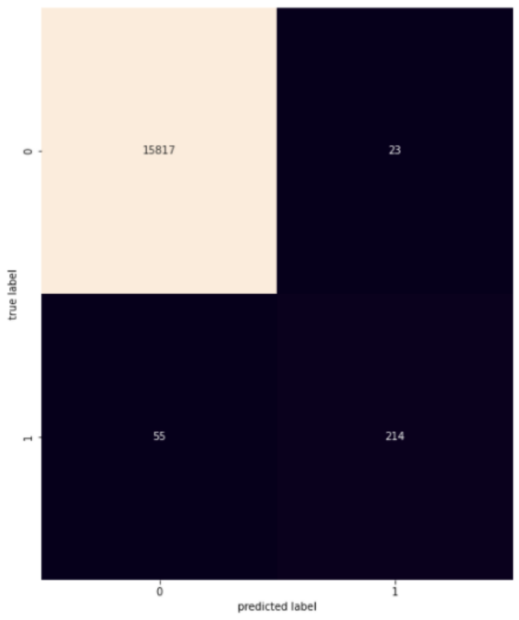

Figure 6.11. Catboost (Instance evaluation)

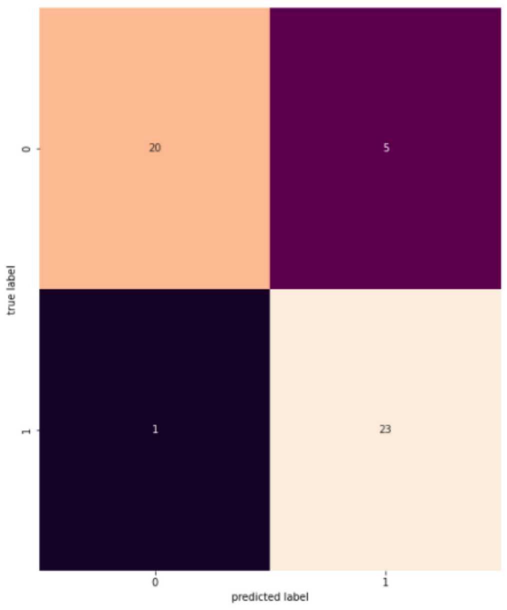

Figure 6.12. Catboost (user evaluation)

E. Chmpion Model

Table 6.3 represents the results of the models. It is obvious that XGBoost outperforms both Random forest and CatBoost in terms of f1 score by achieving 0.86. It has a slightly higher false positive rate than the random forest, which means a bit of more work for the threat analyst, but this is a price one has to pay for more true positives as well. So, XGboost is our Champion model.

According to XGBoost results, Instance Evaluation has only 8 false positives out of almost 16 thousand records, which is a massive decrease in the number of instances requiring investigation. It can be further tuned to decrease the number of false positives as well, but we can see on User Evaluation that almost all the malicious actors were eventually detected, except only two, so here what matters more is how late the detection was.

Models/metrics	f1-score	Instance Evaluation	User Evaluation
Random Forest	0.79	FN: 88 FP: 6	FN: 3 FP: 0
XGBoost	0.86	FN: 61 FP: 8	FN: 2 FP: 1
CatBoost	0.85	FN: 55 FP: 23	FN: 1 FP: 5

Table 6.3. Models Results

Let’s check our results in a little more detail. Figure 7.13 shows the average attack length in days, where attack 4 is typically longer and lasts around 70 days on average, while attack one is shorter and lasts only around 10 days on average. Attack 3 is not plotted because it has a constant value of 1 day always. Figure 7.14 represents the distribution of these attack scenario instances in the testing data, where the majority of the instances belong to attack 2, and most of them were correctly detected. Although the longest in duration and is expected to be more subtle, all instances of attack 4 were correctly classified. Attack 3 seems to be the most challenging since there aren’t a lot of instances in the data for it.

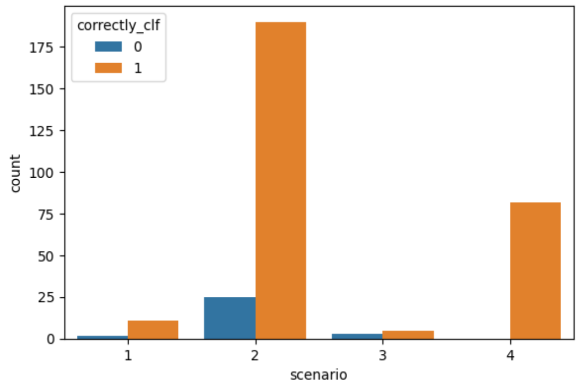

Figure 6.13. Average attack duration.

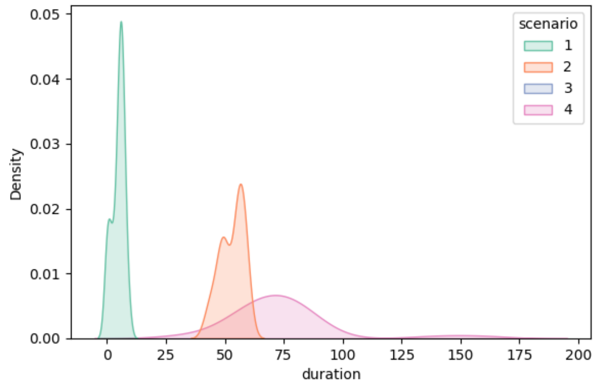

Figure 6.14. Number of detected vs. undetected days of each attack

Figure 6.15 represents average detection delay per scenario. We only use the scenarios that were eventually detected to compute this average, so if a scenario was completed and not detected at all, it would not contribute to this average. And we can see that on average, it takes 0.4 days to detect one attack, and 0 days to detect the rest, meaning that if they are detected, they are detected early on their first days. This supports our decision of using user-based evaluations, since we are able to detect the attacks early enough.

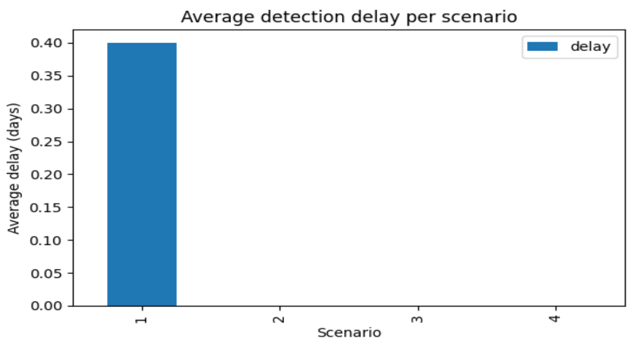

Figure 6.15. Average detection delay per scenario

VII. Conclusion

In this project, we aim to build an industry system to detect insider threat attackers. We faced some issues with the volume of the dataset. Therefore, it needs a higher computational power to process. We decided to down sample our data to decrease its size. In addition, there is an imbalance in the dataset since there is a small number of malicious events compared to the benign ones. Therefore, we sample all the malicious users, which are 99 users in our case, and then randomly sample another 100 benign users. After this stage, we solved two main problems in our case, the size of data and the imbalance problem. In the next stage we split our data based on users, where we sample 75% of the malicious and benign users to be in the training set, and the rest in the testing set. Now we used the random forest algorithm to select the most important features to use for the training stage. We tried different algorithms, and, in the end, the champion model was XGBoost. XGBoost achieved a high F1 score, and this is proportional in our case because precision and recall have the same importance in our business.

VIII. Future Work

One of the most important features is the changing behavior of the insider attacker over time. Since the attacker has access to the organization's resources which vary depending on role, they can carry out all motives over a long period of time. Therefore, to improve the proposed solution we could use a time series model to capture the temporal changes in the behavior of the insider attacker since the beginning of the attack. The LSTM model can be used to capture sequence features over time and build profiles for all users. Then we can use this profile with all behavior for the user to detect any abnormal behavior and store it in the user profile. Now we will have the user profile with the changing in behavior over time. Another approach is to use the profile of the users which is created by the LSTM model as input to our model to improve the model performance in the future.

IX. References

[1] Figures, "Insider threat statistics for 2022: Facts and figures," *Ekransystem.com*, 05-Apr-2022. [Online]. Available: https://www.ekransystem.com/en/blog/insider-threat-statistics-facts-and-figures.

[2] Lindauer, Brian (2020): Insider Threat Test Dataset. Carnegie Mellon University. Dataset. https://doi.org/10.1184/R1/12841247.v1

[3] Zhang, C., Wang, S., Zhan, D., Yu, T., Wang, T., & Yin, M. (2021). Detecting Insider Threat from Behavioral Logs Based on Ensemble and Self-Supervised Learning. Security and Communication Networks, 2021. Doi:https://doi.org/10.1155/2021/4148441

[4] Nasir, R., Afzal, M., Latif, R., & Iqbal, W. (2021). Behavioral Based Insider Threat Detection Using Deep Learning. IEEE Access, 9. doi:https://doi.org/10.1109/ACCESS.2021.3118297

[5] Liu, L., De Vel, O., Chen, C., Zhang, J., & Xiang, Y. (2018). Anomaly-Based Insider Threat Detection Using Deep Autoencoders. 2018 IEEE International Conference on Data Mining Workshops (ICDMW), (pp. 39-48). doi:10.1109/ICDMW.2018.00014.

[6] Jiang, J., Chen, J., Choo, K. R., Liu, K., Liu, C., Yu, M., & Mohapatra, P. (2018). Prediction and Detection of Malicious Insiders' Motivation Based on Sentiment Profile on Webpages and Emails. 2018 IEEE Military Communications Conference (MILCOM). doi:http://doi.org/10.1109/MILCOM.2018.8599790

[7] Ahmad, R., Zainal Abidin, Z., & Al-Mhiqani, N. (2021, January). An Integrated Imbalanced Learning and Deep Neural Network Model for Insider Threat Detection. International Journal of Advanced Computer Science and Applications, 12(1). Retrieved from https://www.researchgate.net/publication/348909288

[8] H. Eldardiry, E. Bart, J. Liu, J. Hanley, B. Price and O. Brdiczka, "Multi-Domain Information Fusion for Insider Threat Detection," 2013 IEEE Security and Privacy Workshops, 2013, pp. 45-51, doi: 10.1109/SPW.2013.14.

[9] D. C. Le, N. Zincir-Heywood and M. I. Heywood, "Analyzing Data Granularity Levels for Insider Threat Detection Using Machine Learning," in IEEE Transactions on Network and Service Management, vol. 17, no. 1, pp. 30-44, March 2020, doi: 10.1109/TNSM.2020.2967721.