Contrastive Learning for Insider Threat Detection

M. S. Vinay[1], Shuhan Yuan[2], and Xintao Wu[1(✉)]

[1] University of Arkansas, Fayetteville, AR 72701, USA
{vmadanbh,xintaowu}@uark.edu
[2] Utah State University, Logan, UT 84322, USA
Shuhan.Yuan@usu.edu

Abstract. Insider threat detection techniques typically employ supervised learning models for detecting malicious insiders by using insider activity audit data. In many situations, the number of detected malicious insiders is extremely limited. To address this issue, we present a contrastive learning-based insider threat detection framework, CLDet, and empirically evaluate its efficacy in detecting malicious sessions that contain malicious activities from insiders. We evaluate our framework along with state-of-the-art baselines on two unbalanced benchmark datasets. Our framework exhibits relatively superior performance on these unbalanced datasets in effectively detecting malicious sessions.

Keywords: Insider threat detection · Contrastive learning · Cyber-security

1 Introduction

Insider threat refers to the threat arising form the organizational insiders who can be employees, contractors or business partners etc. These insiders usually have an authorization to access organizational resources such as systems, data and network etc. A popular approach to detect malicious insiders is by analyzing the insider activities recorded in the audit data [14] and applying supervised learning models. Usually, the insider audit data is unbalanced because only a few malicious insiders are detected. Hence, applying supervised learning models on such unbalanced datasets can result in poor detection accuracy. To address this limitation, we present a framework, CLDet, to detect malicious sessions (containing malicious activities from insiders) by using contrastive learning.

Our CLDet framework has two components: self-supervised pre-training and supervised fine tuning. Specifically, the self-supervised pre-training component generates encodings for user activity sessions by utilizing contrastive learning whereas the supervised fine tuning component classifies a session as malicious or normal by using these encodings. Contrastive learning requires data augmentations for generating augmented versions of an original data point and ensures

A. Bhattacharya et al. (Eds.): DASFAA 2022, LNCS 13245, pp. 395–403, 2022.
https://doi.org/10.1007/978-3-031-00123-9_32

that these augmented versions have close proximity with each other when compared to the augmented versions of the other data points. Since each user activity session can be modelled as a sentence and each activity as a word of this sentence [14], we adapt sentence based data augmentations from the Natural Language Processing (NLP) domain [10] in our framework. We conduct an empirical evaluation study of our framework and evaluation results demonstrate noticeable performance improvement over state-of-the-art baselines.

2 Related Work

Insider Threat Detection. Traditional insider threat detection models employ handcrafted features extracted from user activity log data to detect insider threats. Yuan et al. [11] argued that utilizing hand crafted features for detecting insider threats can be tedious and time consuming and hence proposed to utilize deep learning model to automatically learn the features. Specifically they employed a LSTM model to extract encoded features from user activities and then detected malicious insiders through a Convolutional Neural Network (CNN). Similarly, Lin et al. [5] used unsupervised Deep Belief Network to extract features from user activity data and applied one-class Support Vector Machine (SVM) to detect the malicious insiders. Lu et al. [6] modeled the user activity log information as a sequence and extract user specific features through a trained LSTM model. Yuan et al. [13] combined a RNN with temporal point process to utilize both intra- and inter-session time information. The closely related work is [14] where the authors proposed a few-shot learning based framework to specifically addresses the data imbalance issue in insider threat detection. The developed framework applies the word-to-vector model for generating encoded features from user activity data and then uses a trained BERT language model to refine the encoded features. We refer readers to a survey [12] for other related works.

Contrastive Learning. Contrastive learning has been extensively studied in the literature for image and NLP domains. Jaiswal et al. [3] presented a comprehensive survey on contrastive learning techniques for both image and NLP domains. Marrakchi et al. [7] effectively utilized contrastive learning on unbalanced medical image datasets to detect skin diseases and diabetic retinopathy. The developed algorithm utilizes a supervised pre-training component, which is designed by employing a Residual Network, and generates image representations. These generated image representations are further fed as input to a fine tuning component which is designed by using a single linear layer. In our framework, we utilize some of data augmentation concepts presented in [10] and [9]. Wu et al. [10] presented a contrastive learning based framework for analyzing text similarity. Their framework employs sentence based augmentation techniques for self-supervised pre-training. Wang et al. [9] presented a new contrastive loss function for the image domain.

3 Framework

User activities are modeled through activity sessions. Specifically, each session consists of multiple user activities. Let S_k denote the k^{th} activity session of a user. Here, $S_k = \{e_{k_1}, e_{k_2}, \ldots, e_{k_T}\}$, where $e_{k_i} (1 \leq i \leq T)$ is the i^{th} user activity. Let $D = \{S_i, y_i\}_{i=1}^{m}$ denote the insider threat dataset where m denotes the number of sessions, y_i is the label of S_i. Here, $y_i = 1$ and $y_i = 0$ denote that S_i is malicious and normal session respectively. The two main components of our CLDet framework are self-supervised pre-training and supervised fine tuning. The pre-training component is responsible for generating session encodings and the fine tuning component, using these session encodings as input classifies a given input session as a malicious or normal session.

3.1 Self-supervised Pre-training Component

3.1.1 Encoder and Projection Head

Each activity in the session is represented through trained word-to-vector model. Let $\mathbf{x}_{k_i} \in \mathbb{R}^d$ denote the word-to-vector model representation of activity e_{k_i}, where d denotes the number of representation dimensions. Each activity of an input session is converted to its corresponding word-to-vector representation and it is fed as an input to a specially designed *Encoder*. We choose Recurrent Neural Network (RNN) to design our encoder. The encoder is responsible for generating the session encoding $\mathbf{x}_k \in \mathbb{R}^d$ of session S_k. Finally, a projection head will project $\mathbf{x}_k$ to a new space representation $\mathbf{z}_k \in \mathbb{R}^d$. The projection head is only used in the training of the self-supervised component. After this training, the projection head will be discarded and only the encoded session representation will be used as an input to the supervised fine tuning component.

The encoder consists of a RNN and a linear layer. The RNN consists of two hidden layers denoted as $H^{(1)}$ and $H^{(2)}$ respectively. The first hidden layer $H^{(1)}$ is represented as $\mathbf{h}_{k_t}^{(1)} = tanh(W_1^1 \mathbf{x}_{k_t} + \mathbf{b}_1^1 + W_2^1 \mathbf{h}_{k_{t-1}}^{(1)} + \mathbf{b}_2^1)$ where $1 \leq t \leq T$, W_1^1 and W_2^1 are $(d \times d)$ weight matrices, $\mathbf{b}_1^1 \in \mathbb{R}^d$ and $\mathbf{b}_2^1 \in \mathbb{R}^d$ are the bias vectors, and $\mathbf{h}_{k_t}^{(1)}$ denotes the encoded output of $H^{(1)}$ for the input $\mathbf{x}_{k_t}$. The second hidden layer $H^{(2)}$ is similarly represented as $\mathbf{h}_{k_t}^{(2)} = tanh(W_1^2 \mathbf{h}_{k_t}^{(1)} + \mathbf{b}_1^2 + W_2^2 \mathbf{h}_{k_{t-1}}^{(2)} + \mathbf{b}_2^2)$ where W_1^2 and W_2^2 are $(d \times d)$ weight matrices, $\mathbf{b}_1^2 \in \mathbb{R}^d$ and $\mathbf{b}_2^2 \in \mathbb{R}^d$ are the bias vectors, and $\mathbf{h}_{k_t}^{(2)}$ denotes the encoded output of $H^{(2)}$ for the input $\mathbf{h}_{k_t}^{(1)}$. Finally $\{\mathbf{h}_{k_i}^{(2)}\}_{i=1}^{T}$ is flattened to denote the output of RNN as $\mathbf{v}_k \in \mathbb{R}^{Td}$, which is then fed to the linear layer $L^{(1)}$ to obtain the session encoding $\mathbf{x}_k$. This linear layer is represented as $\mathbf{x}_k = A_1 \mathbf{v}_k + \mathbf{b}_1$ where A_1 is a $(d \times Td)$ weight matrix and $\mathbf{b}_1 \in \mathbb{R}^d$ is a bias vector. The projection head is denoted as $L^{(2)}$ and is represented as $\mathbf{z}_k = A_2 \mathbf{x}_k + \mathbf{b}_2$ where A_2 is a $(d \times d)$ weight matrix and $\mathbf{b}_2 \in \mathbb{R}^d$ is a bias vector.

3.1.2 Contrastive Loss

A contrastive learning loss function is used for a contrastive prediction task, i.e., predicting positive augmentation pairs. We adapt the *SimCLR* contrastive loss function [10] in our framework and augment each batch of sessions. Let $B_s = \{S_1, S_2, ..., S_N\}$ denote a batch of sessions. Each $S_k \in B_s$ is subjected to data augmentation and two augmented sessions denoted as S_k^1 and S_k^2 are generated. Let $B_s^a = \{S_1^1, S_1^2, S_2^1, S_2^2, ..., S_N^1, S_N^2\}$ denote a batch of augmented sessions. The augmented sessions (S_k^1, S_k^2) form a positive sample pair and all the remaining sessions in B_s^a are considered as the negative samples. Let $\mathbf{z}_k^1$ and $\mathbf{z}_k^2$ denote the projection head representations of the augmented sessions S_k^1 and S_k^2 respectively. The loss function for the positive pair $(\mathbf{z}_k^1, \mathbf{z}_k^2)$ is represented as

$$l(\mathbf{z}_k^1, \mathbf{z}_k^2) = -log \frac{exp(cos(\mathbf{z}_k^1, \mathbf{z}_k^2)/\alpha)}{exp(cos(\mathbf{z}_k^1, \mathbf{z}_k^2)/\alpha) + \sum_{i=1}^{N} \mathbf{1}_{[i \neq k]} \sum_{j=1}^{2} exp(cos(\mathbf{z}_k^1, \mathbf{z}_i^j)/\alpha)} \quad (1)$$

Here, $cos()$ denotes the cosine similarity function, $\mathbf{1}_{[i \neq k]}$ denotes an indicator variable, and α denotes the tunable temperature parameter. This pair loss function is not symmetric, because $l(\mathbf{z}_k^1, \mathbf{z}_k^2) \neq l(\mathbf{z}_k^2, \mathbf{z}_k^1)$. For the batch of augmented sessions B_s^a, we can easily see there are N positive pairs. The contrastive loss function for B_s^a, which is defined as the sum of all positive pairs' loss in the batch, is represented as $CL(B_s^a) = \sum_{i=1}^{N} l(\mathbf{z}_i^1, \mathbf{z}_i^2) + l(\mathbf{z}_i^2, \mathbf{z}_i^1)$.

For session S_k, we adapt three basic NLP based sentence augmentation techniques [10]: 1) Activity Replacement (Rpl), we generate the augmented session S_k^1 (S_k^2) by randomly replacing g_1 (g_2) number of activities with a set of token activities; 2) Activity Reordering (Rod), we generate the augmented session S_k^1 (S_k^2) by randomly selecting a sub-sequence with length g_1 (g_2) and shuffling all activities in the chosen sub-sequence while keeping all other activities unchanged; 3) Activity Deletion (Del), g_1 and g_2 number of activities in S_k are deleted to generate the augmented sessions S_k^1 and S_k^2 respectively. We will investigate the effectiveness of other complex data augmentation techniques in our future work.

3.2 Supervised Fine Tuning Component

The supervised fine tuning component has two layers denoted as $L^{(3)}$ and $L^{(4)}$. The first layer $L^{(3)}$ is represented as $\mathbf{m}_k = A_3\mathbf{x}_k + \mathbf{b}_3$. Here, A_3 is a $(d \times d)$ weight matrix, $\mathbf{b}_3 \in \mathbb{R}^d$ is a bias vector, and $\mathbf{m}_k \in \mathbb{R}^d$ denotes the output encoding of $L^{(3)}$. The output layer $L^{(4)}$ is represented as $\mathbf{o}_k = Softmax(A_4\mathbf{m}_k + \mathbf{b}_4)$. Here, A_4 is $(2 \times d)$ weight matrix, $\mathbf{b}_4 \in \mathbb{R}^2$ is a bias vector, and $\mathbf{o}_k$ denotes the output of the supervised fine-tuning component. We use the Softmax activation function in the output layer. The supervised fine tuning component is trained by using the cross entropy loss function.

4 Experiments

4.1 Experimental Setup

4.1.1 Datasets

The empirical evaluation study of our proposed framework is conducted on two datasets: CERT Insider Threat [2] and UMD-Wikipedia [4]. In CERT, each user activities are chronologically recorded over 516 days. To perform our empirical analysis, we split the dataset into training and test sets using chronological ordering. Specifically, the user activities recorded until the first 460 days and between 461 to 516 days are used in the training and test sets respectively. Additionally, we further split the training set for training the pre-training and fine tuning components, wherein, the user activities recorded until the first 400 days and between 401 to 460 days are used for training the pre-training and fine tuning components respectively. For the supervised fine-tuning component, four scenarios are utilized in the training phase. Each scenario involves different number of malicious sessions. Specifically, $5, 8, 10$ and 15 malicious sessions are utilized in the training phase. The UMD-Wikipedia dataset is relatively more balanced than CERT dataset. Since our framework is specifically designed to effectively operate on unbalanced datasets, we only use a limited number of malicious sessions for training the supervised fine tuning component. The training set is split between pre-training and fine tuning components, wherein, 4436 and 50 normal sessions are used for training the pre-training and fine tuning components respectively and similarly, 3577 and 50 malicious sessions are used for training the pre-training and fine tuning components respectively. Again, we use four scenarios in the training phase of the supervised fine-tuning component. Specifically, $5, 15, 30$ and 50 malicious sessions are utilized in the training phase. We show the detailed settings in Table 1.

Table 1. Training and test sets

Dataset	Partition		# of Malicious Sessions	# of Normal Sessions
CERT	Training set	Pre-Train	23	1,217,608
		Fine Tune	15	50
	Test set		10	1000
UMD-Wikipedia	Training set	Pre-Train	3577	4436
		Fine Tune	50	50
	Test set		1000	1000

4.1.2 Training Details

The activity features are extracted through a trained word-to-vector model. Specifically, the word-to-vector model is trained through the skip-gram approach with the minimum word frequency parameter as 1. The hyper-parameters g_1 and g_2 employed in our data augmentation techniques control the amount of distortion caused by augmenting the original session. For activity replacement and deletion-based data augmentation techniques we set $g_1 = 1$ and $g_2 = 1$, and for activity reordering based data augmentation technique we set $g_1 = 3$ and $g_2 = 3$. We set the number of dimensions of activity and session encodings d as 5 and the temperature parameter α in the contrastive loss function as 1. Four metrics are utilized to quantify the performance of our framework: Precision, Recall, F_1 and FPR.

4.1.3 Baselines

We compare our CLDet framework with three baselines: Few-Shot [14], Deep-Log [1] and BEM [8]. Few-Shot has a similar design as our framework, wherein, it has both self-supervised pre-training and supervised fine tuning components. Specifically, the self-supervised pre-training component is used for generating session encodings and these encodings are utilized to detect malicious sessions through the supervised fine tuning component. The session encodings are generated through the BERT language model. The self-supervised pre-training component is trained by using the Mask Language Modeling (MLM) loss function. We train both self-supervised pre-training and supervised fine tuning components by using the same settings shown in Table 1. BEM employs LSTM to model user activity sessions. Specifically, it considers the past user activities and predicts the probabilities of future activities through LSTM. If the predicted probability of an activity in the session is low, then that session is flagged as a malicious session. The LSTM model employs a single hidden layer and the model training is performed by using cross entropy loss. We train this baseline by using the same training set which we have used for training the fine-tuning component of our framework. Deep-Log differs from BEM in two ways: (1) It employs two hidden layers in its LSTM model. (2) It predicts the probabilities of the top-K future activities, if some activity in the session is not in the list of predicted top-K activities, then that session is flagged as a malicious session. Deep-Log training is performed by using cross entropy loss. We use the same training settings which was used for BEM to train this baseline.

Table 2. Performance of our framework and baselines under different scenarios. The higher the better for Precision, Recall, and F1. The lower the better for FPR. The cells with—indicate the extreme scenario where all sessions are predicted as normal. Best values are bold highlighted. M denotes the number of malicious sessions.

Models	Scenario	CERT					UMD-Wikipedia				
		M	Precision	Recall	F1	FPR	M	Precision	Recall	F1	FPR
Deep-Log	1	5	—	—	—	—	5	—	—	—	—
	2	8	—	—	—	—	15	—	—	—	—
	3	10	0.7600	0.8125	0.7294	0.4500	30	—	—	—	—
	4	15	**1.0000**	0.5875	0.7394	**0.0000**	50	0.6765	0.9200	0.7797	0.4400
BEM	1	5	—	—	—	—	5	—	—	—	—
	2	8	0.5000	0.3125	0.3846	**0.0000**	15	—	—	—	—
	3	10	0.6724	0.5165	0.4971	0.4500	30	0.5000	0.1500	0.2307	**0.0000**
	4	15	0.7500	0.8100	0.7179	0.5000	50	0.6282	**0.9800**	0.7656	0.5800
Few-Shot	1	5	—	—	—	—	5	—	—	—	—
	2	8	0.3666	0.1125	0.1709	0.1861	15	—	—	—	—
	3	10	0.5833	0.1875	0.2832	0.1361	30	0.4286	0.1200	0.1875	0.1600
	4	15	0.4000	0.4125	0.3709	0.5111	50	0.4894	0.9200	0.6389	0.9600
CLDet(Rpl)	1	5	**0.9444**	0.5875	0.6195	**0.0000**	5	0.6234	**0.9600**	0.7559	0.5800
	2	8	0.9158	0.9026	0.9070	0.0812	15	**0.8718**	0.6800	**0.7640**	**0.1000**
	3	10	0.9111	0.9210	0.9117	0.0812	30	**0.8750**	0.7000	0.7778	0.1000
	4	15	0.9236	0.9333	0.9243	0.0715	50	0.8039	0.8200	**0.8119**	0.2000
CLDet(Del)	1	5	—	—	—	—	5	**0.6935**	0.8600	0.7679	**0.3800**
	2	8	**0.9444**	0.6500	0.7013	**0.0000**	15	0.7551	**0.7400**	0.7475	0.2400
	3	10	**1.0000**	0.7250	0.7806	**0.0000**	30	0.7636	**0.8400**	0.8000	0.2600
	4	15	**1.0000**	0.9000	0.9150	**0.0000**	50	**0.8222**	0.7400	0.7789	**0.1600**
CLDet(Rod)	1	5	0.9206	**1.0000**	**0.9584**	0.0800	5	0.6667	**0.9600**	**0.7860**	0.4800
	2	8	**0.9444**	**1.0000**	**0.9706**	**0.0000**	15	0.7826	0.7200	0.7500	0.2000
	3	10	**1.0000**	**1.0000**	**1.0000**	**0.0000**	30	0.7778	**0.8400**	**0.8077**	0.2400
	4	15	**1.0000**	**1.0000**	**1.0000**	**0.0000**	50	0.8039	0.8200	**0.8119**	0.2000

4.2 Experimental Results

We consider the three versions of our CLDet framework based on the specific data augmentation technique employed for pre-training. The performance of the three versions of our framework and the baselines w.r.t four different scenarios is shown in Table 2. Our CLDet framework consistently shows better overall performance than the baselines in all the considered scenarios and datasets. The main reason for this performance is that the self-supervised pre-training component by utilizing contrastive learning generates favorable encoding for each session and by using these favorable encodings as inputs, the supervised fine-tuning component can effectively separate the malicious and normal sessions. We would point out that we purposely introduce the first scenario where the number of malicious sessions used in the training is only 5 for both CERT and UMD-Wikipedia datasets. Under this extreme setting, all baselines completely fail (all sessions in the test data are predicted as normal). On the contrary, our framework can still achieve reasonable performance except the version of using the activity deletion on CERT. There is a no clear winner among the three data augmentation techniques used in our framework when all the scenarios and

datasets are considered. However, all the three data augmentation techniques can be considered as quite effective in achieving the main goal of our framework.

Table 3. Ablation analysis results. M denotes the number of malicious sessions.

Dataset	Scenario	M	Precision	Recall	F1	FPR
CERT	1	5	—	—	—	—
	2	8	0.2531	0.5000	0.3361	0.5000
	3	10	0.4423	0.3026	0.3594	0.0384
	4	15	0.4706	1.0000	0.6400	1.0000
UMD-Wikipedia	1	5	—	—	—	—
	2	15	—	—	—	—
	3	30	0.9294	0.4210	0.5795	0.0320
	4	50	0.6487	0.9750	0.7791	0.5280

Ablation Analysis. We conduct one ablation study by removing the self supervised pre-training component from our framework and only utilizing the supervised fine-tuning component. The supervised fine-tuning component consists of only linear layers and cannot model sequence data. To resolve this limitation for our ablation study, we suitably format the input data and layer $L^{(3)}$ of the fine tuning component. Consider the word-to-vector representations of the activities belonging to the session $S_k = \{e_{k_1}, e_{k_2}, ..., e_{k_T}\}$ which are denoted as $\{\mathbf{x}_{k_1}, \mathbf{x}_{k_2}, ..., \mathbf{x}_{k_T}\}$. We flatten this sequence $\{\mathbf{x}_{k_t}\}_{t=1}^{T}$ into a vector and feed this flattened vector as an input to layer $L^{(3)}$ of the fine-tuning component. Table 3 shows the results of this ablation study. For both datasets, the supervised fine tuning component when used in isolation for detecting malicious sessions, underperforms against our framework in all the four scenarios. Clearly, this ablation study demonstrates that self-supervised pre-training component is crucial for our framework to achieve good performance.

5 Conclusion

We presented a contrastive learning-based framework to detect malicious insiders. Our framework is specifically designed to operate on unbalanced datasets. Our framework has self-supervised pre-training and supervised fine tuning components. The former is responsible for generating user session encodings. These session encodings are generated through the aid of contrastive learning and are then used by the supervised fine tuning component to detect malicious sessions. We presented an empirical study and results demonstrated our framework's better effectiveness than the baselines.

Acknowledgement. This work was supported in part by NSF grants 1564250, 1937010 and 2103829.

References

1. Du, M., Li, F., Zheng, G., Srikumar, V.: Deeplog: anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017. pp. 1285–1298. ACM (2017)
2. Glasser, J., Lindauer, B.: Bridging the gap: a pragmatic approach to generating insider threat data. In: 2013 IEEE Symposium on Security and Privacy Workshops, San Francisco, CA, USA, May 23–24, 2013. pp. 98–104. IEEE Computer Society (2013)
3. Jaiswal, A., Babu, A.R., Zadeh, M.Z., Banerjee, D., Makedon, F.: A survey on contrastive self-supervised learning. CoRR arXiv:2011.00362 (2020)
4. Kumar, S., Spezzano, F., Subrahmanian, V.: Vews: a wikipedia vandal early warning system. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, p. 607–616. KDD 2015 (2015)
5. Lin, L., Zhong, S., Jia, C., Chen, K.: Insider threat detection based on deep belief network feature representation. In: 2017 International Conference on Green Informatics (ICGI), pp. 54–59 (2017)
6. Lu, J., Wong, R.K.: Insider threat detection with long short-term memory. In: Proceedings of the Australasian Computer Science Week Multi-conference. New York, NY, USA (2019)
7. Marrakchi, Y., Makansi, O., Brox, T.: Fighting class imbalance with contrastive learning. In: de Bruijne, M., Cattin, P.C., Cotin, S., Padoy, N., Speidel, S., Zheng, Y., Essert, C. (eds.) MICCAI 2021. LNCS, vol. 12903, pp. 466–476. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-87199-4_44
8. Tuor, A., Baerwolf, R., Knowles, N., Hutchinson, B., Nichols, N., Jasper, R.: Recurrent neural network language models for open vocabulary event-level cyber anomaly detection. CoRR arXiv:1712.00557 (2017)
9. Wang, X., Qi, G.: Contrastive learning with stronger augmentations. CoRR arXiv:2104.07713 (2021)
10. Wu, Z., Wang, S., Gu, J., Khabsa, M., Sun, F., Ma, H.: CLEAR: contrastive learning for sentence representation. CoRR arXiv:2012.15466 (2020)
11. Yuan, F., Cao, Y., Shang, Y., Liu, Y., Tan, J., Fang, B.: Insider threat detection with deep neural network. In: Shi, Y., Fu, H., Tian, Y., Krzhizhanovskaya, V.V., Lees, M.H., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2018. LNCS, vol. 10860, pp. 43–54. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93698-7_4
12. Yuan, S., Wu, X.: Deep learning for insider threat detection: Review, challenges and opportunities. Comput. Secur. **104**, 102221 (2021). https://doi.org/10.1016/j.cose.2021.102221
13. Yuan, S., Zheng, P., Wu, X., Li, Q.: Insider threat detection via hierarchical neural temporal point processes. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 1343–1350 (2019)
14. Yuan, S., Zheng, P., Wu, X., Tong, H.: Few-shot insider threat detection. In: CIKM 2020: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19–23, 2020. pp. 2289–2292. ACM (2020)