A Novel Network Alert Classification Model based on Behavior Semantic

Zhanshi Li[a], Tong Li[a,*], Runzi Zhang[b], Di Wu[a], Zhen Yang[a]
[a] Beijing University of Technology
[b] NSFOCUS Technologies Group Co., Ltd., Beijing 100089, China
*litong@bjut.edu.cn

***Abstract*—With the ever-increasing complexity and scale of today's information systems, security personnel have to face massive alerts every day. Efficiently and precisely classifying such alerts in terms of different threat levels is essential for protecting systems from serious threats. However, existing works mainly focus on binary classification which are coarse-grained and cannot pragmatically reduce the number of alerts examined by security personnel. This challenge is further exacerbated when dealing with large numbers of alerts. Moreover, existing approaches only focus on textual and temporal features without considering behavior semantic information when training classifiers. In this paper, we propose a behavior semantic-based alert multi-classification model. Specifically, our proposal classifies alerts into three threat levels according to the real practice of our industrial collaborator. In the meantime, we extract attribute, temporal, and behavioral semantic features to help classifiers to better learn classification boundaries. To extract behavior semantic information, we embed the alert attributes by learning their contextual behavior semantics and use the embedding vector as behavioral semantic features. We conducted experiments using a real enterprise network alert dataset. The experimental results demonstrate that our approach outperforms baseline methods in terms of false positive rate and classification performance.**

***Index Terms*—Alert Classification, Semantic Feature, Machine Learning**

I. Introduction

In large enterprises, security personnel need to continuously monitor, analyze, and classify security alerts to defend against malicious attacks [1]. Security alerts are typically classified into multiple levels by security personnel. However, modern large-scale systems face a vast number of security alerts every day, desperately requiring an effective and efficient multi-classification approach. While there has been some work exploring the task of alert classification, it still has shortcomings.

Firstly, existing works [2], [3] mainly focus on binary classification which are coarse-grained. Coarse-grained alerts cannot pragmatically reduce the number of alerts examined by security personnel. Especially, when dealing with a huge number of alerts, existing approaches cannot effectively reduce the workload of security personnel. For example, Lin et al. [2] classify alerts into alerts triggered by attacks and by non-attacks, respectively. They believe that alerts triggered by attacks are all important high-threats. However, this is not the case in practice. Because some attacks (e.g., worm attacks) are low-threat and do not cause actual loss to the enterprise. The security personnel's real needs are identifying high-threats, which are usually not that many and feasibly to be manually reviewed. In conclusion, the alert classification method based on binary classification is not reasonable.

Secondly, existing approaches only focus on textual and temporal features without considering behavior semantic information when training classifiers. Specifically, these methods are simple in representing alert attributes without considering the behavior semantic information of the alert attributes. For example, Shittu et al. [4] represent IP addresses as binary strings to calculate the distance between alerts. The distance between IP addresses is measured by the length of the longest common prefix. The way the distance between IP addresses is calculated shows that the longer the common prefix, the closer the logical distance between different IP addresses. However, the longest common prefix length for two IP addresses only implies whether they are in closer subnets and does not indicate similar behavior.

To address the above shortcomings, we propose A behavioR seMantic-based alert multi-Classification mOdel (ARMCO). Specifically, ARMCO extracts a series of alert features from the attributes of historical alerts, including attribute, temporal, semantic features. Attribute features encode discrete attributes of alerts with a wide range of values into a low-dimensional vector space, which save space and facilitates subsequent classification model learning. Temporal features can characterize the temporal aspects of alerts for different threat levels. To extract behavioral semantic information, we embed the alert IP addresses and ports by learning their contextual behavior semantics, and use the embedding vector as semantic features. Therefore semantic features can characterize the contexts of IP addresses and ports at different threat levels alerts. In the meantime, the representation of the IP addresses and ports is kept consistent with their behavior characteristic. In addition, ARMCO classifies alerts into high-threat, low-threat, and non-threat alerts according to the real practice of our industrial collaborator. Fine-grained classification of alerts more conform to the real network situation, and more friendly to security personnel as well. The major contributions of this paper are the following:

- We classify alerts into non-threat, low-threat, and high-threat, which are more in accordance with the threat situation of real network alerts.
- We embed the alert attributes by learning contextual

DOI reference number: 10.18293/SEKE2022-116.

behavior semantics to consistent their representation and behavioral characteristics.
- We conducted a number of experiments on a real network dataset, the results of which demonstrate the effectiveness of ARMCO.

II. Related Work

Recently there is a lot of work dedicated to the alert post-processing, such as alert correlation [5]–[7], aggregation [8], clustering [9]. These works are interdependent, and the dividing line is blurred. However, they aim to reduce the number of alerts or detect the attack. Shittu et al. [4] extracted the association conditions from historical alerts by using posterior probabilities. They use these conditions to correlate alerts, called meta-alert. Finally, the different local outliers of meta-alert are mapped to different alert threat levels. Low-threat alerts can then be filtered. Hofmann et al. [10] proposed an online intrusion alert aggregation system, which uses a finite mixture distribution to measure the similarity between alerts. However, the effectiveness of this approach depends on the assumptions of the distributions. Siraj et al. [11] design domain-specific similarity calculation methods for different alert attributes. The similarity of two alerts is a weighted sum of the similarity of attributes. Then, the alerts are aggregated by similarity for attack detection. Ahmadinejad et al. [12] use some temporal windows to reduce the comparison of new alerts with the full history alerts. After that, they use a probability-based evaluation function to determine the threshold value of the alert association. Two alerts are associated if their temporal similarity is higher than the threshold value. The correlated alerts are treated as hyper-alerts, which can be used for attack detection. In summary, the above works aim to relieve the pressure on security personnel. However, many alerts still need to be examined, and security personnel cannot focus on examining high-threat alerts.

Identifying the threat level of alerts is a relevant research topic in intrusion detection systems (IDS) [13]. Shittu et al. [4] mines possible correlation conditions in the alerts history and set correlation thresholds for different correlation conditions. Two alerts can be correlated when they satisfy multiple correlation conditions simultaneously. Directly or indirectly related alerts form a correlation graph. High-threat correlation graphs are discovered by calculating the similarity among correlation graphs. Lin et al. [2] jointly model alert temporal dependencies and textual dependencies to discover actual attacks in alerts. Temporal dependence exploits Bayes' rule and prefix tree to extract attack patterns. Text dependencies measure the co-occurrence probability between attributes. They combine the two dependencies to sort the alerts in order of probability of being an actual attack. Chen et al. [14] encode and decode high-threat alerts, and the model computes a higher reconstruction error when non-high-threat alerts appear. More specifically, they identify high-threat alerts by two encoding and decoding layers. The above work can also not relieve the pressure of security personnel very well. They didn't make a finer division of non-high-threat alerts, which is unreasonable. In addition, they capture inadequate factors that influence the threat level of alerts.

III. A Behavior Semantic-based Alert Multi-classification Model

Fig. 1 illustrates the overall framework of the ARMCO. As shown in Fig. 1, alerts are input to the semantic, attribute and temporal feature extraction modules. After the three feature extraction modules are computed, their outputs are concatenated to generate the feature vectors. Then the alerts feature vector, including multi-dimension features, is used to train the classification model. Finally, the trained model is used for alert threat level classification. After threat level classification, alerts will be classified as high-threat, low-threat, and non-threat.

A. Extract Attribute Features

Most alert attributes are discrete variables such as source IP address (Sip), destination IP address (Dip), source port (Sport), destination port (Dport), the rule id utilized to trigger the alert (RuleId) and the attack category to which the traffic belongs (AttackType). The above alert attributes include important information about an attack, which is useful for classifying the alert threat level. To facilitate the learning of the classification model, these discrete variables need to be encoded as feature vectors. In this paper, we use hash coding to encode discrete attributes. The hash encoding approach is more suitable for cases where the variables have a large range of values. And it can map variables with any range of values to feature vectors of a small length. Let the set of values of a variable v be V. v including n different values, we use hash coding approach to map a ($a \in V$) to a feature vector $\vec{a}$ of length $\sqrt{n}$. Specifically, we use the signed 32-bit variant of MurmurHash3 as hash function. MurmurHash3 is an extensively tested and fast non-cryptographic hash function [15]. It has good distributivity and is suitable for machine learning. The RuleId and AttackType attributes of the alert can be encoded directly as described above. The remaining four attributes need to be coded in two separate groups, Sip, Dip and Sport, Dport, respectively. Because Sip and Dip belong to the IP address, they need to be encoded together, so do Sport and Dport. The following description is how to encode Sip and Dip, the same for Sport and Dport. Let the set of values of Sip be S, the set of values of Dip be D,and $V = A \cup D$, and $n = |A| + |D|$. Then for b ($b \in V$) and n, they also input above-mentioned hash function to get the feature vector.

B. Extract Temporal Features

In this section, the construction of temporal features is described. For simplicity, no-threat, low-threat and high-threat are represented by 0, 1 and 2 respectively.

Alert count. We define the number of alerts in a specific time window as the alert count (e.g., the total number of alerts in the 5 minutes before the current alert was triggered). Intuitively, security personnel should pay more attention to an alert with a sharp change in the number of alerts in a short period [16].

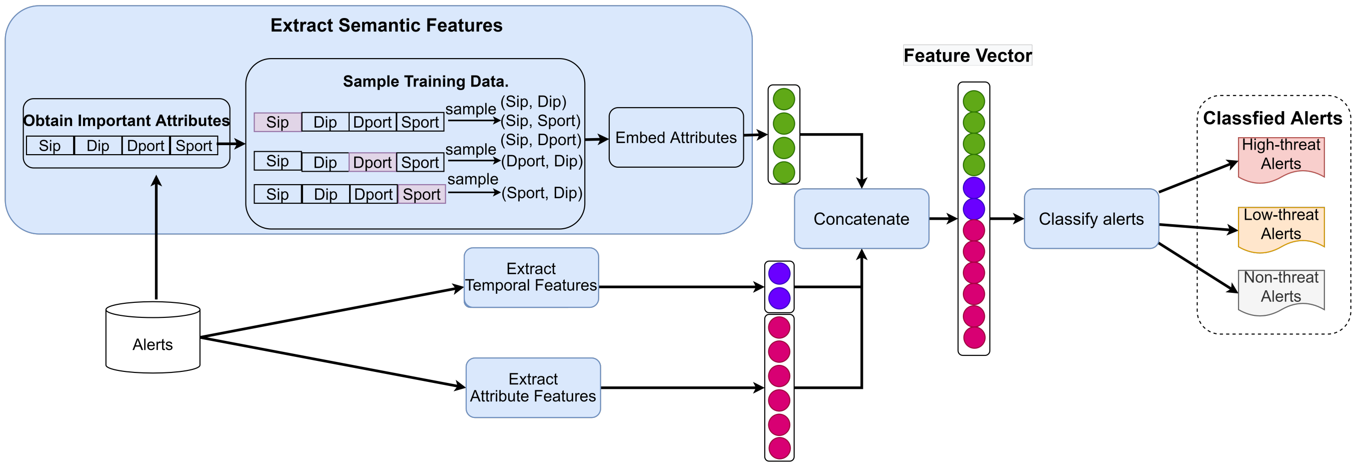

Fig. 1. The architecture of ARMCO.

Inter-arrival time. We define the time difference between the current alert and the previous alert as the inter-arrival time [16] (e.g., if alert A was triggered at 2021.03.22 00:00:00 and alert B was triggered at 2021.03.22 00:01:00, the inter-arrival time for alert B is 60 seconds). Intuitively, the first alert after a long period of no alerts should be focused on by security personnel [16].

C. Extract Semantic Features

For simplicity, we consider Sip, Dip, Sport and Dport as important attributes of the alert. Although the important attributes are also encoded by the attribute features, they are encoded for different purposes. The attribute features are designed to numerically encode discrete attributes.And semantic features aim to encode important attributes with contextual behavioral semantics. The contextual behavioral semantics of attributes refers to different IP addresses or ports that are similar if they appear in similar contexts. The approach to embedding important attributes of alerts follows the idea of IP2Vec [17] but is still somewhat different. While IP2Vec embeds IP addresses in network traffic, our approach embeds both IP addresses and ports in alerts. We extract the context of IP addresses and ports from historical alerts and construct training samples to input the embed attributes module. Fig. 1 shows the architecture of embedding the IP addresses and ports. Several key tasks are explained in detail next.

Selection of context. How to select behavioral semantics of important attributes is a core issue. As introduced in Section I, existing methods are simple in their representation of alert attributes and fail to capture the differences between their behaviors. Each alert describes the information related to specific anomalous traffic, as shown in Table I. Not all attributes are helpful. It is not difficult to find that the important attributes themselves are closely related because host communication requires this information. Therefore, we choose the important attributes themselves as their context.

Sampling of training data. The Sip, Dip, Sport and Dport of each alert are considered a ”sentence”, we use a subset of the ”sentence” to construct the input and output words. As shown in Fig 1, it shows how to sample training data. When using Sip as input words, Dip, Sport, and Dport are used as context words. When Dport is used as the input word, only Dip is selected as the context word. When using Sport as the input word, only Dip is selected as the context word. After the above sampling, input-output word pairs can be obtained, which are inputted into the module of embedding attributes.

Embed attributes. After receiving the input-output pairs, we need to convert them into knowledge graphs. The nodes in the knowledge graph are the set of all inputs and outputs. For each input-output pair, a directed edge from input to output is added to the knowledge graph. There are at most two edges with different directions between two nodes. In the module of embedding attributes, we adopt interactE([18]) to embed the attributes. InteractE is based on three key ideas – feature permutation, a novel feature reshaping, and circular convolution. The interactE optimizes the performance of embedding by increasing feature interaction.

D. Classify alerts

After feature extraction, each alert is represented by a set of features, and each alert also includes a threat level label that is examined and labeled manually. The XGBoost classification model is adopted for the alert threat level classification task. XGBoost is a gradient boosting tree-based model that is widely used by data analysts and has achieved good performance on many problems [19]. The algorithm creates and combines a large number of separate weak but complementary classifiers to produce a powerful estimator. This combination can be done in two ways: bagging (random forests) and boosting. Gradient boosting is established sequentially. In fact, a new weak learner is constructed to have a maximum correlation with the negative gradient of the loss function of the entire set in each iteration [20]. XGBoost belongs to a group of widely used tree learning algorithms [21]. Decision trees allow prediction of output variables based on a series of rules arranged in a tree structure. They consist of a series of segmentation points, or nodes, determined according to the values of the input features. The last node is a leaf that gives us the specific value of

TABLE I
AN EXAMPLE ALERT.

Attribute	Value
Timestamp	2021-03-22 00:00:00
AlertLevel	2
RuleId	18622209
AttackType	Directory traversal
Sip	*.*.*.*
Dip	*.*.*.*
Sport	48045
Dport	80
Payload	\x085q\x1024\x9cW\xadopu\x08...
q_body	POST /login/login.htm...
r_body	HTTP/1.1 200 OK \r\n Server...

the output variable. Tree learning algorithms do not require linear features or linear interactions between features. They are significantly better classifiers than other algorithms [22]. In addition, XGBoost, a gradient boosting algorithm, has two major improvements: accelerated tree construction and proposed a new distributed tree search algorithm.

IV. EXPERIMENT

In this section, we use real network dataset to evaluate ARMCO, aiming to answer the following research questions:

- RQ1: Can ARMCO effectively improve the alert threat level classification performance?
- RQ2: How much do different features affect classification performance?

A. Datasets

We first present information about the attributes of the alerts and the dataset used.

1) Alert Description: an alert, which has multi-dimensional alert attributes, is the smallest core data structure we study and analyze. Table I presents an example alert with several major attributes. The Timestamp attribute indicates the time when the alert was triggered. The AlertLevel attribute indicates the threat level determined by the traditional rule-based approach. The higher the value, the higher the threat level. The RuleId attribute indicates the rule ID utilized to trigger the alert. The AttackType attribute indicates the attack category to which the traffic belongs. The Sip, Dip, Port, and Dport attributes indicate the source IP address, destination IP address, source port, and destination port of the traffic, respectively. The payload attribute indicates the alert payload, including IP layer and lower layer data. The q_body attribute indicates the request body of the web access. The r_body attribute indicates the response body of the web request.

2) Dataset Description: the network alert dataset used in this paper is collected from a real security company. This dataset records a day's total of about nine million alerts. As Table I shows, while each alert log includes its threat level, that threat level is evaluated by a rule-based approach that has poor classification performance. For further analysis and to get better classification performance, the security personnel examine each alert and label it with the real corresponding threat level.

B. Metrics

As described in Section III, the trained classification model classify the test set alert. After all the alerts are classified, the precision (P) /recall (R) /F1-score (F1) /false positive rate (FPR) be calculated to evaluate the performance of each class. Each class can calculate P, R, F1 and FPR. Finally, we use the weighted P, R, F1 and FPR to evaluate the model performance. The weight of each class is equal to its proportion in the test set. Precision measures the percentage of identified threat levels that are the same as the actual threat level. Recall measures the percentage of alert threat levels that are correctly identified. F1-score is the harmonic mean of precision and recall. FPR measures the percentage of alert threat levels that are incorrectly identified. Therefore a better classification model has a higher P, R, F1 and a lower FPR.

C. Parameter Settings

We tune the hyperparameters of the ARMCO in the validation set (split from trainset). In the attribute features, the length of the feature vectors after hashing Sip, Sport, Dip, Dport, RuleId and AttackType are 24,124,24,124,5,5, respectively. In the temporal features, the window size of the alert count is set to 165 seconds. In interactE model, the attributes are embedded with a feature vector dimension of 32. For each positive sample, 50 negative samples are randomly sampled. Also, we set input dropout rate to 0.2, feature dropout rate to 0.5, hidden dropout rate to 0.5. It is trained via stochastic gradient descent over shuffled mini-batches with a batch size of 128. It uses an Adam optimizer with a learning rate is 0.0001. In the XGBoost classification model, we set lambda to 2, gamma to 0.1, max_depth to 6, subsample to 0.6, colsample_bytree to 0.9, min_child_weight to 3 and eta is 0.1.

D. Experimental Settings

To answer the research questions raised above, we designed two experiments. The following experiments uses the dataset introduced in Sec. IV-A. We use the top 80% of the alerts that have been sorted in time order as the training set and the remaining 20% as the test set.

1) Experiment 1: To demonstrate the performance of the ARMCO, we compare it with two baseline methods. The compared methods are listed below.

- Rule-based. The traditional rule-based approach classifies alerts into different threat levels (e.g., high-threat, low-threat, and non-threat). Experienced security personnel is required to keep the rule database updated regularly.
- Bug-KNN [23]. Bug-KNN calculates the similarity between bugs by textual similarity. It uses K-Nearest Neighbor to calculate the distribution of severity levels in historical bug reports most similar to the new bug report.

2) Experiment 2: The features ARMCO extracted include the attribute, temporal, semantic features. To demonstrate the effect of different features on the performance, we prepare three variants of ARMCO: $ARMCO_{TS}$, $ARMCO_{AT}$ and $ARMCO_{AS}$. They are differentiated below.

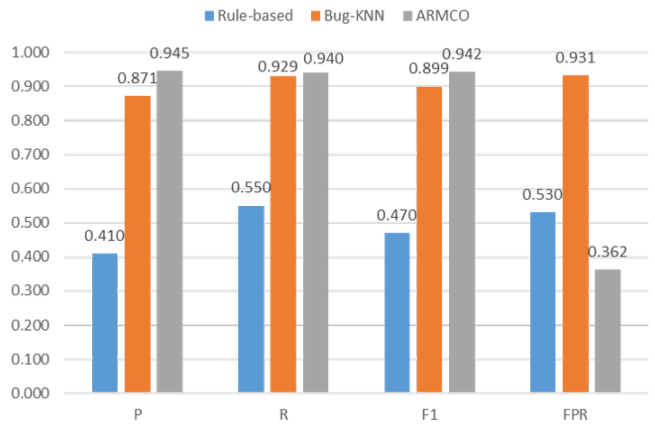

Fig. 2. Performance comparison of different methods.

TABLE II
EFFECT OF DIFFERENT FEATURE.

	P	R	F1	FPR
$ARMCO_{TS}$	0.944	0.939	0.941	0.366
$ARMCO_{AT}$	0.937	0.930	0.933	0.406
$ARMCO_{AS}$	0.892	0.666	0.750	0.497
ARMCO	**0.945**	**0.940**	**0.942**	**0.362**

- $ARMCO_{TS}$: This variant removes the attribute features.
- $ARMCO_{AT}$: This variant removes the semantic features.
- $ARMCO_{AS}$: This variant removes the temporal features.
- $ARMCO$: This is our complete model which involves all the proposed features.

E. Experimental Results

1) Experiment 1: Fig.2 shows the performance comparison of ARMCO and baseline methods in P, R, F1 and FPR. ARMCO outperformed other baselines, achieving an F1 of 0.942, higher than others, and an FPR of 0.362, lower than others. Compared with Rule-based, ARMCO's F1 is increased by 100%, and FPR is reduced by 31.6%. Compared with Bug-KNN, ARMCO's F1 increased by 4.7% and FPR reduced by 61.1%. As shown in Fig.2, the Rule-based approach has an F1 of 0.470, FPR of 0.530, which is weakly effective. Due to various factors influencing the threat level of alerts, using rules alone can't completely capture these factors. Besides, the Rule-based approach requires security personnel to spend plenty of time and effort to maintain the rules.

Bug-KNN measures the threat level by calculating the threat distribution of the historical alerts that are most similar to the current alert. As shown in Fig.2, Bug-KNN has an F1 of 0.899, FPR of 0.931. The results show that the Bug-KNN is weakly effective and has a very high time complexity. Due to various factors influencing the threat level of alerts, using textual information alone can't completely capture these factors. In summary, the experimental results demonstrate the effectiveness of ARMCO on classifying the alert threat level.

2) Experiment 2: Table II shows the performance of ARMCO with its three variants. From Table II, we can see that ARMCO can achieve the highest F1 of 0.942 and the lowest FPR of 0.362 when all features are used. The following is a detailed introduction to the impact of each ARMCO variant on performance.

- $ARMCO_{TS}$: When this variant was used, F1 drops from 0.942 to 0.941, and FPR increases from 0.362 to 0.366 compared to ARMCO. The phenomenon shows that attribute features have little effect on improving F1 and reducing FPR. Therefore, attribute features have weakly impact on improving classification performance.
- $ARMCO_{AT}$: When this variant was used, F1 drops from 0.942 to 0.93, and FPR increases from 0.362 to 0.406 compared to ARMCO. Compared with the attribute features, the semantic features have a greater effect on improving F1 and reducing FPR. Semantic features can be seen as a way to encode alerting attributes. Semantic features encode only four attributes. Due to the encoding process considering the semantic information of the attributes, the performance leads to significant improvement. This experimental result demonstrates the effectiveness of the semantic feature and reflects our second contribution.
- $ARMCO_{AS}$: When this variant was used, F1 drops from 0.942 to 0.75, and FPR increases from 0.362 to 0.49 compared to ARMCO. Compared with the previous three features, the temporal features have the biggest effect on improving F1 and reducing FPR. This indicates that the temporal features significantly affect classification performance.

In summary, the experimental results show that different features have different extent effects on the performance. And the temporal feature has the greatest effect, followed by semantic feature, and attribute feature.

F. Effects of Parameters

Our model has important parameters that need to be tuned. Here, we evaluate the impact of one parameter on performance, i.e., the embedding size in semantic features.

To investigate the effect of the embedding size in semantic features, we vary it in the set of 4, 8, 16, 32, 64, 128 and record performance results. From Fig. 3, we can see that the

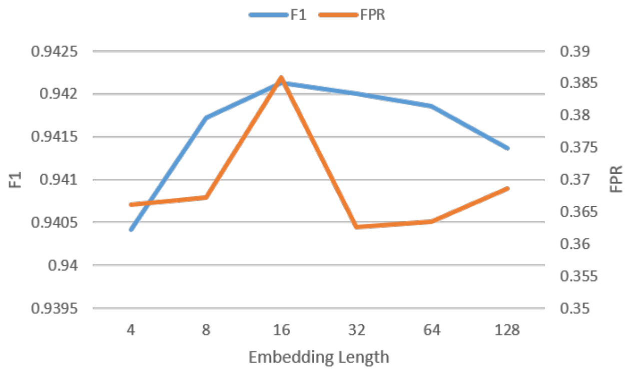

Fig. 3. Effects of embedding size.

F1 first improves with the increase in embedding size, and then starts to decrease at size 16. From Fig. 3, we can see that the FPR first improves with the increase in embedding size and then starts to decrease at size 16, finally starts to increase at size 32. As mentioned in Section IV-B, we expect larger F1 and smaller FPR. When the embedding size is smaller, the alert's important attributes may overlap in the low-dimensional representation space and cannot be represented accurately, so F1 and FPR are a bit worse. When the embedding size is larger since the behavioral semantics of the alert's important attributes are stationary, it leads to a deviation of the learning direction during the embedding process, so F1 and FPR are a bit worse. Although F1 achieves a maximum of 0.9422 at size 16, the corresponding FPR is high at 0.3859. FPR achieves a minimum of 0.362 at size 32, and although the corresponding F1 of 0.942 is 0.0002 lower than the maximum, it is ignorable. So the optimal performance can be obtained at the embedding size 32.

V. Conclusion and Future Work

In this paper, we propose a behavior semantic-based alert multi-classification model, which contains a set of powerful features to measure alerts of different threat levels. The results demonstrate the effectiveness of ARMCO and achieve an F1 of 0.942 and an FPR of 0.362. This dramatically reduces the pressure on security personnel to examine and minimize losses to the enterprise.

Alerts do not occur in isolation. Some attacks require relatively fixed attack steps to exploit the target, and different attack steps trigger different alerts. This can lead to the alert under different alert contexts having different threat levels, and how to incorporate this information into the alert threat assessment needs further study. Besides, we will also evaluate the proposed approach to real-time scenarios in future work.

Acknowledgement

This work is partially supported by the Major Research Plan of National Natural Science Foundation of China (92167102), the Beijing Nova Program (Z211100002121150), the Project of Beijing Municipal Education Commission (No.KM202110005025), and Engineering Research Center of Intelligent Perception and Autonomous Control, Ministry of Education.

References

[1] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[2] Y. Lin, Z. Chen, C. Cao, L.-A. Tang, K. Zhang, W. Cheng, and Z. Li, "Collaborative alert ranking for anomaly detection," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, ser. CIKM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1987–1995. [Online]. Available: https://doi.org/10.1145/3269206.3272013

[3] M. E. Aminanto, T. Ban, R. Isawa, T. Takahashi, and D. Inoue, "Threat alert prioritization using isolation forest and stacked auto encoder with day-forward-chaining analysis," *IEEE Access*, vol. 8, pp. 217 977–217 986, 2020.

[4] R. Shittu, A. Healing, R. Ghanea-Hercock, R. Bloomfield, and M. Rajarajan, "Intrusion alert prioritisation and attack detection using post-correlation analysis," *Computers & Security*, vol. 50, pp. 1–15, 2015.

[5] S. A. Mirheidari, S. Arshad, and R. Jalili, "Alert correlation algorithms: A survey and taxonomy," in *International Symposium on Cyberspace Safety and Security*. Springer, 2013, pp. 183–197.

[6] A. A. Ramaki, M. Khosravi-Farmad, and A. G. Bafghi, "Real time alert correlation and prediction using bayesian networks," in *2015 12th International Iranian Society of Cryptology Conference on Information Security and Cryptology (ISCISC)*. IEEE, 2015, pp. 98–103.

[7] S. Salah, G. Maciá-Fernández, and J. E. Díaz-Verdejo, "A model-based survey of alert correlation techniques," *Computer Networks*, vol. 57, no. 5, pp. 1289–1317, 2013.

[8] D. Man, W. Yang, W. Wang, and S. Xuan, "An alert aggregation algorithm based on iterative self-organization," *Procedia Engineering*, vol. 29, pp. 3033–3038, 2012.

[9] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 1630–1639.

[10] A. Hofmann and B. Sick, "Online intrusion alert aggregation with generative data stream modeling," *IEEE transactions on dependable and secure computing*, vol. 8, no. 2, pp. 282–294, 2009.

[11] A. Siraj and R. B. Vaughn, "Multi-level alert clustering for intrusion detection sensor data," in *NAFIPS 2005-2005 Annual Meeting of the North American Fuzzy Information Processing Society*. IEEE, 2005, pp. 748–753.

[12] S. H. Ahmadinejad and S. Jalili, "Alert correlation using correlation probability estimation and time windows," in *2009 International Conference on Computer Technology and Development*, vol. 2. IEEE, 2009, pp. 170–175.

[13] K. Alsubhi, E. Al-Shaer, and R. Boutaba, "Alert prioritization in intrusion detection systems," in *NOMS 2008-2008 IEEE Network Operations and Management Symposium*. IEEE, 2008, pp. 33–40.

[14] C. Chen, D. Zhang, P. S. Castro, N. Li, L. Sun, S. Li, and Z. Wang, "iboat: Isolation-based online anomalous trajectory detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 2, pp. 806–818, 2013.

[15] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 1113–1120.

[16] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei, "Automatically and adaptively identifying severe alerts for online service systems," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2420–2429.

[17] M. Ring, A. Dallmann, D. Landes, and A. Hotho, "Ip2vec: Learning similarities between ip addresses," in *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017, pp. 657–666.

[18] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. Talukdar, "Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 03, 2020, pp. 3009–3016.

[19] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[20] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[21] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, 2014, pp. 1–9.

[22] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 161–168.

[23] T. Zhang, J. Chen, G. Yang, B. Lee, and X. Luo, "Towards more accurate severity prediction and fixer recommendation of software bugs," *Journal of Systems and Software*, vol. 117, pp. 166–184, 2016.