Proceedings of the 2025 28th International Conference on Computer Supported Cooperative Work in Design

Autumn: An Unsupervised APT Detection via Detailed Process-Level Analysis

Zehui Wang[†‡], Yunxiang Wang[†‡], Wenhao Yan[†‡], Yinhao Qi[†‡], Tian Tian[*†‡], Bo Jiang[†‡], Zhigang Lu[†‡],
[†]Institute of Information Engineering,Chinese Academy of Sciences, Beijing, China
[‡]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{wangzehui, wangyunxiang, yanwenhao, qiyinhao, tiantian, jiangbo, luzhigang}@iie.ac.cn

***Abstract*—Advanced Persistent Threats (APTs) are exceptionally challenging to detect due to their high stealthiness. Audit logs, which provide detailed process-level information and record all activities before and after an APT attack, are crucial for detecting such threats. However, the sheer volume of data in audit logs also poses a significant challenge. Current methods suffer from the following issues: 1) difficulty in extracting information from the complex contextual relationships within audit logs, 2) reliance on prior knowledge for detecting APT attacks, and 3) coarse-grained detection signals.**

In this paper, we introduce Autumn, an APT detection method focused on processes as the primary research object. Autumn is an unsupervised learning model that does not rely on prior knowledge, making it more suitable for real-world APT detection scenarios. Autumn can swiftly identify critical information from vast audit logs and provides fine-grained detection signals by focusing on processes. We begin by constructing a graph from audit logs, segmenting it into subgraphs based on time, and applying strategies to reduce data volume. We then learn the characteristics of processes. By converting process information into input vectors using word2vec and calculating the reconstruction error for each subgraph through the encoding and decoding process of a transformer autoencoder, we train the model by associating the processes' IDF scores. Finally, we train the model on benign data and test it on a separate set of subgraphs containing attack events. Compared to other unsupervised learning methods, Autumn shows significant improvement in detection performance.

***Index Terms*—APT, attack detection, provenance graph**

I. Introduction

Advanced Persistent Threats (APTs) have become increasingly sophisticated in recent times, emerging as a predominant method of attack. These attacks are characterized by their extended duration and high level of stealth, posing significant challenges to detection. In an APT attack, threat actors systematically exploit vulnerabilities in system defenses with the ultimate goal of gaining control and maintaining prolonged access unnoticed.

Traditional defense mechanisms only protect the network perimeter and rely on rule-based features [1] [2]. For example, most Web Application Firewalls (WAF) and Intrusion Detection Systems (IDS) monitor network traffic and system logs of specific events. These mechanisms use rules based on the network, transport, and application layers to control and monitor traffic, prevent unauthorized access and attacks, and promptly identify and respond to security threats. However, such defense measures rely heavily on behaviors and traffic with prominent characteristics, making them easily bypassed when dealing with APT attacks that have high stealth characteristics.

Therefore, existing defense mechanisms and detection data are no longer sufficient to detect new types of attacks, necessitating the extraction of more information from networks and systems. To address the above issues, using new data sources to mine deep information has become the best option. Host audit logs record process-level operational information on endpoints and theoretically contain all operational information of the machine during that period. Current research [3] [4] has chosen host audit logs as the object of analysis and detection. They treat the system entities in host audit logs (such as processes, files, memory, etc.) as nodes and the events that associate these nodes as edges. These elements are collectively represented as a graph, referred to here as a Provenance Graph [5] [6], which includes the relationships between attack targets, thereby better uncovering potential attack behaviors.

The fundamental challenge in detecting network threats lies in inferring attacks from vast amounts of fine-grained audit logs. The logs contain a large number of benign events and only a very small number of malicious events .Detecting malicious events among them is like "finding a needle in a haystack," which greatly increases the difficulty of detecting malicious behavior. We notice that in host audit logs, process entities are the main actors of system call operations, associated with all entities (such as files, networks, memory, etc.). This means that by focusing on process entities, we can more quickly establish connections with other entities and clarify the system's call relationships.

In this paper, we propose Autumn, a threat analysis system that focus focuon processes and associates causal relationships. Autumn segments the provenance graph by time slices, dividing it into multiple subgraphs and using processes as the mainline to associate various entities. Autumn is an unsupervised method that detects anomalous subgraphs by learning the characteristics of normal subgraphs, without requiring prior knowledge or expert knowledge bases. Through this approach, we detect malicious behavior from subgraphs at different granularities. The results indicate that, under the premise of an acceptable false positive rate, we achieve high precision and recall rates, and can accurately identify attack subgraphs. We make the following contributions:

[*] corresponding author

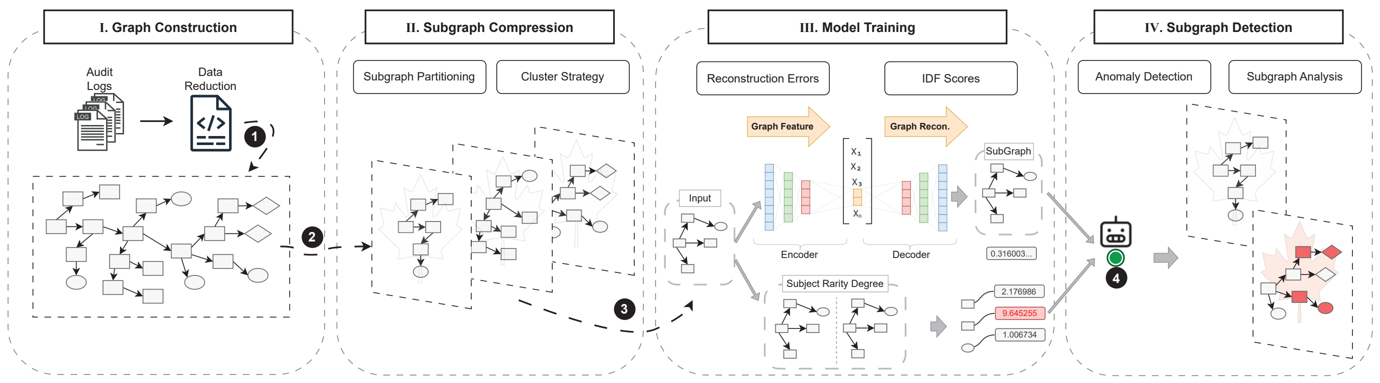

Fig. 1. Overview of Autumn

- We propose a new approach to graph construction. By creating a graph centered on processes and associating various entities, this method allows us to focus on the most critical elements, simplifying the complex provenance graph and clarifying the main relationships and associations.
- We design a threat detection system targeting APT attacks, capable of analyzing audit logs for threats without relying on prior knowledge or expert knowledge bases. By learning benign subgraphs and detecting anomalous subgraphs through reconstruction error, unknown risks can be effectively identified.
- We implement a fine-grained detection scheme that allows for detection at different granularities. This approach can greatly simplify the input size of the model and reduce its complexity. By using fine-grained detection methods to identify suspicious processes, threat analysis can be conducted more rapidly.
- We implement Autumn and conducted a systematic evaluation against real-world attacks. The results demonstrate that Autumn significantly outperforms other unsupervised learning methods. It can detect most attack subgraphs with a low false positive rate and false negative rate, and its detection covers multiple attack scenarios.

II. Background

Host audit logs. Host audit logs record various activities and events at the process-level within computer systems or networks, collected using the built-in logging capabilities of operating systems [7] [8] [9]. They provide detailed records of system operations, user behaviors, application activities, and more. Audit logs consist of entity information and event information, where entity information includes processes, files, network sockets, memory, etc., and event information records the interactions between these entities. Events are typically represented as sextuples <*src node*, *src type*, *dst node*, *dst type*, *edge type*, *time*>, reflecting the relationships between entities. Audit logs have become a crucial data source for tracking and analyzing system behavior, monitoring security, and ensuring compliance.

Causal analysis. Causal analysis has become an essential method for attack detection and investigation. It is based on analyzing dependencies within audit logs. Typically, researchers construct audit logs into a directed graph, known as a provenance graph (PG). A PG<E,V> consists of "node-edge-node" relationships established by multiple event information, where each connection can be represented as *e(u, v)*, with $u \in V$, $v \in V$, and $e \in E$. The direction of the edges in the graph indicates the direction of information flow. By constructing a PG, researchers can effectively trace information flows from audit logs. The PG includes relationships between edges at different times, essentially provides a snapshot of the system's historical behavior during that period. Researchers analyze these historical behavior snapshots to search for attack activities and conduct attack investigations.

III. Methodology

A. Approach Overview

The overall approach of Autumn is illustrated in Fig.1. It consists of four key components: graph construction, subgraph compression, model training, and subgraph detection. In the graph construction module, Autumn first parses the audit logs and constructs a graph ❶. Then, in the subgraph compression module, it segments the subgraphs ❷, compresses them using multiple strategies, and clusters the processes. In the model training module, the subgraphs ❸ are inputted, subgraph features and process features are extracted, and unsupervised methods are used to detect subgraphs containing attacks. Finally, in the subgraph detection module, the reconstruction error and idf scores of the subgraphs are comprehensively evaluated ❹, and the detection results are analyzed.

B. Graph Construction

TABLE I
Element of Graph Construction

Src	Dst	Connection Relationship
Process	Process	Clone
	File	Read, Open, Execute, Modify_file_attributes
	Netflow	Connect, Unlink, Send(Sendto, Sendmsg), Recv(Recvfrom, Recvmsg)

Autumn takes audit logs as input and converts them into a PG structure. The provenance graph contains rich semantics but also includes a significant amount of redundant data, which can add unnecessary complexity to the detection process. To address this, we first reduce the data to lower the complexity of detection. Table I presents the elements and relationships used in the graph construction in this study.

C. Subgraph Compression

Autumn employs two data reduction strategies and one clustering strategy to compress subgraphs.

Subgraph Partitioning. The provenance graph is made up of time-ordered events, so that subgraphs sliced by time can represent behaviour over this time period. We divide the logs into different-sized time windows of 5 minutes, 15 minutes, 30 minutes, and 120 minutes, to explore how different subgraph partitioning methods impact the detection results. During subgraph construction, we employ two data reduction strategies: **Strategy 1** involves deleting isolated nodes and isolated edges (subgraphs containing only up to two edges), and **Strategy 2** involves retaining only the first event when the source object, destination object, and the type of operation between them remain unchanged. Table II shows the compression effects of these two strategies at different time granularities.

TABLE II
REDUCTION EFFECT OF EACH STRATEGY OVER PREVIOUS

	5min	15min	30min	120min
strategy 1	9.49%	8.96%	8.40%	8.36%
strategy 2	65.82%	64.94%	64.48%	62.97%
strategy 3	64.39%	75.43%	79.33%	80.27%

Cluster Strategy. In audit logs, the UUIDs of processes and files change with state updates. Therefore, the same process often has multiple UUIDs in different states, posing a challenge for our correlation analysis. In each subgraph, we cluster each type of process, referred to as **Strategy 3**. If two processes have the same process name and the same UUID for their parent process, we classify them as the same type. The unchanged UUID of the parent process ensures that these operations are performed within a short time frame, confirming that these processes are executing the same operation. Table II demonstrates the compression efficiency of Strategy 3, significantly reducing redundant information in the graph.

D. Model

Reconstruction Errors. Reconstruction Errors refer to the differences between the reconstructed data and the original data when an autoencoder attempts to rebuild the input data. An autoencoder compresses the input data into a lower-dimensional representation using an encoder, and then restores this low-dimensional representation back to the high-dimensional original space using a decoder. Reconstruction errors can measure the effectiveness of the restoration process and, in the absence of labels, can be used to detect differences between the input data and the predicted results. Algorithm 1 illustrates the workflow of calculating reconstruction errors.

Algorithm 1 Reconstruction Errors

```
Require: Subgraph set $\mathcal{G}_V$
Ensure: Reconstruction Error R.E.
 1: for $\mathcal{G} \in \mathcal{G}_V$ do
 2:     for $process\ in\ \mathcal{G}$ do
 3:         Word2Vec(process)
 4:         return $SubgraphProcessEmbeddings$
 5:     end for
 6:     $DataGenerator(SubgraphProcessEmbeddings)$
 7:     $GraphFeature \leftarrow DataGenerator(...)$
 8: end for
 9: function $TransformerAutoencoder(GraphFeature)$
10:     $encoded = encoder(GraphFeature)$
11:     $decoded = decoder(encoded)$
12:     $error = MSE(GraphFeature, GraphRecon.)$
13:     return $error$
14: end function
```

Our encoder section employs a transformer [10] architecture, with the output layer consisting of a fully connected layer. The model takes all events in a subgraph as input. It first extracts all first-order neighbor nodes of a particular node. For each neighbor node, information is extracted in the format [node ID, event type, neighbor node ID, event timestamp, neighbor node attributes] and stored in a list. These lists form sequences that serve as input for that node's Word2Vec [11] model. The embeddings generated for multiple nodes from a subgraph are then used as input for the next step in a transformer, representing the graphical feature information of that subgraph. In the transformer, the input sequence initially undergoes self-attention computation via a multi-head attention mechanism. Subsequently, residual connections and layer normalization are applied to enhance the model's stability and learning capacity. The sequence is further processed through a feed-forward network with ReLU activation for additional feature extraction and transformation. Dropout layers throughout prevent overfitting, effectively encoding the complex dependencies within the sequence data. After the decoding phase, mean squared error (MSE) is utilized to compute the reconstruction error.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 \tag{1}$$

where *n* represents the number of input and reconstructed data points, x_i represents the *i*-th actual value in the input data, and $\hat{x}_i$ represents the *i*-th predicted value in the reconstructed data generated by the model.

IDF Scores. Inverse Document Frequency (IDF) [12] is a method used in information retrieval and text mining to measure the importance of a word within a corpus. IDF can reduce the significance of common words in document evaluation while increasing the weight of rare words. The formula for IDF is as follows:

$$\text{IDF}(t) = \log\left(\frac{N}{n_t}\right) \tag{2}$$

where *t* is the term, *N* is the total number of documents, and n_t is the number of documents containing the term *t*.

In APT attacks, attackers perform unusual operations that differ from commonly used processes. Therefore, the frequency with which a process appears in the system is a crucial indicator for detecting APT attacks. In this paper, we calculate the IDF values for all processes within each subgraph to identify rare processes, which serve as a crucial basis for subsequent analysis.

E. Subgraph Detection

Anomaly Detection. We use the aforementioned reconstruction error and IDF scores for anomaly detection. Given the variability in the number of processes per subgraph, normalization is necessary due to significant differences in average IDF calculations. We define the IDF anomaly score as amss (Arithmetic Mean Size Score).

$$amss_i = idf_{mean} * \frac{process\ num_i - min}{max + min} \tag{3}$$

where *i* represents the subgraph index, and *max* and *min* represent the maximum and minimum number of processes across all subgraphs, respectively.

Due to the significant differences in IDF scores among each process, we query the IDF of each process in each subgraph to obtain the average IDF score of all processes, denoted as $mean_{all}$. We use the following score as the overall rating for the subgraph, which is calculated as a weighted combination of two scores.

$$score_i = reconstruction\ errors_i + w * amss_i \tag{4}$$

where w is used to limit the weight of the $amss$ value. where $w = \pm min(\frac{max_{process}}{min_{process}}, 100)$.

We can judge the subgraphs based on the comprehensive results, and Algorithm 2 demonstrates our method for evaluating malicious subgraphs.

Subgraph Analysis. By thoroughly analyzing the detected malicious subgraphs through our detection results, we can observe that in different attack scenarios, each attack subgraph exhibits distinct process characteristics. These unique characteristics are indicative of the varied techniques and strategies employed by attackers. These variations in process behavior are crucial as they provide a deeper understanding of the attackers' methodologies and the specific actions they undertake during an attack.

This detailed analysis not only enhances our real-time detection capabilities but also provides valuable insights for developing more robust defense mechanisms in the future.

Algorithm 2 Detecting Malicious Subgraphs

```
Require: Subgraph set $\mathcal{G}_V$
Ensure: prediction value
 1: $idf\_list \leftarrow []$
 2: $mean_{all} = average(idf\_score)$
 3: for $\mathcal{G} \in \mathcal{G}_V$ do
 4:     for $process\ in\ \mathcal{G}$ do
 5:         if $any(process).idf\_score > mean_{all}$
 6:             return True
 7:         if $all(process).idf\_score < mean_{all}/2$
 8:             return False
 9:         elif $score_i > threshold$
10:             return True
11:         else
12:             return False
13:     end for
14: end for
```

IV. Evaluation

To evaluate the effectiveness of Autumn, We focus on the following research questions:

Q1. How does Autumn perform in detecting different scenarios and at different granularities? (§IV.*B*)

Q2. What is the detection coverage for attack events, and can it effectively distinguish normal events? (§IV.*B*)

Q3. Compared to other unsupervised learning methods, what are the advantages of Autumn? (§IV.*C*)

Q4. How much does each module of the experiment affect the Autumn model? (§IV.*D*)

A. Dataset

We evaluate Autumn on two datasets: the StreamSpot dataset and the DARPA TC dataset.

The StreamSpot dataset [13] contains information flow graphs from six scenarios, including five benign operations and one malicious operation. Table III summarizes this dataset.

TABLE III
Characteristics of the StreamSpot dataset

Experiment	Dataset	# of Graphs	$Avg.\mid E\mid$	$Avg.\mid V\mid$
StreamSpot	YouTube	100	113229	8292
	Gmail	100	37382	6827
	VGame	100	112958	8831
	Attack	100	28423	8637
	Download	100	310814	8900
	CNN	100	294903	8891

The DARPA TC dataset [14] was generated during the third red team vs. blue team exercise over a two-week period in April 2018 and collected from host networks. We select the dataset from the THEIA team for our study. This dataset contains data from Ubuntu Linux machines simulating an enterprise setup. The red team could carry out attacks using methods like Firefox backdoors, Nginx backdoors, and phishing emails. Table IV summarizes this dataset.

TABLE IV
CHARACTERISTICS OF THE DARPATC DATASET

Experiment	Duration	# of Graphs	$Avg. \mid E \mid$	$Avg. \mid V \mid$
DarpaTC THEIA	5min	1633	3498	40
	15min	548	10698	103
	30min	276	21790	192

TABLE V
EFFECTIVENESS ANALYSIS

Dataset	Duration	Conf. Matrix	Scen. A	Scen. B	Scen. C
Streamspot	/	[**499**,1] [0,**100**]	/	/	/
DarpaTC	5min	[**627**,12] [5 , **8**]	✓	✓	✓
	15min	[**203**,6] [2 , **7**]	✓	✓	✓
	30min	[**106**,5] [1 , **5**]	✓	✓	✓

B. Effectiveness Analysis

To evaluate the performance of Autumn in detecting APT attacks, we examine the detection effectiveness at different time granularities in this section. Table V shows Autumn's detection performance under various time granularities. Since the time granularity of StreamSpot is less than 5 minutes, we do not segment its graphs based on time.

Experimental Setup. We train our model using benign data and then predict other data based on the trained model. In the Streamspot dataset, we randomly select 10% of subgraphs from each of the five types of normal behavior as learning objects and use all subgraphs as detection objects. In the Darpa TC dataset, we divide the training subgraphs and prediction subgraphs based on time. Subgraphs before April 9th are used as learning objects, which are all benign subgraphs. Subgraphs from April 10th to 12th are used as detection objects, and which include three attack events.

Result Analysis. From Table V, Autumn is capable of distinguishing most normal subgraphs from malicious ones, with only a few false negatives. Further analysis of the detection results shows that Autumn's detection outcomes at different granularities cover the three attack scenarios. Autumn demonstrates good detection performance across different time granularities. It can detect malicious subgraphs under acceptable false-positive conditions.

C. Comparative Analysis

In this section, we compare the results of Autumn and other unsupervised methods on the StreamSpot and DarpaTC datasets.

Experimental Setup. We extract statistical features such as file size, number of events, number of processes, event categories, and quantities. We used K-means clustering, Spectral Clustering methods, and the Louvain algorithm for community detection to obtain the highest F1 scores. We conduct validation on two datasets. For the different data formats of the two datasets, we use appropriate other unsupervised methods for comparative analysis.

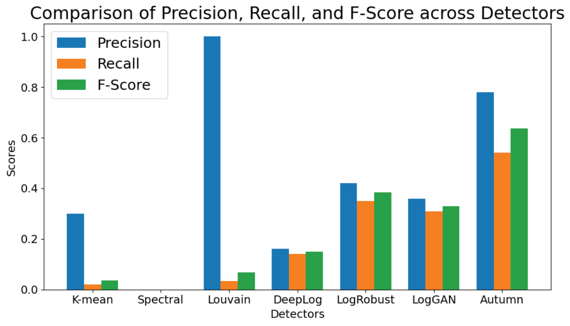

Fig. 2. Performance Comparison between Autumn and Other Methods

TABLE VI
COMPARATIVE ANALYSIS OF STREAMSPOT DATASET

Detector	Precision	Recall	Accuracy	F-Score	FPR
K-mean	0.2564	1.0000	0.5167	0.4032	0.5800
Spectral	0.3636	1.0000	0.7083	0.5385	0.3500
Louvain	1.0000	0.1500	0.8583	0.2609	0.0000
StreamSpot	0.7331	0.9120	0.9331	0.8147	0.0664
Unicorn	0.9470	0.9728	0.9864	0.9597	0.0109
Autumn	0.**9901**	**1.0000**	**0.9980**	**0.9950**	**0.0025**

TABLE VII
COMPARATIVE ANALYSIS OF DARPA DATASET

Detector	Precision	Recall	F-Score	FPR
K-mean	0.3000	0.0192	0.0364	0.0178
Spectral	0.0000	0.0000	-	0.0275
Louvain	**1.0000**	0.0345	0.0675	0.0000
DeepLog [15]	0.1600	0.1400	0.1499	0.0053
LogRobust [16]	0.4240	0.3500	0.3849	**0.0034**
LogGAN [17]	0.3560	0.3100	0.3300	0.0040
Autumn	0.7778	**0.5385**	**0.6364**	0.0098

Results Analysis. As shown in Table VI and Table VII, Autumn outperforms other unsupervised methods in key metrics such as precision, recall, f-score and false positive rate. Compared to StreamSpot and Unicorn, Autumn can uncover more fine-grained information. DeepLog, LogRobust, and LogGAN do not detect anomalies in graphs. Therefore, they struggle to identify the anomalous contextual information within a graph, while Autumn can effectively learns the contextual relationships within subgraphs. Fig. 2 visually demonstrates that Autumn's overall performance significantly exceeds that of other methods.

D. Ablation Study

In this section, we analyze the contribution of each component in Autumn to its overall performance.

Experimental Setup. R.E. stands for reconstruction error. Here, we use only the R.E. as the evaluation criterion, computing it with a transformer autoencoder. We adjust the threshold to find the result with the highest F1 score. Another important part is the IDF score. We calculate the IDF score for each subgraph based on the number of processes and category features of each subgraph, and use SVM for classification to obtain the result with the highest F1 score. We conduct validation on DARPA dataset.

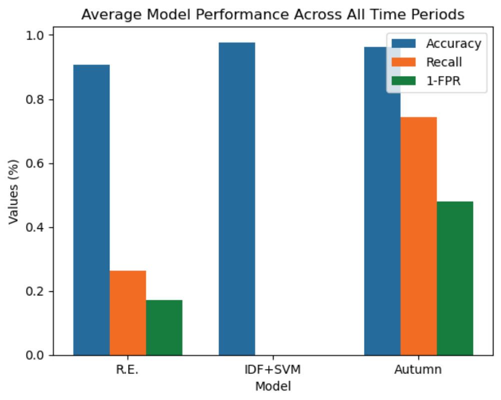

Fig. 3. Comparison of performance between Autumn and its modules

Results Analysis. Fig.3 visually demonstrates that the Autumn method outperforms each of the individual modules. Each module of Autumn performs with relatively high accuracy, but there is a significant deficiency in detecting malicious subgraphs, with a large number of false positives and false negatives. Using only IDF features tends to sacrifice the ability to finely distinguish malicious subgraphs in pursuit of overall performance, such as improving accuracy. None of the individual modules can effectively detect and distinguish malicious subgraphs.

E. Runtime Overhead

We verify the real-time capability of this method through experiments. The experiments in this subsection are done in an Ubuntu 20.04.6 LTS machine with 20 vCPUs and 96GiB of memory. Using the Darpa dataset as the experimental data, each subgraph takes 52.5 ms on average to go from input graph information to calculating the reconstruction error and the average IDF score. Each subgraph takes 0.05 ms on average to make a comprehensive decision based on the subgraph's reconstruction error and the IDF scores of the processes it contains. We repeat the experiments and report the average results.

V. Conclusion

In this paper, we introduce Autumn, an unsupervised learning-based APT attack detection system. Autumn filters out useless information through policy reduction, divides log events by time windows and constructs graphs. Autumn focuses on processes in audit logs as the primary research object, obtains embedded representations of processes by word2vec, uses them as inputs for the transformer autoencoder to compute the reconstruction error for each subgraph. Combining this with the IDF score features of each process, Autumn evaluates the detection results of the subgraphs. We assess Autumn on two real-world APT attack datasets, and the results indicate that Autumn can distinguish between benign and malicious subgraphs with a high recall rate and a low false positive rate.

Acknowledgment

This research is supported by National Key Research and Development Program of China (No.2023YFC2206402), Youth Innovation Promotion Association CAS (No.2021156). This work is also supported by the Program of Key Laboratory of Network Assessment Technology, the Chinese Academy of Sciences, Program of Beijing Key Laboratory of Network Security and Protection Technology.

References

[1] K. Mukherjee, J. Wiedemeier, T. Wang, J. Wei, F. Chen, M. Kim, M. Kantarcioglu, and K. Jee, "Evading Provenance-Based ML Detectors with Adversarial System Actions."
[2] "APT Attacks: Exploring Advanced Persistent Threats - ThreatDown by Malwarebytes." [Online]. Available: https://www.threatdown.com/blog/apt-attacks-exploring-advanced-persistent-threats-and-their-evasive-techniques/
[3] N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. N. Venkatakrishnan, "SLEUTH: Real-time Attack Scenario Reconstruction from COTS Audit Data."
[4] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-Time APT Detection through Correlation of Suspicious Information Flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, May 2019, pp. 1137–1152, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/8835390
[5] X. Shu, F. Araujo, D. L. Schales, M. P. Stoecklin, J. Jang, H. Huang, and J. R. Rao, "Threat Intelligence Computing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 1883–1898. [Online]. Available: https://dl.acm.org/doi/10.1145/3243734.3243829
[6] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "POIROT: Aligning Attack Behavior with Kernel Audit Records for Cyber Threat Hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 1795–1812. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3363217
[7] "linux-audit/audit-kernel: GitHub mirror of the Linux Kernel's audit repository." [Online]. Available: https://github.com/linux-audit/audit-kernel
[8] "Security Issues in Network Event Logging (syslog)." [Online]. Available: https://datatracker.ietf.org/wg/syslog/charter/
[9] markdefalco, "Event Viewer," Jan. 2019. [Online]. Available: https://learn.microsoft.com/en-us/shows/inside/event-viewer
[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," Aug. 2023, arXiv:1706.03762 [cs]. [Online]. Available: http://arxiv.org/abs/1706.03762
[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," Sep. 2013, arXiv:1301.3781 [cs]. [Online]. Available: http://arxiv.org/abs/1301.3781
[12] G. Salton and C. Buckley, "Term Weighting Approaches in Automatic Text Retrieval," Cornell University, USA, Technical Report, Oct. 1987.
[13] "StreamSpot: Detecting network anomalies in edge streams." [Online]. Available: https://sbustreamspot.github.io/
[14] "Transparent-Computing/README-E3.md at master · darpa-i2o/Transparent-Computing." [Online]. Available: https://github.com/darpa-i2o/Transparent-Computing
[15] "DeepLog | Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security." [Online]. Available: https://dl.acm.org/doi/10.1145/3133956.3134015
[16] "Robust log-based anomaly detection on unstable log data | Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering." [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3338906.3338931
[17] "LogGAN: A Sequence-Based Generative Adversarial Network for Anomaly Detection Based on System Logs | SpringerLink." [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-34637-9_5