Replication: Network-based Lateral Movement Detection Methods Using Machine Learning

Vladimír Bouček, Martin Husák
Faculty of Informatics, Masaryk University, Brno, Czech Republic
492927@mail.muni.cz, husak@fi.muni.cz

***Abstract*—Pivoting is a technique commonly employed by advanced adversaries to perform lateral movement within a network. In this process, an attacker leverages an intermediary host to relay commands to otherwise inaccessible systems. In this work, we survey the current state-of-the-art lateral movement detection techniques and identify approaches best suited for detecting pivoting behavior. Specifically, we focus on methods analyzing network traffic, not system logs, since we are looking for network-wide solution. We present the results of a replicability study, in which we find that only a few proposed approaches also publish a usable implementation, but their results are promising.**

***Keywords*—Lateral movement, pivoting, link prediction**

I. Introduction

Advanced Persistent Threats (APTs) represent some of the most severe cyber threats, characterized by their persistence, sophistication, and potential to remain undetected within compromised systems for extended periods [1]. Often backed by state-sponsored entities, APTs use a combination of advanced tools and techniques to achieve their objectives, whether data exfiltration or critical disruptions. A key strategy APTs employ is lateral movement, which involves spreading through a network to expand control and access. One particularly challenging technique within lateral movement is pivoting, where attackers compromise a public-facing host and leverage it to reach internal systems. Due to the covert nature of pivoting and the frequent use of legitimate credentials or encrypted protocols, detecting such behavior remains a significant challenge, particularly in dynamic and heterogeneous environments.

The detection of pivoting and lateral movement in general was a subject of research in the past years with some promising results. The most effective are the pivoting detection methods based on authentication log analysis [2]. Such approaches, however, require the agents installed on the pivots (devices abused for pivoting) or system logs to be centrally collected and can be difficult to deploy in large infrastructures. The network-based methods that would detect such activities network-wide were also proposed [3]. Such methods work fine in private networks with not much background traffic. However, in Internet-facing settings, the network-based pivoting detection methods suffer from enormous amounts of false positives and plethora of benign activities appearing as pivoting [4]. In such settings, it is challenging to (1) detect pivoting and pivoting-like activities without too much false positives and (2) distinguish benign and suspicious (or potentially malicious) activities.

In this work, we survey current lateral movement detection methods, focusing on network-based pivoting detection and evaluating their applicability to real-world network traffic. We are especially interested in the availability of the implementations and replicability of the proposed solutions.

The remainder of this work is structured as follows. Section II present the existing methods. Section III describes the datasets used in the replicability study. Section IV presents the results of the experimental evaluation. Section V concludes the paper.

II. Pivoting detection methods

When surveying the related work, we identified only three methods of pivoting detection that use machine learning and have a publicly available implementation. Even though there are more research papers on the topic of lateral movement detection [2], we had to criteria to include the related works in the replicability study. First, the methods should use only network traffic data, such as NetFlows or PCAPs, which excluded numerous methods using system or authentication logs, which would be out of scope of this work. Second, the methods should have a publicly available implementation, which excluded two methods, the pioneering work by Apruzzese et al. [3] and advanced method called APIVADS [5]. Nevertheless, both methods are pattern-based and do not use machine learning.

To provide a structured comparison of the key attributes of the reviewed methods, we summarized them in Table I. Our assessment covers their overall detection performance, reproducibility, and practical applicability, intending to identify suitable candidates for use in our work.

A. PIKACHU

The first approach for detecting lateral movement we will examine, introduced in 2022 by Paudel and Huang, is PIKACHU [6]. Similarly to methods like node2vec [9], PIKACHU is based on performing graph walks to learn graph structure. However, unlike previous work in this field (e.g., [10]), which used static graphs, PIKACHU employs dynamic graphs. Specifically, PIKACHU uses a discrete-time dynamic graph $G = \{G_1, G_2, \ldots, G_T, \ldots\}$, where $G_1, G_2, \ldots, G_T$, are static directed graphs. PIKACHU employs temporal walks to capture the graph's topological structure in these snapshots. A key difference from node2vec, which performs random biased walks, is that temporal walk preserves the chronological order of events according to their timestamps. Once the temporal walks are constructed, PIKACHU utilizes a Skip-Gram model to learn node embeddings that maintain both topological and temporal properties. This enables the model

978-3-903176-75-1 ©2025 IFIP

TABLE I
COMPARISON OF DETECTION METHODS AND REPRODUCIBILITY OF THEIR EVALUATION.

Method	Type	Learning	Data	Dist. Processing	Datasets	Implementation
PIKACHU [6]	ML	Unsupervised	Temporal graph (discrete)	✗	LANL, DARPA OpTC	https://github.com/rpaudel42/Pikachu
Euler [7]	ML	Unsupervised	Temporal graph (discrete)	✓	LANL, DARPA OpTC	https://github.com/iHeartGraph/Euler
Jbeil [8]	ML	Self-supervised	Temporal graph (continuous)	✗	LANL, Pivoting	https://github.com/surender08/Jbeil

to capture dynamic changes on a short-term scale effectively. Formally, the Skip-Gram model in PIKACHU aims to maximize the probability of observing a temporal walk:

$$\max_{f} \sum_{v_j \in V_T} \log P(W_T | f(v_j)) \tag{1}$$

where W_T represents a temporal walk at time T for a node v conditioned by its node embedding given by the learned function f. The optimization ensures that nodes frequently appearing in similar temporal contexts have embeddings that maximize their co-occurrence probability.

To model long-term dependencies, PIKACHU utilizes a GRU-based autoencoder. This component refines the learned embeddings by capturing patterns across different snapshots in time. The training is conducted in a fully unsupervised manner, requiring no labeled attack data. Anomalies are then detected by estimating the conditional probability of edges, flagging unexpected interactions as potential lateral movement.

B. Euler

Euler, introduced in 2023 by King and Huang, aims to detect network lateral movement using temporal graph-based link prediction [7]. It addresses key limitations of previous approaches, which either failed to capture the dynamic nature of network interactions by relying on static graphs or lacked the expressiveness of modern graph neural network architectures. As the authors suggest, and to the best of our knowledge as well, Euler was the first lateral movement detection approach incorporating graph neural networks with temporal link prediction.

Rather than a single model, Euler is a framework integrating two model-agnostic components: a graph neural network and a sequence encoder, typically a recurrent neural network (RNN). For this purpose, it uses interactions $\mathcal{I}$ between nodes in the network in the form of tuples (src, dst, ts). These interactions are aggregated into discrete time windows of size δ to construct a discrete temporal graph $G = \{G_0, \ldots, G_T\}$. Each of these snapshots is passed to the GNN to capture topological relationships between nodes in the network at time t. The RNN component then processes these encodings to capture the evolving nature of the network between snapshots.

This can be formally defined within the encoder-decoder framework, where the encoder $f(\cdot)$ can be described as:

$$\begin{aligned} Z &= f(\{G_0, G_1, \ldots, G_T\}) \\ &= RNN([GNN(X_0, A_0), \ldots, GNN(X_T, A_T)]) \end{aligned} \tag{2}$$

where X_t represents the node feature matrix at time t, A_t is the adjacency matrix, encoding edges between nodes at t, and Z is the final snapshot embedding, capturing both topological and temporal properties of the network.

The decoder function $g(\cdot)$ reconstructs or predicts edges by estimating the probability of a node connection. Anomalous edges are then those with a low probability of occurring. This is achieved using an inner-product decoder, where the dot product of the latent vectors $Z_t[u]$ and $Z_t[v]$ represents the log-odds that an edge (u, v) exists at time $t + n$:

$$g(Z_t) = \sigma(Z_t Z_t^T) = \tilde{A}_{t+n} \tag{3}$$

where σ represents the logistic sigmoid function, and $\tilde{A}_{t+n}$ is the decoded adjacency matrix at time $t + n$.

This encoder approach enables scalability, one of the key advantages of the Euler framework. Traditional GNN-based temporal link predictors, such as Variational Graph Recurrent Neural Networks (VGRNN) [11], operate sequentially, where the GNN embedding at time t depends on the RNN output from time $t - 1$. In contrast, Euler's framework decouples topological encoding from temporal processing, allowing GNN computations to run in parallel on multiple machines using a master/slave paradigm. Since message passing in GNN models is the most computationally expensive step, this design significantly improves efficiency and scalability.

The authors evaluated the Euler using several GNNs: GCN, GAT, and GraphSAGE. As for the RNN, Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) were used. These variants were then compared to the previous state-of-the-art temporal link predictors like VGRNN using LANL and DARPA OpTC datasets. Euler's scalability was also evaluated through theoretical complexity analysis and empirical runtime benchmarks.

C. Jbeil

While Euler introduced key advancements in lateral movement detection through temporal link prediction, it also has some limitations. Euler is trained in a transductive setting, meaning the full adjacency matrix (including all nodes) is available during training, with only edges being masked. In contrast, Jbeil [8], proposed by Khoury et al. in 2024, employs an inductive approach, enabling it to generalize to unseen data. This makes Jbeil particularly well-suited for dynamic enterprise networks, where new events frequently arise.

Another key distinction from Euler lies in the graph representation used. Jbeil is built upon the Temporal Graph Networks

(TGN) framework [12], which employs a continuous-time temporal graph, enabling finer temporal granularity. Formally, Jbeil represents the network graph structure as a set of events $G = \{e_{t1}, e_{t2}, e_{t3}, \dots\}$. Each event is then defined as:

$$e = \{t, src, dst, l, v_{src}, v_{dst}\} \tag{4}$$

where t denotes the timestamp of the event, src and dst are the source and destination nodes, and l represents the event label indicating benign or malicious activity. The vectors v_{src} and v_{dst} capture dynamic graph features that encode node interactions, including in-degree, out-degree, and temporal activity statistics.

To capture a node's interaction history, Jbeil utilizes a temporal node memory mechanism. Initially, each node's memory is initialized with its features. Since Jbeil operates in a continuous-time manner, message passing and subsequent memory updates occur whenever an interaction involving the node takes place. The memory update process follows these equations:

$$m_{src} = msg_{src}(m_{src}(t-1), m_{dst}(t-1), t) \tag{5}$$

$$m_{dst} = msg_{dst}(m_{src}(t-1), m_{dst}(t-1), t) \tag{6}$$

where m_{src} and m_{dst} represent the memory of the source and destination nodes, respectively, and msg_{src} and msg_{dst} are learnable functions. In this work, these functions are modeled using recurrent neural networks, specifically Gated Recurrent Units (GRUs).

The final embedding $z_i(t)$ for node i at time t is computed by aggregating the memories of its neighbors:

$$z_i(t) = \sum_{(0,t)} h(s_{src}(t), s_{dst}(t), v_{src}(t), v_{dst}(t)) \tag{7}$$

where $h()$ is a learnable function ranging from a simple identity function to more complex attention mechanisms. Resulting node embeddings are then used to perform the link prediction task.

III. Datasets

We leverage three publicly available datasets to evaluate and compare the aforementioned methods. The properties of all datasets are summarized in Table II. The Pivoting Dataset is a snippet of data utilized by Apruzzese et al. to evaluate the pattern-based detection algorithm [3]. It comprises of internal NetFlow data from a single working day within a large organization [15]. The dataset contains 74,551,643 flows, each characterized by 11 attributes, including source and destination IP addresses, ports, and other flow parameters, accompanied by a label indicating whether a flow is associated with pivoting activity.

Comprehensive, Multi-Source Cyber-Security Events [13] is a dataset gathered from five sources in the internal network of Los Alamos National Laboratory [16]; for convenience, most sources refer to this dataset as the "LANL dataset." Spanning 58 consecutive days in 2015, it comprises authentication logs and process execution data from Windows-based computers and servers, NetFlow records from routers, and DNS lookup events from the central DNS server. Additionally, the dataset includes a set of authentication events known to be malicious, originating from red team operations.

The Operationally Transparent Cyber (OpTC) study, conducted by the Defense Advanced Research Projects Agency (DARPA) in 2019, examined Advanced Persistent Threat (APT) attacks within an enterprise network [14]. As part of this study, DARPA introduced a large dataset spanning seven days, structured using the Extended Cyber Analytics Repository (eCAR) data model, an extension of MITRE's Cyber Analytics Repository (CAR) model [17]. The dataset primarily consists of network flow data, with the remaining portion capturing host-level logs, including process activity, file operations, and user sessions [18]. Unlike LANL, the red team activities in OpTC are documented in a text file rather than just a list of isolated malicious events. However, some ambiguity may exist regarding the exact labeling of the data. To address this, we adopted the same labeling approach used by the authors of Euler. Specifically, OpTC includes three days with documented red team activity, each spanning an entire day. An IP address is considered malicious if a successful attack regarding this IP was observed earlier that day and resets at midnight [7]. The scripts used for parsing and labeling the data are provided in the appendix of this work.

IV. Results and Discussion

The previous section identified PIKACHU, Euler, and Jbeil as the most suitable candidates for our work. In this section, we evaluate their implementations, attempt to reproduce the performance reported in their respective papers, and test all three methods across the public datasets introduced earlier. This practical assessment aims to verify the usability of methods, the reproducibility of their results, and explore their generalization capabilities across different environments. All tests were conducted on a machine with a 10-core Apple M1 Pro CPU and 16 GB of memory, macOS Sequoia 15.2, and Python 3.10.

A. PIKACHU

Before using PIKACHU for testing, we had to resolve several technical issues in the codebase. The repository lacked specific versioning for its Python dependencies, and the requirements file was incomplete. We first added a proper requirements file with the latest package versions to address this. However, this led to API version mismatches, which we had to fix before proceeding with the tests. Furthermore, the code contained an index out-of-bounds bug, which we also had to fix.

We initially attempted to run the code on the provided OpTC sample. However, even though the sample represented only about 10% of all flows, the performance was unsatisfactory, as noted in an unresolved issue on GitHub. The primary bottleneck was calculating probability scores due to slow Python-level cycles. By introducing caching and leveraging vectorization, we

TABLE II
SUMMARY OF PIVOTING DETECTION DATASETS USED IN RELATED WORK.

Dataset	Data Type	Hosts	Events	Duration
Pivoting dataset [3]	NetFlow	1,015	74,551,643	1 Day
LANL [13]	Logs & NetFlow	15,610	49,341,300	58 Days
DARPA OpTC [14]	Logs & NetFlow	1,000	17,433,324,390	7 Days

TABLE III
PERFORMANCE COMPARISON OF PIKACHU, EULER, AND JBEIL ON DIFFERENT DATASETS. VALUES IN THE FIRST COLUMN ARE THOSE REPORTED BY THE AUTHORS; VALUES IN THE SECOND COLUMN ARE FROM OUR TESTING.

PIKACHU [6]

Dataset	TPR (%)		FPR (%)		AUC	
LANL	95.1	94.302	5.05	5.6975	0.94	0.9521
OpTC	98.7	68.7108	0.42	31.2893	0.99	0.7760
Pivoting	—	52.6316	—	46.2994	—	0.6211

Euler [7]

Dataset	TPR (%)		FPR (%)		AUC	
LANL	89.65	89.5938	0.5723	0.57	0.9913	0.9871
OpTC	17.8	12.7059	0.168	0.1385	0.882	0.6540
Pivoting	—	55.56	—	33.80	—	0.6453

Jbeil [8]

Dataset	Setting	TPR (%)		AUC		Precision	
LANL	Trans.	99.22	53.60	0.9982	0.7580	0.9893	0.00025
	Induct.	99.25	54.47	0.9973	0.7650	0.9848	0.00053
OpTC	Trans.	—	25.60	—	0.2350	—	0.0020
	Induct.	—	12.00	—	0.2120	—	0.0040
Pivoting	Trans.	97.09	75.00	0.9812	0.8340	0.9912	6.31×10^{-7}
	Induct.	97.42	50.00	0.9826	0.7040	0.9909	2.25×10^{-7}

significantly reduced the computation time from approximately 4 hours to just a few minutes on our test machine.

Following the author's setup, we first ran PIKACHU on the LANL dataset. We randomly downsampled benign users to address the dataset's severe class imbalance to achieve a 20:1 ratio of benign to anomalous users, as specified in their approach. Due to PIKACHU's high memory requirements, we used a default embedding size of 64—lower than the 100 used by the authors in their paper—and a training context of 5 hours only. Despite this adjustment, our results closely matched those reported by the authors, with most metrics differing by only one percentage point, as seen in Table III.

In the case of the OpTC dataset, there is some ambiguity. The authors do not explicitly state whether they applied the same type of downsampling as in the LANL dataset. However, our fully parsed OpTC dataset contains approximately 92 million flows, whereas the authors report their dataset containing only 40 million flows. This discrepancy suggests that downsampling was likely used. To maintain consistency, we applied the same 20:1 downsampling ratio as in the LANL dataset. Despite this, we were unable to reproduce the authors' results. One possible explanation is the difference in labeling. Upon inspecting the provided sample, we identified several labeling inconsistencies. For instance, a flow recorded at 11:26:00.277, originating from the compromised IP address 142.20.56.202 and communicating with a command-and-control server at 132.197.158.98, is clearly a malicious edge but is incorrectly labeled as benign.

For our experiments on the Pivoting dataset, we attempted to downsample it like the previous two datasets. However, the dataset remained too large to train on our test machine even after downsampling. A quick exploration revealed that most benign data originated from a single host. To reduce the dataset size while preserving its structure, we downsampled traffic from this host to 10%, resulting in a dataset approximately one-sixth of its original size. All anomalous flows were retained in full. The results in this case are not particularly impressive, with a high number of false positives and performance only marginally better than a random classifier, as shown in Table III.

From our testing, we observed that one significant drawback of PIKACHU is its lack of efficiency, particularly in scaling with larger graph sizes. We attribute this primarily to the complexity of combining short-term and long-term node embeddings. Processing larger datasets took hours on our test machine, even after downsampling. For LANL, running the test took almost one whole day. As a result, we did not repeat the experiments five times as the original authors did. While a more powerful server could reduce processing time, the overall efficiency remains suboptimal, especially when compared to more scalable methods like Euler.

B. Euler

Euler stands out as one of the better methods regarding code quality and overall functionality. The code is well-structured and thoroughly documented, including a descriptive README

file with instructions on how to run the experiments. However, compared to the details provided in the paper, the public repository is incomplete. Notably, the OpTC dataset parser is only partially implemented and exists only on a secondary branch. Similarly, an augmented version of Euler incorporating a softmax function within the decoder (EulerSM) is included as a work-in-progress on a separate branch.

To assess the reproducibility of the reported results and identify the most effective configuration for our use case, we tested several model combinations on the LANL dataset. Since our pipeline focuses on analyzing already collected data rather than forecasting future events, we exclusively employed Euler's link detector setting. Following the methodology described in the original paper, each experiment used a 30-minute snapshot size, with results averaged over five independent runs. Our findings are consistent with those reported by the authors, confirming that the best performance is achieved when combining GCN with either GRU or LSTM. LSTM performed slightly better in our tests, and we reported this result in Table III. These results further support the authors' claim that, despite its simplicity, GCN is a robust and generalizable model capable of delivering strong performance [7].

Next, we conducted similar tests on the OpTC dataset. As with the LANL dataset, we used a portion of the data up to the first detected anomaly for training and the remainder for testing. We again employed snapshot windows of 30 minutes and used a GCN encoder with LSTM and GRU models. Similarly to the LANL dataset, LSTM again performed negligibly better, and we used this result in Table III. Our results were slightly worse than those reported by the original authors, likely due to differences in preprocessing. In the work-in-progress version of the code (according to the authors, the version from the article was lost due to a hard disk failure), we observed that they applied additional filtering steps, such as removing broadcast communications. However, we chose not to apply this filter, as broadcast traffic can also be potentially malicious and may provide valuable signals for detection.

For the Pivoting dataset, we followed a methodology similar to that used with the previous two datasets. Specifically, we used only the benign portion before the first malicious event for training, reserving the remaining data for testing. We also segmented the data into 30-minute snapshots and employed a GCN encoder in combination with an LSTM recurrent model. However, we experimented with different values of the lambda parameter in the Euler model, which controls the trade-off between the true positive rate (TPR) and the false positive rate (FPR). The default value of 0.6 failed to detect anomalous edges, while lowering it to 0.5 resulted in only a single detection. However, a lambda value of 0.4 yielded better results and more effectively identified anomaly edges. As shown in Table III, the performance was still suboptimal, but it outperformed PIKACHU.

The only issues we encountered with Euler were several random crashes during the scoring phase on the LANL dataset. According to the authors, this is likely due to insufficient memory, as Euler was primarily designed for distributed environments and tested on systems with several hundred gigabytes of RAM. This explanation seems plausible, as we experienced no issues when we tried to reduce the test data size on the LANL dataset.

C. Jbeil

The Jbeil model repository is in the poorest technical condition among those evaluated. Although the codebase includes a README file, it does not function as intended. The preprocessing script assumes an already preprocessed dataset, but the necessary steps to achieve this are neither documented nor provided. Furthermore, the code structure is suboptimal, featuring lengthy functions with extensive commented-out sections. Attempting to run the model also results in errors.

An even more significant issue arises in the evaluation part of Jbeil (evaluation.py), which is used for validation and testing. The code fails to incorporate ground truth labels. Instead, it performs a standard link prediction task, treating edges in the graph as true positives and negatively sampled edges as true negatives. If the authors used this code for evaluation, they inadvertently assessed a different task altogether.

Since a fork[1] of the repository addresses the most critical bugs, we used it as our starting point. While this fork resolves the major issues present in the original Jbeil implementation, training the model on the LANL dataset still proved to be challenging. The forked version relies solely on properties from the dataset; however, the original Jbeil authors used extracted graph features during training. While the feature extraction code is provided, it is only available as a disorganized collection of inconsistently named Jupyter notebooks. To address this, we refined the notebook-based code and extracted the features from the LANL dataset.

Another major issue is that the authors did not specify how they split the dataset. The code suggests a 70:15:15 split for training, validation, and testing. However, this approach is unsuitable for the LANL dataset, as the dataset's last quarter does not contain any malicious events. The authors may have used a smaller subset of the dataset, but this is not clearly stated in the paper as well. Initially, we attempted a similar split to the one used by Euler (training and validation on the portion before the first anomaly, testing on the remaining data). This approach did not yield promising results (TPR of 0.401, AUC of 0.659, and precision of 0.0000688).

Since the paper claims that Jbeil can detect malicious patterns within the training data, we conducted a second experiment. This time, we split the data so that approximately half of the anomalous events were included in the training/validation set and the other half in the testing set. This approach produced slightly better results, which are presented in Table III; however, the performance remains significantly below the results reported in the paper.

For the OpTC dataset, we employed a methodology analogous to that used with the LANL dataset. The entire dataset was leveraged, with 30% of the nodes masked to evaluate

[1] https://github.com/TristanBilot/Jbeil

the inductive learning capabilities of Jbeil. Feature selection adhered to the same principles as in the LANL setup, except for user-related features, as only NetFlow data was utilized from the OpTC dataset. Consistent with the previous approach, approximately half of the anomalous edges were allocated for training and validation, while the remainder were reserved for testing. Once again, the results were suboptimal, with Jbeil underperforming compared to PIKACHU and Euler.

We also evaluated Jbeil on the Pivoting dataset. In this case, we masked 40% of the nodes for inductive evaluation, following the same masking strategy described by the original authors. Apart from this, we maintained a similar experimental setup. However, the results again fell far short of those presented in the original publication.

A further complication is that the Jbeil paper does not disclose the specific hyperparameters used during training. Consequently, we initially relied on the default parameters provided by the implementation. However, the paper refers to using node memory, a feature not enabled by default. Therefore, we conducted additional tests on the Pivoting dataset with node memory enabled to account for this. However, this adjustment did not lead to any noticeable improvement in performance.

These results clearly suggest that the authors either uploaded a different version (which is also almost identical to the original TGN code) of Jbeil to GitHub or reported results from the link prediction task instead of anomaly detection. When we applied the evaluation code from the original Jbeil repository, we obtained results that closely matched those reported by the authors. However, this corresponds to a fundamentally different task: predicting the mere existence of a link between nodes rather than determining whether the link is anomalous. We attempted to contact the authors via email for clarification, but have not received a response.

V. Conclusion

In this paper, we explored the malicious tactic of pivoting and the various methods used to detect it. We began by examining lateral movement within the broader context of advanced persistent threats (APTs) and other sophisticated adversaries, with a particular emphasis on pivoting maneuvers. We then reviewed existing pattern-based and machine learning-based pivoting detection techniques, focusing on methods analyzing solely network traffic and not needed data, such as system logs, from the hosts in the network. Our contribution is the replicability study of network-based pivoting detection methods. We found out there is only a limited number of implementations or functional prototypes. On the contrary, the publicly available implementations deliver the expected results outlines in the respective papers.

Although our work highlights and replicates promising results, the technique of pivoting remains a difficult problem—particularly in complex network environments. As part of future work, we aim to develop a pivoting detection tool that would use the proposed methods and deploy it in an operational setting for a longer period of time to evaluate its real-world performance. The source codes of the tools used in the experiments were forked on GitHub and made publicly available with all the fixes required for successful run of the experiments. See the repositories for PIKACHU[2], Euler[3], and JBeil[4] for more details.

References

[1] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, "A Survey on Advanced Persistent Threats: Techniques, Solutions, Challenges, and Research Opportunities," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1851–1877, 2019.

[2] C. Smiliotopoulos, G. Kambourakis, and C. Kolias, "Detecting lateral movement: A systematic survey," *Heliyon*, vol. 10, no. 4, p. e26317, 2024.

[3] G. Apruzzese, F. Pierazzi, M. Colajanni, and M. Marchetti, "Detection and threat prioritization of pivoting attacks in large networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 8, no. 2, pp. 404–415, 2020.

[4] M. Husák, G. Apruzzese, S. J. Yang, and G. Werner, "Towards an efficient detection of pivoting activity," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 980–985.

[5] R. S. Marques, H. Al-Khateeb, G. Epiphaniou, and C. Maple, "Apivads: A novel privacy-preserving pivot attack detection scheme based on statistical pattern recognition," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 700–715, 2022.

[6] R. Paudel and H. H. Huang, "Pikachu: Temporal walk based dynamic graph embedding for network anomaly detection," in *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, 2022.

[7] I. J. King and H. H. Huang, "Euler: Detecting network lateral movement via scalable temporal link prediction," *ACM Transactions on Privacy and Security*, vol. 26, no. 3, pp. 1–36, 2023.

[8] J. Khoury, Đ. Klisura, H. Zanddizari, G. De La Torre Parra, P. Najafirad, and E. Bou-Harb, "Jbeil: Temporal graph-based inductive learning to infer lateral movement in evolving enterprise networks," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 3644–3660.

[9] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[10] B. Bowman, C. Laprade, Y. Ji, and H. H. Huang, "Detecting lateral movement in enterprise computer networks with unsupervised graph AI," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, 2020, pp. 257–268.

[11] E. Hajiramezanali, A. Hasanzadeh, N. Duffield, K. R. Narayanan, M. Zhou, and X. Qian, "Variational graph recurrent neural networks," 2020.

[12] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *arXiv preprint arXiv:2006.10637*, 2020.

[13] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events," Los Alamos National Laboratory, 2015. [Online]. Available: https://csr.lanl.gov/data/cyber1/

[14] R. Arantes, C. Weir, H. Hannon, and M. Kulseng, "Operationally Transparent Cyber (OpTC) Data Release," 2019. [Online]. Available: https://github.com/FiveDirections/OpTC-data?utm_source=chatgpt.com

[15] G. Apruzzese, "Pivoting detection dataset," 2017. [Online]. Available: https://weblab.ing.unimore.it/people/apruzzese/datasets/pivoting.html

[16] A. D. Kent, "Cybersecurity Data Sources for Dynamic Network Research," in *Dynamic Networks in Cybersecurity*. Imperial College Press, Jun. 2015.

[17] The MITRE Corporation, "MITRE Cyber Analytics Repository - Data Model," 2022. [Online]. Available: https://car.mitre.org/data_model/

[18] M. M. Anjum, S. Iqbal, and B. Hamelin, "Analyzing the Usefulness of the DARPA OpTC Dataset in Cyber Threat Detection Research," in *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*, ser. SACMAT '21. ACM, Jun. 2021, p. 27–32.

[2]https://github.com/vboucek/Pikachu

[3]https://github.com/vboucek/Euler

[4]https://github.com/vboucek/Jbeil