Advancing Cloud-Native Cyber Threat Detection with Graph-Based Feature Engineering

Advancing Cloud-native Cyber Threat Detection with Graph-based Feature Engineering

Tailai Song*, Mukharbek Organokov†, Lennart Gulikers†, Giulio Grassi†, Giovanna Carofiglio†, Michela Meo*
*Politecnico di Torino, Turin, Italy, †Cisco Systems Inc., Paris, France
*first.last@polito.it, †morganok/lguliker/gigrassi/gcarofig@cisco.com

***Abstract*—In light of the unprecedented proliferation of cloud-native applications, cyber threats targeting cloud services have escalated markedly, emerging as a critical concern for stakeholders. The intrinsic nature of cloud infrastructures renders them particularly susceptible to diverse attacks. In this context, effective attack identification becomes pivotal, facilitating swift responses and preventive measures to mitigate risks and bolster overall security resilience. Despite a range of solutions in literature, attack classification remains an arduous task in industrial environments. To address this, we propose a comprehensive and deployment-friendly graph-based framework. It leverages cloud activity traces, transforming system events to graph structures, and we enumerate 4 different types of attack within a controlled environment. We frame a multi-classification problem, and construct multi-level features to characterize distinct attacks amidst background activities, bypassing the complexity of Deep Learning (DL). To evaluate the efficacy, we compare with multiple Graph Neural Networks (GNNs), and our solution yields comparable performance, demonstrating a promising candidate for the practical and efficient cyber threat detection.**

***Index Terms*—Cloud system, cybersecurity, attack classification, graph, feature engineering, ML, DL, knowledge discovery**

I. Introduction

The cloud-native paradigm underpins multitudinous modern internet applications, serving as the de facto standard for deploying scalable systems. In the wake of the rapid expansion of cloud services, cybersecurity for cloud-native applications has become a critical imperative. The inherent vulnerabilities of cloud infrastructures create potential entry points and attack surfaces, perplexing the enforcement of security countermeasures and real-time threat detection [1], [2], and attacks continually diversify in forms, further exacerbating an already complicated threat landscape. Consequently, cyber threats can significantly disrupt cloud environments, posing severe risks to both service providers and users.

To this end, there is a compelling impetus to develop an effective threat detection system capable of differentiating among various threat types and safeguarding the reliability of cloud services. Pertinent studies have drawn significant attention but fall short mainly in 3 aspects: (1) focusing narrowly on attack classification itself, while overlooking the importance of a whole comprehensive system, (2) relying on relatively simplistic datasets, which lack the complexity to represent the nuanced characteristics of diverse attacks, thus resulting in deceptively high performance (e.g., 99% accuracy) that is overly optimistic and unlikely to generalize well in real-world scenarios, and (3) overly pursuing intricate, computationally intensive deep learning (DL) models that, while powerful, impose restrictive demands. These limitations substantially constrain the practicality and applicability of existing solutions, particularly in real-world environments.

In response, we introduce a graph-based framework, leveraging conventional yet efficacious feature engineering to characterize attacks, therefore bypassing the complexity associated with DL methods. Specifically, we employ SysFlow [3] and Falco [4] to trace and log cloud system activities, which are then transformed into knowledge graphs (KGs). As a result, malicious or benign processes are encapsulated and portrayed by a series of graphs with enriched information. Subsequently, for each graph, we apply feature engineering to generate comprehensive numerical features that can be utilized by a lightweight Machine Learning (ML) model to classify different attacks. Additionally, we emulate 4 distinct types of cyber attack in a controlled environment to produce representative and dependable datasets with guidance from domain experts. Formally, we formulate a multi-classification problem, and compare with various graph neural networks (GNNs). Conclusively, our method yields competitive classification performance, and achieves markedly enhanced temporal efficiency, demonstrating a viable solution to advance contemporary cloud-native cyber threat detection.

II. Problem context

A. Background, objective, and motive

Cloud-native applications, engineered to capitalize on cloud infrastructures, are typically deployed within a cluster architecture orchestrated by containerization systems such as Kubernetes (K8s). While container-based virtualization enables rapid deployment and optimizes resource utilization, it also introduces unique security risks, such as vulnerabilities in container images and misconfigurations in K8s clusters. Any breach in these environments can lead to severe repercussions, e.g., unauthorized data access and disruption of microservices. Such threats can allow adversaries to establish persistent access, modify or delete critical content, or execute malicious programs, ultimately causing significant financial and reputational damage. Fortunately, cyber attacks are generally neither instantaneous nor ephemeral; rather, they unfold over a series of stages, as outlined in frameworks like MITRE ATT&CK [5], including phases such as execution of malicious code, privilege escalation, and more. These often leave identifiable traces across system logs, network traffic, and application layers, which, if detected and analyzed, can

This work was supported by Cisco Systems Inc., the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART", Focused Project R4R), and the SmartData@PoliTO center on Big Data and Data Science. The authors would like to thank Jacques Samain, Jordan Auge, and Luca Muscariello for their contributions.

reveal patterns that signal an impending or ongoing attack. By tracking these breadcrumbs, security teams can trace the attack's origin, monitor its progression, and pinpoint affected systems, allowing for prompt or even preemptive responses. Effective threat detection systems leverage these traces to mitigate current threats and strengthen future defenses, proactively fortifying resilience against evolving attack vectors.

To achieve this, we resort to SysFlow and Falco to facilitate comprehensive monitoring and threat detection in cloud-native environments. SysFlow is an open-source telemetry format and data pipeline designed to efficiently capture system-level events, and Falco is a cloud-native security framework that applies custom rules on system calls to enrich runtime data with contextual metadata. These tools depict interactions between processes and environments by tracing cloud activities, linking relational dual-objects to form triplets, and aggregating these interconnected triplets to build graph-like data structures. This aids to recognize malicious TTPs (Tactics, Techniques & Procedures), thus enabling the reasoning and tracking of cyber attack footprints. Moreover, such compact summarizations with essential attributes can be smoothly converted into KGs to aptly represent attack chains for further analysis and classification.

To cater the pragmatic requirements of industrial production in tandem with the aforementioned factors, our objective is to distinguish between graphs of different types of attack and benign processes through an efficient feature engineering approach. Our motivation is fourfold: 1) Established solutions based on network traffic or audit logs are suboptimal for cloud environments; 2) Prevailing DL technologies are often costly regarding training (especially when re-training is frequently needed to adapt to evolving threats) and inference (as it is beneficial to classify attacks and enact countermoves rapidly, empowering the system to function in real time); 3) DL models are generally black boxes lacking interpretability, which is, however, crucial for understanding decision-making processes, fostering trust in sensitive scenarios, communicating insights effectively to stakeholders, and establishing preliminary defenses; 4) Classical ML approaches have a mature ecosystem with well-established frameworks (e.g., Kubeflow) that can be swiftly deployed and readily integrated into modern DevOps cycle, while DL methodologies typically characterize multifarious architectures and impose specific requirements, such as custom containerization, substantially restricting their practicality, maintainability and scalability.

B. Related work

Cyber attack detection can be generally categorized into network- and system-based, where the former scheme falls outside the scope of our work. System-based detection focuses primarily on the host level, monitoring system logs, file integrity, and network activity to discover malicious processes [6], [7], and thus excelling at recognizing threats against specific cloud-native applications. In recent years, DL technologies have essentially reshaped attack classification [8], [9]. For instance, [10] introduced a Recurrent Neural Network (RNN)-based autoencoder to detect network intrusion using privacy and system log data from clients,

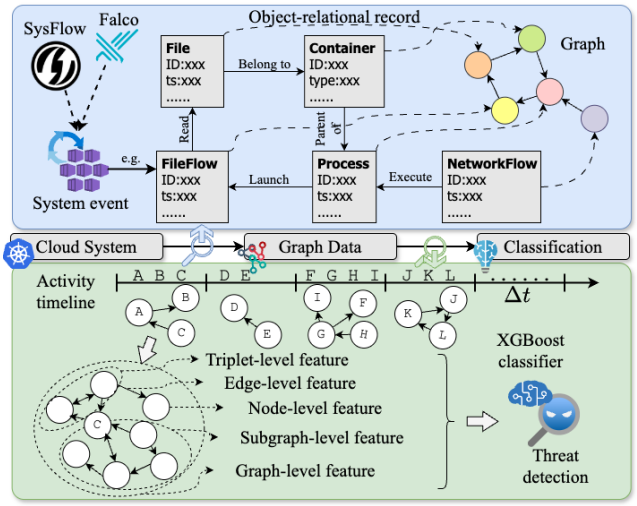

Fig. 1: System overview.

while [11] proposed a DL-based ensemble model to classify ransomware attacks through transfer learning and cloud-encrypted data. The authors in [12] developed a sequence-learning approach to identify attack sequence out of system activity causal graphs based on audit logs, by utilizing Natural Language Processing (NLP) methods to generate sequence embeddings, and a Long Short-term Memory (LSTM) Neural Network (NN) to model sequences. Moreover, the recent surge of Large Language Model (LLM) was not left behind, with [13] employing BERT [14] to detect malicious logs for Unix shell attacks. Furthermore, KGs have also gain considerable attentions in cybersecurity [15]–[18]; however, the majority of them focused on named entity recognition or relation extraction with high-level network or system data, which are not applicable to our specific cloud scenario, not to mention their widely-adopted DL solutions. Additionally, while "shallow" learning methods, such as conventional kernel-based and boosting-based graph classification [19], remain viable options, their effectiveness in modern cybersecurity domain with enriched graph features is limited due to the relatively outdated and deficient performance [20].

While the hype of DL for cyber threat detection continues to ferment in academia, with NNs demonstrating impressive performance, their practical applicability and effectiveness for real-world applications remain skeptical in industry scenarios [21], [22]. This gap prompts us to rethink the necessity of DL approaches and to revisit the traditional ML domain, wherein the technologies, while not as trendy, are comparatively reliable and implementable.

III. Methodology

Herein, we delineate the proposed graph-based framework with feature engineering, as described in Fig.1.

Dataset generation: SysFlow serves as the backbone of our framework by tracing and formatting cloud activities. When a process, either malicious or benign, initiates, SysFlow tracks its progression and correlations with system elements/resources, logging system events to create relational objects following chronological order in real time. As exemplified by Fig.1 (upper), a network flow executes a process within the container, which triggers a file flow to read a file

belonging to the container. At any given moment during the process, the corresponding system events and interactions are recorded to form a triplet (e.g., `process` → `delete` → `file`), which can represent a segment of the attack chain.

To conduct a thorough analysis of cyber threats, we generate 4 typical attack types that are prevalent and predominant in cloud-native environments [23], [24]: crypto-jacking (3,480 triplets in total), data destruction (14,631), data exfiltration (7,956), and ransomware (1,343) in a controlled environment, emulating attacker behaviors, replicating attack procedures, and recording the whole process. More importantly, we incorporate diverse attack technologies/strategies for each attack type. As a target of attacks, we use the vulnerable OWASP Juice Shop [25] application, exploiting its initial access vulnerability. For instance, once access is obtained, the attacker performs discovery operations — accessing to environment variables (`env` command), ongoing processes (`pwd`), list of open ports (`netstat`), file systems (`find`, `ls`), etc. Then, depending on the threat category, the attacker proceeds to the impact phase, compressing and sending data over the network (e.g., via `nc`), encrypting data (ransomware), deleting data (data destruction), or downloading and executing binaries for crypto-mining (e.g., `xmrig`). To introduce variability among attacks, both the specific attack steps and the targeted data vary, and in some cases, the attacker escalates privileges within the target cluster to deploy pods via `kubectl`, executing impact commands within these newly created environments. Additionally, we also generate benign activities by automating browsing activities on the target web-service, to simulate normal application usage, providing background noises (12,253). Crucially, despite being emulated, both the process and dataset are meticulously crafted at an industrial scale and standard, ensuring a high degree of fidelity to real-world conditions.

Graph construction: As a result of the above-mentioned procedure, a certain process (malicious or benign) generates a series of triplets, each resembling the basic element of a graph, i.e., node–edge–node. Specifically, each triplet features a directional relationship, i.e., `from_node` → `relation` → `to_node`, where a node represents one of 10 system element types (e.g., file, process, path), and the edge denotes one of 17 relation types (e.g., create, open, update). Additionally, each triplet contains auxiliary information including a timestamp and possible node descriptions. As SysFlow is able to trace a process from its onset till its termination without discontinuity, thereby forming an activity sequence, a `to_node` can also be a `from_node` in another, and each node may belong to multiple triplets, thus reflecting an interrelated structure. Subsequently, as portrayed by Fig.1 (lower), we segment a process into multiple consecutive time windows of duration Δt, aggregating all interconnected triplets within each window to form a directional graph. Importantly, we adopt a key modelling assumption that each resulting graph (sub-process) inherits the class label of its parent graph (the entire cloud activity of either an attack or noise). The rationale stems from two factors: (1) besides the affinity between a graph and SysFlow outputs, graph structures are naturally instrumental in capturing contextual, relational, localized, and global traits, and (2) one or few standalone triplets are not sufficiently representative to characterize an attack or to form an informative graph.

Attack classification: After previous steps, each class of cloud activity yields multiple graphs, and our task becomes a multi-classification problem, distinguishing graphs among background noises and 4 types of attacks. Noteworthily, our framework could intervene and query the process at any point, starting to construct graphs and classify their properties. Upon the detection of a threat in real time, several actions may be taken, such as terminating/blocking the process, or reporting to the security team, at which point it is unnecessary to continue monitoring and generating graphs. In contrast, we opt to observe the entire progress of an activity and generate all corresponding graphs, because we aim to unveil the full patterns of the attack progression while also gathering more data for model training.

Feature engineering: We devise numerous features constructed directly from graphs as follows:

Node-level: At the most granular level, we construct features for the basic element of a graph (nodes) in 3 categories: ***(i)*** **Node type**—Each node within the 10 node types can function as either a source or a target, as the edge is directional. We compute the absolute count and frequency of each node type across `from_node`, `to_node`, and all nodes. ***(ii)*** **Node statistics**—We derive 4 sets of features: 1) Node degree—average node/in/out degree, the total degree of a node type, and whether a node type corresponds to the maximum node degree; 2) Average node centrality for each node type; 3) Mean clustering coefficient for each node type; 4) Mean overlap measure between pairs of distinct node types for all possible node combinations. ***(iii)*** **Keyword-related**—Attacks often exploit cloud infrastructures using distinct techniques, leaving trails that SysFlow and Falco capture in node descriptions, which may contain specific keywords. We consider four of them: `bash`, `xmrig`, `busybox`, `tmp`, e.g., XMRig, a CPU/GPU miner, may denote the operation of cryto-jacking. We compute the quantity and frequency of nodes encompassing a specific keyword, the number of different `from_node` or `to_node` types associated with nodes containing each keyword, and the diversity of source or target node types when the corresponding target or source node includes a keyword. Consequently, we potentially identify anomalies, such as a process creating a temporary directory, which may signal atypical activity rarely observed in normal operations.

Edge-level: The second feature group pertains to edges, focusing primarily on edge types. We consider two categories of features: ***(i)*** **Edge type**—With a total of 17 edge types, we calculate the number and frequency of each edge type, analogous to the approach for node types. ***(ii)*** **Keyword-related**—Considering the same keyword list, we derive the count of different edge types when `from_node` of the edge contains a certain keyword and when `to_node` does.

Triplet-level: Triplets constitute the most important element of graphs, manifesting explicit relationships between system events induced by an attack. As attackers often adhere to specific steps, exploiting the system in a manner seldom replicated by benign activities, certain triplets (i.e., partic-

ular actions) can reveal anomalous behaviors. We develop 2 primary categories of features: ***(i)*** **Node/edge type**—we aim to uncover the interdependence between bipolar nodes and the connecting edge by computing (1) the number of triplets for each unique triplet type, (2) the number of each edge type and the number of each `to_node` type for a given `from_node` type, (3) the number of each edge type and the number of each `from_node` type for a given `to_node` type, and (4) the number of each edge type when its connected `from_node` or `to_node` belongs to a certain node type. ***(ii)*** **Keyword-related**—We count the occurrences of each distinct source/target node-edge type pair for each keyword in target/source nodes with a given node type.

Graph-level: We extend our analysis to a global scale, investigating topological properties and graph-level features. In addition to basic features, such as the total number of triplets or nodes, we leverage spectral graph theory [26], manipulating a graph's Laplacian matrix and its eigenvalues to compute 10 features, including the largest eigenvalue, the summation/average/variance of eigenvalues, the spectral gap/radius, the graph energy, and the Laplacian energy/rank/nullity.

Subgraph-level: We partition a graph into multiple subgraphs to explore quasi-localized features. During the progress of a cloud activity, various correlated or uncorrelated processes are activated with a mutual link to a centralized node, `cluster`. Given this, we split a "big" graph at `cluster` node to generate multiple independent "small" ones, wherein processes remain interconnected but are disconnected between subgraphs. These process-decoupled subgroups can offer distinctive insights from granular vantage points. We calculate 2 categories of subgraph features: ***(i)*** **Quantity-based**—We check the total subgraph amount, and compute the average/maximum/minimum number of nodes/triplets. ***(ii)*** **Node/edge-based**—We calculate the average/maximum/minimum frequency of each node/edge type.

In consequence, we condense the graphs into various features totaling 2895, and we perform a standard data preprocessing pipeline to deal with constant, low-variance, highly-correlated, and non-informative features. As a result, we derive 82 numerical values per graph, which can be fed into a downstream ML model for classifications. Importantly, while the original feature set was extensive, practical implementation merely entails the final 82 features, substantially reducing the complexity of feature construction in real time, not to mention other optimization techniques, such as recursive feature elimination [27], that can curtail the feature quantity even further during the model development phase, enhancing efficiency without compromising performance.

IV. Experimental result

A. Experimental setup

In our case, we set the duration to $\Delta t = 500$ ms, which contains an average of 91 triplets, and consequently, we generate 175 (noise), 70 (crypto-jacking), 102 (data destruction), 69 (data exfiltration), and 22 (ransomware) samples (i.e., graphs). Noteworthily, the actual number of triplets or graphs varies across scenarios, influenced by factors such as the nature of cloud activities and number of cluster pods, while our framework is designed to be generic, with an adjustable window duration Δt to accommodate different requirements.

Proposed ML model: With the consolidated features, we select eXtreme Gradient Boosting (XGB) [28], a renowned tree boosting algorithm, as the base model for three main reasons: (1) XGB has proved to be a highly competent ML model, widely adopted in industrial environments thanks to its robust, lightweight design, (2) it can produce feature importance, thereby offering a certain degree of interpretability, and (3) it is equipped with well-established frameworks such as *KServe InferenceService*, enabling a seamless and effortless deployment with minimal configuration and without customized containers, ensuring a scalable and streamlined production workflow. To improve performance, we explore different sampling strategies and employ Bayesian optimization [29] to tune hyper-parameters systematically.

Comparative models: To justify the competitiveness of our proposed framework, we first implement a baseline model from [30] that generates spectral features derived from the adjacency matrix, degree matrix, and their eigenvalues of a graph. We then refer to a broad technology spectrum, choosing 8 DL models for comparison, including 4 general GNNs based on message passing: Graph Attention Networks (GAT) [31], GENeralized Aggregation Networks (GEN) [32], Graph Isomorphism Networks with edge features (GINE) [33], Unified Message Passing Model (UniMP) [34], 1 large-scale Transformer-based model tailored for graph data: Graphormer [35], 1 graph embedding model, which maps graphs into latent embeddings subsequently injected into a basic multi-layer perceptron (MLP) for classification: Graph Matching Networks (GMN) [36], 1 brand-new state-of-the-art GNN specialized for intrusion detection in the cybersecurity domain: FLASH [37], and 1 freshly-released GNN designed for imbalanced graph classification: edge-enhanced causal attention learning (ECAL) [38]. Each model undergoes necessary architectural and hyperparameter tuning to achieve optimized performance. All models operate on an identical input structure, comprising a tuple (*edge index, num node, class label, node attribute, edge attribute*) for each graph, wherein *edge index* indicates the graph topology via node pairs, *node attribute* contains node features of one-hot encoding for node types, and *edge attribute* represents edge features including edge types, types of the connected `from_node`/`to_node`, and presence of a certain keyword in `from_node`/`to_node`. As a result, each graph is fully described, guaranteeing null information loss for fair comparisons with feature engineering.

Model evaluation: To ensure an unbiased assessment of model performance, we conduct a 5-trial evaluation process, where each trial features a different random and stratified partition of the dataset into 80% for training (an extra split for validation set if needed) and 20% for testing. Notably, our problem involves an extremely imbalanced dataset, which poses an inherent issue in ML, often leading to models disproportionately favoring the majority class. To tackle this, we incorporate class weights during model training, and to provide insights into performance, we average and showcase class recalls and macro average metrics of all trials, present-

TABLE I: Experimental results in terms of class recalls (first block) and macro average metrics (second block).

Model	Noise	Crypto-jacking	Data destruction	Data exfiltration	Ransomware	Recall	Precision	F1-score
Baseline	0.75±0.08	0.37±0.10	0.36±0.06	0.29±0.06	0.00±0.00	0.35±0.02	0.37±0.03	0.35±0.02
GAT	0.61±0.04	0.53±0.12	0.62±0.14	0.31±0.17	0.30±0.19	0.47±0.03	0.56±0.07	0.47±0.03
GEN	0.69±0.04	0.81±0.06	0.56±0.10	0.53±0.12	0.25±0.16	0.57±0.04	0.68±0.08	0.57±0.05
GINE	0.64±0.06	0.79±0.05	0.54±0.11	0.30±0.05	0.00±0.00	0.45±0.02	0.48±0.02	0.43±0.02
UniMP	0.75±0.03	0.74±0.11	0.57±0.10	0.41±0.08	0.25±0.00	0.55±0.04	0.69±0.09	0.56±0.04
Graphormer	0.61±0.05	0.81±0.07	0.50±0.10	0.36±0.10	0.15±0.12	0.49±0.02	0.58±0.10	0.48±0.04
GMN	0.70±0.06	0.53±0.15	0.55±0.13	0.31±0.03	0.05±0.10	0.43±0.02	0.47±0.08	0.43±0.02
FLASH	0.61±0.03	0.77±0.05	0.57±0.09	0.33±0.12	0.30±0.10	0.52±0.02	0.62±0.07	0.52±0.03
ECAL	0.58±0.09	0.64±0.16	0.58±0.10	0.36±0.15	0.35±0.00	0.48±0.04	0.62±0.07	0.49±0.04
XGB	0.61±0.06	0.57±0.18	0.70±0.12	0.41±0.08	0.55±0.24	0.57±0.01	0.54±0.02	0.52±0.01

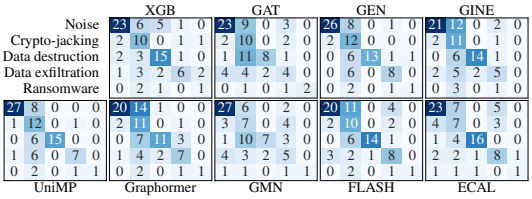

Fig. 2: Example confusion matrices of all models (except the baseline).

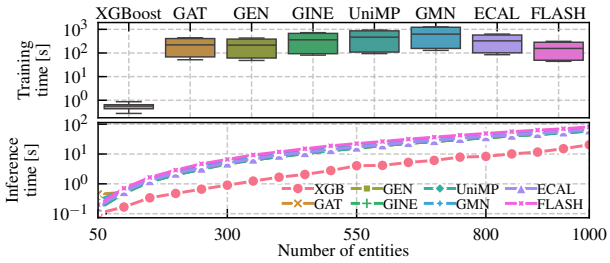

Fig. 3: Time consumption for model training and inference.

ing individual class outcomes and overall performance.

B. Quantitative result

Table I presents the experimental results with composite evaluation metrics of all models. In brief, XGB trained based on feature engineering achieves a comparable performance, even surpassing several DL models. Additionally, all models exhibit subpar outcomes for the minority classes (data exfiltration and ransomware) due to class imbalance. Notably, the baseline model exhibits inferior performance, verifying the deficiency of traditional approaches and rendering it meaningless for inclusion in subsequent analyses.

Concerning benign activities (noise), XGB only attains a recall of approximately 0.61, which is 14% lower than the top-performing model, UniMP. Nonetheless, this is deemed a minor issue in cybersecurity, as false negatives for noises are generally less critical, given that missed noise does not directly compromise security integrity. For individual attacks, XGB appears to deliver only mediocre performance at a first glance, as evidenced by its second-lowest recall around 0.57 for crypto-jacking. However, it performs exceptionally well for other threats, boasting one second-highest and two highest recalls. Notably, XGB demonstrates the only model that yields a recall above 0.7 and 0.5 for data destruction and ransomware, respectively. In contrast, other models struggle to identify ransomware, with their best recall barely reaching 0.35, which can be ascribed to the sensitivity of DL models to class imbalance. In contrast, feature engineering alleviates this issue, successfully classifying over half of samples occasionally. Among DL methods, two general GNNs exhibit remarkable outcomes, achieving the highest F1-score, whereas the behaviors of the giant Transformer model and the graph embedding approach are not commensurate with their complexity. The GNN specialized for security tasks produces respectable results, yet its adaptability to our specific problem remains suboptimal, and the GNN tailored for imbalanced problems indeed identifies minorities, as evidenced by its second-highest recall for ransomware, but its overall performance is hindered by suboptimal results for majority classes. Overall, feature engineering assists to balance the performance among different classes, enhancing the identification of minority without compromising the majority. As a consequence, we obtain a decent overall result, as indicated by the macro average evaluation metrics: the highest recall, moderate precision, and the third-best F1-score.

To provide deeper insights into the performance, we showcase a confusion matrix for each model (Fig.2). Reflecting the numerical metrics, XGB produces the most distinct diagonal phenomenon. Interestingly, all models share a common pattern of misclassified samples tending to be recognized as crypto-jacking. Moreover, despite ransomware comprising only 4 samples in the test set, none of them are misclassified as noises, except for GMN. The aforementioned observations alongside the generally satisfactory precisions for normal activities (only few false positives for noises) highlight the effectiveness of leveraging system-event graphs to distinguish among malicious and harmless cloud activities, with feature engineering standing out as a feasible solution to supersede DL models. Although the results are not groundbreaking due to the inherent challenges of class imbalance and problem complexity, wherein different attacks may exhibit overlapping patterns, such as similar initial access and exploration phases, the performance is deemed acceptable in the cybersecurity domain, especially under the backdrop of the identification between normalcy and attack, where our detailed classification of diverse attacks offers valuable and manifold insights.

C. Overhead analysis

To align with our primary objective of efficient and practical attack detection, we analyze the temporal overhead by examining training and inference times, as outlined in Fig.3. Notably, we deliberately exclude Graphormer due to its enormous scale with nearly 50 million parameters, that results in an excessive time consumption even under the simplest condition. Additionally, the training of DL models is conducted on a single GPU of NVIDIA Tesla V100-16GB, while XGB training and all inferences are carried out on a server-tier CPU of Intel Xeon Gold 6140.

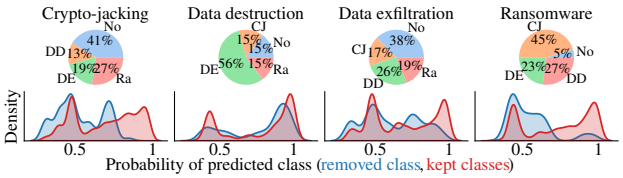

Fig. 4: Outcomes for unknown attacks.

Regarding model training, we evaluate various data quantities and window durations, as more training samples or longer windows that contain more triplets to yield large graphs affect time consumption. The box plots (upper figure) illustrate that XGB consumes a negligible time regardless of the graph complexity and data volume, thanks to a constant feature set and its lightweight nature. In contrast, DL models exhibit substantially higher time requirements and are notably sensitive to training complexity, with consumption increasing alongside graph size and data quantity. Note that while FLASH requires relatively short time, the training time of Word2Vec is not accounted for, suggesting a potentially prolonged duration when the corpus is large. Although the overall consumption seems acceptable (under 1000 s), it remains a concern in today's threat hunting landscape with rapidly evolving attack measures, let alone the redundancy and overhead incurred by repeated reconfiguration and deployment. For inference, we assess the consumption also including feature construction, with respect to increasing triplet quantities, since more triplets produce large graphs, complicating feature computation and inference. The line plots (lower figure) reveal that all models incur greater costs as triplet counts rise, while XGB demonstrates significantly reduced time consumption, which proffers a distinct advantage with more flexibility to enact real-time security measures owing to the extended time preserved. While we acknowledge that feature engineering with XGB may require more time than DL methods for exceedingly large graphs, particularly due to complex features like graph Laplacian, the use of time windows up to a few seconds ensures that the number of triplets remains manageable, essentially eliminating an overly-elongated feature construction time.

D. Unknown attack

We focus on 4 attacks, which may not be exhaustive enough to cope with the ever-growing security risks, given that emerging attacks unbeknownst to detection systems are particularly detrimental. In light of this, we aim to explore the capability of feature engineering in handling unknown attacks by retraining the model with an attack type omitted each time. The pie charts in Fig.4 depict the occupancy of removed attacks predicted as remaining classes. Remarkably, only at most 41% of removed threats are misclassified as benign activities, while the majority, especially when data destruction and ransomware are withheld, are still correctly identified as malicious, which demonstrates the ability to apprehend the underlying patterns of normal cloud processes, and to generalize beyond known attack types, underscoring the resilience in detecting potential threats even when faced with previously unseen attacks. More importantly, we also

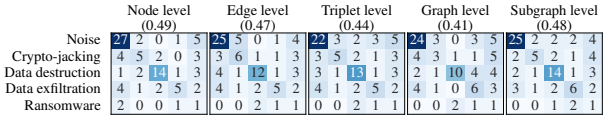

Fig. 5: Confusion matrices: removed feature category (F1-score in bracket).

investigate the predicted probabilities (Fig.4). Evidently, a predominant difference is observed between removed and kept classes, except when data destruction is discarded, where only 15% of attacks are misclassified. This suggests that XGB is more confident with learned threats but uncertain about novelties. This tradeoff offers the potential to implement a probability-based strategy, where lower likelihood scores for unfamiliar patterns could prompt further scrutiny or secondary verification, enhancing detection robustness against emerging threats. Specifically, one could frame an auxiliary binary classification problem alongside the multi-classification one while developing multiple XGB models trained on different data portions. Consequently, we obtain diverse predictions from distinct perspectives, which can then be aggregated using voting strategies with a predefined probability threshold to select confident predictions. Admittedly, addressing long-term variations remains a challenge, ultimately necessitating continuous learning. However, this concern is relatively minor, as XGB retraining is trivial and feature engineering can be readily streamlined, not to mention the well-established MLOps framework and data pipeline that can seamlessly integrate XGB, facilitating incremental updates and effortless scalability to accommodate new attacks.

E. Model interpretability

Rather than merely analyzing feature importance, we examine performance by training XGB for 5 trials, each omitting a feature set (level). The confusion matrices (with F1-scores) in Figure 5 present declined overall performance, affirming the significance of all features and the efficacy of feature engineering. Upon closer inspection, informative patterns elucidate the contributions of different features. For example, the exclusion of node-level features is the only one misclassifying ransomware as noises, and while graph-level features constitute the smallest portion of the feature set, their absence leads to excessive misidentification of cryto-jacking and data destruction, considerably diminishing the F1-score.

V. Conclusion

In this paper, we propose a graph-based framework to characterize cloud activities while avoiding the complication from DL approaches in cyber threat detection. We employ SysFlow and Falco to format system events and convert relational objects into graphs. Subsequently, we generate 4 types of attack and apply feature engineering methods. We frame a multi-classification problem to distinguish graphs, leveraging a base model of XGB and comparing with 8 DL models. Consequently, our feature engineering approach achieves comparable performance and dramatically reduces temporal overhead, demonstrating a feasible candidate for a more practical and efficient cloud-native cyber threat detection. Future work could consist of an investigation into feature importance and additional data generation campaigns.

REFERENCES

[1] Q. Zeng, M. Kavousi, Y. Luo, L. Jin, and Y. Chen, "Full-stack vulnerability analysis of the cloud-native platform," *Computers & Security*, vol. 129, p. 103173, 2023.

[2] T. Theodoropoulos, L. Rosa, C. Benzaid, P. Gray, E. Marin, A. Makris, L. Cordeiro, F. Diego, P. Sorokin, M. D. Girolamo, *et al.*, "Security in cloud-native services: A survey," *Journal of Cybersecurity and Privacy*, vol. 3, no. 4, pp. 758–793, 2023.

[3] T. Taylor, F. Araujo, and X. Shu, "Towards an open format for scalable system telemetry," in *2020 IEEE International Conference on Big Data (Big Data)*, pp. 1031–1040, IEEE, 2020.

[4] Sysdig, "Falco: Open source security tool for containers, kubernetes and cloud.," 2022.

[5] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, "Mitre att&ck: Design and philosophy," in *Technical report*, The MITRE Corporation, 2018.

[6] S. Khan, A. Gani, A. W. A. Wahab, M. A. Bagiwa, M. Shiraz, S. U. Khan, R. Buyya, and A. Y. Zomaya, "Cloud log forensics: Foundations, state of the art, and future directions," *ACM Computing Surveys (CSUR)*, vol. 49, no. 1, pp. 1–42, 2016.

[7] J. Svacina, J. Raffety, C. Woodahl, B. Stone, T. Cerny, M. Bures, D. Shin, K. Frajtak, and P. Tisnovsky, "On vulnerability and security log analysis: A systematic literature review on recent trends," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 175–180, 2020.

[8] A. B. Nassif, M. A. Talib, Q. Nasir, H. Albadani, and F. M. Dakalbab, "Machine learning for cloud security: a systematic review," *IEEE Access*, vol. 9, pp. 20717–20735, 2021.

[9] U. A. Butt, M. Mehmood, S. B. H. Shah, R. Amin, M. W. Shaukat, S. M. Raza, D. Y. Suh, and M. J. Piran, "A review of machine learning algorithms for cloud computing security," *Electronics*, vol. 9, no. 9, p. 1379, 2020.

[10] S. Priya and R. Ponmagal, "Network intrusion detection system based security system for cloud services using novel recurrent neural network-autoencoder (nrnn-ae) and genetic," *Advances in Science and Technology*, vol. 124, pp. 729–737, 2023.

[11] A. Singh, Z. Mushtaq, H. A. Abosaq, S. N. F. Mursal, M. Irfan, and G. Nowakowski, "Enhancing ransomware attack detection using transfer learning and deep learning ensemble models on cloud-encrypted data," *Electronics*, vol. 12, no. 18, p. 3899, 2023.

[12] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "Atlas: A sequence-based learning approach for attack investigation," in *30th USENIX security symposium (USENIX security 21)*, pp. 3005–3022, 2021.

[13] M. Boffa, I. Drago, M. Mellia, L. Vassio, D. Giordano, R. Valentim, and Z. B. Houidi, "Logprécis: Unleashing language models for automated malicious log analysis: Précis: A concise summary of essential points, statements, or facts," *Computers & Security*, vol. 141, p. 103805, 2024.

[14] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] L. F. Sikos, "Cybersecurity knowledge graphs," *Knowledge and Information Systems*, vol. 65, no. 9, pp. 3511–3531, 2023.

[16] I. Sarhan and M. Spruit, "Open-cykg: An open cyber threat intelligence knowledge graph," *Knowledge-Based Systems*, vol. 233, p. 107524, 2021.

[17] K. Liu, F. Wang, Z. Ding, S. Liang, Z. Yu, and Y. Zhou, "Recent progress of using knowledge graph for cybersecurity," *Electronics*, vol. 11, no. 15, p. 2287, 2022.

[18] K. Liu, F. Wang, Z. Ding, S. Liang, Z. Yu, and Y. Zhou, "A review of knowledge graph application scenarios in cyber security," 2022.

[19] K. Tsuda and H. Saigo, "Graph classification," *Managing and mining graph data*, pp. 337–363, 2010.

[20] B. Yan, C. Yang, C. Shi, Y. Fang, Q. Li, Y. Ye, and J. Du, "Graph mining for cybersecurity: A survey," *ACM Transactions on Knowledge Discovery from Data*, vol. 18, no. 2, pp. 1–52, 2023.

[21] M. F. Ansari, B. Dash, P. Sharma, and N. Yathiraju, "The impact and limitations of artificial intelligence in cybersecurity: a literature review," *International Journal of Advanced Research in Computer and Communication Engineering*, 2022.

[22] J. P. Bharadiya, "Ai-driven security: how machine learning will shape the future of cybersecurity and web 3.0," *American Journal of Neural Networks and Applications*, vol. 9, no. 1, pp. 1–7, 2023.

[23] J. Ferdous, R. Islam, A. Mahboubi, and M. Z. Islam, "A state-of-the-art review of malware attack trends and defense mechanism.," *IEEE Access*, 2023.

[24] S. Varlioglu, N. Elsayed, Z. ElSayed, and M. Ozer, "The dangerous combo: Fileless malware and cryptojacking," *SoutheastCon 2022*, pp. 125–132, 2022.

[25] B. Kimminich, "Owasp juice shop project." https://owasp.org/www-project-juice-shop/. Accessed: Nov. 22, 2024.

[26] D. Spielman, "Spectral graph theory," *Combinatorial scientific computing*, vol. 18, p. 18, 2012.

[27] X.-w. Chen and J. C. Jeong, "Enhanced recursive feature elimination," in *Sixth international conference on machine learning and applications (ICMLA 2007)*, pp. 429–435, IEEE, 2007.

[28] 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*, pp. 785–794, 2016.

[29] P. I. Frazier, "Bayesian optimization," in *Recent advances in optimization and modeling of contemporary problems*, pp. 255–278, Informs, 2018.

[30] N. de Lara and E. Pineau, "A simple baseline algorithm for graph classification," *arXiv preprint arXiv:1810.09155*, 2018.

[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[32] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv preprint arXiv:2006.07739*, 2020.

[33] W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec, "Strategies for pre-training graph neural networks," *arXiv preprint arXiv:1905.12265*, 2019.

[34] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[35] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?," *Advances in neural information processing systems*, vol. 34, pp. 28877–28888, 2021.

[36] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *International conference on machine learning*, pp. 3835–3845, PMLR, 2019.

[37] M. U. Rehman, H. Ahmadi, and W. U. Hassan, "Flash: A comprehensive approach to intrusion detection via provenance graph representation learning," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 139–139, IEEE Computer Society, 2024.

[38] F. Zhang, Y. Yin, H. Li, Y. Chen, and T. Qu, "Catch causal signals from edges for label imbalance in graph classification," *arXiv preprint arXiv:2501.01707*, 2025.