RANK: AI-Assisted End-to-End Architecture for Detecting Persistent Attacks in Enterprise Networks

Hazem M. Soliman, Dušan Sovilj, Geoff Salmon, Mohan Rao, and Niranjan Mayya

Abstract—Modern government and enterprise networks are the target of sophisticated multi-step attacks called Advanced Persistent Threats (APTs), designed and carried out by expert adversaries. The prolonged nature of APTs results in overwhelming the analyst with an increasingly impractical number of alerts. As a result, the challenge of APT detection is ideal for automation through artificial intelligence (AI). In this paper, we propose the first, up to our knowledge, end-to-end AI-assisted architecture for detecting APTs – RANK. We propose advanced algorithms and solutions for four consecutive sub-problems: 1) alert templating and merging, 2) alert graph construction, 3) alert graph partitioning into incidents, and 4) incident scoring and prioritization. Additionally, we discuss the necessary optimizations and techniques enabling the system to operate in a real-time fashion. We evaluate our architecture against the 2000 DARPA, Mordor, as well as a large number of real-world datasets from enterprise networks. Extensive results are provided showing four orders-of-magnitude reduction in the amount of data to be reviewed, innovative extraction and security-aware scoring of incidents. The extracted incidents can be further used for downstream tasks. In our experiments where we have access to a portion of alert labels, we are able to achieve 87% balanced accuracy.

Index Terms—Advanced persistent threats, enterprise networks, intrusion detection, machine learning, mathematical optimization, security management architecture.

I. INTRODUCTION

A

Advanced Persistent Threats (APTs) form a crucial part of the attack campaigns targeting enterprise networks [1]. A combination of stealth actions, sophisticated exploits and long-term operation marks them as one of the most dangerous and hardest to detect cybersecurity attacks [2]. As a planned multi-step attack, APTs do not target a single vulnerability. Instead, an APT typically starts by gaining access to the network, exploiting several vulnerabilities, and remaining silent for extended periods of time [3]. With data exfiltration as the common end-goal of these attacks, the economic damage is particularly severe due to the risk of losing intellectual properties or sensitive data [4]. Various companies and government agencies have fallen victim to these attacks [5], [6]. In light of their severity and impact, detecting APTs has increasingly become the focus of both academia and industry.

A pattern is emerging in the literature where the detection of APTs is composed of numerous small and simple detectors each focused on a single step in the APT chain. Examples of this methodology are [7], [8]. These small detectors are based on either signature-based intrusion detection systems (IDS), user entity behavior analytics (UEBA) [9], or a combination of both. On one hand, signature-based IDS still act as the bulwark of enterprise detection systems with decades of accumulated security knowledge built into it. On the other hand, IDS and UEBA have well-documented problems such as high false-positive ratio, expensive cost of error, difficulty of explainability for anomaly-based detection, lack of training data and continuously improving intelligent adversaries [10]. APT-based detection methods attempt to solve some of those issues in order to significantly reduce the false-positive ratio. The main premise behind these methods is that an alert is malicious if it is part of a larger group of correlated alerts that together represent a proper APT plan.

A leading effort to study APT attack plans is the ATT&CK matrix developed by the MITRE corporation [11]. The study observed that attacks tend to follow similar patterns, and provided a taxonomy describing both offensive and defensive operations. The matrix describes the tactics, commonly known as the attack phases, the techniques for carrying out a tactic, and the observed instances of such techniques in known attacks. Detection and extraction of ATT&CK-aware security incidents has been one of our major design goals.

In this paper, we propose RANK, the first end-to-end architecture for detecting persistent attacks through incorporating numerous alerts in a meaningful and automated approach. The architecture is composed of several steps:

1) Alert templating and merging: In this step, the aim is to reduce the number of generated alerts by merging together the ones pertaining to the same step of the attack. These merged alerts are referred to as generalized alerts.

2) Alert graph construction: Once the number of alerts is reduced, we establish relationships between generalized alerts according to their MITRE tactics, and form an alert graph representing those relationships.

3) Alert graph partitioning: The partitioning part focuses on finding the smallest subgraphs within the alert graph that correspond to concise incidents. We refer to these as incidents or incident graphs.

4) Incident scoring: Scores are assigned to each incident graph found in previous step, utilizing factor graphs (FG)
and alert scores. For each incident graph, we can compute the likelihood of presence for all MITRE tactics, a useful and important summary. A final single score is predicted for the graph indicating maliciousness and whether the graph should be reviewed by an analyst. For graphs with high scores, the incident graph along with score and summary are delivered to the security analyst for final review. The proposed architecture is also able to integrate analyst feedback as part of incident investigation and threat hunting. This is enabled by the evidence-based queries provided by the scoring factor graph.

To summarize, our contributions in this paper are:

- We present the first architecture for end-to-end detection of persistent attacks, consuming both network and endpoint data, to the best of our knowledge.
- As an extension of our earlier work [12], we explore a novel method for partitioning an alert graph into distinct incidents and how it fits into the overall architecture.
- We demonstrate how the investigation burden on the security analyst can be greatly lessened through the proposed architecture. For instance, our experiments demonstrate that as few as 60 incident graphs may be created from more than 70 thousand alerts. This is a far more succinct, yet equally informative, representation of the data.
- We present an incident graph classification mechanism, achieving up to 96% accuracy and 87% balanced accuracy, significantly reducing the investigation effort needed.
- We discuss a real-time version of the system that can function at the data line speed, along with all necessary techniques and optimizations.
- We discuss various practical aspects of the involved algorithms, such as meaningful clustering of security alerts, calculating correlation between IP addresses, and building all the algorithms with minimal manual tweaking effort due to the lack of labelled datasets.

The remainder of the paper is organized as follows: in Section II we provide a summary of related works in the literature. Section III provides an overview of the proposed architecture and its components. These components are then detailed in the following Sections IV, V, VI, and VII. Experimental results are provided in Section VIII. The streaming architecture is discussed in Section IX. A discussion of the related peripheral problems is provided in Section X. Section XI discusses potential future work, and the paper is concluded in Section XII.

II. RELATED WORK

APT attacks, as mentioned earlier, are planned and executed by sophisticated actors and/or nation states, resulting in high sophistication which pose a significant challenge for detection teams. In [13] the authors carried out a survey of APT attacks focusing on relevant techniques and solutions. The authors organized the attacks into five broad stages representing attack progress irrespective of the end-goal. These five stages are \textit{Reconnaissance, Establish Foothold, Lateral Movement/Stay Undetected, Exfiltration/Impediment and Post-Exfiltration/Post-Impediment}. Stages 1 and 2 are about gaining access to the victim network. In stage 3 the attackers spread through the network in search of data and/or critical resources. Finally, in stages 4 and 5 the attacker could exfiltrate sensitive data and/or hold the victim resources for ransom among other scenarios. A taxonomy of APT attacks based on available data was carried out in [14]. This work has highlighted the wide-spread threat of social networking techniques as a first step for target identification, that Windows systems are widely targeted due to their popularity, HTTP(s) are frequently used for command and control, and that zero-day exploits are observed in more than half of the available attacks.

On the detection front, the sophisticated nature of APT attacks along with their multi-step aspect has led to detection efforts being developed along two directions. The first direction focused on accurate detection of steps shared among most attacks. For example, in [15] two new features, response packet load fluctuation and bad packet rate, were proposed for detecting command and control traffic. The other direction, and the one where most of the current research is carried out, focuses on correlating and integrating multiple detection methods together. This approach has the advantage of building upon a large body of work on anomaly and intrusion detection, and is more robust to differences between various attack scenarios. A framework is proposed in [16] where eight detection methods are first employed to detected various steps in APTs. The outcomes of these detectors are then correlated together into APT attack scenarios. Finally, a threat prediction step tries to identify potential future development of the attack based on past occurrences. The main disadvantage in this work is its focus on the proposed eight statistical methods, ignoring a plethora of signature-based methods, among others, such as SNORT. These methods still form the bulk of intrusion detection systems used today. Another line of work on alert correlation leverages Fisher’s method for \textit{p}-value combination [7]. This approach for detecting APTs assigns a joint score for multiple simple detectors, and the resulting score signals not only the likelihood of individual behaviors, but also the likelihood of all detected behaviors taking place together in normal operating circumstances. A special class of correlation-based approaches focused on utilizing graph approaches, typically through building attack graphs. Attack graphs are a powerful representation of APTs [17]. Besides their visually attractive properties, the topology of a directed graph is a representation of the attack plan progression. In such graphs, each vertex represents an individual alert, which might come from a variety of security solutions, and each edge represents the progression between two alerts [8]. With their vertices comprised of security alerts, attack graphs are also sometimes called alert graphs, which is the terminology we use in this work. A provenance based approach for building alert graphs is proposed in [18]. Data provenance refers to representing operating system execution as a directed acyclic graph of interaction between system processes and objects such as files or sockets. This approach, while powerful, requires access to system-level data that are hardly available.

In a series of works, [19], [20], [21], the authors have developed a framework for building attack graphs from intrusion alerts. The framework starts with grouping alerts together
into attack episodes, then grouping attack episodes into attack sequences. A final probabilistic step tries to highlight severe, albeit infrequent, episodes and the paths leading to them. This line of work has a lot of similarity to ours. However, our framework provides few advantages:

- Subject matter expertise is well-integrated into our framework, especially during the correlation and scoring steps, as will be shown for example in V, X-D and Appendix C, available online.
- We provide a comprehensive graph partitioning framework tailored to the security applications, discussed in Section VI.
- We propose a comprehensive scoring step, integrating security knowledge and interactively updated with investigative input from the analyst, discussed in VII and Appendix C, available online.

Despite its advantageous features, an alert graph containing all alerts is usually overwhelmingly large due to the high false-positive rate of the individual alert detectors. This problem can be addressed in two ways. The first approach aggregates alerts corresponding to the same event together into a Hyper/Generalized alert [22], while the second approach works by partitioning the alert graph into smaller sub-graphs. Both approaches aim at producing smaller alert graphs in order to reduce the investigation burden on the security analyst. An example of the second approach is given in [23], where a community-detection approach is proposed with each resulting community corresponding to a single attack step. In [24], alert prerequisites and consequences are defined for each alert to limit the number of edges. However, this approach requires heavy manual work, and can not handle cases when multiple attacks are carried out simultaneously. Finally, in [8] a separate graph is defined for each single asset. This approach results in manageable graphs, but it ignores lateral movement between assets, a common tactic in APT attacks.

It is worth emphasizing again the difference between the two main approaches in reducing the size of the graph. The first approach tries to combine together alerts corresponding to the same step in an APT into a single alert, equivalent to a node in the graph. The second approach aims at collecting all steps for a single coherent APT into a separate subgraph representing the incident. Identifying the difference between the two approaches and having them work together in the same architecture is an important contribution of this work.

Intrusion detection is also still an active area of research, especially in newer settings such as Internet of Things (IoT). Generative adversarial networks (GANs) were utilized in [25] to solve the class imbalance problem prevalent in the security domain. In [26], the IoT security problem is studied, with the focus being identifying spam and phishing emails in the Industry 4.0 setting. In [27] with a focus on the android platform, system calls along with byte-level image representation were utilized for efficient malware detection. A data privacy preserving, federated learning approach for intrusion detection in IoT settings was proposed in [28]. We would like to re-emphasize that the our proposed architecture is detection-agnostic, and is not focused on building a new detection as much as correlating different detections together into a single cohesive attack story.

Deep learning approaches are also becoming increasingly popular in detecting APT attacks. An approach based on event sequence embedding and labelling has been proposed in [29]. Similarly, an approach based on Graph Neural Networks (GNNs) was proposed in [30]. Both approaches require access to system call data, and deep learning models in general need extensive labelled datasets for training. These are two factors that may impede the widespread adoption of deep learning approaches. On a different note, game theory has been used to model the APT response problem in [31], providing an interesting approach in understanding attacker-defender interaction.

In summary, our paper improves upon the existing literature through addressing the following gaps:

- Most of the existing works focus on individual steps, being new detections or small scale correlation of alerts. There is a lack of works on comprehensive end-to-end solutions, which we address with this paper.
- Most of the literature is a large deviation from the current security best practices, through focusing solely on data-driven approaches, and completely dropping the plethora of rule- and signature-driven security systems. These systems are entrenched in the existing security deployments for good reasons, since they capture decades of security subject matter expertise. Any novel architecture should leverage and integrate with these systems. Our proposed system is compatible with legacy IDS and integrates subject matter expertise throughout its different steps.
- Most of the existing works utilize out of the box box solutions for solving problems in the security domain, such as graph partitioning in particular. Our work shows how these methods can be adapted for the security context by rebuilding them from first principles.
- The literature has focused mainly on batch processing utilizing open source, albeit mostly unrealistic, datasets. We are the first to discuss the real-time aspect of the problem at hand, as well as provide results from testing on the largest real world dataset to date.

III. SYSTEM DESCRIPTION

The goal of RANK is to process tens/hundreds of thousands of alerts produced by various IDS and Ueba systems into a much smaller set of proper security incidents representing truly malicious behavior. The general architecture is shown in Fig. 1. First, we identify the inputs and outputs of the system:

- **Input**: Alerts coming from sources such as Suricata\(^1\) and Snort\(^2\), Active Directory\(^3\), Office 365\(^4\), anomaly detectors and custom user-defined rules.
- **Output**: A number of tactic-aware security incidents, where each incident is represented as a directed graph of related

---

1. [https://suricata-ids.org/](https://suricata-ids.org/)
2. [https://www.snort.org/](https://www.snort.org/)
3. [https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/active-directory-domain-services](https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/active-directory-domain-services)
4. [https://docs.microsoft.com/en-us/microsoft-365](https://docs.microsoft.com/en-us/microsoft-365)
Fig. 1. Incident extraction architecture, with corresponding section labels for each step. The architectures takes as input alerts coming from a variety of sources such as IDS and custom rules. These alerts are first analyzed for common patterns and matching alerts are merged together into generalized alerts. An alert graph is built from these generalized alerts and then partitioned into independent incident sub-graphs. Resulting incident graphs are scored through their corresponding factor graphs and presented to the analyst for final investigation.

generalized alerts. Each incident also includes a maliciousness score, the set of MITRE tactics present as well as their respective scores. Throughout the paper, we use the terms “incident” and “incident graph” interchangeably.

The mapping between those inputs and outputs is accomplished through the following four stages:

A. Alert Templating and Merging

The first step in the architecture is to merge together alerts representing the same attack step. A generalized alert is the result of aggregation over that particular group of alerts. For example, a port scan alert would include alerts corresponding to all ports being scanned by a single adversary. The output of this phase is a set of generalized alerts.

B. Alert Correlation and Alert Graph

Once similar alerts have been merged together, our next objective is to establish correlation relationships between generalized alerts. The outcome of this step is what we refer to as an alert graph. The correlation between alerts is measured using the correlation of their attributes, the time interval in-between, and how each alert fits into the MITRE ATT&CK chain.

C. Alert Graph Partitioning

The previous stage still provides large graphs that do not necessarily lead to easily identifiable incidents. To address this issue, the alert graph is partitioned into smaller sub-graphs – incident graphs – each representing a distinct, potential APT attack.

D. Incident Scoring

Finally, once an incident graph has been built, our final goal is to assign a score, or set of scores, to each incident. In particular, we assign a score for each MITRE tactic present in the incident. This is achieved through a Factor Graph (FG), a type of probabilistic graphical models [32]. The FG considers the scores of all alerts in the incident, their sequence and their MITRE mapping, and produces a score for each tactic present. A subsequent step processes the tactics’ scores, along with other graph and alert features as will be discussed later, and produces a single predictive score for each graph whether it should be presented for manual investigation. The predictive output has the potential to reduce cognitive load on the analyst and improve efficacy of investigations that are warranted.

In the next sections, we provide details of each of the stages above.

IV. ALERT TEMPLATING AND MERGING

The first step in the RANK architecture is to merge together alerts representing the same attack step. In this section, we follow the methodology and notation of Julisch [22].

A. Motivation

Whether the alerts are coming from IDS signature, behavioral anomalies or custom rules, the end result is an overwhelming flood of alerts [10]. However, it has been observed that there is a significant amount of redundancy in these alerts. Hence, it is our goal for the first step in the detection architecture to remove such redundancy and merge together all alerts triggered by the same underlying cause. For example, consider the Suricata alert GPL RPC sadmind query with root credentials. This alert can be triggered when an
external adversary is scanning internal hosts for the Sadmind exploit. Even when only a single external IP is scanning, the IDS may produce a separate alert for each internal host scanned. A much more concise but similarly informative alert would indicate this behavior occurring between the external adversary and all affected internal hosts. This process of merging together alerts produced by the same cause is called alert generalization [22], and is the focus of this section as explained next. While manual rules can be used to group alerts together, a data-driven approach as discussed here is more scalable and efficient.

B. Alert Structure

Mathematically, an alert, \( a \in A \) with \( A \) being the set of alerts, is defined as a tuple of the Cartesian product

\[
a \in \text{dom}B_1^a \times \text{dom}B_2^a \times \ldots \times \text{dom}B_{N_a}^a
\]

where \( \{B_1^a, B_2^a, \ldots, B_{N_a}^a\} \) is the set of alert attributes, \( N_a \) is the number of attributes in alert \( a \) and \( \text{dom} \) is the domain, representing the set of values an attribute can take. The value an attribute \( B_i^a \) takes in alert \( a \) is denoted as \( b_i^a \). Examples of alert attributes are sourceIP, destinationIP, sourcePort, destinationPort, username.

Each alert has a source, \( s^a \in S \), with \( S \) being the set of all alerts’ sources. The source of an alert refers to the rule or analytic generating it. For example, for an IDS alert the source is the rule ID, while for anomaly alerts the source is the name of Ueba algorithm. The set of alerts coming from a source \( s \) is denoted as \( \mathcal{A}_s \), and the set of attributes in all alerts in \( \mathcal{A}_s \) is denoted as \( \mathcal{B}_s \). All alerts coming from the same source have the same set of attributes.

Each alert also has a score \( p^a \in [0, 1] \). An alert score is a measure of its severity. For example, a Suricata and Snort IDS rule has a severity entry, which can be translated into a numerical score. For Ueba-style alerts, the score is usually derived from the \( p \)-value of the statistical test employed, as in [7].

Finally, each alert is also indicative of a set of MITRE tactics, \( \mathcal{M}^a \), and is affecting a set of internal assets/IPs \( \mathcal{L}^a \) and accounts/usernames \( \mathcal{U}^a \). The corresponding MITRE tactics for each alert are manually assigned based on the alert source.

C. Attribute Hierarchy Trees

Generalizing an attribute \( B_i^a \) refers to the concept of extending its domain \( \text{dom}B_i^a \) with a new set of values, \( \{t_1, t_2, \ldots, t_{N_i}\} \), each representing a distinct subset of \( \text{dom}B_i^a \), resulting in the new domain \( \text{dom}B_i^a = \text{dom}B_i^a \cup \{t_1, t_2, \ldots, t_{N_i}\} \). To achieve this goal, for each attribute \( B_i^a \) we define an associated hierarchy tree \( T_{B_i^a} \) representing the splitting of its domain. In this case, the number of new values \( N_i \) is the number of non-leaf nodes in the corresponding hierarchy tree \( T_{B_i^a} \).

An attribute \( B_i^a \) is generalized by replacing the value it takes in an alert, \( b_i^a \), with the parent value in its hierarchy tree. In particular, the new attribute value becomes \( b_i^a = \text{parent of } b_i^a \) in \( T_{B_i^a} \). This new value represents the subset of the attribute’s domain containing the original value. We show examples of hierarchy trees for IP and Port attributes in Figs. 2 and 3, respectively.

D. Generalized Alerts

A generalized alert \( v \in V \) is a tuple in the new Cartesian product

\[
v \in \text{Dom}B_1^v \times \ldots \times \text{Dom}B_{N_v}^v \times \text{dom}B_{gl_1}^v \times \ldots \times \text{dom}B_{gl_{N_v}}^v
\]

where \( gl_i \) is the number of attributes to generalize per alert source and \( N_v \) is the number of attributes in the generalized alert \( v \). Note that generalizing an alert does not change its number of attributes. In other words, an alert is generalized when at least one of its attributes is also generalized.

It is worth emphasizing that the alert source \( s^a \) plays a crucial role in the generalization process. First, only alerts coming from the same source can be merged together. This is done in order to keep the security context of the generalized alert meaningful. For example, if an alert is an indication of port scan for a single target machine, its generalized version will indicate a port scan for multiple target machines. The second significance of the alert source is that the parameter \( gl_i \) is indexed per source not per alert. These values are manually chosen, and it is easier and more meaningful to choose the number of attributes to generalize per source. In our experiments, we have selected \( gl = 2 \) for IDS alerts and \( gl = 1 \) for Ueba alerts.

E. Template Extraction and Merging Algorithm

The high-level steps of the template extraction and alert merging are as follows:

- Each alert has at least two attributes, which can include sourceIP, destinationIP, sourcePort,
Algorithm 1: Template Extraction and Alert Merging Algorithm.

Input: A set of alerts \( \mathcal{A} \)

Output: A set of merged and generalized alerts \( \mathcal{V} \)

Initialize
\[ \mathcal{V} = \mathcal{A} \]

for \( s \in S \) do
\[ j = 0 \]

while \( j < g \) do
\[ l = SelectAttribute(\mathcal{V}_s, \mathcal{B}_s) \]
for \( v \in \mathcal{V}_s \) do
\[
// \text{Generalize selected attribute}
\]
\[ b^*_v = \text{parent of } b^*_v \text{ in } \mathcal{T}_i \]
end
while identical alerts \( v, v' \in \mathcal{V} \) exist do
\[
// \text{Merge matching alerts}
\]
\[ p^v = \max(p^v, p^{v'}) \]
\[ \text{delete } v' \text{ from } \mathcal{V} \]
end
\[ j += 1 \]
end

Function SelectAttribute(\( \mathcal{V}_s, \mathcal{B}_s \)) is

for \( B_s \in \mathcal{B}_s \) do
\[
\text{maxCount}^t_{B_s} = \max\{\text{count}(x)|x \in \{b^*_v | v \in \mathcal{V}_s\}\}
\]
end
\[ l = \arg\min_{B_s \in \mathcal{B}_s} \text{maxCount}^t_{B_s} \]
return \( l \)
end

destinationPort, username and one or more process related attributes.

- For every alert source \( s \in S \) and every attribute \( B_s \in \mathcal{B}_s \), calculate the number of occurrences of the most common value \( \text{maxCount}^t_{B_s} \).
- Sort attributes in ascending order according to their \( \text{maxCount}^t_{B_s} \), ties are broken randomly.
- Generalize the sorted attributes each according to its hierarchy tree until the pre-configured limit on the number of generalizable attributes. The severity score of the merged alert is the maximum among all the severity scores of its contributing individual alerts.

In summary, the attribute to generalize at each step is the one whose most frequent value has occurred least number of times across all remaining attributes under consideration. The algorithm itself is shown in Algorithm 1. The output of this stage is a set of generalized alerts. For example, if we have two alerts as GPL RPC sadmind query with root credentials 10.14.14.1 -> 10.0.0.1 and GPL RPC sadmind query with root credentials 10.14.14.1 -> 10.0.0.2 as input to the stage, they will be aggregated together into a generalized alert of the form GPL RPC sadmind query with root credentials 10.14.14.1 -> PRIVATE-IP.

V. ALERT CORRELATION AND ALERT GRAPHS

The previous step reduces the number of alerts to investigate by approximately two orders of magnitude. Our next step is to build an alert graph in which each node represents a generalized alert. In particular, we denote the graph \( \mathcal{G} = \{\mathcal{V}, \mathcal{E}\} \) as the alert graph, where the set of nodes \( \mathcal{V} \) is the set of generalized alerts, and \( \mathcal{E} \) being the set of directed edges connecting alerts together.

The main question in this section is when to add edges between the nodes, and what are the edge weights. In essence, the question is about measuring correlation between alerts. We measure correlation using two metrics: 1) correlation between the alert attributes, and 2) where each alert fits in the MITRE ATT&CK sequence. For example, if the same external IP is targeting one or more internal hosts, with each attempted exploit triggering a different alert, then attribute correlation between said alerts would be high. Meanwhile, the MITRE ATT&CK correlation between two alerts will increase if the second alert’s tactic follows naturally from that of the first alert. For example, if the first alert represents lateral movement and the second alert represents data exfiltration, then the correlation is high since this transition is in line with MITRE ATT&CK. On the other hand, the correlation will be low if the first alert represents lateral movement and the second alert represents initial access.

In summary, each alert is associated with one or more MITRE tactics, and the MITRE ATT&CK correlation weight represents how likely it is for an attack to transition from the tactics associated with the first alert to the tactics associated with the second.

After many experiments, we have found the following expression for the alert correlation measure, \( C(\cdot, \cdot) : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}_{\geq 0} \), to yield the best results:

\[
C(v, v') = \max_{t \in M^v \cap t' \in M^{v'}} T_{KC}(t, t') \ast \max_{ip \in L^v \cap ip' \in L^{v'}} C_{IP}(ip, ip')
\]

\[
\ast \max_{u \in U^v \cap u' \in U^{v'}} C_{UN}(u, u')
\]

where \( C_{IP}(\cdot, \cdot) \) is an IP correlation measure, defined to be one for matching IPs and zero otherwise

\[
C_{IP}(ip, ip') = \delta(ip, ip')
\]

where \( \delta \) is the Kronecker delta function. Similarly the \( C_{UN}(\cdot, \cdot) \) is the username correlation measure, defined as

\[
C_{UN}(u, u') = \delta(u, u')
\]

\( L^v \) is the set of IPs in alert \( v \), \( U^v \) is the set of usernames, and \( M^v \) is the set of tactics. \( T_{KC}(\cdot, \cdot) \) is a MITRE tactic transition matrix defined according to Table I.

We have manually assigned the transition weights and mappings, taking care to limit the number of parameters needed. This is due to the lack of properly labelled datasets containing real attacks covering many facets of the MITRE ATT&CK matrix. The matrix is designed around two main concepts: 1) Attacks tend to move forward in the kill-chain, 2) Attacks progress in small steps. For example an attack is more likely to go from Execution to Privilege Escalation than the opposite direction. Along the same lines, an attack is more likely to move from the
Execution stage to Privilege Escalation than it is to go directly to Impact. We also use low weights for transitioning to the same tactic in order to avoid loops in the alert graph. This criteria can be seen in the transition matrix, Table I, with diagonal weights being small, and weights in the k-diagonals starting large and getting smaller as k increases.

An edge $v \rightarrow v'$ is added to the graph $G$ between the two alerts $(v, v')$ if their correlation measure $C(v, v')$ is greater than a pre-defined threshold. This correlation measure is also used as the edge weight. The threshold value has to be manually tuned and we have found a value of 0.4 to give us the best results.

The output of this stage is an alert graph $G = (\mathcal{V}, \mathcal{E})$, where nodes are generalized alerts and edges represent the progress of the attack between these alerts.

### VI. ALERT GRAPH PARTITIONING

With the alert graph built, the next step is to partition said graph into smaller subgraphs each representing a distinct security incident. In its general form, the graph partitioning problem is concerned with splitting the vertex set $\mathcal{V}$ of the graph $G$ into a collection of non-empty subsets. The standard objective in such problems is to minimize the total weight of the edges connecting any two subsets [33].

We utilize two approaches for graph partitioning: the ego-splitting overlapping community detection [34], and our proposed optimization approach [12]. The motivation behind proposing a novel approach is that community detection algorithms focus mainly on the topological properties of graphs, and fail to incorporate a security context representing the validity of the resulting sub-graphs as proper attack plans. Moreover, existing graph partitioning approaches act as a black-box and are hard to extend or improve given the security analyst feedback.

In our previous work [12], we have provided the first rigorous formulation of the attack partitioning problem enabling the extraction of security-aware incident graphs. Recognizing that community detection algorithms approximate the graph partitioning problem [35], we leveraged the power of convex optimization to incorporate security context into the problem. In particular, our objectives when partitioning a graph are:

- To guarantee a proper incident, we would like to maximize the number of attack phases in each partition.
- To avoid impractical incidents, we would like to minimize the number of assets involved in a partition.
- To provide the analyst with an easy-to-visualize graph, we would like to have an upper limit on the number of alerts included in each partition.
- To balance how many incidents involve a single alert, our formulation allows having a reasonable and flexible limit on the number of partitions an alert might appear in. For example, limiting an alert to a single incident is overly optimistic while sharing it across many incidents is overly pessimistic. This is analogous to the concept of overlapping communities [34].
- Perhaps most importantly, to allow more flexibility in addressing security analysts feedback, we would like to have a more tractable formulation of the problem, unlike black-box community detection algorithms.

In [12, Eq. (11)], we showed how the overall graph partitioning optimization problem can be formulated as a mixed integer

| TACTIC                  | Initial Access | Execution | Persistence | Privilege Escalation | Defense Evasion | Credential Access | Discovery | Lateral Movement | Collection | Command and Control | Exfiltration | Impact |
|-------------------------|----------------|-----------|-------------|----------------------|----------------|--------------------|-----------|------------------|------------|---------------------|-------------|--------|
| Initial Access          | 0.1            | 0.8       | 0.8         | 0.8                  | 0.8            | 0.8                | 0.5       | 0.5              | 0.3        | 0.3                 | 0.3         | 0.3    |
| Execution               | 0.5            | 0.1       | 0.7         | 0.7                  | 0.7            | 0.7                | 0.8       | 0.8              | 0.5        | 0.5                 | 0.5         | 0.5    |
| Persistence             | 0.5            | 0.7       | 0.1         | 0.7                  | 0.7            | 0.7                | 0.8       | 0.8              | 0.5        | 0.5                 | 0.5         | 0.5    |
| Privilege Escalation    | 0.5            | 0.7       | 0.7         | 0.1                  | 0.7            | 0.7                | 0.8       | 0.8              | 0.5        | 0.5                 | 0.5         | 0.5    |
| Defense Evasion         | 0.5            | 0.7       | 0.7         | 0.1                  | 0.7            | 0.7                | 0.8       | 0.8              | 0.5        | 0.5                 | 0.5         | 0.5    |
| Credential Access       | 0.3            | 0.5       | 0.5         | 0.5                  | 0.5            | 0.5                | 0.8       | 0.8              | 0.5        | 0.5                 | 0.5         | 0.5    |
| Discovery               | 0.3            | 0.5       | 0.5         | 0.5                  | 0.5            | 0.7                | 0.8       | 0.8              | 0.8        | 0.8                 | 0.8         | 0.8    |
| Lateral Movement        | 0.3            | 0.5       | 0.5         | 0.5                  | 0.5            | 0.7                | 0.8       | 0.8              | 0.8        | 0.8                 | 0.8         | 0.8    |
| Collection              | 0.3            | 0.3       | 0.3         | 0.3                  | 0.3            | 0.5                | 0.7       | 0.7              | 0.7        | 0.7                 | 0.7         | 0.7    |
| Command and Control     | 0.3            | 0.3       | 0.3         | 0.3                  | 0.3            | 0.3                | 0.5       | 0.5              | 0.7        | 0.7                 | 0.7         | 0.7    |
| Exfiltration            | 0.3            | 0.3       | 0.3         | 0.3                  | 0.3            | 0.3                | 0.5       | 0.5              | 0.7        | 0.7                 | 0.7         | 0.7    |
| Impact                  | 0.3            | 0.3       | 0.3         | 0.3                  | 0.3            | 0.3                | 0.5       | 0.5              | 0.7        | 0.7                 | 0.7         | 0.7    |
linear programming (MILP) problem, which can be solved using commercial and open-source solvers such as CBC.\textsuperscript{5} As a multi-objective optimization problem, the goal is to minimize the graph cut between any two partitions, maximize the number of kill chain stages included in a partition, minimize the number of assets and limit the size of the partition for easy understanding by the analyst. Solving the problem gives us the optimum alert-to-partition assignments, where each partition now represents a distinct security incident.

### VII. INCIDENT SCORING

With the security incidents now extracted from the bigger alert graph, the final step is to evaluate whether the sequence of alerts present in each incident graph represents a proper attack plan. For example, a sequence of Initial Access $\rightarrow$ Execution $\rightarrow$ Lateral Movement is more valid as an attack plan compared to Lateral Movement $\rightarrow$ Execution $\rightarrow$ Initial Access, since the first follows more closely the order in MITRE ATT&CK. Even though the way we calculate correlation between alerts when building the alert graph tries to incorporate these relations, depending on the pair-wise correlation only is not enough. On one hand, pair-wise correlation does not look at the graph as a whole, on the other hand a single alert can be mapped to multiple MITRE tactics. Our solution here leverages factor graphs\textsuperscript{32}, a type of probabilistic graphical model\textsuperscript{36}, to assign a score for each MITRE tactic present in the incident. The tactic scores are based on the scores of the individual alerts and their sequence in the incident graph.

### A. Overview of Factor Graphs

A factor graph is a bipartite graph used to factor a global function into a product of smaller local functions\textsuperscript{32}. The graph has two types of nodes: variable nodes that represent the arguments of the function being calculated, and factor nodes defining the relationships between the variable nodes. There are three main problems under study in any application of FGs: structure, inference and learning\textsuperscript{36}. Structure is about defining the variable and factor nodes, as well as the overall topology of the graph. The inference step is concerned with designing the inference algorithm to calculate the marginal probabilities of the variable nodes. The choice of an inference algorithm is a trade-off between accuracy and feasibility, since in general the inference complexity in FG is NP-hard\textsuperscript{36}. Finally, the learning step is about the choice of parameters in the graph, such as the factor functions and the prior distribution of the variable nodes. FGs have achieved wide-success in a variety of applications such as decoding of linear codes and pattern recognition\textsuperscript{36}.

### B. Building a Factor Graph From an Incident Graph

Given an incident graph, i.e., a set of generalized alerts and their correlations, the question is how can we assign a score to the different MITRE tactics present in the incident. These scores should be calculated based not on the individual alerts, rather, on the incident graph as a whole. This question can be seen as that of calculating a joint probability distribution for all alerts and tactics present in the incident. This joint probability distribution is the global function that the FG is calculating. There are many challenges here:

- The number of alerts and tactics is different for each incident, the score calculation capability should be dynamic.
- The exponential growth in the number of states. For a binary state space of alerts $\mathcal{V}$ and tactics $\mathcal{M}$, with each state either active or inactive, the total number of states is $2^{(|\mathcal{V}|+|\mathcal{M}|)}$.
- With the lack of labelled datasets, how can we minimize the number of parameters in the FG?

In light of these challenges, we build the FG as follows:

- **Variables:** every MITRE tactic present in the incident is mapped to a variable node.
- **Factors:** we have two classes of factor nodes
  - **Alert-Factors:** we also have two types of alert-factors
    - Single tactic alert: if an alert $v$ is mapped to a single tactic, then the alert becomes a one-variable factor with $f(Active) = p^v$ where $f(.)$ is the factor function and $p$ is the alert score, as shown in Table II. This factor node is connected to the corresponding variable node representing the tactic.
    - Multi-tactic alert: if an alert $v$ is mapped to multiple tactics, then the alert becomes a multi-variable factor with the same number of variables as the number of mapped tactics. The factor table entry for every possible combination of the variables’ values is equal to the minimum between either the alert score or the maximum transition matrix entry between every possible pair of active tactics. The factor table is shown in Table III for the case when the alert is mapped to two tactics, i.e., $\mathcal{M}^v = \{t_A, t_B\}$.
  - **Transition-Factors:** for every pair of tactics present in the incident, we define a new factor connecting the two. The factor function is equal to the transition matrix entry depending on the temporal order of the alerts leading to each tactic. This is shown in Table IV, where

| Tactic State | Factor Value |
|--------------|--------------|
| Active       | $p^v$        |
| Inactive     | $1 - p^v$    |

| Tactic A State | Tactic B State | Factor Value |
|----------------|----------------|--------------|
| Active         | Active         | $\min(p^v, \max(T_{KC}(t_A, t_B), T_{KC}(t_B, t_A)))$ |
| Inactive       | Active         | $p^v$        |
| Active         | Inactive       | $p^v$        |
| Inactive       | Inactive       | $1 - p^v$    |

\textsuperscript{5}https://github.com/coin-or/Cbc
Finally, the resultant graphs can be used in downstream tasks such as classifying whether a graph should be ticketed. Since the resolution can be either a ticket or a false positive, ticket label indicates a positive case, where a security analyst has identified a security compromise. Using this dataset, we trained a graph classifier to predict whether a graph should be ticketed. Since the resolutions are for individual alerts and not incident graphs, we need a way to construct a graph label from its node labels. We opted for a worst-case approach, where a graph’s label is considered to be a ticket if any of its nodes were ticketed, and false-positive otherwise. It is also worth noting that with the dataset being for individual alerts, it cannot be used to learn inter-alert parameters such as the transition matrix in the previous sections.

The features used in building the graph classifier are detailed in Appendix B, available online. The features include the final scores of the MITRE tactics calculated by the factor graph, counts of alert types present in the incident graph, as well as graph features such as numbers of nodes and edges. Alert types are a broad categorisation of various alerts that can be tied to a particular source $s$. For example, snort-high encapsulates all alerts coming from Snort IDS with high severity. We have trained multiple standard models including support vector machines (SVM), random forest (RF), multi-layer perceptron (MLP), as well as an ensemble model of the aforementioned ones.

### VIII. Experimental Results

The lack of good APT datasets, as well as the well-documented challenges of applying algorithmic and machine-learning approaches in cybersecurity [10], makes evaluating any work particularly challenging. In evaluating this work, we are looking at three metrics:

- Reduction in the amount of data reviewed by the analyst.
- The quality of the resultant graphs and the meaningfulness of alert correlations.
- Finally, the resultant graphs can be used in downstream tasks such as classifying whether a graph should be ticketed, and this is measured in terms of classification accuracy.

For evaluation datasets, we use the following:

1. **DARPA** – The publicly available 2000 DARPA intrusion detection dataset [37].
2. **Enterprise Network** – A private dataset from a medium enterprise network. This network has around 50 employees using primarily Windows machines, as well as a few internal servers.
3. **Enterprise Aggregate** – Another private dataset consisting of large collection of security events and alerts from several thousand enterprise networks of varying sizes.

The datasets are used to measure the evaluation metrics according to:

- **Data Reduction**: DARPA and Enterprise Network.
- **Graphs and Correlation Quality**: DARPA and Enterprise Network.
- **Graph classification downstream task**: Enterprise Aggregate

Although DARPA is very old data set in terms of attack patterns, it still represents a good benchmark for our purposes here since it contains a well documented attack vector. Throughout our experiments, we used Suricata with its open rule-set as our IDS, and we also ingest active directory, Office-365 events, Sysmon, Windows along with UEBA anomalies.
A. DARPA 2000

DARPA 2000 is an intrusion detection dataset with the attacker trying to gain access to victim machines in order to launch Distributed Denial of Service (DDoS) attack. We focus on the first scenario here, and the data has around 7000 alerts.

1) Data Reduction: a) Templates: First, we study the effect of alert templating and merging on reducing the number of alerts for the later steps. In Table V, we show three examples of templates extracted from the DARPA dataset. The first column is the Suricata rule signature. The attributes count column means that the most common sourcePort value was seen only 5 times, while the most common sourceIP value was seen 70 times for the same data. This is expected for such an alert as it is likely that a single external adversary, sourceIP, is trying to gain root access in any of the enterprise hosts it can find. Hence, the template is chosen for a single sourceIP and single destinationPort, while the destinationIP and sourcePort are up-levelled according to their hierarchy tree, as seen in the second column. Note that the resulting generalized alert still represents a specific security behavior, making it easy to investigate further. This is in contrast to generic clustering approaches which can aggregate different alerts together resulting in loss of context.

b) Overall Reduction: As discussed multiple times, one major benefit of the proposed architecture is reducing the amount of investigations to be done by the analyst without extensive data losses, all while automating as many parts of the process as possible and giving better representations of the data for more efficient investigation. In terms of reduction, we have achieved the following with the DARPA dataset

| Signature                      | Generalized Form                                                                 | Attributes Count      | Alert Examples |
|-------------------------------|----------------------------------------------------------------------------------|-----------------------|---------------|
| GPL RPC sadmind query         | {dstIP: private-IP, dstPort: <port number>, arcp: <IP address>, srcPort: Non private - Port} | (srcPort, 5), (dstIP, 30), (dstIP, 50), (arcIP, 70) | 172.16.115.20 32773 202.77.162.213 60251 |
| with root credentials        |                                                                                   |                       | 172.16.112.10 32774 202.77.162.213 60542 |
|                               |                                                                                   |                       | 172.16.1122.50 32773 202.77.162.213 60569 |
| GPL Telnet Bad Login          | {srcIP: <IP address>, dstPort: Non private - port, arcp: private-IP, srcPort: <Port number>} | (srcPort, 5), (srcIP, 10), (srcIP, 20), (arcPort, 30) | 195 115.218.108 43886 172.16.113.50 23 |
|                               |                                                                                   |                       | 202.77.162.213 46956 172.16.115.20 23 |
|                               |                                                                                   |                       | 202.77.162.213 46986 172.16.112.10 23 |
| GPL RPC portmap sadmind request UDP | {dstIP: private-IP, dstPort: <port number>, arcp: <IP address>, srcPort: Non private - Port} | (dstIP, 60), (dstPort, 60), (arcIP, 450), (arcPort, 450) | 172.16.115.20 111 202.77.162.213 54790 |
|                               |                                                                                   |                       | 172.16.115.87 111 202.77.162.213 54793 |
|                               |                                                                                   |                       | 172.16.112.10 111 202.77.162.213 60540 |

and for the enterprise network, for one month of data

77000 Alerts → 1100 Generalized Alerts → 60 Incidents

We observe around two orders of magnitude reduction through templating and another one order of magnitude reduction through graph partitioning. The end result is a considerably more concise representation of the alerts, significantly easing the investigation burden on the analyst.

2) Graph Quality: a) Graph partitioning: Here we study the performance of the ego-splitting framework from [34] and our proposed approach (Section VI). The ego-splitting framework utilizes non-overlapping algorithms underneath, for which we use Louwain and Modularity from the NetworkX package, as these two have given the best results in practice. The evaluation metric is first and foremost a manual inspection of the resulting graphs, as well as statistics on the quality of those graphs, such as the average number of alerts.

The parameters for our approach are $\gamma_0 = 1.0, \gamma_1 = 0.5, \gamma_2 = 0.5, MaxMem = 2$ and $MaxCard = 20$. For the experiment with the DARPA dataset we solved the integer version of problem [12, Eq. (11)] while a relaxed version without the binary constraint is used for the experiment with the enterprise network dataset.

The overall attack graph in the DARPA scenario is shown in Fig. 4. The goal of the attack is to exploit the Solaris sadmind vulnerability on eligible hosts, install a trojan and launch a DDoS attack against a remote server. The attack progression and corresponding Suricata alerts are summarized in Table VI.

The incident extracted through our approach is shown in Fig. 6, while the ego-splitting community detection is shown in Fig. 5. Examining the outcomes of both approaches, we see that our method is able to extract a single path containing all attack phases which is a desirable result for the security analyst. On the other hand, the ego-splitting method contains two potential pathways from the initial to the final stage of the attack, requiring additional effort to identify the correct one.

B. Enterprise Network

This network has around 50 employees using primarily windows machines, as well as a few internal servers. Suricata alerts were collected from this network over a period of three weeks.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
Fig. 4. DARPA overall alert graph prior to partitioning. The contents of each node are the source ip(s) (top), destination ip(s) (middle) and signature of the Suricata rule triggering the alert (bottom). Thick red nodes indicate the real steps of the attack. Some of the edges are omitted for visual clarity.

TABLE VI
DARPA ATTACK PROGRESSION AND IDS ALERTS FOR THE CASE SHOWN IN FIG. 4

| Attack Step | IDS Alert(s) |
|-------------|--------------|
| An IP sweep of the enterprise hosts. | The IP scan is not reported. |
| Probing the IPs from step 1 to look for sadmind hosts. | Detected by rule "GPL RPC portmap sadmind request UDP". |
| Exploiting the Solaris Sadmind vulnerability (CVE-1999-0977). | Detected by rule "GPL RPC sadmind query with root credentials attempt UDP". |
| Installation of the DDoS trojan on exploited machines. | Detected by rule "GPL TELNET Bad Login", followed by "GPL RPC sadmind query with root credentials attempt UDP" and the actual file download in "ET POLICY Executable and linking format (ELF) file download". |
| Launching the DDoS attack. | Detected but not connected to the other alerts due to the trojans using source IP spoofing. |

Fig. 5. DARPA incident graph through community detection.
C. Data Reduction

a) Templates: Another way to consider data reduction is to look into the growth of the number of generalized versus increasing the analysis period, and consequently the number of alerts. This expansion of analysis time is necessary if we aim to discover long-term APTs. In Fig. 7, we show the number of generalized alerts versus the number of individual alerts from the enterprise network we study. These alerts are collected from a period of three weeks. We can see a pattern where the number of generalized alerts is almost two orders of magnitude smaller than the number of individual alerts. This is quite useful by itself, as the merged alerts still keep track of all the attributes of individual ones. Meanwhile, the reduced number of alerts greatly simplifies visualization, as well as the next steps in the architecture for graph building, partitioning and scoring.

1) Graph Quality: a) Graph partitioning: Similar to the previous point, one interesting metric when at looking at graph and correlation quality is to look into how much the size of the incident graph grows as we lengthen the analysis period, following which the number of alerts also increases. In Fig. 8 we show the average incident size and its standard deviation versus the length of the analysis period. With the increase in the number of alerts, the community detection approach results in larger communities thwarting the ability of the analyst to study them, a weakness alleviated in our model. Note that we have set $\text{MaxCard} = 20$ which explains the divergence of the two lines once the incident size extracted by the community detection exceeds the upper limit set in the optimization approach. This value was chosen based on analyst feedback where any graph with more than 20 nodes is hard to visually investigate. MaxCard value can also be chosen based on the number of steps in the attack-chain or attack-lifecycle, where that number is usually less than 15.

b) Factor Graphs: Next, we take a deeper look at the scoring operations performed by the proposed factor graphs. In Fig. 9 we show an example of an incident detected in our enterprise network. This incident represents exploiting an open SQL port to take part in a distributed denial-of-service (DDoS) attack. Running the belief-propagation algorithm on the FG for this...
incident yields the scores shown in Table VII. In short, the FG rejected the Initial Access tactic and its alert corresponding to the branching off from the main attack path. The first two alerts are much more likely to lead to a Command and Control and other tactics representing the advanced progress of attack as opposed to a new attempt at accessing the network. This example shows the ability of the FG to take all alerts into consideration when scoring each individual tactic.

D. Enterprise Aggregate

1) Graph Classification: As the final experimental step, we wish to evaluate the usefulness of generated incidents for downstream tasks. For this purpose, we have collected three months of data containing security alerts and their resolution by security analysts from thousands of enterprise networks. The data contains Snort, Active Directory and Office-365 alerts. We split the data into a 75% training set and a 25% validation set. After ingesting the data through our framework, the validation dataset contains a total of 19409 graphs. We drop any graphs with less than three nodes as suggested by security analysts. Once these graphs are dropped, we are left with 2761 validation graphs.

The performance metrics for a variety of classification models are shown in Table VIII. The ensemble model (Ensemble) is built from DecisionTree, RandomForest, Adaboost and MLP instances. The same holds for (Ensemble with Search) with the addition of grid search over its hyper-parameters such as the class weights. More details on the contribution and importance of each feature is provided in B. Due to the imbalanced nature of the data, i.e., majority of graphs are benign, we consider balanced accuracy as our main metric. Balanced accuracy is defined as the average of recall obtained on each class. The ensemble model achieves the best balanced accuracy at 87%. Upon further investigation of mislabelled graphs, a common scenario is when the analyst label is based on meta-data unavailable to the model. The meta-data might indicate that the network is undergoing a penetration test, or more generally that the alerts are expected for the network under study.

When all four steps of the proposed architecture are considered, i.e., alert merging, graph construction, partitioning and classification, we end up with around four orders of magnitude reduction in the data volume represented to the analyst. Two orders of magnitude reduction are a result of alert merging, one a result of graph operations, and the final reduction comes from graph classification. The final classification step also ensures high-fidelity in the graphs to be investigated.

In order to provide a more extensive set of results, we have tried our system on the APT29 emulation plan from the recent Mordor dataset, whose results are provided as supplementary material for this manuscript, corresponding to days 1 and 2 in the emulation.

IX. REAL-TIME ARCHITECTURE

In the discussion so far, the graph building, partitioning and incident scoring are all run as periodic batch jobs. A need arises in industrial applications for a real-time architecture in order to ensure timely detection. In order for the architecture to be real-time, it needs to satisfy two requirements:

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
1) **Causal Data View**: in the signal processing sense [38]. The system must be able to make a decision based on past and present data without needing access to future data.

2) **Data Rate**: the rate at which the system processes a new data point must be less than the input data rate. This is in order to guarantee the system does not lag behind the input stream of data, i.e., prevent backlog [39]. If the system does not lag behind it is sometimes called to be operating at line-speed.

We have improved the architecture for real-time operation through the following changes:

- **Template Extraction**: Templates are built offline. During the real-time operation, alerts are matched to one of the generated templates, and merged accordingly.
- **Alert Correlation**: we propose a real-time version of the correlation algorithm building on ideas from locally-sensitive hashing.
- **Alert Graph Partitioning**: we adopt a real-time graph partitioning algorithm from [40] that is highly applicable to data rates we encounter in industrial deployments.
- **Incident Scoring**: We drop the factor graph element from the scoring and stick only to the classification step.

A diagram detailing the streaming architecture is shown in Fig. 10.

### A. Locally Sensitive Hashing for Fast Correlation

Calculating correlation between generalized alerts in order to determine the edges of the graph is an instance of the k nearest neighbours problem, for which the complexity is $O(n^2)$. Locally-sensitive hashing [41] is a way to reduce the complexity of the problem through splitting data into buckets, and only searching for neighbours within the same bucket. The choice of the hashing function is crucial, since data points that are supposed to be neighbours should ideally have the same hash and fall within the same bucket.

We note that in order for two generalized alerts to have an edge, a prerequisite is for them to share at least one attribute value. Hence, if we hash generalized alerts based on their attribute values, we guarantee that all alerts sharing the same attribute value are in the same bucket. Indeed, our choice of the hashing function for the alert is its attribute values. In other words, the generalized alert will be put in any bucket corresponding to at least one of its attribute values.

We note that in order for two generalized alerts to have an edge, a prerequisite is for them to share at least one attribute value. Hence, if we hash generalized alerts based on their attribute values, we guarantee that all alerts sharing the same attribute value are in the same bucket. Indeed, our choice of the hashing function for the alert is its attribute values. In other words, the generalized alert will be put in any bucket corresponding to at least one of its attribute values.

The rest of the correlation algorithm, i.e., checking the MITRE correlation, will continue as usual but only against the generalized alerts within the same bucket.

### B. Linear Greedy Graph Partitioning

Real-time graph partitioning is an active research area aiming for low-complexity algorithms and a streaming API. Stanton et al. discussed different algorithms for the problem in [40] and bench-marked their performance. For our work, we opted to go with the (Weighted) Linear Deterministic Greedy version. This method has the advantages of:

- An upper limit on the graph size, necessary for visual clarity of the output graphs
- A customizable weight which can take into account the security context, such as number of assets and tactics

The streaming architecture outlined in this section is currently under review as a patent application [42].

### C. Real-Time Performance

The streaming architecture was implemented and deployed in our environment. Events are first grouped into 5-minute mini-batches. The main performance measure when it comes to streaming is whether the system can handle the line-speed of the data, or does it lag behind. In other words, for a 5-minute mini-batch, if the system can finish processing the mini-batch in less than 5 minutes, then we can match the line-speed. However, if it takes more than 5 minutes to process the mini-batch, then the system is lagging behind. The mini-batch processing time is shown in Fig. 11. These metrics were captured from running the system on a m5.8xlarge AWS instance for a period of 7 days.
The $y$-axis represents execution time in seconds, while the $x$-axis represents the time axis. Some important conclusions from the figure are:

- The system can operate at line speed for up to 5 days.
- Being a stateful system, the state grows steadily over time, eventually causing the system to lag behind.
- While the system lags during peak hours, it is still able to catch up during off-peak hours.
- While we got the results by running the system on a single machine for all enterprise networks, the system can be horizontally scaled through assigned different instances to different groups of networks and processing them in parallel.

X. PERIPHERAL PROBLEMS

In this section, we outline a few crucial areas that heavily influence the design and implementation of our architecture. These areas do not constitute a separate set of stages, instead, they tie closely to the design of the four stages discussed in Section III.

A. Scalable Implementation

In terms of implementation, the process of converting alerts into distinct incidents, each composed of multiple generalized alerts, is divided into three main steps:

- Learning alert templates from batches of alerts.
- Processing alerts as they are created and merging them into generalized alerts.
- Constructing alert graphs from batches of generalized alerts, partitioning them into incident graphs, and scoring the incidents.

The first and second step, being independent for each alert source, lend themselves naturally to a Map-Reduce implementation [43]. Spark\(^9\) is a popular choice here and is what we have used internally. The third step, focusing on graph processing, is best implemented through a message-passing framework [44]. Together, these solutions enable horizontal-scaling of our architecture across a dynamic number of computing instances, and have allowed us to process data from thousands of devices in large enterprise networks.

B. IP Similarity

Measuring similarity between IP addresses is an important task in the daily operations of any enterprise network. Applications that depend on an IP similarity measure include measuring correlation between security alerts, building baselines for behavioral modeling, debugging network failures and tracking persistent attacks. For example, in our case a novel IP similarity measure $C^\parallel_{IP}(\cdot,\cdot)$ can be plugged into (3) to measure correlation between two alerts. However, IP addresses do not have a natural similarity measure by definition. Deep Learning (DL) architectures are a promising solution here since they are able to learn numerical representations for IP addresses directly from data. Given these numerical representations, various distance functions can be applied to measure similarity between the corresponding IP addresses. In a recent work [45], we have leveraged a variant of the word2vec algorithm [46] to measure similarity between IPs learned from network log data, with promising results in incident extraction and investigation. In a follow-up work [47], we have extended [45] by leveraging Graph Neural Networks (GNN) for IP similarity. GNNs have the advantage of being inductive models, i.e., they can measure similarity between new IPs not necessarily encountered during the training phase. This is a crucial feature for these models necessary in order to scale to real-world networks.

C. MITRE Tactic Mapping

Mapping alerts into one or more MITRE tactics is a necessary requirement in the proposed work. Such mapping is typically done manually, based on subject matter knowledge. However, recent language models can of great value here. Language models can be fine-tuned on a small dataset of labelled alerts and used to infer for the rest. Moreover, powerful models such as ChatGPT [48] can be used as well. In fact, the mappings for Sysmon and Windows events we currently use were produced by ChatGPT.

D. Security Analyst Feedback

The analyst can interact with the extracted incident graph as well as the general process of the extraction architecture in a variety of ways. These include:

- An analyst can interact with the generated incident by marking either an alert or a tactic as Inactive, indicating a false-positive. This feedback can be handled by the FG in the form of an evidence-based query [36]. An evidence-based query is an inference process on the FG with a limit on the values the conditioned variable can take.
- In case an analyst has marked an alert as false-positive, then a parallel process can look for similar alerts and suggest them to the analyst for further evaluation. Measuring similarity between alerts can be partially answered by measuring similarity between their IP attributes. The approaches discussed in Section X-B are of value here.
- The analyst may wish to revisit some of the generated templates. For example, the template generation mechanism might choose to generalize sourceIP and destinationIP. Generalizing on both IPs is not usually a desired behavior. Re-arranging the order of attributes for templating can be easily done by the analyst afterwards.

We have only implemented the first point using evidence-based queries as will be discussed in Appendix C, available online, while the integration of the other two is part of our future work.

E. Causality

One important observation from discussing the incident graphs with security analysts is that an analyst expects an edge connecting two alerts to indicate a causal relationship. The
question of learning causality is much more challenging than that of learning correlation [49]. We have tried two approaches to address the problem of causality:

- **Protocol-based:** This approach would use knowledge of the internals of different networking protocols to identify if a previous packet or flow led to the current packet or flow. As an example, a DNS query for a certain domain name together with the response containing the domain’s IP address can be seen as the cause of the next HTTP traffic to that IP. The main challenge here is handling all of the possible protocol combinations expected in the network and all their variants, which is quite a challenging task.

- **Statistical:** This approach utilizes some of the established algorithms for learning a causal relations graph directly from the data [50]. However, the nature of our data and its main representation as low-level TCP/UDP flow logs hinders the applicability of algorithms such as the PC-algorithm. In particular, the layered structured of the networking protocol results in layer-3 events being all of the same type for different applications, and we cannot build a causality graph from these layer-3 flows. This approach has more success for other types of data with more context such as Syslog [51].

**XI. Future Work**

Our planned future works centers on several lines of potential improvements:

1) **Causality:** The logic for chaining a sequence of alerts into an incident graphs can be greatly improved if causal relationships between alerts are available. A potential avenue here is built around leveraging end-point data to determine the universally unique identifier (UUID) [52] for the software process generating each event. Once the network-event to process mapping is done, all subsequent events from the same process can be seen as part of the same causal chain. On the statistical front, recent work [51] has explored building causal graphs between log events in a network operations context. Extending this approach to the security context is a promising venue for extending the present work.

2) **Extending Factor Graphs:** Currently, our FG considers if a sequence of two tactics in an incident is valid from a MITRE ATT&CK perspective. A further enhancement here is to build pairwise factors between alerts, checking if this specific sequence of alerts is valid. This is a more granular approach to build FGs, with the added granularity expected to lead to better scores. On the other hand, labelled datasets and increased computation are necessary to build and use these updated FGs. Integrating causal relationships into factor graph scoring could also prove valuable.

**XII. Conclusion**

In this paper, we have proposed RANK, the first end-to-end architecture for detecting APTs in enterprise networks. Many elements of the data processing pipeline are automated in the proposed architecture, such as learning alert templates, segmenting an alert graph into separate incidents, and utilizing factor graphs to automatically score these incidents. Simultaneously, the architecture is meant to preserve analysts’ critical role in the investigation process through offering a highly efficient and simplified representation of the data for easier examination. Furthermore, MITRE ATT&CK has been carefully integrated into the various stages of our architecture, including alert correlation, alert graph segmentation, and incident scoring.

Our key contribution is up to four-orders-of-magnitude reduction in the amount of data to be investigated by the analyst. We have also proposed innovative solutions for a plethora of sub-problems, including learning alert templates, segmenting an alert graph into separate incidents, and utilizing factor graphs to automatically score these incidents. We have also proposed a graph classification model with up to 87% balanced accuracy. The real-time design on the architecture is also investigated and shown to operate at line-speed. In addition, we addressed unique methodologies for alert correlation, alert graph segmentation, incident scoring, scalable implementation, and IP similarity, as well as how they all fit into the APT detection architecture. Future work for this system can focus on modeling causal relationships between the telemetry events and integrating them into the graph construction logic.

**References**

[1] I. Ghafir and V. Prenosil, “Advanced persistent threat attack detection: An overview,” Int. J. Adv. Comput. Netw. Its Secur., vol. 4, no. 4, 2014, Art. no. 5054.
[2] D. McWhorter, “Apt1: Exposing one of China’s cyber espionage units,” Mandiant, 2013.
[3] IBM Security, “IBM X-force threat intelligence index 2017,” Mar. 2017. [Online]. Available: https://www.ibm.com/reports/threat-intelligence
[4] J. Sexton, C. Stordeur, and J. Neill, “Attack chain detection,” Stat. Anal. Data Mining: ASA Data Sci. J., vol. 8, no. 5/6, pp. 353–363, 2015.
[5] P. Cao, “On preempting advanced persistent threats using probabilistic graphical models,” 2019, arXiv: 1903.08826.
[6] M. Shashanka, M.-Y. Shen, and J. Wang, “User and entity behavior analytics for enterprise security,” in Proc. IEEE Int. Conf. Big Data, 2016, pp. 1867–1874.
[7] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Proc. IEEE Symp. Secur. Privacy, 2010, pp. 305–316.
[8] N. Caplan, “Cyber war: The challenge to national security,” Glob. Secur. Stud., vol. 4, no. 1, pp. 93–115, 2013.
[9] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington, and C. B. Thomas, “Mitre attck: Design and philosophy,” MITRE Corp., Jul. 2018. [Online]. Available: https://www.mitre.org/news-insights/publication/mitre-attck-design-and-philosophy
[10] H. M. Soliman, “An optimization approach to graph partitioning for detecting persistent attacks in enterprise networks,” in Proc. Int. Symp. Netw. Comput. Commun., 2020, pp. 1–6.
[11] A. Alshamrani, S. Myneni, A. Chowdhary, and D. Huang, “A survey on advanced persistent threats: Techniques, solutions, challenges, and research opportunities,” IEEE Commun. Surv. Tutor., vol. 21, no. 2, pp. 1851–1877, Second Quarter 2019.
[12] P. N. Bahrami, A. Dehghantanha, T. Dargahi, R. M. Parizi, K.-K. R. Choo, and H. H. Javadi, “Cyber kill chain-based taxonomy of advanced persistent threat actors: Analogy of tactics, techniques, and procedures,” J. Inf. Process. Syst., vol. 15, no. 4, pp. 865–889, 2019.
