NETHAWK: Hunting Advanced Persistent Threats via Structural and Temporal Graph Anomalies

Benjamin Bowman[§] Isaiah J. King[§] Rimon Melamed H. Howie Huang
GraphLab
The George Washington University
{bbowman410, iking5, rmelamed, howie}@gwu.edu

***Abstract*—Traditional signature-based perimeter defenses are not sufficient for defending enterprise computer networks against modern-day cyber threats. Advanced Persistent Threats can easily evade known techniques and signatures. Behavioral analytics have been proposed as a way to move beyond signatures to detect stealthy and sophisticated threat actors better. However, current implementations of these analytics are either too granular to generate meaningful alerts, or too strict, relying on user-generated patterns to explicitly describe malicious behaviors. In this work, we introduce a new system, NETHAWK, which represents cyber activity within an enterprise network as an attributed graph. It then uses this graph to conduct behavioral anomaly detection to identify structural and temporal graph anomalies. We further leverage the graph structure to generate high-fidelity and complete incident reports on malicious activity based on connected-component analysis. We apply our techniques to two open-source datasets: the DARPA OpTC dataset, and the LANL Comprehensive Multi-Source Cyber Security Events dataset. We detect malicious activity with high accuracy while maintaining minimal CPU and memory utilization. We also demonstrate that our approach generates meaningful, easy-to-understand alerts that align with the human descriptions of the attacks we analyze.**

I. INTRODUCTION

Existing cyber security tools utilized by enterprise defenders suffer from two major challenges: they have difficulty detecting novel attacks, and the alerts they generate are uninformative. Signature-based intrusion detection systems (IDSs) are popular in the academic field [1], [2], [3], [4], as well as industry [5], [6], [7], but they can only detect signatures that experts explicitly specify. On the other hand, anomaly-based intrusion detection systems do not have this weakness. They can detect zero-day attacks but at the cost of higher false alarms [8]. When these systems do have true positives, they are often difficult to interpret, either due to the high volume of uncorrelated alerts [9], [10] or because the alerts are coarse-grained, labeling large swaths of logs over fixed-length periods of time [11], [12], [13].

Signature-based IDSs detect signatures of known bad events, such as malicious file hashes, domain names, IP addresses, etc, or predefined behaviors of an attacker. While these methods can provide a high-fidelity indicator of compromise, they are inherently flawed as they will never be able to detect a sophisticated adversary who will not reuse known attacks. For example, the high profile *SolarWinds* hack was not only based on new malware, but on malware embedded in previously trusted software [14]. Many advanced attack campaigns will perform so-called *living-off-the-land* attacks, utilizing known and trusted software native to the environment to accomplish their mission, making them very challenging to detect [15]. For an IDS to be resilient against advanced persistent threats (APTs), it must detect zero-day attacks.

The second challenge comes from the fact that security tools often generate a high volume of weak indicators of comprise. This requires a human analyst to then "connect-the-dots" to understand the full impact of a security event. This challenge is exacerbated by the fact that many security alerts are false positives [16], [17], [18], [19] leading to so-called "alert fatigue" of the cyber analysts. Differentiating true positives from false positives, while simultaneously stitching together the true positive alerts into a security incident requires a significant level of expertise, as well as time and effort. Attacks can succeed just because defenders do not properly understand their full scope quickly enough to respond. For an IDS to be usable, it is crucial that its alerts are not only precise but understandable to security analysts.

In this work, we introduce our system, NETHAWK, which accurately and scalably models user and system activity within enterprise-grade networks. Our system is anomaly-based, meaning it is not susceptible to the first challenge; it does not require expensive creation or maintenance of rules, patterns, or signatures of attacks. Additionally, the *Cyber Activity Graph* data structure at the heart of our system provides alerts in the form of malicious subgraphs. This means that unlike prior graph-based IDSs [1], [2], [9], [11], [12], [20], [21], NETHAWK aggregates alerts together into explainable clusters of malicious activity, creating highly informative, and comprehensive security alerts. NETHAWK operates at the enterprise level, as opposed to individual hosts, and therefore it can stitch together related events that span hosts and users in the environment.

We evaluate our system on the OpTC dataset [22] generated during the DARPA Transparent Computing program, and the Comprehensive Multi-Source Cyber Security Events dataset [23] from the Los Alamos National Labs (LANL) enterprise network. The former dataset is a simulated enterprise-grade network including roughly 600 hosts, 600 users, five days of benign activity, and three days of malicious red team attack

[§]Equal contribution

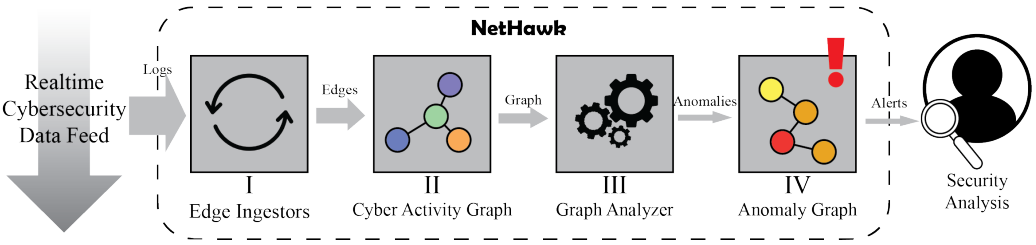


Fig. 1: NETHAWK Detection and Monitoring System Architecture

campaigns. The latter dataset is a real-world dataset captured from the internal LANL network, also including labels from red team activity. Our system was able to learn the benign activity and identify the malicious attack campaigns with 90% precision and 98% recall in OpTC, and 72% precision and 93% recall in LANL. Because alerts are structured as temporal subgraphs of all user and system activity, they are highly explainable. We will show how the alerts from the NETHAWK system can be used to accurately reconstruct the path of an attacker as they pivot from host to host, and how the subgraph alerts generated by our system agree, and even expand upon those reported by the redteams.

II. BACKGROUND

The majority of off-the-shelf security tools and analytics focus on traditional row or columnar data representation and analysis. Representing and analyzing the data in this way imposes limitations on the types of analysis that can be performed [24]. Some prior works design purpose-built query systems entirely for cybersecurity analysis [3], [25]. Many other works [12], [26], [27], [28] have shown that graph representations of enterprise activities provide significant algorithmic advantages over traditional database representations for a variety of cybersecurity-relevant tasks. For this reason, in this work, we represent and process cybersecurity data as a graph. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a collection of vertices, or nodes $\mathcal{V}$ that represent discrete entities, and edges $\mathcal{E} = \{(u, v) \mid u, v \in \mathcal{V}\}$ that represent relationships between them. In this work, systems and users are represented by nodes, and interactions between them are represented by edges.

III. NETHAWK SYSTEM

In this section, we will discuss the details of our detection and monitoring system which we call NETHAWK. We will start with a high-level overview of the components of the system, followed by a deep dive of each element.

A. System Overview

Figure 1 shows the workflow illustrating how the system operates. Each component is summarized below.

I. Edge Ingestors parse cybersecurity log data and convert them into a set of time series edges in the Cyber Activity Graph. This requires custom parsers for each log type in use. The edge ingestors either pull from an existing data store, such as a Security Information and Events Management System (SIEM) (e.g., Splunk [29], or ELK [30]), or from a streaming data feed (e.g., Apache Kafka [31]), and convert the raw logs into a set of node and/or edge updates. These updates are then applied to the Cyber Activity Graph.

II. Cyber Activity Graph is the primary data structure of our system. It contains real-time graph information pertaining to how users and systems interact across an enterprise network. The graph holds information that is both structural in nature (e.g., who is talking to whom?), and behavioral (e.g., what are they saying?). The graph edges and features are further attributed with timestamps of the first observation of a particular behavior on each edge. The graph analyzer anomaly detection algorithm uses this fine-grained time series information per entity in the graph.

III. Graph Analyzer processes a snapshot of the Cyber Activity Graph and assigns an anomaly score to all edges in the graph using our anomaly detection algorithm.

IV. Anomaly Graph receives anomalous edges from the Graph Analyzer and will commit a subset of those edges to the Anomaly Graph data structure for inspection by a security analyst when the Anomaly Score exceeds a certain threshold.

B. Edge Ingestors

The Cyber Activity Graph is only as expressive as the data on which it is generated. As we are focused on granular, user-driven behaviors and activity, it is necessary that we have an equally granular data source. As a result, we focus on analyzing host-based telemetry. Fortunately, this type of information is already generated by many enterprise solutions such as Endpoint Detection and Response (EDR) solutions [32], antivirus solutions (AV), Sysmon [33], or even native solutions provided by operating systems, such as Event Logs in Windows [34], or `auditd` in Linux [35]. In most of these cases, information such as which user is logged in, and what actions they are performing on a particular system will be logged and optionally forwarded to a centralized system, such as a SIEM for analysis. The responsibility of the Edge Ingestors is to pull log data from the SIEM or similar centralized log source, transform the content into a time series edge list, and update the Cyber Activity Graph.

C. Cyber Activity Graph

The Cyber Activity Graph is a graph structure $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of vertices of type $\{system, user\}$ and $\mathcal{E}$ is a set of edges of type $\{file, process, network\}$. All nodes and

edges are attributed with a timestamp corresponding to the first time NETHAWK observed that node or edge. Edges are further attributed with behavioral features which themselves are also each associated with a timestamp of the first observation. Figure 2 provides a diagram representation of the information captured in the Cyber Activity Graph. Note that all graph elements are tagged with timestamps.

We consider two kinds of nodes, and three kinds of edges in our representation of the network. *User Nodes* are the entities that correspond to an account, persona, or credential. These nodes are designed to capture identities within an enterprise network. *System Nodes* are the entities that correspond to systems or services. They are attributed with information such as an IP address and/or hostname. These nodes are designed to capture services within an enterprise network.

Edges in the graph represent interactions between users and systems. A *File Edge* represents a user interacting with a file on a system; *Process Edges* represent a user executing a process on a system; *Network Edges* represent a user accessing a network service on a system. Each of these edge types has a timestamp for the first time that kind of interaction was observed, and a unique identifier that categorizes the relationship. File edges are uniquely identified by the filetype that was being accessed, process edges are identified by the process name, and network edges are identified by a port number if it was non-ephemeral, or a feature for ephemeral port use. They are also attributed with a direction specifier to indicate if the user experienced flow activity to or from the system.

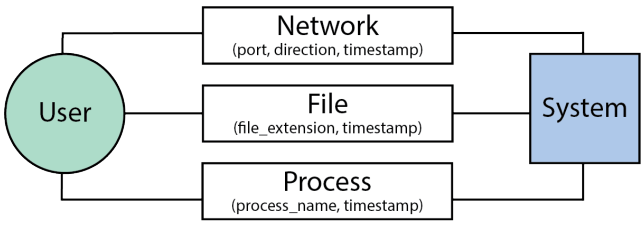


Fig. 2: Cyber Activity Graph Schema

D. Graph Analyzer

The purpose of the Graph Analyzer is to periodically analyze the Cyber Activity Graph and assign anomaly scores to edges of interest. The analysis consists of temporal and structural analysis and a training phase.

Temporal Analysis. The Graph Analyzer is powered by a custom anomaly detection algorithm which was influenced by the threat-hunting technique of long-tail analysis. The idea of long-tail analysis is that the most infrequent events are a good place to start when looking for potentially malicious activity. As NETHAWK is a system designed for monitoring real-time enterprise network data feeds, we relax long-tail analysis slightly; instead of just focusing on infrequent events, we focus on new, previously unobserved events.

Equation 1 shows the most basic form of the edge feature anomaly calculation in our system. This equation is an exponential decay function that returns values between $(0, 1]$ based on the age of an edge feature f.

$$\begin{aligned} A(f) &= e^{\frac{ln(0.5)}{t_{1/2}} * age(f)} \\ &= 0.5^{\frac{age(f)}{t_{1/2}}} \end{aligned} \tag{1}$$

The $age()$ function is simply the number of time units that have elapsed between the current time of analysis, and the first observation of edge feature f. The $t_{1/2}$ parameter, represents a concept we call anomaly half-life. This is the amount of time that must pass for the anomaly score of a particular edge feature to decrease by half. As the value of $t_{1/2}$ decreases, the anomaly score of a newly observed event will decay more quickly. A value of $t_{1/2}$ that is too low will cause the system to "forget" about activity too quickly, meaning it will not properly stitch together related malicious events. A value that is too high will cause the system to "remember" activity for too long. This increases the likelihood of unrelated but anomalous activity being incorrectly associated together, ultimately resulting in false positives. In our experiments, we have found a half-life of 6 hours to provide good performance.

Structural Analysis. A common challenge with temporal-based threat hunting is that there are many new or infrequent events that occur in large-scale, diverse computing environments that are not malicious. Because of this, it is not sufficient to only look at new or infrequent events as this would cause far too many false positives. To handle this challenge we utilize the graph structure to reduce the anomaly score of edges that are unlikely to be malicious based on the neighborhoods of the nodes involved.

The *Neighbor Similarity Scaling* technique aims to reduce the anomaly score of an edge feature based on the similarity of the anomalous edge feature to other edges that have been previously observed and are no longer considered anomalous. In other words, if a new edge is observed between node a and b, we reduce the anomaly score based on whether that same activity was observed between nodes (a, b') where $b \approx b'$. There are many ways to compute node similarity, however, in this work, we use a simple function based on nodes with a similar degree in the 1-hop neighborhoods of the nodes involved in the original edge. Equation 2 shows the similarity function. The β parameter controls how strict this equation is, and the $D(\cdot)$ function returns the degree of the node.

$$sim(a, a') = \begin{cases} true & |D(a') - D(a)| < \beta * \frac{D(a') + D(a)}{2} \\ false & otherwise \end{cases} \tag{2}$$

After identifying suitable nodes based on our similarity function in the 1-hop neighborhood, the anomaly score of the same feature (e.g. port, process name, etc) is used as a scaling factor on the original anomaly score s. For each shared feature f' in a similar node, we adjust the anomaly score to be $s = s * A(f')$. As the values returned by the anomaly function $A(\cdot)$ are necessarily ≤ 1, this will always result in either no change or a reduction of the anomaly score s.

It is important to note that we do not want to blindly apply the neighbor similarity scaling technique for all edges in the graph. For edges involving low-degree nodes (e.g., users and their workstations), new activity, despite its similarity to previously observed activity on other edges, is inherently interesting. For example, a user logging into their co-worker's workstation for the first time may look exactly like how they log into their own workstation, however, it is still extremely interesting from a security perspective. Conversely, an edge involving a node with a high degree, such as a web server or file server, utilizing that resource for the first time is less interesting from a security perspective. Therefore, edges are divided into two groups we call: individual resources and shared resources.

Intuitively, individual resources are those which are tied to specific users or services. For these nodes, we do not apply the neighbor similarity scaling technique to reduce anomaly scores. On the other hand, shared resources are services that are used by large groups of users, or even the entire enterprise. For these nodes, we do apply the neighbor similarity scaling technique in order to reduce the anomaly score for edges involving these entities. To separate the two types of entities, we use a metric based on the mean and standard deviation of node degrees across the enterprise. Nodes with more than the average plus one standard deviation number of edges are labeled as shared resource modes. In the datasets we analyzed, we found that there was a clear distinction between these two distributions.

Algorithm 1 Cyber Activity Graph edge scoring routine

```
1: procedure COMPUTEEDGESCORES(G)
2:     for edge ∈ G.edges do
3:         scores = []
4:         for f ∈ edge.features do
5:             s = A(f)
6:             if (a or b) ∈ shared resources then
7:                 s = s * neighborScale(G, a, b, f)
8:             scores.push(s)
9:         G(edge).scores.push(sum(scores))
```

The pseudocode for the edge anomaly scoring algorithm can be seen in Algorithm 1. The input graph $\mathcal{G}$ is a snapshot of the Cyber Activity Graph at a particular point in time. On lines 2 and 4 we are iterating over each feature of each edge in the graph. We compute the edge anomaly score for the particular feature on line 5, followed by the neighbor similarity scaling technique on lines 7 and 8 if either of the nodes are shared resources. Finally, on line 10 the edge anomaly scores are summed and added to the list of edge scores stored in the graph.

Training. Since the edge anomaly algorithm is based on first observation time, it is necessary that we have a training window. Otherwise, everything that is new will be identified as anomalous as soon as the system starts. Additionally, our goal is not to simply detect new entities within the network, but rather to detect new behaviors between known entities. To that end, we build and maintain a data structure that indicates if a particular node is in training or not. No edges involving nodes that are in the training phase will be added to the anomaly graph.

To handle new nodes and nodes that require different amounts of training time, we have designed an online, per-node training algorithm. All nodes start in training mode when they are first observed, with the exception of external IP nodes, which are inherently untrusted and can generate anomalies on their first observation. For nodes from internal entities, a new node is in training until two conditions are met: (1) a minimum number of observations have been recorded, and (2) the anomaly score reaches equilibrium. The first condition is trivial to implement. For the second, we calculate the variance of the previous N observations and declare that equilibrium has been met once the variance of the anomaly terms falls below some threshold, γ. In our experiments, we found a value of $\gamma = 100$ performed well. If this equilibrium has been achieved, the node is labeled as no longer training and will be incorporated into the Anomaly Graph if anomalous edges are identified.

E. Anomaly Graph

The Anomaly Graph is a structure that stores the anomalous components identified by the Graph Analyzer and presents them to a human analyst for investigation. At this point in the algorithm, individual edge anomalies are still relatively weak indicators of compromise, and a human analyst would still be required to do a lot of manual inspection in order to piece together an attack campaign. To address this, we again develop a technique that utilizes the graph structure to further weed out false positive events from true malicious activity. We accomplish this by analyzing the structure of the graph formed by connecting the anomalous edges and amplifying the most anomalous graph structures.

In graph theory, connected components are a traditional way to analyze graph structures, and are defined as subgraphs in which there is a path that connects each node. We modify this type of analysis slightly and define our anomalous connected components as those components that share edges that are identified as anomalous. Additionally, this part of the algorithm has the ability to baseline the anomaly score of edges based on their history of anomaly scores. For edges that contain an anomaly score history of at least the number of minimum training samples N, anomalous edges are identified as those edges with an anomaly score which is τ standard deviations away from the mean for that particular edge. For edges that do not have sufficient history, anomalous edges are simply labeled with the anomaly score as computed at that time step. This way, the system has a way to baseline anomalous behavior for particular edges such that they won't make their way into the Anomaly Graph if it is typical for a particular edge.

After identifying the anomalous connected components, we use the size of the component as a way to further amplify their signal. The intuition here is that a single edge anomaly is a relatively weak indicator of malicious activity, however,

as anomalies connect together both structurally via anomalous connected components, as well as temporally by our time-based anomaly detection algorithm, they become increasingly more likely to be malicious. Therefore we use the size of the connected anomalous component as a force-multiplier of the anomalous activity. After the anomaly amplification, we have a threshold based calculation that determines if the activity should be published to the anomaly graph, ultimately escalating the activity to a human for investigation.

Algorithm 2 Anomaly Graph Create Routine

```
1:  procedure CREATEANOMALYGRAPH(G_R)
2:      G_A = new Graph()
3:      for (a, b) ∈ G_R.edges do
4:          lastScore = G_R(a, b).scores.last
5:          mean = mean(G_R(a, b).scores)
6:          std = std(G_R(a, b).scores)
7:          if lastScore > mean + τ * std then
8:              δ = lastScore − (mean + τ * std)
9:              G_A.addEdge(a, b, score = δ)
10:     ccs = connectedComponents(G_A)
11:     for cc ∈ ccs do
12:         score = |cc.nodes| * |cc.edges| * sum(cc.scores)
13:         if score > ν then
14:             alert(cc)
```

One full analysis iteration is shown in Algorithm 2. On line 4, the real-time Cyber Activity Graph accesses the output of computeEdgeScores (Algorithm 1) for each edge. Next, the edge anomaly scores are analyzed and if the score exceeds a threshold based on deviation from the mean, the edge is added to a temporary graph structure (lines 5-9). After all edges are added to the temporary graph for the current time step, we apply the connected component anomaly amplification by scoring entire components of the graph as opposed to individual edges (lines 10-12). If an anomalous component exceeds a predefined threshold ν then an alert is raised on the connected component (line 14).

IV. Evaluation

We evaluate NetHawk on two public datasets: OpTC [22] and the LANL Comprehensive Multi-Source Cyber Security Events [23]. Table I shows high-level dataset details. The LANL dataset, as it spans 58 days serves to demonstrate our approach's ability to correlate temporally distant events. The OpTC red team attempted to emulate APT behavior; we evaluate NetHawk on this dataset to support our claim that it can detect APT-like attacks.

The NetHawk system is developed entirely in Python, utilizing the highly efficient NetworkX graphing library [36], in about two thousand lines of code. The NetHawk system was developed as a single-threaded application. The reported runtimes in the experiments below are from a single Intel Xeon E5-2683 CPU core. In all experiments, unless otherwise specified, the hyperparameters used by NetHawk are those delineated in Table II, with the exception of ν, which is set to 100 for the OpTC experiments, and 1,000 for LANL, due to its larger scale. That these parameters can be reused between such different datasets illustrates the robustness and applicability of our model.

TABLE I: Evaluation Datasets

	OpTC	LANL
Duration (Days)	8	58
Attacks (Days)	3	18
Computers	625	17666
Users	627	10941

TABLE II: NetHawk System Configuration

Parameter Name	Variable	Value
Anomaly Half-Life	$t_{1/2}$	6
Neighbor Similarity Factor	β	0.3
Minimum Training Samples	N	24
Training Variance Threshold	γ	100
Anomalous Edge Factor	τ	5
Anomaly Graph Threshold	ν	{100, 1,000}

A. Accuracy Analysis

1) OpTC: First, we will evaluate the detection accuracy of our algorithm compared with the state-of-the-art on the OpTC data. Table III shows the raw metrics each model achieves. We compare to three popular graph-based IDS approaches, Argus [10], Euler [9], and Unicorn [12]. The first two approaches detect anomalies on the edge-level by analyzing a temporal graph representation of the network, split into discrete (one-hour) time steps. Unicorn analyzes fixed-length edge lists representing activity in the network. It classifies periods of time as malicious or benign but does not differentiate between benign and malicious regions within each edge list it analyzes. Because of these different data splits, it is difficult to fairly compare these methods with ours, which detects anomalies at the node level. However, as we are primarily interested in limiting alert fatigue, and producing explainable alarms, we also compare the methods in terms of how many false alarms they produce, and how informative the true alarms are.

The first row of Table III shows the scores the models attained across the full dataset. In the NetHawk row, this includes the full 5 days of benign data that offline models need for training. For Euler and Argus, the metrics are only from the three attack days, as the benign days were needed for training. Despite this, NetHawk attains higher precision and recall than both of those approaches. Because NetHawk classifies nodes individually, it has fewer alarms than edge-centric methods; of those alarms, they are overwhelmingly TPs. We report 126 TPs, and 14 FPs during the attack period of the OpTC data. Compare this to the 1,175 TPs and 2,795 FPs, and the 753 TPs and 840 FPs emitted by Argus and Euler, respectively. Because the NetHawk alerts are correlated into attack graphs, analysts may have an easier time interpreting them. We explore this in the following subsection. Prior works only provide isolated edges, devoid of context; of these edge alerts, the majority of them are FPs for both approaches.

TABLE III: Accuracy results on OpTC Dataset

Data	Approach	Precision	Recall	F1
All	NETHAWK	**0.9000**	**0.9800**	**0.9400**
	Argus	0.2960	0.9647	0.4530
	Euler	0.5273	0.6897	0.5977
Attack Day-1	NETHAWK	0.9200	0.9600	0.9400
	Unicorn	0.6000	1.0000	0.7500
Attack Day-2	NETHAWK	0.8600	0.9900	0.9200
	Unicorn	0.8000	1.0000	0.8900
Attack Day-3	NETHAWK	1.0000	1.0000	1.0000
	Unicorn	0.1400	1.0000	0.2500

For our comparison with Unicorn, we construct host-based provenance graphs from the OpTC data, similarly as with NETHAWK. We construct the host graphs for each attack day and the benign period. The graphs from the 5-day benign period are used for training the models. We use the same parameters as Unicorn used for their analysis of the Darpa TC3 dataset [37].

The Unicorn approach presents several issues. While Unicorn could detect all of the attacks (recall was always 1.0), it suffers from a much higher false positive rate. Additionally, Unicorn's system is more aligned with analyzing homogeneous data, like enterprise servers, which execute predefined tasks with little to no deviation. However, the OpTC data is much more heterogeneous because it consists of users working on various machines, which can result in much more unstable and unpredictable behavior. Using these periodic sub-models for clustering results in less robustness when there are sudden changes in system behavior, causing the FPR to increase. In addition, unlike NETHAWK, which produces a comparatively small anomaly graph based on connected components within the overall graph, Unicorn does not provide any reconstruction or temporal attack analysis, causing the analyst to have to manually determine anomalies within the graph sketches, which can include thousands of edges.

2) LANL: Next, we analyze the models on the larger, but more opaque LANL dataset. During analysis, we found that an individual system always dominated our results, and generated a large Anomaly Graph component that dwarfed all others. Upon investigation, we found that this was related to a system labeled as *C15244*. This is a shared resource in the computing environment, which around the time of the red team events started providing a new service on port 69 (typically associated with the TFTP protocol) which was utilized by many systems in the environment. This is highly anomalous and possibly malicious, however, it does not coincide with any of the labeled red team events. Therefore, we decided to note the activity and whitelist the system to best analyze the other results without the noise added by this individual system. Note that this interference likely affected the results of the prior works to which we compare NETHAWK, as we use the results reported by [10].

Table IV shows the raw scores of each IDS on the LANL dataset. Due to memory constraints, we had to omit Unicorn from the evaluation, as the LANL dataset was too large for it to process. We again observe that NETHAWK achieves higher metrics than the prior works. However, as the three models are classifying different components of the graph, it is important to consider more than just the raw precision and recall scores. We must compare these models from the perspective of a security analyst. Both of the edge-detection works had more false positives than true positives, while 74% of NETHAWK's alerts were true positives. Additionally, the results of NETHAWK are easier to interpret. The Anomaly Graph contained 555 TPs, and 219 FPs, and did not include 41 FNs, and 102,822 TNs. It captured the vast majority of entities involved in the red team activity, while also remaining highly precise at 72%.

TABLE IV: Accuracy results on the LANL Dataset

Approach	Precision	Recall	F1	TPs	FPs
NETHAWK	**0.7200**	**0.9300**	**0.8116**	**555**	**219**
Argus	0.2171	0.8269	0.3439	363	1,309
Euler	0.0318	0.8565	0.0613	376	11,464

B. Attack Analysis

Next, we will dig into the results on a per-day, per-attack basis. Because LANL is from a real-world environment, the data is highly anonymized, which makes a granular interpretation of the results challenging if not infeasible. Thus, we only analyze the alerts NETHAWK produced on the OpTC dataset. We will analyze where our system succeeded, where it failed, and how it could be improved. Here, we attempt to justify our claim that the alerts produced by NETHAWK are not only precise but also explainable.

Attack Day 1. Figure 3 shows the Anomaly Graph for the activities that were detected during the day-1 attack campaign. Nodes are colored based on the time when they were aggregated in the Anomaly Graph. The graph was updated three times throughout the day, as indicated by the three individual colors. At the center is the compromised user account *zleazer*. We observe that *zleazer* had anomalous activity detected on their workstation, *sysclient0201*, in addition to many other systems in the environment. The C2 communication is clearly visible as the external IP node in the Anomaly Graph. The *administrator* account the attacker pivoted to, as well as the systems involved in that lateral movement, sysclient0660 and sysclient0402, are also visible. Despite not having visibility into the log files for the domain controller, we were able to detect the large amount of lateral movement that occurred after the domain controller compromise due to the usage of the compromised *zleazer* account.

In addition to this large connected component displayed in Figure 3, there is also a smaller, 3 node component that was detected on day-1 (omitted in the figure) that was counted as a false positive. This component involved a large amount of new activity between a couple of users in the system. Despite making its way into the anomaly graph, this

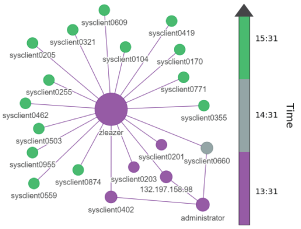


Fig. 3: OpTC Attack Day-1

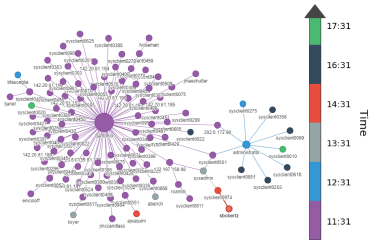


Fig. 4: OpTC Attack Day-2

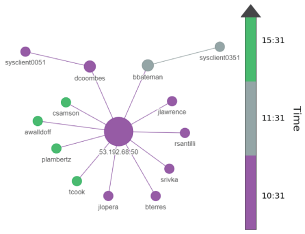


Fig. 5: OpTC Attack Day-3

component is easily distinguishable from the malicious activity as it covers only a small amount of activity. The only false negative our system incurred involved the compromised user account *hdorka*. However, this account was compromised on the domain controller, and we did not have visibility into this system.

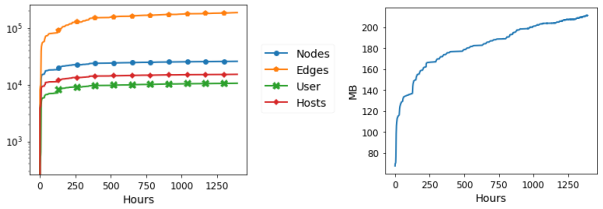
Attack Day 2. The Anomaly Graph for the activity that occurred during the day-2 attack campaign is shown in Figure 4. The attack began with the large cloud of connections around the *bantonio* user as the result of running the *DeathStar* automatic exploitation tool. It is clear that this level of activity is very noisy, and very obvious when analyzed as a graph. We can also see the other compromised user accounts of *administrator* and *sysadmin* which were added to the graph after the domain admin was compromised by the *DeathStar* tool. Again the C2 server is easily observable in the graph, shared between *bantonio* and the *administrator* account.

This day introduced the most false positives, and consequently had the lowest precision. This was due largely to the fact that the *bantonio DeathStar* process touched so many unique systems that it inadvertently amplified some activity that would have never been escalated into the anomaly graph. These manifest in the graph as weakly connected leaf nodes around the perimeter of the large *bantonio* cloud. There is likely some pruning that we could do to remove some of these false positives, which we leave to future work.

Attack Day 3. The attack on Day 3 was based on a malicious update of a well-known text editing program *Notepad++*. Two users, *dcoombes* and *bbateman* performed an update that provided attackers a backdoor into their system. Once on the system, the attackers interacted with the local workstations of the compromised users, only performing some enumeration and local persistence actions. No further lateral movement or exploitation was performed. Figure 5 shows the full attack graph for day-3. We can see that the system captured the anomalous and malicious activity between *dcoombes*, *bbateman*, and both of their respective workstations *sysclient0051* and *sysclient0351*. We can also see the attacker C2 server at *53.192.68.50*, the central hub of the Attack Graph. All of the other users interacting with the C2 IP address were not further exploited by the attackers, though we assume they were compromised, as they were calling back to the C2 infrastructure. Had all of these various accounts not also been compromised, the activity on this day would have likely been much more challenging to detect.

C. Scalability

Memory. Figure 6a shows the size of the Cyber Activity Graph over time as it processes the LANL dataset. Notice the superlinear growth in the first few hours of the dataset, tailing off after about 250 hours of system operation. During the first hours of operation, the system is observing a large quantity of new activity, which leads to this aggressive growth in size. After roughly 250 hours, the size of the graph is relatively static, with only a small upward trend. As shown in Subfigure 6b total memory consumption follows the same trend, with a large increase in memory consumption in the first 250 hours, followed by a slower, sub-linear increase from that point onward. For the full 58-day graph, the total memory used reached less than 300 MB, indicating that this technique could scale to a much larger network.



(a) Graph Size vs Time (b) Memory

Fig. 6: LANL Memory Characteristics

Runtime. Figure 7 shows how long the system took to run one analysis iteration per every hour of the dataset. In general, the analysis runtime is short, less than 3 seconds of analysis per one-hour of real-time data that has elapsed. There is an initial increase in runtime in the first few days as new activity is aggregated into the Cyber Activity Graph, and after this point the runtime only shows a slight trend upwards. This again indicates that this technique can scale to a much larger network with ease. We emphasize that at no point does the time required for computation exceed the amount of wall-clock time that has passed, meaning our approach is indeed usable for real-time analysis.

V. Limitations

a) False Positives: As with other techniques, false positives tend to be one of the biggest issues in threat detection. However, NetHawk effectively tries to quell false positives

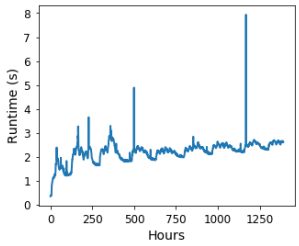


Fig. 7: Runtime Analysis

through various techniques, such as the *Neighbor Similarity Scaling*, and connected component signal amplification. Nevertheless, it may be beneficial to deploy NETHAWK in conjunction with a SIEM, such as Elastic, to verify anomalous edges and lower false positive scores.

b) Slow and Evasive Threat Actors: Threat actors that act extremely slowly may be more difficult to detect. Since we must set the anomaly half-life parameter, threat actors can try to evade detection by acting much more slowly than expected. While this is possible, it is unlikely, as multiple actions must be performed when the threat actors log in or perform malicious activity. Threat actors can also attempt to model their malicious activity as closely as possible to benign system activity in order to dodge detection, as their activities would be less likely to be marked by NETHAWK. In these cases, as we mentioned earlier, pairing NETHAWK with a SIEM would aid in detection of evasive threats.

c) Hyperparameter Tuning: In order for NETHAWK to provide robust results, additional analysis is required to select optimal hyperparameters for different environments. This entails the knowledge of the enterprise network and deployment strategies, which may take some time initially to setup. Testing would also need to be performed in order to ensure system robustness. It is important to note that while it may be nontrivial to correctly choose the hyperparameters, other approaches, such as signature-based rules, also require tuning, and in many cases rule modification must be performed for hundreds, or thousands of different rules.

VI. RELATED WORK

The vast majority of commercial and open-source tools for enterprise security rely on signature-based detentions. Snort [5] and Suricata [38] are two popular open-source IDSs, but they both require rules that define what malicious activity looks like and thus will always be vulnerable to APT circumvention. On the other hand, Endpoint Detection and Response (EDR) systems, such as the open-source OSSEC [39], run software on hosts and can detect malicious activity and some behavioral anomalies. Unfortunately, they often lack the big picture visibility across the full enterprise and are also highly dependent on signatures. Security Information and Events Management Systems (SIEMs) such as Splunk [29] or ELK [30] aim to collect and correlate activity across an enterprise, however, they are also typically rule-driven when it comes to detecting malicious activity.

In the academic space, many works model and detect malicious activity via provenance graphs or causal analysis of host events as we did. Poirot [1] and Holmes [2] model attacker behavior as a query graph and search for attacks on a system via the kernel audit records. RapSheet [21] is another technique where attacker activity is modeled as a graph and identified in a host provenance graph. Unlike NETHAWK, these approaches, as well as many other provenance-based works [4], [40], [41], [42], focus on a set of pre-defined attacker behaviors on individual hosts.

Argus [10] is the current state-of-the-art anomaly-based IDS. They too model the network as a temporal graph, but their alerts are upon isolated edges, making interpretability difficult. Link prediction-based systems, in general, suffer from this issue [9], [43], [44]. The NoDoze system [20] is similar to our approach. They identify anomalous paths in host-based provenance graphs to triage alerts and produce useful alarms, but rely on domain-specific expert knowledge to generate anomaly scores. Similarly, PrioTracker [45] has a strong emphasis on temporal features and the rarity of edges in provenance graphs to generate anomaly scores. While they also provide explainable attack subgraphs, their approach has a greater focus on causality analysis, and forensics than live detection.

By comparison, the NETHAWK system models network-wide dynamics, as opposed to individual hosts, which allows for the system to provide a more complete account of malicious activity that spans multiple systems and services across an enterprise. Finally, due to the compact yet expressive representation of the Cyber Activity Graph, the NETHAWK system can monitor large-scale real-world enterprise computer networks with minimal memory and compute requirements.

VII. CONCLUSION

In this work we introduced NETHAWK, a system for monitoring and detecting Advanced Persistent Threats in enterprise computer networks based entirely on anomalous activity. We discussed the pitfalls of traditional signature-based approaches, and how behavioral detection techniques are the key to our future security. We described the Cyber Activity Graph and the Anomaly Graph data structures, and our anomaly detection algorithm based on structural and temporal graph anomalies. Then, we applied our system to two datasets representing real-world enterprise computer networks and demonstrated how NETHAWK can detect sophisticated malicious activity spanning multiple users and systems with high precision and recall. Further, we showed how our system can analyze large-scale networks with minimal memory and compute requirements. Importantly, we showed that this approach generates highly interpretable alerts. By correlating suspicious behavior and analyzing large connected components of anomalous activity, we can tell the story of an adversary traversing a system and can help security analysts end it.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their suggestions. This research was developed in part with funding from the Defense Advanced Research Projects Agency under agreement number N66001-18-C-4033 and National Science Foundation under grant 2127207. Distribution Statement A (Approved for Public Release, Distribution Unlimited).

REFERENCES

[1] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1795–1812, 2019.

[2] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 1137–1152, IEEE, 2019.

[3] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "Saql: A stream-based query system for real-time abnormal system behavior detection," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 639–656, 2018.

[4] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. Ciocarlie, V. Yegneswaran, *et al.*, "Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation," 2021.

[5] Snort, "Snort." https://www.snort.org/, 2021. Accessed:2021-01-16.

[6] Elastic, "Open security platform unifying siem, endpoint & cloud." https://www.elastic.co/security, 2022. Accessed: 2022-10-1.

[7] SigmaHQ, "Sigmahq/sigma: Generic signature format for siem systems." https://github.com/SigmaHQ/sigma, 2022. Accessed: 2022-10-1.

[8] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, pp. 1–22, 2019.

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

[10] J. Xu, X. Shu, and Z. Li, "Understanding and bridging the gap between unsupervised network representation learning and security analytics," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 12–12, IEEE Computer Society, 2023.

[11] I. J. King, X. Shu, J. Jang, K. Eykholt, T. Lee, and H. H. Huang, "EdgeTorrent: Real-time temporal graph representations for intrusion detection," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses*, pp. 77–91, 2023.

[12] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime provenance-based detector for advanced persistent threats," in *Proceedings 2020 Network and Distributed System Security Symposium*, Internet Society, 2020.

[13] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "Fast memory-efficient anomaly detection in streaming heterogeneous graphs," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1035–1044, 2016.

[14] FireEye, "Highly evasive attacker leverages solarwinds supply chain to compromise multiple global victims with sunburst backdoor." https://www.fireeye.com/blog/threat-research/2020/12/evasive-attacker-leverages-solarwinds-supply-chain-compromises-with-sunburst-backdoor.html, 2020. Accessed:2021-01-16.

[15] F. Barr-Smith, X. Ugarte-Pedrero, M. Graziano, R. Spolaor, and I. Martinovic, "Survivalism: Systematic analysis of windows malware living-off-the-land," in *Proceedings of the IEEE Symposium on Security and Privacy*, Institute of Electrical and Electronics Engineers, 2021.

[16] S. Axelsson, "The base-rate fallacy and the difficulty of intrusion detection," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 3, pp. 186–205, 2000.

[17] G. P. Spathoulas and S. K. Katsikas, "Reducing false positives in intrusion detection systems," *computers & security*, vol. 29, no. 1, pp. 35–44, 2010.

[18] T. Pietraszek, "Using adaptive alert classification to reduce false positives in intrusion detection," in *International workshop on recent advances in intrusion detection*, pp. 102–124, Springer, 2004.

[19] C.-Y. Ho, Y.-C. Lai, I.-W. Chen, F.-Y. Wang, and W.-H. Tai, "Statistical analysis of false positives and false negatives from real traffic with intrusion detection/prevention systems," *IEEE Communications Magazine*, vol. 50, no. 3, pp. 146–154, 2012.

[20] W. U. Hassan, S. Guo, D. Li, Z. Chen, K. Jee, Z. Li, and A. Bates, "Nodoze: Combatting threat alert fatigue with automated provenance triage," in *Network and Distributed Systems Security Symposium*, 2019.

[21] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1172–1189, IEEE, 2020.

[22] OpTC, "DARPA." https://github.com/FiveDirections/OpTC-data, 2021. Accessed: 2021-07-21.

[23] A. D. Kent, "Comprehensive, Multi-Source Cyber-Security Events." Los Alamos National Laboratory, 2015.

[24] P. Kumar and H. H. Huang, "GraphOne: A data store for real-time analytics on evolving graphs," *ACM Transactions on Storage (TOS)*, vol. 15, no. 4, pp. 1–40, 2020.

[25] P. Gao, X. Xiao, Z. Li, F. Xu, S. R. Kulkarni, and P. Mittal, "Aiql: Enabling efficient attack investigation from system monitoring data," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pp. 113–126, 2018.

[26] B. Bowman and H. H. Huang, "Towards next-generation cybersecurity with graph AI," *ACM SIGOPS Operating Systems Review*, vol. 55, no. 1, pp. 61–67, 2021.

[27] S. Noel, E. Harley, K. H. Tam, M. Limiero, and M. Share, "Cygraph: graph-based analytics and visualization for cybersecurity," in *Handbook of Statistics*, vol. 35, pp. 117–167, Elsevier, 2016.

[28] Y. Jia, Y. Qi, H. Shang, R. Jiang, and A. Li, "A practical approach to constructing a knowledge graph for cybersecurity," *Engineering*, vol. 4, no. 1, pp. 53–60, 2018.

[29] Splunk, "Splunk." https://www.splunk.com/, 2021. Accessed:2021-01-16.

[30] Elastic, "Elastic." https://www.elastic.co/, 2021. Accessed:2021-01-16.

[31] Apache, "Kafka." https://kafka.apache.org/, 2021. Accessed:2021-01-16.

[32] Microsoft, "Overview of endpoint detection and response." https://docs.microsoft.com/en-us/microsoft-365/security/defender-endpoint/overview-endpoint-detection-response, 2021. Accessed:2021-01-16.

[33] Microsoft, "sysmon." https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon, 2021. Accessed:2021-01-16.

[34] Microsoft, "Windows events." https://docs.microsoft.com/en-us/windows/win32/events/windows-events, 2021. Accessed:2021-01-16.

[35] Linux, "auditd." https://man7.org/linux/man-pages/man8/auditd.8.html, 2021. Accessed:2021-01-16.

[36] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[37] TC3, "DARPA." https://github.com/darpa-i2o/Transparent-Computing, 2020. Accessed: 2022-10-1.

[38] Suricata, "Suricata." https://suricata.io/, 2021. Accessed:2021-01-16.

[39] OSSEC, "Ossec." https://www.ossec.net/, 2021. Accessed:2021-01-16.

[40] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "Sleuth: Real-time attack scenario reconstruction from cots audit data," in *26th USENIX Security Symposium (USENIX Security 17)*, pp. 487–504, 2017.

[41] W. U. Hassan, L. Aguse, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *Network and Distributed Systems Security Symposium*, 2018.

[42] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications.," in *NDSS*, 2020.

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

[44] J. Khoury, D. Klisura, H. Zanddizari, G. D. L. T. Parra, P. Najafirad, and E. Bou-Harb, "Jbeil: Temporal graph-based inductive learning to infer lateral movement in evolving enterprise networks," in *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 9–9, IEEE Computer Society, 2023.

[45] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a timely causality analysis for enterprise security.," in *NDSS*, 2018.