Carbon Filter: Scalable, Efficient, and Secure Alert Triage for Endpoint Detection & Response

Muhammad Adil Inam,†§ Jonathan Oliver,‡§ Raghav Batta,‡ and Adam Bates†
† *Siebel School of Computing and Data Science, University of Illinois at Urbana-Champaign*
‡ *Carbon Black, Broadcom Inc.*

***Abstract*—Endpoint Detection & Response (EDR) products detect threats by pattern matching endpoint telemetry against behavioral rules that describe potentially malicious behavior. However, EDR can suffer from high false positives that distract from actual attacks, leading to an "alert fatigue" problem. While provenance-based alert triage techniques have shown promise, historical provenance analysis is prohibitively slow when applied to the stream-based event processing pipelines that dominate industry today; provenance-based systems may take over a minute to inspect a single alert, while *individual EDR customers can face tens of millions of alerts per day*. At present, these approaches cannot scale to production environments.**

We present Carbon Filter, an automated alert triage mechanism that reduces false alerts by upwards of 82% and is already in use by thousands of Carbon Black EDR customers today. Our key insight is that the vast majority false alerts are triggered by programs that share a common initiation context, and thus the specific false alerts associated with an initiation context can be identified. However, rather than turning to costly provenance analysis, we hypothesize that it is sufficient to use the command line arguments of alert-triggering processes as the initiation context. Through prioritizing speed for similarity-preserving hashing, clustering, and search, we demonstrate that our approach scales to millions of alerts per hour (>5K/sec). In evaluations customer alert data, we demonstrate that Carbon Filter can identify 82% of false alerts nearly a 6-fold improvement in signal-to-noise ratio. Further, when comparing to provenance-based approaches, we show that Carbon Filter (AUC=0.94) actually outperforms NoDoze (AUC=0.60) and RapSheet (AUC=0.90) while reducing analysis time by 5,064x and 26,723x, respectively.

***Index Terms*—Intrusion detection, Cyber threat intelligence, Cyber attack**

I. Introduction

Endpoint Detection & Response (EDR)[1] products continuously record and store endpoint event activity, enabling security analysts to detect and hunt for threats [1]. EDRs are a staple of any enterprise security stack, representing a 3 billion dollar market [2] within the security product ecosystem. EDR detections are based on analyzing the event telemetry stream against potential attacks. AI is not typically relied upon for this critical task. Instead, EDRs employ a set of detection queries that were manually written by cyber analysts based on their review of past incidents or other threat intelligence. This pattern matching approach is extraordinarily efficient and scalable in the event data stream paradigm. By annotating queries with MITRE ATT&CK techniques [3], detections are also explainable, providing much-needed context to analysts.

However, EDRs are also a major contributor to the growing "alert fatigue" problem – security products are prone to generating high volumes of alerts. Industry reports observe that sufficiently-large organizations will face *at least* tens of thousands of alerts per day, the majority of which are false alerts [4], [5], [6]. Among Carbon Black[1] customers, some organizations generate *tens of millions of alerts* of alerts per day; in an extreme case, a large customer with many tens of thousands of endpoints faced *38 million to 82 million alerts each day* over a one week period. This forces analysts to identify attacks from an information stream where the majority of alerts are low in severity and high in volume, making it impossible to examine all alerts [7], [8]. The reasons for poor signal-to-noise ratio are numerous. Notably, attackers have become adept at leveraging legitimate system utilities, so-called "Living-off-the-land" attacks [9], blurring the boundaries between malicious and benign activity. MITRE ATT&CK [3] is also populated with misuse of legitimate programs, such that many attack techniques are difficult or impossible to distinguish from normal activity [10], [11].

Encouragingly, recent work has begun to recognize *automated alert triage* – methods of (de)prioritizing alerts based on their perceived risk – as an important security task. Most notably, data provenance analysis has proven effective at identifying many of the false alerts in an alert stream [12], [13], [14]. Data Provenance is a general approach to auditing that captures causal dependencies between entities in a computing system, with applications to intrusion detection, threat investigation, and beyond. In the case of alert triage, provenance-based approaches analyze the historical context of the process that caused an alert to fire, examining the suspiciousness of past activity to determine the suspiciousness of the current execution context. On small evaluation testbeds (tens to hundreds of machines), provenance-based triage has been shown to be capable of filtering out 84% [12] to 97% [13] of false alerts, a tremendous improvement in signal-to-noise ratio. Unfortunately, provenance-based triage is not nearly scalable enough for widespread deployment in commercial EDR products. Because provenance needs to retrieve and process many historic events for triage, it typically takes tens of seconds (if not minutes) to analyze a single alert [12], [13].

[1]The "EDR" described and used in this work is actually an eXtended Detection & Response (XDR) product that supports queries over endpoint events, authentication events and network events. We use the better-known EDR term; the EDR/XDR distinction is not relevant to this paper.

§Equal contribution.

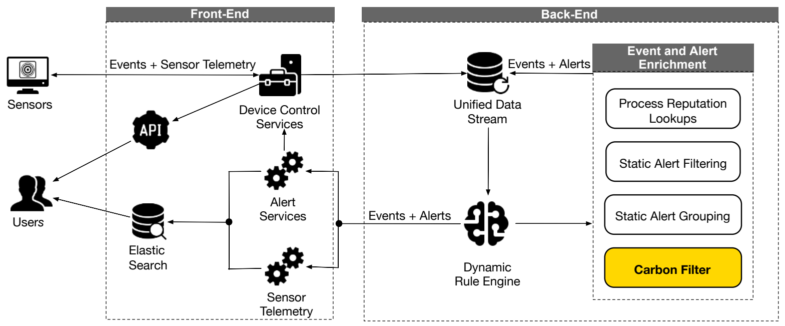


Fig. 1: A simplified cloud-based EDR system based on the Carbon Black data collection architecture.

While this is not an intrinsic limitation of provenance analysis – for example, prior work has observed that provenance can be analyzed extremely efficiently in a vertex-centric graph streaming paradigm [15] – achieving such efficiency requires rearchitecting today's products, which are deeply entrenched in tabular and event data stream representations. What is needed is a technique that approaches the efficacy of provenance-based triage that is natively compatible with today's non-graphical streaming event analysis engines.

Within the Carbon Black EDR/XDR product, we observe that the vast majority of false alerts are issued by "well-formed" processes – programs whose execution has been automated by the operator, or are otherwise share a common *initiation context* across numerous users and organizations. Further, the well-formed processes that are prone to triggering false alerts will consistently match against the same detection queries. While such detections may seem suspicious from the perspective of an individual endpoint or organization, the sheer volume of detections seen from the vendor's perspective provides strong assurance that they are false alarms. Provenance analysis would certainly be able to identify these noisey initiation contexts, but is prohibitively costly at present. Instead, we hypothesize that initiation contexts can be effectively clustered by examining the command line arguments of processes that trigger alerts. Because it is already common practice to record and match against command lines in EDRs [16], this approach is a candidate for immediate deployment.

We distill these insights into Carbon Filter, a highly scalable and secure automated alert triage system. Carbon Filter performs similarity-preserving hashing of every process command line that triggers an alert, then identifies large dense clusters with nearly-identical command lines. A profile of the alerts associated with each cluster is then built. To triage alerts, Carbon Filter hashes the command line of the offending process and then performs nearest neighbor search of clusters associated with that alert type. Distance in the embedding space forms the basis for alert prioritization. To prevent abuse of this mechanism, Carbon Black continuously monitors clusters for evidence of evasion and poisoning by profiling additional features (e.g., process path, username, privilege level) of the triaged processes and their parents. By prioritizing efficient analysis, we are able to triage $\sim 5.5K$ alerts per second on a single multi-core machine.

Our evaluation is based on hundreds of millions of alerts generated by Carbon Black customers, including over 10,000 labeled alerts that were verified by professional analysts within Carbon Black. To validate our hypothesis that command lines are an effective indicator of initiation context, we first demonstrate that 102 million customer alerts map to just $763K$ initiation contexts (134x reduction), which Carbon Filter reduces to a compact representation of $16K$ clusters (6259x reduction). We find that Carbon Filter is able to filter out 82% – 84% of false alerts while mis-classifying 0% – 0.1% of malicious alerts, resulting in a 6-fold improvement in the signal-to-noise ratio of the alert stream. Further, in a comparison to the NoDoze [17] and RapSheet [13] systems on a public dataset, we find that Carbon Filter actually outperforms provenance-based triage (0.94 AUC compared to 0.6 and 0.9 AUC, respectively) while enjoying up to a 26,723x speed boost. By acting as an efficient prefilter that removes the majority of false alerts, we demonstrate a synergistic relationship between Carbon Filter and costlier provenance-based triage techniques. *Carbon Filter is already deployed in Carbon Black products to help customers triage threats.*

The key contributions of this paper are as follows:

- *Triage millions of alerts per hour.* We demonstrate a scalable method of automated alert triage through granular clustering of command lines.
- *Global-scale Evaluation.* We demonstrate the efficacy of our approach on a global snapshot of Carbon Black customer data comprised of hundreds of millions of alerts.
- *Open Evaluation Comparing to Prior Work.* We replicate our results on the open-source ATLASv2 dataset, comparing our approach to the NoDoze [12] and RapSheet [13] provenance-based triage techniques. We will release our scores for the ATLASv2 alerts upon publication.

The rest of this paper is organized as follows. Section II gives an overview of EDR architectures and the limitations of current alert management strategies. Section III presents the design and implementation of Carbon Filter. Section IV evaluates the performance of Carbon Filter. We provide additional commentary on related work in Section V, discuss future directions in Section VI, and conclude in Section VII.

II. Background & Motivation

A. Endpoint Detection and Response

Endpoint Detection and Response (EDR) products continuously monitor and record the event and network telemetry on end-user devices to detect and promptly respond to cyber threats. Expert-defined behavioural rules are applied to this event telemetry to generate alerts. Cyber analysts can then investigate alerts by querying the underlying event stream to verify attacks and craft appropriate responses. Analysts can also threat hunt by directly querying the event stream.

A high level overview of a cloud-based (i.e., SaaS) EDR is given in Figure 1, based on the architecture of the Carbon Black EDR. It is structured such that the front-end interface is accessible to customers through APIs, while the complexities of the backend remain invisible to end-users. We go over the individual components of the architecture in detail below.

Endpoint sensors are deployed on target machines, where generate a continuous stream of event telemetry. The sensors may also perform additional monitoring, such as scanning new executable content on the endpoint (i.e., antivirus), but this is not the focus of our work. The event streams generated by the sensors are passed to the device control service, where specific device-based configurations are applied to the event stream. After being authenticated, the updated event stream, augmented with device-specific metadata, is transmitted to the backend infrastructure in a binary compressed format.

The event stream proceeds to the Unified Data Stream (UDS), which serves as the data processing platform at the backend. From the UDS, the event stream is sent over to the Dynamic Rule Engine (DRE). The DRE, referred to as the ”backend brain”, is responsible for applying state-of-the-art expert-defined behavioral rules to the event telemetry. The DRE is responsible for comparing the event stream to deployed behavioral rules, making throughput of paramount importance. The Carbon Black DRE processes trillions of events per second.

Detection queries, i.e., behavioral rules, are founded on the concept of ”state changes” as described in MITRE ATT&CK [3]. This enhances the resilience of the system to evasion tactics as compared to rules based on procedural descriptions of actions. For instance, technique T1037.004 [18] establishes persistence by modifying RC scripts referenced during UNIX startup. The behavioral EDR rule is defined as follows:

```
IF File-modified(/etc/rc.d, /etc/init.d,
    /etc/rc.local)
    AND Is-Platform(Linux, macOS)
THEN FIRE
```

Irrespective of the method employed to acquire the necessary privileges for this state change, any events involving alterations to these files trigger alerts. The DRE also incorporates various event and alert enrichment techniques into the event telemetry, such as process reputation lookups, static alert filtering, and static alert grouping.

Following the application of behavioral rules and alert enrichment techniques, the resultant security alerts are stored within the Carbon Black Cloud (SHC) and transmitted to the alert and sensor telemetry services at the front-end. If a known malicious process is identified on the endpoint, alert services notify the endpoint sensor to terminate the process. Moreover, analysts operating at the front-end of the infrastructure can utilize Elastic Search to query and retrieve specific alert or telemetry data for further analysis.

B. Limitations of Prior Approaches

It is common practice for EDRs to employs event and alert enrichment techniques in order to enhance alert metadata and mitigate false alerts. Widely practiced strategies include process reputation lookups, and static alert filtering, and static alert grouping. Additionally, we discuss data provenance and causality-based approaches for alert triage [12], [13], [14].

Process Reputation Lookups - EDR products assess the reputation of a running process, considering factors such as the reputation of the process's primary executable and its parent process. Typically, process reputation lookups rely on a repository of known processes and malware signatures. This methodology may identify known attack signatures but struggles with more sophisticated threats.

Static Alert Filtering - Due to high alert volume, EDR products incorporate predefined rules to filter out false alerts. EDR products may modify and adjust existing behavioral rules or introduce new filtering rules based on the ongoing alert generation patterns. This approach is often reduces to “turning off” high volume detection queries. Static alert filtering can reduce the volume of false alerts. It is especially necessary in organizations with an under-provisioned security teams. However, it also introduces the risk of missed detections.

Static Alert Grouping - EDR and SIEM products have long offered alert correlation and clustering features based on “static” grouping across alert and event attributes, such as originating hosts, usernames, alert types, or severity levels. In some cases, alert grouping can serve effectively as a triage technique by allowing analysts to consider the cluster of alerts as a whole rather than individual detections.

Provenance-based Approaches - Provenance-based strategies delve into the historical causal relationships between processes [12], [13], [14]. Provenance can triage EDR alerts by examining the causal relationships that caused the alert-triggering process to execute, but retrieving these historical events can be costly. While these approaches have shown promise in small evaluation testbeds, their scalability in commercial EDR systems remains a significant challenge. Systems like Kairos [19] represent a notable advancement in processing speed. When processing events in 15 minute batches, the authors observe an effective throughput of 11k events per second. However, Kairos is a detection mechanism, not an automated triage solution for EDR. The performance of Kairos' as compared to commercial event streams, which may process trillions of events per second, remains unclear.

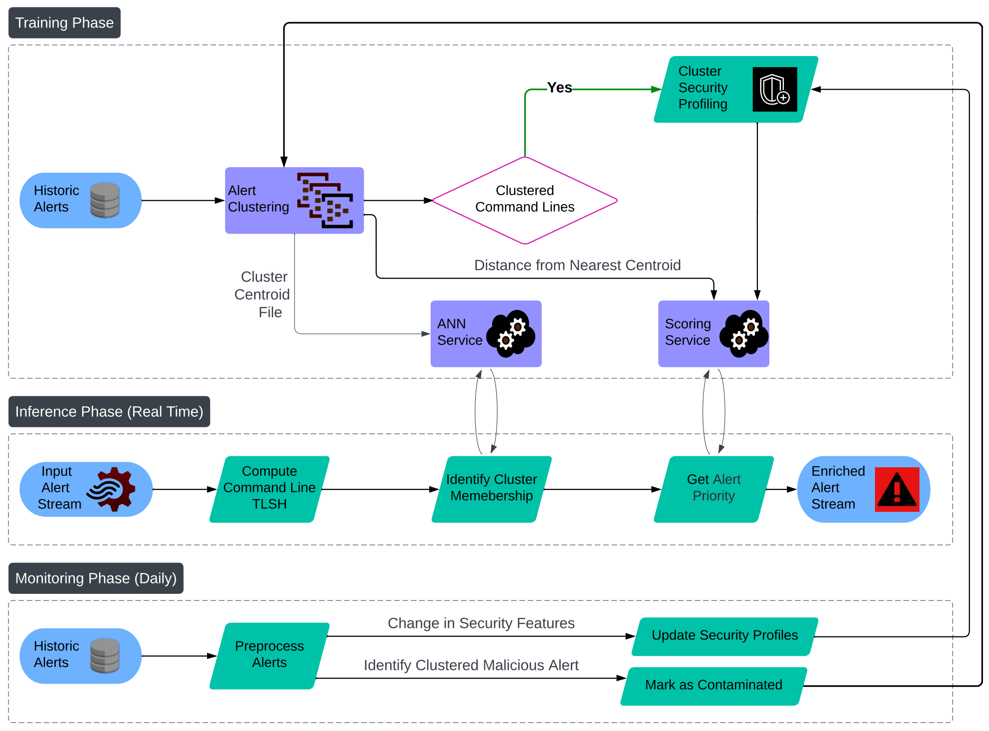


Fig. 2: Carbon Filter's training, inference, and monitoring phases.

III. Carbon Filter Methodology

A. Threat Model & Assumptions

This work considers an enterprise security challenge from the global perspective afforded a security software vendor. We assume the presence of a cloud-based EDR product like the one described in Section II. This including endpoint sensors installed on customer machines that collect and transmit event telemetry, as well as a distributed rule-based detection architecture. However, the EDR emits a high volume alerts of which many are false alarms, taxing the resources of individual clients' Security Operations Centers (SOC's) and creating the risk of a true alert going unnoticed or uninvestigated.

Like other work in this space (e.g., [17], [13]), we assume that the distributed EDR architecture is not compromised. EDR evasion is an important consideration (e.g., [20]), but is outside the scope of this work. Similarly, we do not consider attacks on log integrity [21], [22], [23], [24]. However, we do consider the possibility that the attacker will attempt to abuse the automated alert triage system in order for legitimate detections to go unnoticed. Most notably, we assume that the adversary is capable of launching living-off-the-land attacks that cause legitimate programs to engage in malicious activities, and that the attacker is able to achieve this without causing a change to the programs' command line arguments.

B. Design Goals

We set out to design an automated alert triage system that satisfies the following requirements.

1) **Highly Scalable.** Carbon Black sensors running on customer endpoints can generate tens of millions of alerts per day, a volume that far exceeds the capacity of prior triage approaches in the literature. Our solution must be able to support massive streams of alerts during both training phase and inference (triage) phase.
2) **Effective with Limited Labeled Data.** Scarcity of labeled data is a major challenge for the cybersecurity industry, especially when leveraging machine learning [25]. While it is possible to have SOC analysts manually triage alerts for labeled validation data, this is a labor intensive task that does not scale. Our solution must therefore effectively filter out false alerts from the alert stream with access to only limited labeled samples.
3) **Resilient to Class Imbalance** The majority of the activity that happens on endpoints is legitimate and benign, with attacks occurring very rarely. This creates the conditions for high false alert rates that lead to alert fatigue, a problem that spans across the majority of organizations in the security industry [26]. In addition, this also leads to the potential for bias in ML models trained on alert data. Our solution must be able to effectively differentiate false alerts from malicious alerts in spite of this class imbalance.

Command Line	TLSH Label	T1 Dist	T2 Dist	T3 Dist
`"C:\Windows\System32\rundll32.exe" shwebsvc.dll, AddNetPlaceRunDll`	T1	0	371	380
`"C:\WINDOWS\System32\WindowsPowerShell\v1.0\ powershell.exe" ((New-Object System.Net.WebClient).OpenRead('https:\\www.google.com')).CanRead`	T2	371	0	23
`"C:\WINDOWS\System32\WindowsPowerShell\v1.0\ powershell.exe" ((New-Object System.Net.WebClient) .OpenRead('https:\\www.microsoft.com')).CanRead`	T3	380	23	0

TABLE I: Examples of command lines and the TLSH distances between those command lines.

C. Overview

An overview of the Carbon Filter architecture is shown in Figure 2. As an alert enrichment mechanism, Carbon Filter interposes on Unified Data Stream (UDS) presented as part of the exemplar EDR (Figure 1), denoted in the figure by the "Input alerts stream" and the output "Enriched alerts stream." The key components of Carbon Filter are as follows:

Training Phase (§III-D) - During a training phase, historic alerts are collected from the S3 [27] data storage. The command line for the process that triggered the alert are pre-processed and vectorized using a computationally-efficient similarity-preserving hash (TLSH) [28]. The HAC-T clustering algorithm [29], which is optimized for fast search, then identifies clusters. Once clusters are identified, baseline statistics of 28 security features are extracted from each cluster.

Inference Phase (§III-E) - During the inference phase, Carbon Filter leverages Apache Flink [30] as a scalable streaming framework. The real-time alerts are transferred to the Flink pipeline from UDS via AWS Kinesis [31] streams in the form of Protobuf objects [32]. The alerts are pre-processed, TLSH is computed for their command lines and the *Approximate Neareast Neighbor (ANN) Service* is queried to identify the cluster ID for the alert. Once the cluster ID is determined, the enriched alert is sent via a post request to the *Scoring Service* which returns the score for both the clustered and unclustered alerts. The enriched protobuf objects are then transferred back to the UDS Kinesis stream.

Monitoring Phase (§III-F) - To prevent evasion attempts and adapt to changes in attack patterns over time, the Carbon Filter cluster model is continuously reviewed by internal analysts at Carbon Black. We have also introduced a feedback mechanism in the Carbon Black EDR interface to receive tips from customers on missclassified alerts.

D. Training Phase

1) Extracting Process Initiators: We observe that the vast majority of false alerts are issued by "well-formed" processes – programs whose execution has been automated by the operator, or are otherwise share a common *initiation context* across numerous users and organizations. Further, well-formed processes consistently trigger the same alerts because their behavior consistently matches the same detection queries. Provenance analysis is capable of expressing initiation contexts – for example, the lineage of system entities that caused a processes' execution – but is prohibitively costly. Instead, we predict that a processes' command line arguments [2] can be used to extract a process's initiation context. Intuitively, command line arguments inform the behaviors of the program execution, and at times will directly map to causal dependencies found in a provenance graph. For example, the command line `vim paper.tex` will result in the provenance dependency $paper.tex - read \rightarrow vim$. Further, living-off-the-land attacks will require the attacker to manipulate a process so that the command line is identical to legitimate activity and the process behavior has changed. We detect such attacks by examining the process behavior across a spectrum of behaviors. However, command lines have the advantage of being on hand in the EDR alert, as opposed to requiring the retrieval of historic events.

At the same time, analysis of command lines presents challenges. A naïve approach would be to group command lines that are perfect matches for one another, but this fails to account for how various sources of nondeterminism can affect semantically-equivalent process command lines. For example, a logging procedure may reference a timestamped output file on each execution, or a process accessing/generating cached data may specify a randomized path value. In order to identify commonality between distinct command lines in a generic and scalable way, we turn to similarity-preserving hashing. Popular similarity digests include Ssdeep [33], Sdhash [34], LZJD [35], TLSH[28], etc.. We rule out a number of candidates that may not be fast enough keep pace with the alert stream of a commercial EDR (e.g., [36], [37]). Further, as command lines are much shorter than other digest targets, we must also consider support for small input lengths. Ssdeep and Sdhash are excluded for this reason; with minimum input lengths of 4096 bytes and 512 bytes, respectively.

We ultimately choose TLSH for its high throughput and support for smaller inputs. TLSH has also been shown to be effective in related security tasks [29]. Before computing the TLSH of a command line, we perform minimal preprocessing to remove known sources of non-determinism. Specifically, we use regular expressions to identify globally unique identifiers (GUIDs), timestamps, and cryptographic hashes, then replace them with constant strings of the same length.

An example of TLSH on process command lines is given in Table I. Three example command lines are shown; as the actual TLSH hash is more than 70 characters long, we replace

[2]There are multiple possible initiators for a process. The most common initiator is the command line which started the process. Other common initiators include a user a user clicking an icon on the desktop, process creation from within another process and scheduled tasks. We construct a 'prox' command line for these initiators.

the digests with labels T1, T2 and T3. T1 is known to be a suspicious command based on feedback from expert analysts. In contrast, T2 and T3 are legitimate use command lines that vary in URL. It can also be seen that the command lines for T2 and T3 are very similar to each other, while T1 is quite different. This observation is confirmed by comparing the distance between TLSH digests, where the distance between command lines T2 and T3 (23) is an order of magnitude less than their respective distances from T1 (371 and 380).

We experimented with implementing the TLSH distance function [38] using both the Numba [39] and Cython [40] packages. We chose to make use of a Numba-based implementation because it achieved a 2x speed up compared to the Cython-based implementation.

2) Clustering Process Initiators: Having established a mechanism for measuring distance between process initiators, we now seek to cluster similar initiators for alert-triggering processes. While clustering is an extremely common task, our requirements differ meaningfully from other applications. While other clustering models are used for best-effort identification of class labels, anticipating variance between class instances, the goal of our clusters will be to admit instances that are syntactically near-identical to prior instances. This is because it is far safer for a process initiator to fall outside of all clusters – and therefore be considered as an actual alert – then to force it into a cluster, thereby risking the pruning of a true alert. In other words, we seek to define a model that defines *extremely low entropy* clusters, even at the expense of many samples falling outside of all clusters. The need for low entropy applies not only to the command line digests themselves, but also other security attributes associated with the alert, for reasons that will become clear in Section III-F.

An additional advantage of defining a clustering model is improving scalability during the inference phase. During inference, we need to compare new process initiators to our training set to identify known false alert sources. A naïve approach would compare a new process initiator to hundreds of thousands of raw TLSH values in our training set. This number of comparisons is very expensive given expected alert volumes. A clustering model allows us to dramatically reduce the number of comparisons required.

While a variety of clustering algorithms would be effective for this task (e.g., DBSCAN [41]), we choose the Hierarchical Agglomerative Clustering (HAC-T) algorithm [29] using the TLSH distance function. The output of HAC-T clustering is a list of cluster centroids (in the form of a TLSH), the radius of the cluster and number of points that belong to each cluster. HAC-T stops growing a cluster when the radius of the cluster is greater than a distance threshold parameter ($CDist$).

The optimal value for $CDist$ is a trade-off between the number of clusters and the homogeneity within each cluster. Our extreme performance requirements heavily incentivize us to define a clustering model that contains a smaller number of clusters, as this will reduce the number of distance comparisons required in the inference phase. At the same time, our security requirements demand that clusters be homogeneous (low entropy) to minimize the likelihood that a truly malicious alert is identified as noise. We explore this tradeoff and identify an optimal $CDist$ threshold through analysis of a global snapshot of Carbon Black alert data in Section IV.

E. Inference Phase

In the Inference Phase, we first calculate the TLSH digest of the new alert-triggering process' command line before searching the clustering model. The search function will calculate the distance between the new digest and the nearest cluster that is associated for triggering alerts of the same type. If the distance falls within $CDist$ of the cluster, the new alert is a candidate for identification as noise.

At time of writing, Carbon Black customers trigger an average of 921 alerts per second ($\sigma = 396$); there are also bursts in which alerts per second spikes above 10k. This underscores the need for a scalable approach to compare the alert stream to our clustering model in real time. We therefore choose make use of an Approximate Nearest Neighbor search algorithm function that supports the specification of custom distance functions (TLSH). ANN methods typically make use of either tree-based data structures (e.g., ANNOY [42]) or neighborhood graphs (e.g., HNSW [43], NN-Descent [44]). We are largely agnostic to this distinction, choosing instead to consider approaches that perform best for recall and search in in benchmarking [45], which includes NN-Descent[44], NGT[46] and HNSWlib[43].

We select the NN-Descent (PyNNDescent[47]) for its strong recall, scalability, and flexible ease of use. NN-Descent for example constructs a nearest neighbor graph and then performs search for approximate nearest neighbors. This graph is built by connecting each data point to k-most similar points from the cluster centroid dataset. It act as an index structure over which we perform breadth-first search and only keeping the best k-points at any points in the traversal. NN-Descent will reduce per-alert inference time in our production system, improving centroid look-up speed from linear time to logarithmic. It also improves the modularity of our architecture; for example, if we choose at some time in the future to replace TLSH vectorization with a neural-network based embedding function, we can do so while leaving the overall architecture intact.

F. Monitoring Phase

Carbon Filter introduces an automated alert triage system to the EDR that allows the possibility of alerts to be pruned without analyst review. This creates a new attack vector in which an adversary might attempt to perform malicious acts via a well-formed process that falls within one of our clusters. There are two possible scenarios where this might occur. The first is active manipulation – an attacker crafts a command line process initiator such that a semantically-unlike process behavior is steered into a cluster, or simply relabels malware to match the name of a clustered program. The second is that an attacker is able to make malicious use of a clustered well-formed process without manipulating the command line. We address the first concern by *profiling additional security* *features* associated with a cluster. We address the second concern by *tracking contaminated clusters*.

1) Profiling Additional Security Features: When a cluster is identified during cluster training, we also construct a profile of 28 additional security features of the alerts that fall within the cluster. We track the MITRE ATT&CK Tactics and Technique annotations attached to the alert. We also track numerous features of the alert-triggering process and its parent, including executable path, reputation, username, privilege level, integrity level, and digital signature publisher. Finally, we include other statistical features of the alert-triggering process as number of network connections, registry modifications, file modifications and module loads. Like the command line itself, we expect that all of these features will exhibit extremely low entropy among well-formed tasks that fall within a cluster.

During the inference phase, we perform outlier analysis over these features. We test the categorical features against a singular (or a small set of values) within the cluster profile to establish consistency. Similarly, we test the numerical features against the respective bounds of the cluster profile. If these features vary from the expected values we interpret this as an indicator of attack, causing the alert to be passed to analysts with a high priority score. Thus, an alert will only by classified as a false alert when all the security features are consistent with the baseline defined by the alert's associated cluster.

2) Tracking Contaminated Clusters: On the other hand, it may be possible for an attacker to make illegitimate use of a well-formed process cluster. This is especially true as attack patterns change over time, causing a "known" false alert to suddenly become a true alert. Moreover, an attack technique that occurs with high enough frequency may even warrant the formation of its own cluster should it be admitted into our training data. In both cases, it is necessary to continuously monitor the correctness of clusters for evidence of *contamination*, a true alert falling within a known cluster.

In addition to continuous review by our own internal expert analysts, we have introduced an explicit feedback mechanism for customers to signal that an alert has been misclassified. If a malicious alert is found to be present in a cluster, it is immediately marked as contaminated at which point all future alerts that fall within the cluster proceed to the analyst. While this introduces a "manual" step into our automated triage pipeline, the required human efforts differ from the status quo approach by orders of magnitude. Previously, individual customers were wasting dozens of hours of analyst time per week investigating false alarms [6], [48]. Under Carbon Filter, the limited manual effort required has been redistributed primarily to internal teams within Carbon Black. Rather than inspecting millions of individual alerts, these analysts instead monitor the composition of thousands of clusters. Further, if a customer assists in identifying a misclassification, it is a one-time effort that immediately benefits all other customers.

G. Security Analysis

We now provide a holistic argument for Carbon Filter's security. We limit our consideration to the unique attack surface presented by Carbon Filter. We do not address underlying issues, such as EDR False Negatives or tampering, which are important but outside the scope of this work. Specifically, we will demonstrate that Carbon Filter does not increase the likelihood that a threat goes unnoticed or uninvestigated by enumerating all possible outcomes for a single malicious alert.

Case 1: No Matched Cluster. If a malicious alert passes through Carbon Filter without matching a cluster, it is passed to the analyst just like before.

Case 2: Matched Cluster, Wrong Process/Alert. If the malicious alert falls within $CDist$ of a cluster, but the cluster is associated with a different process name or a different alert type, Carbon Filter still passes the alert to the analyst.

Case 3: Matched Cluster, Wrong Security Features. If the malicious alert falls within $CDist$ of a cluster that matches the process name and alert type, it must also fall within the acceptable range for 28 additional security features in order to be classified as a false alert. The features include the location of the executable, username, privilege level, publisher signature, behavioral features for both the alert-triggering process and its parent. Thus, an attacker that renames malware to match a cluster's well-formed process' name, or runs an unsigned copy of malware at the trusted process' path location, will be trivially detected. An attacker that runs a well-formed process from the wrong account or at the wrong privilege level will be detected. An attacker that invokes the well-formed process in a way that changes the statistical behavioral features of the process or its parent will be detected. This places extraordinary restrictions on the attacker's ability to abuse the triage mechanism.

Case 4: Matched Cluster, Matched Security Features. In extraordinary cases, it may be possible for an attacker to match a well-formed process cluster, its name and alert type, and fall within acceptable range for all other security features. For example, a critical zero-day vulnerability for remote code execution could be discovered for a well-formed process. However, not only will this case be infrequent, but the attacker is still constrained only to the ATT&CK techniques (alerts) that were associated with the benign behavior of the well-formed process. The alert that, without Carbon Filter, would be triggered hundreds or thousands of times per day by the benign process and thus would be "tuned out", unlikely to provide significant insight into the presence of the attack. If the attacker uses the exploit to engage in more aggressive malicious activity, it will no longer match the cluster's features / techniques and will be immediately passed to the analyst. Further, as internal analysts are continuously monitoring alert samples from the clusters, or as customer feedback or other threat intelligence becomes available, the cluster will be marked as contaminated and future exploits will not be filtered.

The above arguments are predicated on Carbon Filter's ability to build a clustering model of extremely low-entropy process initiator clusters that admit only benign well-formed process activities. Higher variance in clusters may increase the risk that malicious activity could be admitted and pruned without human review. In Section IV, we evaluate Carbon

Filter's ability to achieve secure clustering.

IV. Evaluation

We now set out to evaluate the performance and efficacy of Carbon Filter, specifically considering the following questions:

RQ1 What is the optimal parameterization of Carbon Filter's clustering algorithm?

RQ2 Does Carbon Filter's clustering model reduce the number of process initiator comparisons required for triage?

RQ3 How effectively can Carbon Filter perform triage alerts?

RQ4 Can Carbon Filter scale to the alert volumes of a commercial EDR environment?

RQ5 How does the efficacy and performance of Carbon Filter compare to provenance-based triage techniques?

Unless otherwise noted, all experiments were conducted on an AWS hosted EC2 machine with Intel(R) Xeon(R) Platinum 8175M CPU @ 2.50GHz, L1d cache: 32K, L1i cache: 32K, L2 cache: 1024K, and L3 cache: 33792K.

A. Experimental Data

We make use of the following data.

Unlabeled Alerts Dataset: We train our primary clustering model on a full month of unlabeled customer data comprised of $\sim$ 102 million alerts, To do obtain alert data with ground truth labels for evaluation purposes, we enlisted the help of two expert analyst groups with Carbon Black.

TAO Dataset: Carbon Black's Threat Analysis Org (TAO) creates and maintains the detection rules for the Carbon Black EDR. Over a 3 month period, we collected $7,193$ alerts from the alert stream and asked TAO to label them. Due to the high class imbalance between malicious and false alerts, randomly sampling the alert stream would not have yielded a sufficient number of true alerts. To address this challenge, we instead adopted a greedy approach to collect alert feedback. We leveraged TAO team's domain knowledge to identify rules that rarely triggered but had a higher probability of identifying malicious activity. We then sampled half of the alerts from this set and the other half randomly from the alert stream. To minimize human error, we had each alert reviewed by multiple TAO analysts and removed alerts that had conflicting annotations. 74 of the $7,193$ alerts were labeled malicious.

MDR Dataset: Carbon Black's Managed Detection and Response (MDR) team actively triages alerts for customers and works with them to prevent security attacks. Unlike TAO, we did not ask MDR for labels specifically for this project; instead we passively collected investigation outcomes for $2,822$ alerts, of which 88 were designated malicious. Here, we did not need to implement a greedy sampling approach as the MDR analysts only review customer alerts above a specified threat score.

ATLASv2 Dataset: Because the above is proprietary and cannot be released, we also make use of a publicly available dataset. The *ATLAS Attack Engagements, Version 2* (ATLASv2) dataset [49] represents an enriched version of the ATLAS engagement initially conducted by Alsaheel et al [50]. The ATLASv2 dataset features an enhanced methodology that includes an four-day period of naturalistic benign background activity, a one-day attack period in which benign activities continue to occur, and the integration of additional telemetry sources, notably an EDR sensor. In total, this dataset consists of 491 alerts, among which 50 are categorized as malicious alerts. We generated Carbon Filter cluster labels for the ATLASv2 dataset and will release them upon publication.

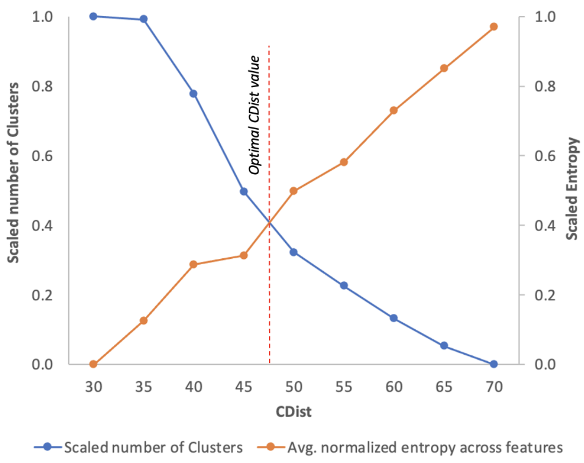


Fig. 3: Relationship of *CDist* with the number of clusters and average entropy across behavioral features.

1) Bias Correction: The feedback on alerts from the TAO team and MDR analysts predominantly skewed towards the malicious class due to the nature of their work. As a result, their distributions do not accurately reflect the broader population of alerts, potentially biasing our results. To mitigate this, we perform bias correction to more closely represent the distribution of alert classes within the general population. We duplicated specific alerts in the MDR and TAO datasets in proportion to their real-world occurrence rates, as illustrated in Table II. Following bias-correction, the representation of malicious alerts significantly decreased to 0.15% and 1.2% for TAO and MDR data, respectively, from the original labels of 1.4% and 3.1%. We use the bias-corrected datasets throughout the remainder of the evaluation.

	Before Bias-Correction		After Bias-Correction	
	Malicious	False	Malicious	False
TAO	32	2,218	108	71,156
MDR	88	2,734	1,003	81,294

TABLE II: Alert distribution for the TAO and MDR datasets before and after performing bias correction.

B. Parametrization Experiments (RQ1)

To obtain the optimal value of for our cluster distance threshold, $CDist$, we randomly sampled 5 million alerts from the unlabeled alerts dataset. We then test various values in the range 30 – 70 in increments of 5, following best practices established in prior work [28]. Testing each value, we compute the number of resulting clusters. The number of clusters to decrease as $CDist$ increases; concomitantly, as larger clusters

	CDist								
Feature Name	**30**	**35**	**40**	**45**	**50**	**55**	**60**	**65**	**70**
Path	0.072	0.079	0.084	0.089	0.095	0.087	0.088	0.095	0.099
Reputation	0.101	0.113	0.124	0.132	0.136	0.138	0.144	0.156	0.159
Username	0.271	0.304	0.333	0.358	0.385	0.375	0.368	0.388	0.408
Privilege	0.016	0.017	0.02	0.02	0.023	0.021	0.023	0.028	0.03
Integrity	0.03	0.031	0.035	0.035	0.041	0.042	0.045	0.052	0.056
Signer	0.008	0.011	0.013	0.013	0.012	0.011	0.013	0.013	0.016
Parent Path	0.091	0.107	0.123	0.139	0.151	0.146	0.151	0.158	0.173
Parent Reputation	0.117	0.126	0.139	0.142	0.151	0.151	0.157	0.166	0.175
Parent Username	0.212	0.25	0.276	0.314	0.337	0.324	0.316	0.328	0.342
Parent Privilege	0.052	0.052	0.055	0.054	0.058	0.062	0.065	0.065	0.067
Parent Integrity	0.065	0.067	0.07	0.071	0.075	0.079	0.082	0.083	0.086
Parent Signer	0.009	0.012	0.014	0.014	0.014	0.013	0.015	0.015	0.017

TABLE III: Relationship between $CDist$ threshold and entropy values of various security features in the cluster profile.

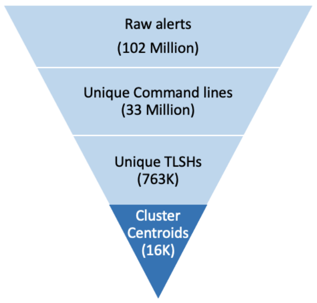
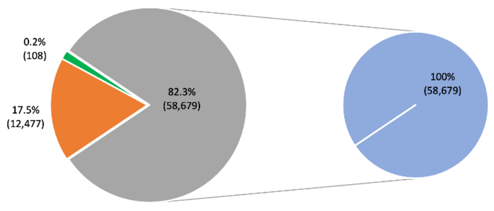
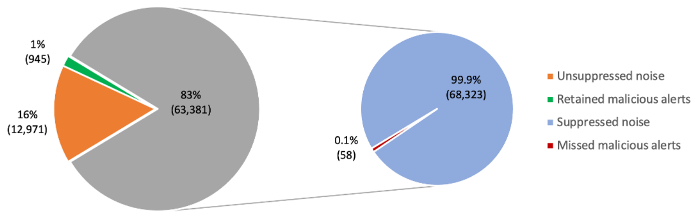


Fig. 4: Reduction in state complexity with Carbon Filter.



(a) TAO



(b) MDR

Fig. 5: Triage results for the TAO and MDR datasets. The lefthand diagram shows the makeup of the entire test sample, with alerts pruned by Carbon Filter shaded in green. The righthand diagram shows the makeup of the pruned alerts only.

get formed, the entropy of the cluster also increases. We use Shannon's entropy [51] to measure the homogeneity for the clusters across the various security features in the cluster profile. We use feature scaling [52] to normalize the entropy values and then calculate the mean across all features and profiles for each $CDist$ value. We also use feature scaling to normalize the number of clusters, such that both values are in range [0,1].

Table III reports the normalized entropy values for various security features under different $CDist$ thresholds. We observe a similar pattern in the homogeneity of the numerical security features such as the number of network connections. Figure 3 reports the relationship between $CDist$, scaled number of clusters, and normalized entropy across all profiles and features. We can see that higher $CDist$ values decreased the homogeneity of the cluster, undermining our security goal. At the same time, higher $CDist$ values also decrease the number of clusters, improving search performance. We determined that a $CDist$ value between 45 to 50 would help us to minimize cluster count while ensuring cluster homogeneity on our data. The remainder of our evaluation reports results for $CDist = 50$.

C. Clustering Experiments (RQ2)

Because the ability to succinctly model process initiators informs both the security and scalability of Carbon Filter, we now consider reduction efficacy of our clustering model. We train Carbon Filter on the unlabeled customer dataset of $\sim$ 102 million alerts with threshold $CDist = 50$. The resulting 16K clusters account for $\sim$101 million (99.1%) of the alerts in our dataset, leaving only 499,946 (0.9%) as unclustered alerts as outliers.

To measure Carbon Filter's clustering efficacy, we consider the complexity of the representation state at each stage of the analysis pipeline. A visual summary of the reduction of the alert stream can be found in Figure 4. The $\sim$ 102 million alerts were comprised of 33 million unique command lines (3X reduction). These command lines produce 736K digests when passed through TLSH (139X reduction). The HAC-T clustering algorithm reduces this representation to 16,299 cluster centroids (6259X reduction). This reduction is not only essential to the scalability of our end-to-end system, but also highlights the limitations of a supervised learning approach. OVer multiple months of prolonged effort we were able to generate a dataset of $\sim$10K labeled samples, but that sample set represents only a small fraction of what we are capable of representing with 16K clusters in an unsupervised model.

D. Triage Experiments (RQ3)

We now discuss the results of Carbon Filter's scoring model. For this experiment we train a separate cluster model using the initial 67% of the TAO dataset, using the remaining 33% of TAO and the complete MDR data as our test set. We specifically set out to determine if out automated alert triage mechanism met 2 key requirements: (1) can Carbon Filter improve the signal-to-noise ratio of EDR alerts by reducing False Positives (erroneously alerting on benign activities)?; and (2) given the potentially catastrophic consequences of a

		Recall				Query Latency					
		Linear Search		ANN		Linear Search		Singlecore ANN		Multicore ANN	
Index Size (IS)		16K	100K	16K	100K	16K	100K	16K	100K	16K	100K
K-Neighbors	1	1	1	0.96	0.96	1.01s	10.42s	0.67s	3.37s	**0.09s**	**0.38s**
	3	1	1	0.97	0.96	1.01s	10.42s	1.10s	3.79s	**0.10s**	**0.44s**
	5	1	1	0.97	0.96	1.01s	10.42s	1.00s	5.00s	**0.10s**	**0.56s**

TABLE IV: Recall and Query Latency for Carbon Filter for batches of 500 alerts under different configurations for Cluster Index Size, K-Neighbors, and Single/Multicore. Linear Search query latency is provided as a baseline.

	Alert Latency (in ms)		Max Supported Alerts per Sec		Factor of High Load (1,706 alerts per sec)	
IS	16K	100K	16K	100K	16K	100K
	0.18	0.76	5,566	1,317	**3.2x**	**0.7x**

TABLE V: Query Latency in *Alerts Per Second* (Batched Query Latency / 500) for Carbon Filter's ANN multicore search with K=1 Neighbors as compared to expected high load volumes in Carbon Black EDR of 1,706 alerts per second.

missed detection, does Carbon Filter minimize False negatives (incorrectly dismissing a genuine threat)? Put another way, is Carbon Filter able to improve precision without meaningfully impacting recall?

The results for the TAO and the MDR datasets are shown in Figure 5. The left circle summarizes the contents of the entire test set for TAO (Fig. 5a) and MDR (Fig. 5a). The grey region denotes the proportion of alerts that were pruned by Carbon Filter, the orange region unpruned false alerts, and the green region true alerts. The right circles summarizes the contents of the pruned alert data, with blue indicating the suppression of false alerts and red indicating the suppression of true alerts.

We find that *Carbon Filter is able to suppress upwards of 82% of false alerts while pruning 0% True Alerts in the TAO dataset and 0.1% True Alerts in the MDR dataset.* This translates to a 5.6x improvement in Signal-to-Noise ratio in both datasets. In the MDR dataset, but not the TAO dataset, Carbon Filter prunes 58 true alerts; while a small number, this actually translates to a 5.7% reduction in recall. Following the experiment, we approached the MDR team for feedback and found that the misclassified malicious alerts fell into two cases. For some, the suspiciousness of the alert was "borderline" and a second analyst simply disagreed investigation outcome. For others, the alerts related to a security policy violation that had been performed by a benign process; most commonly, a legitimate software update routine had temporarily disabled the Data Execution Prevention policy, causing an alert to fire. No misclassified alerts were attributable to outright malicious activity.

E. Scalability Experiments (RQ4)

Carbon Filter's scalability depends on its throughput – the ability to triage alerts in real-time – highlighting the importance of our search algorithm's performance. To evaluate this, we draw an additional test sample of unlabeled customer data comprised of $\sim$ 198 million monthly customer alerts that reduces to $\sim$ 983K unique TLSH's. We match thee TLSH's against our primary model of $\sim 16K$ clusters described in §IV-C.

We evaluate performance along two axes, query speed and recall. Recall differs in definition than from the previous section; here, we refer to the fraction of relevant retrieved results against all relevant database entries:

$$Recall = \frac{\text{Number of relevant results found}}{\text{Total relevant results in the database}} \tag{1}$$

We also manipulate search index size (16K vs. 100K) K-Neighbors count (1,3, or 5) and single versus multi core (1 vs. 16), All queries are issues in batches of 500 alerts. Speed for a Linear Search is reported as a comparison baseline.

The results are reported in Table IV. Multi-core searches were up to $10\times$ faster than single-core and $100\times$ faster than linear searches, with minimal recall differences between index sizes of $16K$ and $100K$. Search times increased approximately fourfold when scaling the index from $16K$ to $100K$. This difference underscores the importance of a parsimonious cluster model to the overall throughput of the architecture.

Let us now consider whether the observed thoughput meets the demands of Carbon Black's operational load. Carbon Black's EDR averages $\mu = 921$ alerts per second with standard deviation $\sigma = 396$, giving a 95% confidence interval of [137, 1706]. Let 1706 alerts per second be an expected high load that Carbon Filter must regularly keep pace with. Our query latency results reported in terms of alerts per second in Table V. With Carbon Filter's current $16K$ index size, it is capable of process $\sim 5,566$ alerts per second, withstanding 3.2 times our current expected high load. Further, even if clustering model's complexity increases by an order of magnitude to 100K, Carbon Filter's current implementation supports 0.7 times our expected high load size on a single EC2 instance. Should the need arise, this ground that could be made up by redeploying Carbon Filter on a better provisioned machine.

F. Comparison to Provenance Baselines (RQ5)

As the state-of-the-art in automated alert triage is based on provenance analysis, we now compare Carbon Filter to two provenance-based approaches that were designed explicitly to triage EDR alerts: NoDoze [12] and RapSheet [13].

1) Experimental Setup: As both systems were closed-source, we re-implemented the published algorithms in consultation with the original authors, These original systems used a fully offline model that processed a full dataset at once. To provide a realistic deployment scenario, we implement the systems as event-driven mechanisms that are invoked each time a new alert arrives. Our NoDoze implementation populates an Event Frequency Database during a training period based on

	Processing Latency (seconds)	
System	**Single Alert**	**All Alerts**
Carbon Filter (Linear)	0.002	0.99
NoDoze	10.2 (5,064X)	5024.2 (5,064X)
RapSheet	53.9 (26,723X)	26503.6 (26,723X)
Carbon→NoDoze	–	1780.0 (1,798X)
Carbon→RapSheet	–	9392.5 (9,486X)

TABLE VI: Query Latency comparison of Carbon Filter to provenance-based triage approaches. The bottom two rows demonstrate that applying Carbon Filter before provenance can improve provenance analysis speeds.

formula (1) in the original paper, assigns scores to each edge in the backtrace of an alert, then aggregates the scores according to formulae (2) and (3) [12]. Our RapSheet implementation performs a backtrace on each alert, creates an alert graph by collapsing vertices in the backtrace that are not associated with an alert, then assigns a score according to formulae (1) and (2) in [13]. The following experiments were conducted on the ATLASv2 dataset [53], with NoDoze trained using benign data from Day 1 to Day 4. All systems were tested on Day 5, which includes a mix of benign and attack activity. We reviewed our results with the authors of the prior works and have incorporated their comments into our remarks.

2) Provenance Query Latency: We begin by comparing the query latency of Carbon Filter to provenance-based approaches. We compare these systems to Carbon Filter's linear search instead of our optimized ANN search function. The query latency for each system is given in Table VI. Carbon Filter is 5,064 times faster than NoDoze and 26,723 times faster than RapSheet. In the final two rows we consider the potential speed-up of applying Carbon Filter as a pre-filtering pass to remove easy-to-triage alerts before employing costly provenance analysis. This reduces the total analysis cost of NoDoze and RapSheet to 1,798x and 9,486x, respectively, suggesting the potential for synergy between the two approaches.

While these results accurately reflect the latency of these systems as they originally appeared, it is worth noting that the NoDoze authors went on to expose a speed/memory tradeoff for provenance-based alert triage [54]. In this work, the NoDoze algorithm was able process 80% of individual alerts in a second or less, and a set of 140 alerts in under two minutes (∼1.16 alerts/sec). This was made possible through the presence of a 300 MB memory cache for retaining alert-related and/or recent events, representing a per-endpoint average memory footprint of ∼1.5 MB. This work demonstrates that provenance analysis may be far closer to feasibility than our own experiments might otherwise suggest. However, Carbon Filter's alert throughput (5,566 alerts/sec) is still three orders of magnitude faster than optimized NoDoze (1 alerts/sec), without the additional memory costs. Using a truly streaming graph analysis architecture for alert triage, akin to the CamQuery framework [15], it may be possible to further reduce the latency of provenance-based approaches.

3) Provenance Triage Efficacy: We compare the triage efficacy of the systems by examining the continuum of classification performance on a traditional ROC curve, depicted in Figure 6. Each point for NoDoze and RapSheet reflects TPR/FPR performance for an anomaly score threshold as described in their respective papers, while each point for Carbon Filter reflects a threshold at which distance from the nearest well-formed process cluster is flagged as anomalous. Carbon Filter achieves an an Area Under Curve (AUC) of 0.94, while RapSheet achieves 0.9 and NoDoze 0.6.

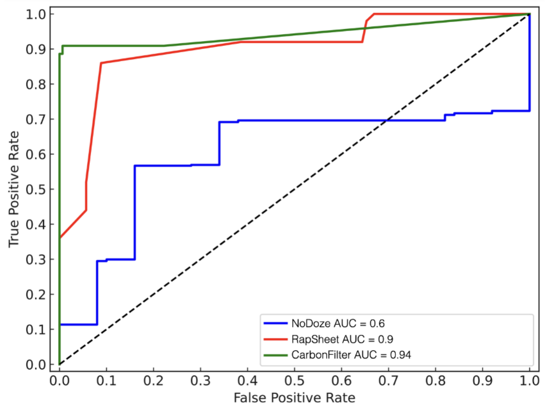


Fig. 6: ROC curve comparison of Carbon Filter with provenance-based approaches on the ATLASv2 dataset.

The poor performance of NoDoze observed here is in part due to the differences between system call logs and EDR telemetry. In the system call auditing configuration commonly used in the literature, every `read` and `write` is associated with a new audit event, improving precision but ballooning the size of the log. By counting event frequency over these logs, NoDoze's edge rarity scores effectively reflect the volume of data flow within the system. In contrast, EDR telemetry does not log individual read and write file events, instead only logging a single file modification event. This flattens the distribution of event frequencies – now extremely common data flows score similarly to anomalous data flows – impairing NoDoze's ability to effectively triage alerts. This helps to explain the discrepancy in our results, but also suggests that NoDoze is not suitable for most EDR telemetry.

While we predicted Carbon Filter to outperform RapSheet in terms of query latency, we were surprised to see it meet or exceed RapSheet in triage efficacy. While the 0.04 difference in AUC between Carbon Filter and Rapsheet may not be significant, Carbon Filter critically achieves TPR of nearly 0.9 at an FPR of 0. That said, while we are encouraged by Carbon Filter's triage performance, we are less confident in the generality of this result. A key property of RapSheet is its ability to identify causal dependencies between alerts, which is not possible with Carbon Filter. It remains likely that the incorporation of past and present event context (via provenance) will allow for subtler distinctions between malicious and benign activity in challenging triage scenarios. The ATLASv2 dataset, which includes just two endpoints and a handful of attack behaviors, may not have provided sufficient opportunity for RapSheet to showcase this capability. Thus, rather than overemphasize an incremental improvement in AUC, we instead conclude that *Carbon Filter can triage the majority of alerts just as effectively as provenance but with orders of magnitude less procesing latency.*

V. Related Work

Carbon Black is not alone in an attempting to mitigate the alert fatigue problem in commercial products. Splunk's Risk Based Alerting (RBA) is a correlation-based feature that also attempts to improve signal-to-noise ratio [55]. Noisey alerts and notables are first correlated into an index based on matching attributes, such as username or endpoint, then prioritized based on analyst-defined weights associated with different risk factors. Microsoft Defender's GraphWeaver performs cross-customer correlation of alerts to identify incidents [56]. Like Splunk RBA, the connectivity between alerts is correlational, based on matching attributes, which here are indicated by edge relationships in a global alert graph. Like Carbon Filter, GraphWeaver supports XDR and features a human-in-the-loop for monitoring and maintenance. While all of these systems are operationally deployed, Carbon Filter is distinct from other commercial systems in that it can affectively triage individual alerts. This makes Carbon Filter synergistic with correlation-based alert prioritization, denoising the inputs to these pipelines, similar to our demonstration of Carbon Filter's synergy with provenance-based approaches.

Provenance-based systems that explicitly consider automated alert correlation and triage are less common than related tasks such as attack reconstruction [57], [58], [59], [60], [61], [62], [54], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74], [75], [76] and threat detection [77], [78], [79], [80], [81], [82], [83], [84], [14], [85], [86], [87], [88]. A comprehensive treatment of these areas can be found in the recent systematization of the provenance literature [89].

Provenance-based alert correlation and triage techniques examine causal relationships to establish dependencies among alerts, which can then be prioritized through scoring functions. Early work in this space traced kernel-level audit logs to enrich IDS alerts with causal information and improve the correlation of alerts [90], [91]. HOLMES employs EDR-like matching logic on rules that define coarse-grained adversary tactics (i.e., match system entities by type only), adjusting for the high volume of false alerts that follow by using a training period to create a baseline of expected false alerts [92]. Carbon Filter provides a more sophisticated approach to false alert baselining that can generalize across many devices/organizations and supports fine-grained detection rules. To our knowledge, NoDoze [12] and RapSheet [13] are the only provenance-based systems designed explicitly for commercial EDR triage. NoDoze employs a network diffusion algorithm to rank alerts and reducing false alarms by 84%, although we show its performance to decline in the absence of finer-grained system call logs. RapSheet identifies causal links between alerts and then generates an anomaly score based on the alert graph's temporal adherence to MITRE ATT&CK Tactics.

While provenance-based systems improve the signal-to-noise ratio of EDR alert streams, scalability and efficiency remain key challenges for deployment. A key factor in this slow down is that provenance-aware systems are often designed for a reactive or postmortem deployment model, needing to iteratively retrieve events ancestor-by-ancestor to reconstruct attacks. This latency is often masked by paging entire datasets into memory, which reduces query latencies by an order of magnitude or more (e.g., [73]), but this solution cannot scale to commercial systems. Hassan et al.'s Swift [93] finds a middle ground by introducing anomaly-based hierarchical storage to audit logs, increasing the likelihood that log events likely to be queried are retained in fast memory. Other systems like CamQuery [15] and Kairos [19] perform edge- or vertex-centric graph streaming analysis to avoid retrieval of historic data. While exhibiting a dramatic improvement on smaller testbeds, these systems have not been tested in large-scale environments, nor have they been demonstrated to be effective at automated alert triage.

VI. Discussion & Future Work

Inspired by the success of recent automated alert triage systems, we present Carbon Filter, an alert triage solution that is compatible with the ubiquitous stream-based event analysis pipelines of today's EDR architectures. Using tens of thousands of Carbon Black alerts labeled by professional analysts, we demonstrate that Carbon Filter is able to prune the vast majority of false alarms ($\sim$82%) while minimally pruning malicious alerts ($<$0.1%). Further, in an open comparison to provenance-based approaches on public data, we find that Carbon Filter meets or exceeds triage performance while improving alert throughput and response latency by 4 to 5 orders of magnitude. Below, we consider some of the broader implications of our findings.

A. The Ongoing Issue of Alert Fatigue.

Carbon Filter is not a panacea to the issue of alert fatigue. While improving these signal-to-noise ratio of the Carbon Black alert stream by a factor of 5.6, false alerts still outnumbered true alerts in our test sets – just 0.9% and 6.8% of the pruned alert stream were truly malicious, up from 0.2% and 1.2% for the TAO and MDR datasets, respectively. The remaining unsuppressed noise in the alert stream will still need to be managed using traditional techniques. Further, there are classes of sophisticated attacks, such as supply-chain attacks, that Carbon Filter may struggle to classify. This is emblematic of a broader issue detecting such attacks in EDR, as the adversary is able to distributed malicious signed code with high reputation scores that appears to be from a trusted vendor. We mitigate this threat through continuously monitoring well-formed clusters for evidence of contamination, ensuring that Carbon Filter does not broaden the EDR's susceptibility to these attacks.

B. The Viability of Provenance-based Triage.

We do not find evidence that current provenance-based mechanisms can outperform the triage efficacy of Carbon

Filter when evaluating on available public datasets. However, we suspect that future iterations of provenance-based triage may prove more effective than Carbon Filter, especially if evaluated against more complex threat scenarios. Provenance analysis mirrors the investigation procedures of professional analysts, and it simply stands to reason that examining *past and present context* provides more information than examining *present context only.* Provenance is particularly suited for complex, multi-stage, and nuanced attacks. For example, provenance is able to identify causal links between multiple alerts (e.g., [13]), where our approach handles each alert in isolation. Instead, we argue that costlier provenance-based analysis should be deployed only when strictly necessary, after lighter-weight mechanisms such as Carbon Filter have already identified and removed the majority of false alerts. We document the potential cost reduction of this hybrid approach in our experiments, where the processing latency of RapSheet is reduced by an order of magnitude. Unfortunately, due to the limitations of available public data, we saw no discernible improvement in triage efficacy when combining these techniques.

Finally, we also stress that provenance analysis is a multi-faceted tool, of which automated alert triage is only a single facet. Even within the scope of endpoint security, provenance can also be brought to bear through the development of orthogonal methods of (anomaly-based) threat detection and through accelerating time-to-insight by automating threat investigation tasks. Our findings should not be interpreted as a general critique of provenance-based methodologies.

C. The Restrained Use of Machine Learning.

Carbon Filter' design exemplifies the philosophy "simpler is better." We chose similarity-preserving digests over word embedding algorithms (e.g., Doc2Vec) because the latter rely heavily on vector calculations that would either have increased inference latency or required the introduction of specialized hardware. An additional benefit of this approach is explainability – it is extremely easy to invert Carbon Filter' classification decisions back into the latent command line data space, removing the possibility that security-critical recommendations are opaque. However, while our digest algorithm (TLSH) is computationally efficient, it has limitations, such as operating in only 132 dimensions with limited cardinality for each feature. Likewise, our HAC-T/ANN clustering approach assigns a single centroid to each cluster, which may not accurately represent the entire cluster.

As discussed above, Carbon Filter left a considerable volume of unsuppressed false alerts "on the table." In addition to provenance-based approaches, it is also worth considering in future work whether more sophisticated machine learning methods might improve on these results. Like provenance, more complex ML could justify its cost by excelling in complex triage scenarios, especially if employed in a hybrid approach as we call for above.

Regardless, as we consider the use of alternate machine learning models in future work, *we must stress the unsuitability of supervised machine learning for the automated alert triage task.* Alert triage should continue to be constructed as an unsupervised learning problem, similar to other anomaly-based intrusion detection tasks. To truly work effectively, supervised learning approaches would need to source an extraordinary volume of alert data with (trustworthy) ground truth labels. Even within Carbon Black, a global leader in the EDR/XDR market, such resources do not exist. Further, if evaluated on a smaller scale with limited public data, supervised learning approaches will be more likely to produce a misleadingly positive result that fails to generalize. In contrast, unsupervised learning is a more conservative approach that assumes less about the true distribution of the alert population, making small scale evaluation results more trustworthy. Unsupervised learning also has the added benefit of generalizing to previously unseen attack behaviors such as zero-day exploits or new living-off-the-land attacks.

VII. Conclusion

Automated EDR alert triage techniques must scale to support the volume of alerts generated by commercial EDR systems. We introduce Carbon Filter, a system performs scalable and effective triage by modeling the command line process initiators of alert-triggering process. Carbon Filter improves the Signal-to-Noise ratio of EDR/XDR alert streams by nearly six-fold while supporting throughputs of 5,666 alerts per second – over 20 million per hour. Carbon Filter is already deployed in the Carbon Black EDR/XDR and is in active use by thousands of customers, demonstrating its real-world efficacy at alert classification.

Availability

Carbon Filter is now integrated into Carbon Black Enterprise EDR and XDR as the "Anomaly Classification" feature,[3][4] which annotates fired alerts with an indicator of whether they are *Anomalous* or *Not Anomalous* alongside the alert's prevalence across all organizations and within the target organization. Users are able to provide direct feedback on these alerts to assist in further refining the model. Currently, anomaly classification is active on over 450 detection rules, including Carbon Black's highest-confidence threat indicators.

Unfortunately, we are unable to release our code or models. To facilitate comparison to future research, we have released the anomaly scores for all alerts in the ATLASv2 dataset. These scores are derived from our global model of Carbon Black customer activity at the time the experiments were conducted. These results can be found alongside the original dataset at https://bitbucket.org/sts-lab/atlasv2. As our prototype is built on open source tools, our core methodology can be replicated by interested researchers. TLSH and HAC-T are available at https://github.com/trendmicro/tlsh and PyNNDescent is available at https://github.com/lmcinnes/pynndescent.

[3]https://techdocs.broadcom.com/us/en/carbon-black/cloud/carbon-black-cloud/index/cbc-user-guide-tile.html

[4]https://www.youtube.com/watch?v=SOEPPn944nA

References

[1] "Endpoint detection and response (EDR)," https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/docs/vmwcb-datasheet-edr.pdf.

[2] Grand View Research, "Endpoint Detection And Response Market Size, Share & Trends Report," https://www.grandviewresearch.com/industry-analysis/endpoint-detection-response-market-report, 2023.

[3] Mitre, "Mitre attack framework," https://attack.mitre.org/, 2013, [Online; accessed 25-Nov-2022].

[4] "Automated Incident Response: Respond to Every Alert," https://swimlane.com/blog/automated-incident-response-respond-every-alert/, 2019.

[5] "New Research from Advanced Threat Analytics," https://prn.to/2uTiaK6, 2019.

[6] FireEye, Inc., "How Many Alerts is Too Many to Handle?" https://www2.fireeye.com/StopTheNoise-IDC-Numbers-Game-Special-Report.html, 2019.

[7] E. Agyepong, Y. Cherdantseva, P. Reinecke, and P. Burnap, "Towards a framework for measuring the performance of a security operations center analyst," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2020, pp. 1–8.

[8] A. Sopan, M. Berninger, M. Mulakaluri, and R. Katakam, "Building a machine learning model for the SOC, by the input from the SOC, and analyzing it for the SOC," in *2018 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 2018, pp. 1–8.

[9] T. Ongun, J. W. Stokes, J. B. Or, K. Tian, F. Tajaddodianfar, J. Neil, C. Seifert, A. Oprea, and J. C. Platt, "Living-off-the-land command detection using active learning," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 442–455.

[10] "Cloud infrastructure is not immune from the solarwinds orion breach," https://securityboulevard.com/2020/12/cloud-infrastructure-is-not-immune-from-the-solarwinds-orion-breach/.

[11] A. Virkud, M. A. Inam, A. Riddle, J. Liu, G. Wang, and A. Bates, "How does endpoint detection use the MITRE ATT&CK framework?" in *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3891–3908. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/virkud

[12] 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 (NDSS) Symposium 2019*, 2019, pp. 24–27.

[13] W. U. Hassan, A. Bates, and D. Marino, "Tactical Provenance Analysis for Endpoint Detection and Response Systems," in *41st IEEE Symposium on Security and Privacy (SP)*, ser. Oakland'20, May 2020.

[14] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: Real-time apt detection through correlation of suspicious information flows," in *40th IEEE Symposium on Security and Privacy*, ser. Oakland'19. Los Alamitos, CA, USA: IEEE Computer Society, may 2019. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2019.00026

[15] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, , and M. Seltzer, "Runtime Analysis of Whole-System Provenance," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, Oct 2018.

[16] "Sigmahq rule example," https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_addinutil_suspicious_cmdline.yml.

[17] 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 *26th ISOC Network and Distributed System Security Symposium*, ser. NDSS'19, February 2019.

[18] Mitre, "Boot or logon initialization scripts: Rc scripts," https://attack.mitre.org/techniques/T1037/004/, 2020, [Online; accessed 30-Nov-2022].

[19] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "Kairos:: Practical intrusion detection and investigation using whole-system provenance," *arXiv preprint arXiv:2308.05034*, 2023.

[20] S. Rothlisberger, "Havoc C2 with AV/EDR Bypass Methods in 2024," https://medium.com/@sam.rothlisberger/havoc-c2-with-av-edr-bypass-methods-in-2024-part-1-733d423fc67b, January 2024.

[21] V. T. Hoang, C. Wu, and X. Yuan, "Faster yet safer: Logging system via Fixed-Key blockcipher," in *31st USENIX Security Symposium*, ser. Security'22. Boston, MA: USENIX Association, Aug. 2022, pp. 2389–2406. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/hoang

[22] A. Ahmad, S. Lee, and M. Peinado, "Hardlog: Practical tamper-proof system auditing using a novel audit device," in *43rd IEEE Symposium on Security and Privacy*, ser. Oakland'22, May 2022, pp. 1791–1807.

[23] R. Paccagnella, P. Datta, W. U. Hassan, A. Bates, C. W. Fletcher, A. Miller, and D. Tian, "Custos: Practical Tamper-Evident Auditing of Operating Systems Using Trusted Execution," in *27th ISOC Network and Distributed System Security Symposium*, ser. NDSS'20, February 2020.

[24] R. Paccagnella, K. Liao, D. Tian, and A. Bates, "Logging to the danger zone: Race condition attacks and defenses on system audit frameworks," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1551–1574. [Online]. Available: https://doi.org/10.1145/3372297.3417862

[25] "The future of machine learning in cybersecurity," https://www.cio.com/article/406441/the-future-of-machine-learning-in-cybersecurity.html/.

[26] "Reaching the tipping point of web application and api security." https://www.fastly.com/web-application-and-api-security-tipping-point/, published July 2021. Research conducted by Enterprise Strategy Group, March 2021.

[27] "Amazon s3," https://aws.amazon.com/s3/.

[28] J. Oliver, C. Cheng, and Y. Chen, "TLSH – a locality sensitive hash," in *2013 Fourth Cybercrime and Trustworthy Computing Workshop*, 2013, pp. 7–13, https://github.com/trendmicro/tlsh/blob/master/TLSH_CTC_final.pdf.

[29] J. Oliver, M. Ali, and J. Hagen, "HAC-T and fast search for similarity in security," in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*. IEEE, 2020, pp. 1–7, https://tlsh.org/papersDir/COINS_2020_camera_ready.pdf.

[30] "Apache flink," https://flink.apache.org/what-is-flink/flink-architecture/.

[31] "Amazon kinesis," https://aws.amazon.com/kinesis/.

[32] "Protocol buffers," https://protobuf.dev/.

[33] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," *Digital investigation*, vol. 3, pp. 91–97, 2006.

[34] V. Roussev, "Data fingerprinting with similarity digests," in *IFIP International Conference on Digital Forensics*. Springer, 2010, pp. 207–226.

[35] E. Raff and C. Nicholas, "Lempel-ziv jaccard distance, an effective alternative to ssdeep and sdhash," *Digital Investigation*, vol. 24, pp. 34–49, 2018.

[36] K. Soska, C. Gates, K. A. Roundy, and N. Christin, "Automatic application identification from billions of files," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 2021–2030.

[37] M. Dell'Amico, "Fishdbc: Flexible, incremental, scalable, hierarchical density-based clustering for arbitrary data and distance," *arXiv preprint arXiv:1910.07283*, 2019, https://arxiv.org/pdf/1910.07283.pdf.

[38] Github, "Tlsh software," https://github.com/trendmicro/tlsh/.

[39] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A llvm-based python jit compiler," in *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 2015, pp. 1–6.

[40] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[41] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: why and how you should (still) use dbscan," *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[42] "ANNOY library," https://github.com/spotify/annoy.

[43] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[44] W. Dong, M. Charikar, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, S. Srinivasan, K. Ramamritham, A. Kumar, M. P. Ravindra, E. Bertino, and R. Kumar, Eds. ACM, 2011, pp. 577–586. [Online]. Available: https://doi.org/10.1145/1963405.1963487

[45] "Benchmarking for fast searching of nearest neighbors," https://github.com/erikbern/ann-benchmarks.

[46] "Neighborhood graph and tree (NGT) for indexing high-dimensional data," https://github.com/yahoojapan/NGT/blob/main/README.md#publications.

[47] "Python implementation of approximate nearest neighbors (nn-descent)," https://github.com/lmcinnes/pynndescent.

[48] Fireeye, "Incident Investigation," https://www.fireeye.com/solutions/incident-investigation.html, 2019.

[49] A. Riddle and K. Westfall, "Atlas v2: An open-source dataset for intrusion detection research," https://bitbucket.org/sts-lab/atlasv2/.

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

[51] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[52] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," *CoRR*, vol. abs/1503.06462, 2015. [Online]. Available: http://arxiv.org/abs/1503.06462

[53] A. Riddle, K. Westfall, and A. Bates, "Atlasv2: Atlas attack engagements, version 2," 2023.

[54] W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, D. Wang, Z. Chen, Z. Li, J. Rhee, J. Gui, and A. Bates, "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *Annual Computer Security Applications Conference*, ser. ACSAC '20. New York, NY, USA: Association for Computing Machinery, Dec 2020, pp. 165–178. [Online]. Available: https://doi.org/10.1145/3427228.3427255

[55] J. Bull, "Implementing risk-based alerting," https://lantern.splunk.com/Security/UCE/Guided_Insights/Risk-based_alerting/Implementing_risk-based_alerting, Sep 2024.

[56] S. Freitas and A. Gharib, "Graphweaver: Billion-scale cybersecurity incident correlation," in *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, ser. CIKM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 4479–4486. [Online]. Available: https://doi.org/10.1145/3627673.3680057

[57] P. Datta, I. Polinsky, M. A. Inam, A. Bates, and W. Enck, "ALASTOR: Reconstructing the provenance of serverless intrusions," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2443–2460. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/datta

[58] P. Fang, P. Gao, C. Liu, E. Ayday, K. Jee, T. Wang, Y. F. Ye, Z. Liu, and X. Xiao, "Back-Propagating system dependency impact for attack investigation," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2461–2478. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/fang

[59] A. Alsaheel, Y. Nan, S. Ma, L. Yu, G. Walkup, Z. B. Celik, X. Zhang, and D. Xu, "ATLAS: A sequence-based learning approach for attack investigation," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3005–3022. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/alsaheel

[60] B. E. Ujcich, S. Jero, R. Skowyra, A. Bates, W. H. Sanders, and H. Okhravi, "Causal analysis for software-defined networking attacks," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, August 2021.

[61] X. Chen, H. Irshad, Y. Chen, A. Gehani, and V. Yegneswaran, "CLARION: Sound and clear provenance tracking for microservice deployments," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3989–4006. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-xutong

[62] J. Zheng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "WATSON: Abstracting Behaviors from Audit Logs via Aggregation of Contextual Semantics," in *28th ISOC Network and Distributed System Security Symposium*, ser. NDSS'21, February 2021.

[63] 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 *Proceedings of the 25th ISOC Network and Distributed System Security Symposium*, ser. NDSS'18, San Diego, CA, USA, February 2018.

[64] W. U. Hassan, N. Aguse, M. Lemay, T. Moyer, and A. Bates, "Towards Scalable Cluster Auditing through Grammatical Inference over Provenance Graphs," in *Proceedings of the 25th ISOC Network and Distributed System Security Symposium*, ser. NDSS'18, San Diego, CA, USA, February 2018.

[65] C. Wang, S. Ma, X. Zhang, J. Rhee, X. Yun, and Z. Hao, "A hypervisor level provenance system to reconstruct attack story caused by kernel malware," in *Security and Privacy in Communication Networks*, X. Lin, A. Ghorbani, K. Ren, S. Zhu, and A. Zhang, Eds. Cham: Springer International Publishing, 2018, pp. 778–792.

[66] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "Mci: Modeling-based causality inference in audit logging for attack investigation," in *Proc. of the 25th Network and Distributed System Security Symposium (NDSS'18)*, 2018.

[67] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. N. Venkatakrishnan, "Propatrol: Attack investigation via extracted high-level tasks," in *Information Systems Security*, V. Ganapathy, T. Jaeger, and R. Shyamasundar, Eds. Cham: Springer International Publishing, 2018, pp. 107–126.

[68] 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)*. Baltimore, MD: USENIX Association, 2018, pp. 639–656. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/gao-peng

[69] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple Perspective Attack Investigation with Semantic Aware Execution Partitioning," in *26th USENIX Security Symposium*, August 2017.

[70] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable attack investigation with on-demand inter-process information flow tracking," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 377–390. [Online]. Available: https://doi.org/10.1145/3133956.3134045

[71] 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)*. Vancouver, BC: USENIX Association, 2017, pp. 487–504. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/hossain

[72] K. Pei, Z. Gu, B. Saltaformaggio, S. Ma, F. Wang, Z. Zhang, L. Si, X. Zhang, and D. Xu, "Hercule: Attack story reconstruction via community discovery on correlated log graph," in *Proceedings of the 32Nd Annual Conference on Computer Security Applications*, ser. ACSAC '16. New York, NY, USA: ACM, 2016, pp. 583–595. [Online]. Available: http://doi.acm.org/10.1145/2991079.2991122

[73] A. Bates, D. Tian, K. R. Butler, and T. Moyer, "Trustworthy Whole-System Provenance for the Linux Kernel," in *Proceedings of 24th USENIX Security Symposium*, Aug. 2015.

[74] Y. Liu, X. Shu, Y. Sun, J. Jang, and P. Mittal, "Rapid: real-time alert investigation with context-aware prioritization for efficient threat discovery," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 827–840.

[75] M. A. Inam, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "FAuST: Striking a Bargain between Forensic Auditing's Security and Throughput," in *Proceedings of the 38th Annual Computer Security Applications Conference*, ser. ACSAC '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 813–826. [Online]. Available: https://doi.org/10.1145/3564625.3567990

[76] M. A. Inam, W. U. Hassan, A. Ahad, A. Bates, R. Tahir, T. Xu, and F. Zaffar, "Forensic Analysis of Configuration-based Attacks," in *29th ISOC Network and Distributed System Security Symposium*, ser. NDSS'22, February 2022.

[77] M. Rehman, H. Ahmadi, and W. Hassan, "FLASH: A Comprehensive Approach to Intrusion Detection via Provenance Graph Representation Learning," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 142–142. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00139

[78] Z. Cheng, Q. Lv, J. Liang, Y. Wang, D. Sun, T. Pasquier, and X. Han, "KAIROS: Practical Intrusion Detection and Investigation using Whole-system Provenance," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 9–9. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00005

[79] X. Han, X. Yu, T. Pasquier, D. Li, J. Rhee, J. Mickens, M. Seltzer, and H. Chen, "SIGL: Securing software installations through deep graph learning," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2345–2362. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/han-xueyuan

[80] Y. Xie, Y. Wu, D. Feng, and D. Long, "P-gaussian: Provenance-based gaussian distribution for detecting intrusion behavior variants using high efficient and real time memory databases," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 6, pp. 2658–2674, 2021.

[81] X. Han, T. Pasqueir, A. Bates, J. Mickens, and M. Seltzer, "Unicorn: Runtime Provenance-Based Detector for Advanced Persistent Threats," in *27th ISOC Network and Distributed System Security Symposium*, ser. NDSS'20, February 2020.

[82] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Zhen, W. Cheng, C. A. Gunter, and H. chen, "You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis," in *27th ISOC Network and Distributed System Security Symposium*, ser. NDSS'20, February 2020.

[83] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2020.

[84] F. Capobianco, R. George, K. Huang, T. Jaeger, S. Krishnamurthy, Z. Qian, M. Payer, and P. Yu, "Employing Attack Graphs for Intrusion Detection," in *New Security Paradigms Workshop*, ser. NSPW'19, September 2019.

[85] X. Han, T. Pasquier, T. Ranjan, M. Goldstein, and M. Seltzer, "Frappuccino: Fault-detection through runtime analysis of provenance," in *9th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 17)*. Santa Clara, CA: USENIX Association, Jul. 2017. [Online]. Available: https://www.usenix.org/conference/hotcloud17/program/presentation/han

[86] E. Manzoor, S. M. Milajerdi, and L. Akoglu, "StreamSpot: Detecting network anomalies in edge streams (Source Code and Data)," https://sbustreamspot.github.io/, 2016.

[87] A. Goyal, G. Wang, and A. Bates, "R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection," in *2024 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2024, pp. 257–257. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00253

[88] J. Liu, M. A. Inam, A. Goyal, A. Riddle, K. Westfall, and A. Bates, "What we talk about when we talk about logs: Understanding the effects of dataset quality on endpoint threat detection research," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025, pp. 112–129.

[89] M. A. Inam, Y. Chen, A. Goyal, J. Liu, J. Mink, N. Michael, S. Gaur, A. Bates, and W. U. Hassan, "Sok: History is a vast early warning system: Auditing the provenance of system intrusions," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2620–2638.

[90] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching intrusion alerts through multi-host causality." in *Proceedings of the 12th ISOC Network and Distributed System Security Symposium*, ser. NDSS'05, 2005.

[91] Y. Zhai, P. Ning, and J. Xu, "Integrating ids alert correlation and os-level dependency tracking," in *International Conference on Intelligence and Security Informatics*. Springer, 2006.

[92] 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 2019 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2019.

[93] W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, D. Wang, Z. Chen, Z. Li, J. Rhee, J. Gui *et al.*, "This is why we can't cache nice things: Lightning-fast threat hunting using suspicion-based hierarchical storage," in *ACSAC*, 2020.