How to Effectively Trace Provenance on Windows Endpoint Detection & Response Telemetry

Jason Liu*, Muhammad Adil Inam*, Akul Goyal, Dylen Greenenwald, Adam Bates
University of Illinois at Urbana-Champaign
{jdliu2, mainam2, akulg2, dgree21, batesa}@illinois.edu

Saurav Chittal
Purdue University
schittal@purdue.edu

***Abstract*—Academic research on provenance analysis is primarily based on high-fidelity event streams captured on Linux/Unix devices (e.g., Linux Audit). Unfortunately, provenance tracing becomes much more complicated on Windows, where microkernel design principles lead to far noisier provenance graphs. These complications further compound when analyzing the efficient, low-fidelity event streams generated by commercial Endpoint Detection & Response products.**

Fortunately, provenance tracing is still possible in spite of these obstacles. We first present a method of recovering whole-system provenance from commercial EDR telemetry. This graph conservatively models all possible information flows, but is even less precise than traditional whole-system provenance graphs – that is, there is more dependency explosion, or false provenance. We go on to present four heuristics that allow us to denoise the provenance graph under realistic threat investigation scenarios. The first two heuristics are process-centric, leveraging domain knowledge of Windows service control flow patterns to mitigate the dependency explosion caused by Windows IPC. The second two heuristics are data-centric, intended to cluster and denoise data accesses on Windows where accesses to environmental configuration data (i.e., Registry keys) are auditable events. In evaluations based on the MITRE Enginuity simulation of the Carbanak APT, we demonstrate that these heuristics reduce graph complexity by up to 98% as compared to a baseline tracing algorithm. These tracing strategies enable further research into provenance integrations for EDR, moving the community towards a more realistic and relevant deployment model.

I. Introduction

Research into security applications for data provenance have led to breakthroughs in intrusion detection (e.g., [2], [8], [17], [36]) and threat investigation (e.g., [1], [25], [35]). These encouraging results suggest the potential for provenance analysis to fundamentally transform the future of "reactive security" [19] in Security Operations Centers. However, there remain threats to the validity and generality of these techniques that the academic community has not sufficiently explored. Among these, one concern is replicating results on the kind of event telemetry that is available in enterprise environments, i.e., Endpoint Detection & Response (EDR) telemetry. Surprisingly, another threat is the Windows operating system. When our research group began to experiment with EDR telemetry and evaluate on Windows datasets, the precision and accuracy of our analyses quickly deteriorated.

Nearly all of the provenance security literature builds on system call audit logs – Linux Audit, Event Tracing for Windows, and the like – that are configured to be extremely voluminous and capture a record of every information flow. While the availability of commodity audit frameworks has been a boon to systems auditing research, including data provenance, in practice organizations do not directly record, analyze, and retain system call (syscall) logs. Instead, EDR sensors act as an intermediary, generating a more compact and efficient event telemetry stream by tapping into syscall logs as well as other system data. EDR logs are commonly generated and retained in enterprise environments, but the suitability of these logs for provenance analysis is underexplored. To the best of our knowledge, Hassan et al.'s RapSheet system is the only provenance application demonstrated to be effective on EDR telemetry [9], but the intricacies of conducting provenance on EDR telemetry are not explored in this work. Further, RapSheet demonstrates the feasibility of single provenance method (forward tracing) for a single application (alert triage); we show how additional challenges emerge when attempting to port the entire provenance toolkit to EDR telemetry.

While not exclusively, prior work on provenance analysis has also heavily focused on *nix systems such as Linux or FreeBSD. With Windows running on 70% of workstations as of May 2025 [31], compared to Linux's 4%, Windows is clearly important to consider for any practical application. Given the evolution of operating systems and the strong similarity in system abstractions – processes, files, sockets, etc. – it seems natural that provenance analysis should more or less work "out of the box" without requiring major methodological changes. Indeed, while fundamental principles of data provenance still apply to Windows subtle design details pose unique complications for provenance analysis. Specifically, we encounter two key challenges when conducting provenance analysis on Windows:

1) **Windows Inter-Process Communication (IPC).** Unlike the monolithic Linux kernel, the Windows NT kernel is a hybrid architecture that borrows heavily from microkernel design principles [30]. Core features like scheduling and

*Equal contribution.

Workshop on Attack Provenance, Reasoning, and Investigation for Security in the Monitored Environment (PRISM) 2026
23 February 2026, San Diego, CA, USA
ISBN 978-1-970672-00-8
https://dx.doi.org/10.14722/prism.2026.23012
http://www.ndss-symposium.org

interrupts are handled in the kernel, with other functionality delegated to user space system services. As a result, OS interactions that appear as "invisible" traps to the kernel in Linux become auditable events on Windows, combining with other IPC to generate significantly more dependency explosion and false provenance.

2) **Windows Data Access Patterns.** In Linux, environment and configuration variables are mapped into process memory and become mostly "invisible," internal process state that is represented by the process vertex. On Windows, accesses to environment and configuration variables are auditable events to the Windows registry. To make matters worse, Windows data paths appear to contain significantly more non-deterministic content, either truly random (UUID's, hashes) or structured data (SIDs). While this does not result in false provenance, it does cause dependency explosion, adding visual noise and increasing the complexity of learning generalizable models of system behaviors.

In this work, we demonstrate that foundational provenance methods like forward and backward tracing remain feasible on Windows EDR telemetry. We show that it is possible to reproduce a conservative approximation of a whole-system provenance graph generated from syscall logs, but with reduced information flow precision. For example, in an attack investigation scenario using EDR telemetry, we produce a concise attack graph on a Linux server (CARBANAKv2, `fs`) comprised of 33 vertices and 53 edges, while the same attack campaign on a Windows server (CARBANAKv2, `dc`) contains 3173 vertices and 6360 edges. While unacceptably large, this result establishes the basic feasibility of provenance analysis on Windows EDR telemetry, as bi-directional information flow connectivity was consistently maintained between the attack's root causes and impacts.

To bring the complexity of the Windows attack graphs down to a more manageable level, we begin to consider how domain knowledge of Windows' design could be instantiated as heuristics that prune false provenance and dependency explosion. We first show that the event-driven nature of most Windows system services make them candidates for execution partitioning, highlighting both inter-process and intra-process opportunities for pruning false information flows during graph traversal. We continue by deduplicating Windows' explosive data access patterns. We train a Doc2Vec-based clustering model group semantically-related data entities together. We also observe that "leaf" data entities can be merged into their parent process without impacting threat investigation results.

These insights are built on a sophisticated attack engagement, CARBANAKv2, conducted over the span of 3 weeks in our research group. Decomposing the attack campaign by device into 6 discrete attack chains, we demonstrate that our domain knowledge heuristics are able to reduce the complexity of Windows EDR graphs by 91%-98%. While further refinement is required, our results combine to demonstrate the feasibility of conducting provenance analysis on EDR telemetry. Our filtering heuristics and datasets, which we look forward to open sourcing upon publication, will facilitate future research into provenance analysis in the more realistic deployment scenario of EDR integration.

TABLE I: A comparison of system call event fidelity to representative EDR telemetry, Carbon Black Cloud XDR v1.1. "Special" indicates an enriched EDR event hook that does not correspond to an individual system call.

Entity	System Event	Information Flow	EDR Event
File	Open	*Process↔File*	`filemod`
File	Read/Write	*Process↔File*	—
File	Special	*Process↔File*	`moduleload`
File	Special	*Process←File*	`scriptload`
File	Close	*End of Flow*	—
Registry	Read/Write	*Process↔Key*	`regmod`
Socket	Open	*Process↔Network*	`netconn`
Socket	Send/Recv	*Process↔Network*	—
Socket	Close	*End of Flow*	—
Process	Fork/Exec	*Process→Process*	`procstart`
Process	IPC	*Process↔Process*	`crossproc`
Process	Special	—	`fileless_scriptload`
Process	Exit/Kill	*End of Flow*	`procend`

II. Background

A. Endpoint Detection & Response Telemetry

At first glance, EDR sensors appear to generate highly similar telemetry to common system call auditing configurations in Linux Audit and Event Tracing for Windows. In Table I, we compare the Carbon Black Cloud XDR's event schema to a standard system call logging configuration. While both event streams capture the occurrence of process activities and data accesses, subtle differences result in concerning implications for provenance analysis:

- **No I/O Events.** EDR schemas mark the start of a data flow, but not record individual read/write events. This makes EDR telemetry much more space efficient than the common system call logging configuration used in the literature. It also means that prior work that leveraged interleaved I/O events to improve resource utilization (e.g., [11], [14], [33]) or improve attack reconstruction (e.g., [5], [10], [24]) is not applicable.
- **No Stop Events.** The only stop event that can be found in the EDR stream is for processes, not data flows. Provided that the process(es) are still alive, this means that it is not possible to infer the end of data flows from EDR logs.

With the benefit of hindsight, it becomes clear that the EDR event schema is highly optimized to support EDR's rule/heuristic-based detection model. In this model, detection analytics will at most test for the occurrence of a data access, not its frequency, volume, or ordering of occurrence relative to other accesses.[1] It is thus an unnatural fit for provenance-based information flow analysis, yet provenance must be made to work on EDR event telemetry in order to successfully transition to practice.

[1]We also note in Table I the presence of other EDR events that do not map cleanly to system call events. Notably, EDR's often log the entire contents of script accesses to the event stream. It is common practice to build detection analytics that match against script contents.

While we have less visibility into other commercial products, Carbon Black XDR's event schema appears similar to several other products. For example, Carbon Black's `filemod` event serves a similar role as SentinelOne's `FileSystemActivity` event and Microsoft Defender's `ObjectAccess` event. We note, however, that other products' event schemas can vary dramatically, and may pose even greater challenges for provenance analysis. For example, CrowdStrike employs 58 specialized events of the form `*FileWritten` (e.g., `ELFFileWritten`, `PDFFileWritten`), but does not appear to have a corresponding read event for files or registry keys [3]. We thus suspect that Carbon Black's schema represents a (realistic) best case scenario for provenance, although analysis may still be possible in Crowdstrike via the process tree.

B. CARBANAKv2 Dataset

Our methods and results are based primarily on an attack engagement dataset generated over a period of three weeks from April 19, 2024 to May 10, 2024. All hosts were instrumented with the Carbon Black Cloud XDR endpoint sensor and Wireshark. The testbed was comprised of four Windows 10 workstations operated by 4 graduate students as their primary workstations throughout the engagement (`h1`-`h4`). The testbed also included a CentOS 7 Linux fileserver (`fs`), which all machines periodically connected to transmit packet capture logs, and a Windows 10 Server AD domain controller (`dc`) which authenticated each workstation's domain accounts and served as a DNS server and relay.

The first week and a half was reserved for benign operation of all machines. Starting on April 30, an additional student initiated an attack campaign based on MITRE Center for Threat-Informed Defense's simulation of the Carbanak APT. [6], While the original Carbanak emulation involved one workstation, one fileserver, and one domain controller, we extended the attack chain to provide a richer evaluation dataset for lateral movement and dwell time. The attacker initially compromises `h1` (Apr 30), spreads to `fs` in order to gain access to the `dc` (May 2), before proceeding to the other hosts on May 7, May 9, and May 10, respectively. The operators of "victim" workstations continued to use the machine as normal throughout the duration of the attack, providing rich naturalistic background activity for future evaluations.

III. Recovering Whole-System Provenance from EDR Telemetry

The good news is that we can use EDR telemetry to approximate the whole-system provenance graph. The EDR graph will *overestimate* information flows, but will not omit any information flows that would be present in the whole-system graph. For a data access event $< s, d, r, t >$ where s and d are vertices representing system entities, r is the event type, and t is the event timestamp, we create an edge $s \rightarrow d$ with attributes $\{$`type=`r`, time=`$[t, \infty]\}$. During graph traversal, any forward trace from time $\geq t$ must traverse the edge, and any backward trace from time $\leq \infty$ must traverse the edge. This extremely permissive time bracket reflects the possibility that a data access event (e.g., `filemod`) at time t reflects the possibility that an information flow could occur at any point in the future so long as the associate process(es) is still alive. If either s or d are associated with a `procend` event at time u, then and only then are we able to update the edge's time bracket to $[t, u]$.

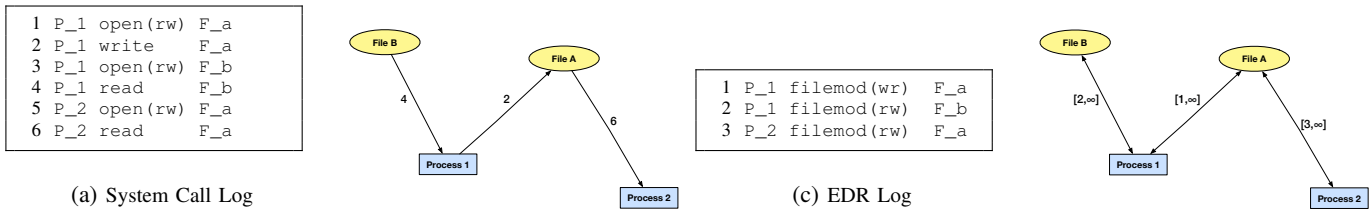
To demonstrate the implications of this change, Figure 1 provides a comparison of provenance graphs generated by system call and EDR event streams. Fig. 1a provides the ground truth of a simple event sequence, which can be used to generate the whole-system provenance graph found in Fig. 1b. From this graph, it is clear that P_2 is not causally dependent on F_b. Fig. 1c shows how the same event sequence appears in EDR telemetry. From this graph, we must assume that P_2 is causally dependent on F_b, but this is *false provenance* as no such information flow occurred on the system. As whole-system provenance is already prone to dependency explosion [21] due to the known semantic gap between system call logs and and intra-process control flow, the exacerbation of false provenance by EDR logs warrants careful consideration.

We identified the need for this tracing approach through trial and error while working the ATLASv2 dataset [29], which also contains Carbon Black Cloud XDR telemetry. When attempting to forward trace the attacks from their root cause(s), we typically found success using a naive tracing algorithm, i.e., the edge $s \rightarrow d$ was associated with the instantaneous timestamp t. However, backward tracing an attack from its impact(s) was unsuccessful because the naive tracing algorithm was unable to identify true causal dependencies. Delving deeper, we soon realized that each data access event implied that the access may continue to occur in perpetuity.

We also note that the transition to EDR event telemetry will pose additional problems for streaming systems that are popular for anomaly detection (e.g., [2], [27]. In streaming processing of chronologically-ordered events/edges, it is not possible to specify start and end timestamps as this violates the chronological ordering. An alternative scheme that would permit streaming processing would be to insert additional "virtual" edges into the stream when the possibility of an undocumented information flow is detected. This would require maintaining state about the active files associated with each process, and vice versa, adding additional processing costs to the streaming application.

IV. Mitigating Windows Process Dependencies

While the imprecision of the EDR graph may cause problems for Linux devices, provenance tracing on Windows devices is an outright disaster. In contrast to the fully monolithic design of the mainline Linux kernel, Windows (NT) kernel adopts a hybrid architecture that borrows heavily from microkernel design principles. Specifically, Windows does not provide a single uniform API for directly interacting with the kernel (e.g., POSIX), instead exposing a range of operating system services that run as programs in user space. By virtue of running as user space programs, these operating system



(a) System Call Log

(b) Whole-system Provenance Graph

(c) EDR Log

(d) EDR Provenance Graph.

Fig. 1: A simplified comparison of whole-system and EDR provenance analysis demonstrating the increased potential for false provenance using EDR telemetry. Line numbers in 1a and 1c are used as Lamport timestamps in 1b and 1d, respectively.

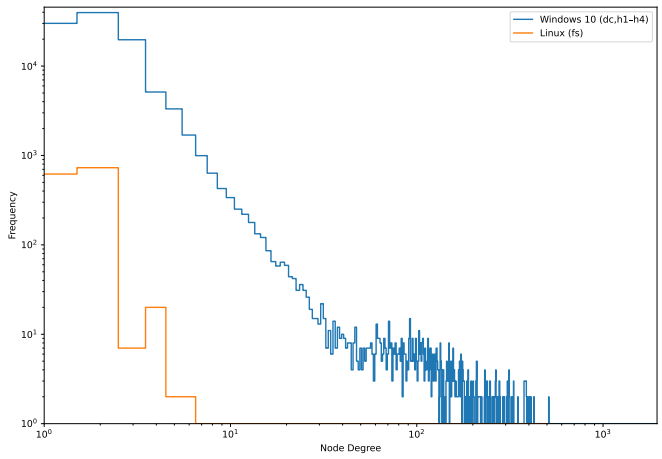


Fig. 2: A comparison of process-to-process connectivity (Node Degree) on Linux (`fs`) and Windows 10 (`dc`, `h1–h4`) in CARBANAKv2. Windows devices have a long tail of highly connected services processes that exacerbate the dependency explosion of provenance analysis.

services are all auditable. On Linux devices, the kernel is effectively invisible in the provenance graph, i.e., we do not consider a value returned from kernel space to be causally dependent on every previous argument sent to kernel space.[2] But on Windows, interactions with the operating system are audited as inter-process events rather than traps to the kernel, meaning that a standard provenance tracing approach would presume causal links between these completely unrelated activities. This problem is not unique to EDR logs, but becomes disastrous when the two sources of imprecision are combined.

As an attempt to quantify this problem, Figure 2 compares the process-to-process connectivity of Windows hosts and the Linux host in the CARBANAKv2 dataset. We observe that Windows hosts exhibit a pronounced long tail of highly connected processes, where a small number of processes demonstrate extremely high fan-out, while the majority interact with only a handful of peers. In contrast, the Linux host contains fewer high-degree processes and a markedly less extreme tail. For example, a single instance of `svchost.exe`, a benign Windows service host process, is connected to over 1,900 processes. Whereas the highest-degree Linux process is connected to only 507 processes. Moreover, the top 10 Windows processes collectively connect to over 13,000 other processes, whereas the top 10 Linux processes connect to fewer than 600. These trends can also be observed in the DARPA Transparent Computing datasets, shown in Figure 11 in the appendix. Thus, while provenance tracing on nix devices also has its problems, *it would be beneficial if we could make our Windows provenance tracing more like Linux provenance tracing!*

[2]Support for this argument – a lesser known observation from Xu et al.'s seminal work [33] is that accesses to virtual files exposed by the kernel do not represent true information flows and should thus be filtered from the graph.

A. Inter-Process Filtering Heuristic

As long-lived server processes, it is unsurprising that Windows operating system services are major contributors to dependency explosion [12], [13], [21], [26]. Our first observation is that, regardless of running as user space processes, interprocess communication with these services can still be handled as communication with the operating system. That is, messages sent by a Windows service are often completely causally independent of prior messages sent to the Windows service. With sufficient confidence that each IPC session with a particular Windows service is causally independent of other sessions, it is no longer necessary to model these events at all. Instead, we can define a domain knowledge constraint [33] that filters all events associated with that service, treating them instead as "invisible" interactions with the operating system.

While Windows has many system services, there exist a finite set on the order of several hundred. Rather than develop a generic approach to modeling these services, we instead manually verified each service. Specifically, we calculated the inter-process connectivity (`crossproc`) of every process in our dataset, then sorted in descending order. We then removed any process' whose images did not reside in a trusted operating system directory (e.g., `c:\windows\system32\`). Next, we reviewed Windows documentation to learn about the API and behavior of the service. The service was added to our filter list if they met the following requirements: (#1) distinct IPC sessions with the service are causally independent on one another, and (#2) an IPC session with the service cannot result in an information flow with other data entities. An example of a service that met this criteria is the Windows Defender Antivirus engine (`msmpeng.exe`), which engages

in IPC with every process. On the other hand, the network service (`msedge.exe`) did not meet this criteria because it acts as an intermediary on all network connections, violating Requirement #2 by establishing an information flow with the network socket. At present, we have used this procedure to identify 80 services whose events can be safely filtered for the purposes of provenance analysis.

B. Intra-Process Filtering Heuristic

Several essential system services acted as an intermediary in information flows (violating Requirement #2) even though each inter-process connection to the service was causally independent (satisfying Requirement #1). In addition to msedge.exe, other services matching this description include the file explorer (`explorer.exe`) and Local Security Authority Subsystem Service (`lsass.exe`). In these cases, traditional execution partitioning can be used to subdivide long-lived service processes into autonomous execution units, eliminating the threat of dependency explosion. One possibility would be to apply Ma et al.'s instrumentation-free approach to execution partitioning based on profiling event logs [26], but inferring execution units from EDR logs would be difficult due to their imprecision.

Instead, we leverage a much simpler heuristic – causally-linked inputs and outputs to a system service exhibit strong temporal locality. In the absence of concurrency or preemption, following the immediate next edge would be sufficient for programs like `msedge.exe` and `explorer.exe`. To account for these effects, we simply define a small time interval Δ and follow all edges within time $t \pm \Delta$ depending on whether the traversal is forwards or backwards. For our datasets, we found that using a Δ of 2 seconds maintained graph connectivity in attack investigations while mitigating the false provenance incurred by these services.

C. Security Analysis

While less technically satisfying than prior work on execution partitioning [21], our approaches adopt the same threat model. All execution partitioning assumes the integrity of the program being partitioned; if the program is compromised, the attacker could manipulate the control flow of the program to manipulate the contents of the log [34]. Similarly, an attacker that is able to exploit one of the operating system services would be able to engage in activities that violate our assumptions about that services' behaviors. Thus, in spite of their simplicity, our heuristics do not expand the attack surface of established methods in provenance analysis.

V. Mitigating Windows Data Dependencies

In addition to providing auditable interactions with the operation system through user space services, Windows also audits accesses to environmental/configuration data for each program. While environment variables are also passed in memory from parent to child processes, as in Linux, individual configuration fields are persisted as auditable key-value pairs in the Windows registry and are regularly accessed when

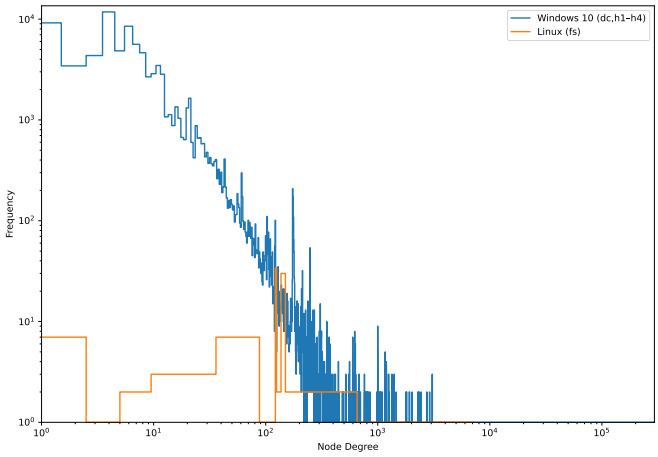


Fig. 3: A comparison of process-to-data connectivity (Node Degree) on Linux (`fs`) and Windows 10 (`dc`, `h1`–`h4`) in CARBANAKv2.

creating new processes. As a result, Windows presents another novel dependency explosion problem in which each process is connected to orders of magnitude more data objects as shown in Figure 3. In CARBANAKv2, the most data-connected Windows process is linked to over 290,000 data objects, whereas the most connected Linux process is linked to fewer than 8,000. Moreover, the median Windows process is connected to 7 data objects, while the median Linux process is connected to none. Most of this data connectivity does not convey inter-process information flow, and is thus less likely to frustrate the results of provenance-based threat investigations. That said, this high degree of data connectivity greatly complicates graph visualization.

Additionally, Windows data paths, including both files and registry keys, seem to more frequently contain non-deterministic path segments, such as Security IDs (SIDs) or hashes, as compared to Linux. For simplicity, we will refer to these non-language components as "random." Due to the high degree of random segments in data paths, the high degree of data connectivity may also cause problems for learning algorithms attempting to create a general model of system activity (e.g., anomaly detection).

To mitigate this problem, our high-level intuition is once again to make Windows analysis more like Linux. On Linux, the opacity of environmental and configuration data was not generally a problem for threat investigation;[3] these values were thought of as an extension of the internal process state. Further, graph visualization and process modeling would greatly benefit from reducing the overall number of data connections, which we can achieve by clustering related data objects together.

[3]Except in the case of configuration-based attacks [16]!

TABLE II: Automated reduction of unique random components in Windows data object paths. The KL divergence filter removes the majority of random components, while the 4-gram model further reduces the remaining components by 9.8% of what the KL divergence filter misses.

Filter	# Components	% Reduction
None	967174	0.0
KL Divergence	217581	77.5
4-gram	650567	32.7
Both	196225	79.7

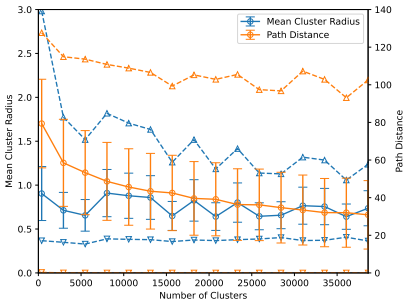


Fig. 4: Cluster metrics versus number of clusters for our path embedding clusters. The solid lines track the mean values with error bars for the standard deviations, while the dotted lines show the minimum and maximum values. Overall, we see that the cluster radii level off very early, while the path distance also quickly levels off by around 10000 clusters.

A. Clustering Data Entities Heuristic

Our first approach is to simply merge semantically-related data entities into a single vertex to reduce graph complexity.

1) Masking Non-Deterministic Path Segments: The first step in our procedure is to replace path segments that are non-deterministic (or, expected to vary between devices) with static pseudonyms. As noted above, some of the "random" components are actually structured data, such as Windows Security Identifiers (SIDs) [28] or UUIDs [4], that we easily identify with regular expressions.

We identify the remaining non-determism in path segments – including segment substrings – using a combination of character entropy and a 4-gram model. We use a basic relative entropy scheme where we compute the total distribution p of all characters in the components, then rank each component's distribution q based on the Kullback-Leibler (KL) divergence from the total distribution [18]:

$$D_{KL}(p,q) = \sum_c p(c) \ln \frac{p(c)}{q(c)} \tag{1}$$

The KL divergence measures how similar a specific component's characters are to the generally expected distribution of characters in path components; thus, we expect $D_{KL}(p, q_r)$ for a random component q_r to be larger than that of a nonrandom component, as the random components do not fit the general distribution of characters in natural language components. However, the separation between random and non-random components is somewhat weak: it is possible for a random component to randomly contain characters that seem plausible, as there is no consideration of the order of the characters. To avoid removing real components, we only filter components above a conservative threshold at which all the components clearly appear to be random.

The KL divergence is able to separate out the vast majority of random components, but permits some random components with lower relative entropies that are not filtered out. To catch these remaining components, we found success using a pretrained 4-gram model designed to separate programming identifiers from random characters, as path components are often structured similarly [15]. The 4-gram model's limitations, such as being unable to handle short strings of 6 characters or less, are covered by the KL divergence model, while it successfully further identifies random components missed by the KL divergence filter. Table II shows the reduction performance of these filters on all unique components in our datasets. As expected, the KL divergence filter removes the majority of the random components, achieving a 77.5% reduction on its own, but the 4-gram model is able to further remove some components that happen to appear non-random, reducing 9.8% of what the KL divergence filter missed, for a total reduction of 79.7% of random components being automatically removed. All segments (or segment substrings) identified as non-deterministic are replaced special tokens, e.g., a UUID is replaced with `UUID` and random components are replaced with `RND`. Vertices whose data paths are identical matches after this procedure are trivially merged.

2) Clustering Related Data Paths: We continue by clustering data entities with non-identical paths, using the path hierarchy as a cue for identifying semantically-related data entities. To achieve this, we train a clustering model on data path embeddings. Data entities whose path fall into the same cluster are merged into a single vertex, using the data path closest to the centroid of the cluster as a vertex label. Data entities that do not fall within a cluster are unchanged.

To cluster the data objects, we first train an embedding model for data paths. Each path converted to lowercase and then decomposed into a sequence of tokens by splitting on all punctuation, including both '\' (the Windows path separator) and other symbols such as underscores or periods. We split on punctuation beyond just the path separator as Windows names often contain long combinations of multiple components. The sequences are then embedded into vectors using Doc2Vec [20]

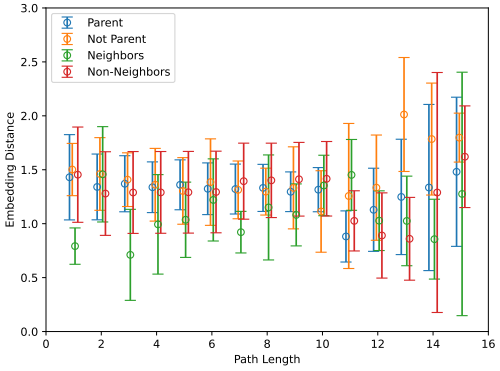
We use k-means clustering due to the volume of data paths (around 4 million in our dataset) and expectation that the number of clusters will be relatively large. While we aim to reduce the number of graph nodes by grouping similar paths together, it is better to err conservatively rather than grouping together unrelated data paths. To optimize the number of clusters, we create a path distance metric by splitting each path into directories and the final object name. At each level in the path, we calculate the entropy of the distribution of

names at that level:

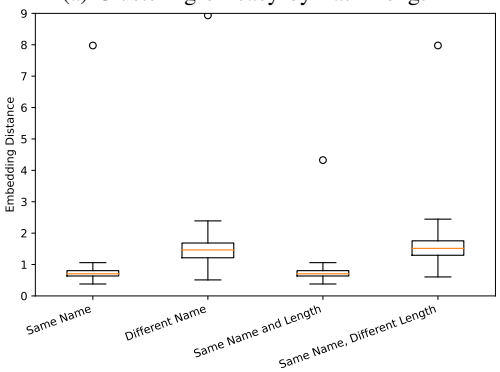
$$H(l) = -\sum_{n} p(n,l) \ln p(n,l) \tag{2}$$

where $p(n,l)$ is the frequency of name n at a level l in the path divided by the total number of names at level l. We then define the total path distance of a cluster to be the sum of these entropies at every level.

We choose the smallest number of clusters that simultaneously achieves a reasonably low path distance. Figure 4 shows our results for both this path distance and the mean radius of each cluster in embedding space, defined as the average distance of each path embedding to the cluster centroid. The path distance per cluster levels off fairly quickly, so for our data, we use 10000 as our number of clusters.



(a) Clustering efficacy by Path Length



(b) Clustering Efficacy for Non-Neighbors

Fig. 5: Box and whisker plot comparisons of Euclidean embedding distance between related data paths in ATLASv2 Host 1. Fig. 5a reports distance between a data object and its *Parent* directory, all *Non-Parent* directories, *Neighbor*ing data in the same parent directory, and all *Non-Neighboring* data. Fig. 5b reports distances between data objects with the same name that reside in different directories.

While embedding-based featurization of data paths is a nearly universal practice in provenance-based machine learning applications (e.g., [7], [8], [17], [32], [36]), we are not aware of any ablation study that specifically reports how well the resulting model encodes path hierarchies. A high-level characterization can be found in Figure 5. Figure 5a compares related data objects while controlling for path length, with shorter path lengths appearing much more commonly in the evaluation dataset. It can be seen that the euclidean distance between neighboring data objects is consistently closer than non-neighbors, except for longer path lengths (10+) where training data was scarce. However, the Doc2Vec model struggles to encode the relationship between a data object and its parent directory, regardless of path length. An additional concern is whether the model encodes similarities between files in different directories. Figure 5b compares the distances between files of the same name that are stored in different directories. We see that the model is extremely effective at placing objects with the same name, but this efficacy is again dependent on path length. These results suggest the opportunity for further improvement on file path embedding, which is a common pre-processing step in provenance-based anomaly detection systems. However, the Doc2Vec model appears sufficient for data object deduplication, particularly for objects in the same parent directory.

B. *Pruning Leaf Data Entities Heuristic*

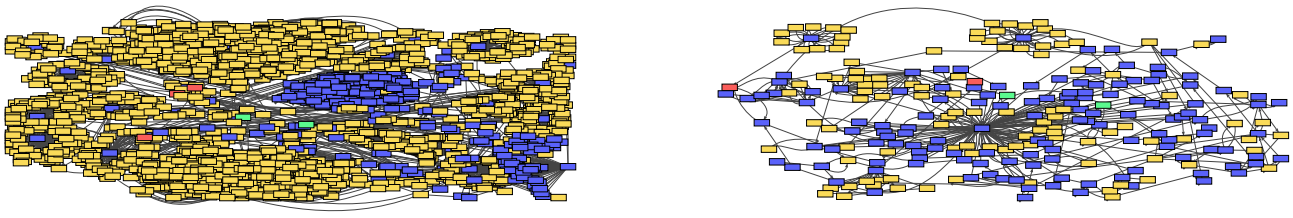
As a large proportion of Window data accesses are to program-specific state and configuration data, it is not surprising that most of these accesses do not facilitate inter-process information flows. That is, most data entity vertices are "leaves" in the graph that are only accessed by a single process. For most applications of data provenance, it is safe to prune these leaves, treating them instead as opaque internal process state. On a live system where the provenance graph is dynamic and evolving, it would likely be possible to identify single process data entities and proactively prune them from the provenance graph. This procedure would be similar to Lee et al.'s LogGC system, which identified and removed accesses to program's temporary file activities that did not convey inter-process information flows [22]. But on a static graph – such as the result of a forward/backward trace in a threat investigation – it is sufficient to simply prune data entities with fewer than two process neighbors. While not conceptually satisfying, we find this heuristic to be extremely helpful when attempting to visualize attack graphs or improve the learning rate of machine learning models on limited training data.

VI. Evaluation

We now evaluate the efficacy of these heuristics by measuring how they reduce graph complexity in threat investigation scenarios. Specifically, we use the REAPr labeling methodology to generate attack graphs [23]. For each host in the CARBANAKv2 dataset, we review the experimenter logs to manually identify the root cause(s) and impact(s) of the attack on each host. Root causes were always the Remote IP address from which the attack was being launched, while Impacts were the Remote IP addresses the attacker was moving lateral to as well as any data artifacts left on the machine. After this limited manual labeling step, we then used the REAPr scripts to procedurally generate the attack chain on each host. These scripts executed a forward trace from the identified root causes,

TABLE III: Attack graph sizes for Windows hosts when applying our dependence explosion mitigations.

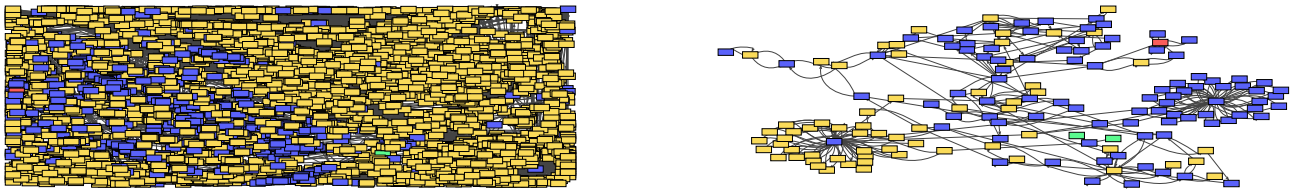
Device	Control		Inter-Program (§IV-A)		Intra-Program (§IV-B)		Clustering (§V-A)		Leaf Data (§V-B)		All Techniques		
	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	$\|V\|$	$\|E\|$	% Reduction
fs	33	53											
dc	3173	6360	531	1013	2410	4847	1728	6360	574	1139	248	478	92.5
h1	6597	12930	6246	12453	6149	12041	2104	12930	1015	1689	159	316	97.6
h2	5755	11210	1463	2435	4641	6415	1385	11210	463	854	130	249	97.8
h3	24124	47793	4592	8846	21931	44361	4853	47793	4893	9932	527	1154	97.6
h4	6083	11258	1620	2604	5181	9880	2379	11258	1489	2669	436	999	91.1



(a) Before filtering

(b) After filtering

Fig. 6: Attack subgraph on `dc` before and after applying all heuristics. The filtered graph exhibits a substantial reduction in fan-out and overall complexity.



(a) Before filtering

(b) After filtering

Fig. 7: Attack subgraph on `h1` before and after applying all heuristics. The resulting graph is compact and visually isolates attack-relevant behavior.

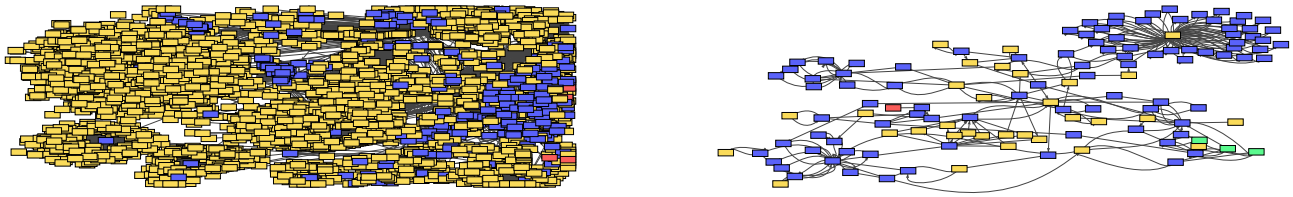
a backward trace from the identified impacts, then returned the intersection of the two traces as the attack graph.

The four heuristics described above – Inter-Process Filtering (§IV-A), Intra-Process Filtering (§IV-B), Clustering Data Entities (§V-A), and Pruning Leaf Data Entities (§V-B) – were all implemented as modifications to the REAPr graph traversal logic that can be independently toggled on and off. To evaluate the efficacy of the heuristics, we enabled each one independently, then all at the same time, comparing the results to a control graph in which no heuristics were activated. We also present the Linux device (`fs`) as an additional control, for which no heuristics were active.

Table III presents our results. While the Linux-based control graph (`fs`) is comprised of just a few dozen vertices, the smallest control graph on Windows (`dc`) is 100 times larger. As these two devices were servers with roughly comparable workloads, we can attribute this explosion of complexity to Windows-specific behaviors

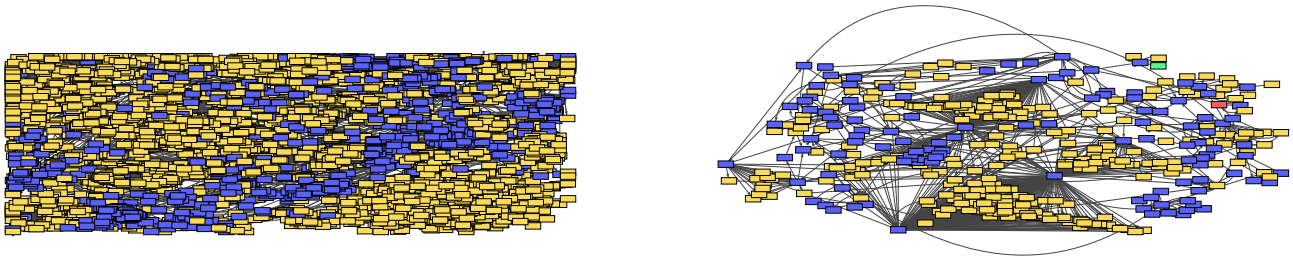
Looking at the performance of each heuristic, we see that all heuristics reduce graph complexity, but are not sufficient in isolation. For example, the inter-program heuristic reduces the number of vertices of most graphs by around 70, but the subgraphs with thousands of nodes are still too large given the scale of the attack. However, with all heuristics applied, most graphs begin to approach a usable size, specifically `dc`, `h1`, and `h2`. Overall, the total reduction for all graphs is quite similar. ranging from around 91% in the worst case to 98% in the best case. The baseline graph for `h3` is about 4 times larger than the other hosts; consequently, its fully-filtered graph is still too large to be very interpretable. We note that the attack log for `h3`, which was the first host compromised after `dc`, was the longest of any in the engagement due to various operational issues that arose during the engagement. `h4`, and to a lesser extent, `dc`, are also notable in responding less to reductions.

Figure 10 first shows the attack subgraph for the Linux host `fs`. As discussed earlier, the Linux control graph is already compact and lacks the high-degree hub structure observed on Windows, and thus does not require additional filtering to be



(a) Before filtering (b) After filtering

Fig. 8: Attack subgraph on `h2` before and after applying all heuristics.



(a) Before filtering (b) After filtering

Fig. 9: Attack subgraph on `h4` before and after applying all heuristics. Although the filtered graph remains comparatively dense, the reduction in complexity is substantial relative to the baseline.

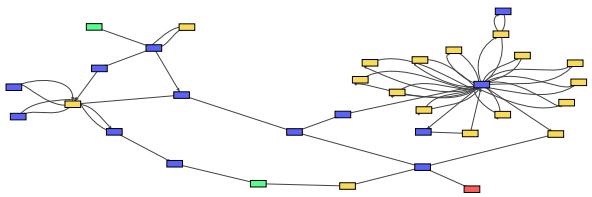


Fig. 10: Attack subgraph for the Linux host `fs`. The graph is already compact and does not exhibit the high-degree hub structure observed on Windows hosts.

interpretable.

Figure 6, Figure 7, Figure 8, and Figure 9 show representative Windows attack graphs before and after applying all heuristics. For `dc`, `h1` and `h2`, the combined heuristics produce a stark reduction in graph size and fan-out, yielding compact attack subgraphs in which attack-relevant behavior is visually isolated. For `h4`, while the fully-filtered graph remains comparatively dense, the reduction is still substantial relative to the baseline, and the resulting subgraph is significantly more structured and interpretable than the unfiltered control. We omit the before-and-after visualization for `h3`, as the unfiltered attack graph was too large to be rendered by our visualization tooling; however, quantitative results in Table III confirm that the applied heuristics still achieve a substantial reduction for this host. Across all cases, we verified that all attack steps remain fully present and logically connected according to the ground-truth attack timeline.

VII. Discussion

Our experimental results underscore the feasibility of performing provenance analysis on EDR telemetry. In spite of the absence of key information flow events in the EDR event schema, we were consistently able to trace attacks from their root causes to their impacts (and vice versa) on EDR telemetry. The heuristics we developed to aid in this analysis did not break graph connectivity and reduced graph complexity by 91% to 98%. That said, the attack chains were still difficult to visualize with all heuristics applied. We believe that simple heuristics like the ones that appear in this work can (and should) be refined to further improve tracing. If we hit a ceiling on the level of reduction that is possible using domain constraints, attack summarization systems (e.g., [1], [17], [36]) can be used to produce a usable attack explanation.

If statistical learning mechanisms will ultimately be necessary to generate precise provenance graphs from EDR telemetry, is there a point to leveraging domain knowledge constraints prior to deploying Machine Learning (ML)? Indeed, it would be astonishing if ML was unable to approximate the heuristics we propose given sufficient training data. One advantage of domain knowledge is that we can specify a clear upper limit on edge pruning. We bound our heuristics to programs for which we were able to verify that an attacker would need to escalate privilege in order to tamper with, making our heuristics more

robust to adversarial abuse. Out-of-band constraints would need to be placed on the ML to arrive at a similar outcome, as this domain knowledge (e.g., file permissions) is not present in the EDR event telemetry. It is also worth noting that a statistical mechanism would suffer from the same downsides, i.e., an attacker that can manipulate a program's control flow could abuse either the domain-based or statistical mechanism.

Another reason to simultaneously pursue statistical and domain knowledge heuristics is to establish ground truth against which to measure statistical mechanisms. Provenance-based intrusion detection research has shown tremendous promise, but these results are predicated on subjective and underdocumented ground truth labeling. The REAPr project is a response to this inconsistency [23], leveraging established provenance analysis techniques to semi-automatically label attack data with only limited experimenter intervention. However, when we attempted to label the CARBANAKv2 dataset using the REAPr methodology and scripts, we quickly realized that the original procedure was designed with Linux/Unix in mind and that far too much dependency explosion was creeping into the attack graph on Windows EDR datasets. We developed these heuristics as a method of adapting the REAPr methodology to Windows EDR, providing deterministic and objective mechanisms for denoising Windows information flows.

The threat model in our work builds on an assumption that attackers cannot manipulate programs and data entities belonging to the operating system. The attacker would need to escalate privilege in order to manipulate the Windows system services or data artifacts that our heuristics are built on, at which point the entire operating system and EDR will have already been compromised. If we were to relax our system-only requirement, it would be possible to apply these heuristics to other programs that generate large volumes of false provenance. For example, the Google and Adobe update programs both engage in tremendous volumes of inter-process communication, both of which ended up appearing in the attack chains of some CARBANAKv2 devices. We believe that further development of domain-based heuristics will be an important part of transitioning provenance analysis to practice.

VIII. Conclusion

In this work we have identified the challenges of conducting provenance analysis on Endpoint Detection & Response telemetry. We also highlighted several novel intricacies of conducting provenance analysis on Windows devices that had gone unreported in prior work on the Windows platform. We go on to demonstrate that in spite of the imprecision of Windows- and EDR-based analysis, domain-based heuristics can be employed to bring provenance analysis more in line with its performance on syscall-based Linux logs. It is our hope that these findings will prompt further research on integrations of provenance with commercial security products.

Acknowledgments

We thank the reviewers for their helpful suggestions. This work was supported in part by NSF grant 2055127 and the VMware University Research Fund.

Availability

Our code is available at https://bitbucket.org/sts-lab/reapr-ground-truth, and our dataset is available at https://bitbucket.org/sts-lab/carbanakv2/.

References

[1] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z. Berkay Celik, Xiangyu Zhang, and Dongyan Xu. ATLAS: A sequence-based learning approach for attack investigation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3005–3022. USENIX Association, August 2021.

[2] Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical Intrusion Detection and Investigation using Whole-system Provenance . In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3533–3551, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[3] Crowdstrike, Inc. Crowdstrike Falcon Events Data Dictionary. https://falcon.crowdstrike.com/documentation/26/events-data-dictionary, April 2024.

[4] Kyzer R. Davis, Brad Peabody, and P. Leach. Universally Unique IDentifiers (UUIDs). RFC 9562, May 2024.

[5] Pengcheng Fang, Peng Gao, Changlin Liu, Erman Ayday, Kangkook Jee, Ting Wang, Yanfang (Fanny) Ye, Zhuotao Liu, and Xusheng Xiao. Back-Propagating system dependency impact for attack investigation. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 2461–2478, Boston, MA, August 2022. USENIX Association.

[6] MITRE Center for Threat-Informed Defense. Adversary emulation library. https://github.com/center-for-threat-informed-defense/adversary_emulation_library, 2024.

[7] Akul Goyal, Xueyuan Han, Adam Bates, and Gang Wang. Sometimes, You Aren't What You Do: Mimicry Attacks against Provenance Graph Host Intrusion Detection Systems. In *30th ISOC Network and Distributed System Security Symposium*, NDSS'23, February 2023.

[8] Akul Goyal, Gang Wang, and Adam Bates. R-CAID: Embedding Root Cause Analysis within Provenance-based Intrusion Detection . In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3515–3532, Los Alamitos, CA, USA, May 2024. IEEE Computer Society.

[9] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical Provenance Analysis for Endpoint Detection and Response Systems. In *41st IEEE Symposium on Security and Privacy (SP)*, Oakland'20, May 2020.

[10] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. NoDoze: Combatting Threat Alert Fatigue with Automated Provenance Triage. In *26th ISOC Network and Distributed System Security Symposium*, NDSS'19, February 2019.

[11] Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Dawei Wang, Zhengzhang Chen, Zhichun Li, Junghwan Rhee, Jiaping Gui, and Adam 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*, ACSAC '20, pages 165–178, New York, NY, USA, Dec 2020. Association for Computing Machinery.

[12] Wajih Ul Hassan, Mohammad Noureddine, Pubali Datta, and Adam Bates. OmegaLog: High-Fidelity Attack Investigation via Transparent Multi-layer Log Analysis. In *27th ISOC Network and Distributed System Security Symposium*, NDSS'20, February 2020.

[13] Md Nahid Hossain, Sanaz Sheikhi, and R. Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1139–1155, 2020.

[14] Md Nahid Hossain, Junao Wang, R. Sekar, and Scott D. Stoller. Dependence-preserving data compaction for scalable forensic analysis. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, pages 1723–1740, Berkeley, CA, USA, 2018. USENIX Association.

[15] Michael Hucka. Nostril: A nonsense string evaluator written in python. *Journal of Open Source Software*, 3(25):596, 2018.

[16] Muhammad Adil Inam, Wajih Ul Hassan, Ali Ahad, Adam Bates, Rashid Tahir, Tianyin Xu, and Fareed Zaffar. Forensic Analysis of Configuration-based Attacks. In *29th ISOC Network and Distributed System Security Symposium*, NDSS'22, February 2022.

[17] Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. ORTHRUS: Achieving High Quality of Attribution in Provenance-based Intrusion Detection Systems. In *34th USENIX Security Symposium*, Sec'25. USENIX Association, August 2025.

[18] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[19] Butler Lampson. Perspectives on protection and security. In *SOSP History Day 2015*, SOSP '15, New York, NY, USA, 2015. Association for Computing Machinery.

[20] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China, 6 2014. PMLR.

[21] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High Accuracy Attack Provenance via Binary-based Execution Partition. In *Proceedings of NDSS '13*, February 2013.

[22] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. LogGC: Garbage Collecting Audit Log. In *Proceedings of the 2013 ACM SIGSAC conference on Computer and Communications Security*, CCS '13, pages 1005–1016, New York, NY, USA, 2013. ACM.

[23] Jason Liu, Adil Inam, Akul Goyal, Kim Westfall, Andy Riddle, and Adam Bates. Reapr: Recovery every attack process. https://bitbucket.org/sts-lab/reapr-ground-truth, 2023.

[24] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a Timely Causality Analysis for Enterprise Security. In *Proceedings of the 25th ISOC Network and Distributed System Security Symposium*, NDSS'18, San Diego, CA, USA, February 2018.

[25] Sadegh M. Milajerdi, Birhanu Eshete, Rigel Gjomemo, and Venkat N. Venkatakrishnan. Propatrol: Attack investigation via extracted high-level tasks. In Vinod Ganapathy, Trent Jaeger, and R.K. Shyamasundar, editors, *Information Systems Security*, pages 107–126, Cham, 2018. Springer International Publishing.

[26] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. Protracer: Towards practical provenance tracing by alternating between logging and tainting. In *23rd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016.

[27] Emaad Manzoor, Sadegh M. Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1035–1044, New York, NY, USA, 2016. Association for Computing Machinery.

[28] Microsoft. Security identifiers, 2025.

[29] Andy Riddle, Kim Westfall, and Adam Bates. Atlasv2: Atlas attack engagements, version 2, 2023.

[30] D.A. Solomon. The windows nt kernel architecture. *Computer*, 31(10):40–47, 1998.

[31] Statcounter. Desktop operating system market share worldwide, 2025.

[32] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Jungwhan Rhee, Zhengzhang Zhen, Wei Cheng, Carl A. Gunter, and Haifeng chen. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *27th ISOC Network and Distributed System Security Symposium*, NDSS'20, February 2020.

[33] Zhang Xu, Zhenyu Wu, Zhichun Li, Kangkook Jee, Junghwan Rhee, Xusheng Xiao, Fengyuan Xu, Haining Wang, and Guofei Jiang. High fidelity data reduction for big data security dependency analyses. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 504–516, New York, NY, USA, 2016. ACM.

[34] Carter Yagemann, Mohammad Noureddine, Wajih Ul Hassan, Simon Chung, Adam Bates, and Wenke Lee. Validating the integrity of audit logs against execution repartitioning attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, 2021.

[35] Jun Zengy, Xiang Wang, Jiahao Liu, Yinfang Chen, Zhenkai Liang, Tat-Seng Chua, and Zheng Leong Chua. Shadewatcher: Recommendation-guided cyber threat analysis using system audit records. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 489–506, 2022.

[36] Bo Zhang, Yansong Gao, Changlong Yu, Boyu Kuang, Zhi Zhang, Hyoungshick Kim, and Anmin Fu. TAPAS: An Efficient Online APT Detection with Task-guided Process Provenance Graph Segmentation and Analysis. In *34th USENIX Security Symposium*, Sec'25. USENIX Association, August 2025.

APPENDIX

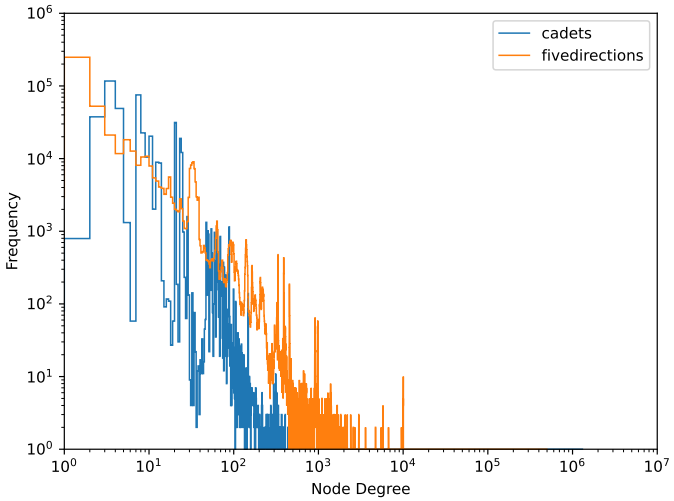


Fig. 11: An additional comparison of process connectivity (Node Degree) in *nix vs. Windows hosts using the DARPA Transparent Computing Engagement 3 datasets. *Cadets* is running a BSD-based Unix distribution, while *FiveDirections* is a Windows device.