Tactical Provenance Analysis for Endpoint Detection and Response Systems

Wajih Ul Hassan
University of Illinois at Urbana-Champaign
whassan3@illinois.edu

Adam Bates
University of Illinois at Urbana-Champaign
batesa@illinois.edu

Daniel Marino
NortonLifeLock Research Group
daniel.marino@nortonlifelock.com

***Abstract*—Endpoint Detection and Response (EDR) tools provide visibility into sophisticated intrusions by matching system events against known adversarial behaviors. However, current solutions suffer from three challenges: 1) EDR tools generate a high volume of false alarms, creating backlogs of investigation tasks for analysts; 2) determining the veracity of these threat alerts requires tedious manual labor due to the overwhelming amount of low-level system logs, creating a "needle-in-a-haystack" problem; and 3) due to the tremendous resource burden of log retention, in practice the system logs describing long-lived attack campaigns are often deleted before an investigation is ever initiated.**

This paper describes an effort to bring the benefits of data provenance to commercial EDR tools. We introduce the notion of *Tactical Provenance Graphs* (TPGs) that, rather than encoding low-level system event dependencies, reason about causal dependencies between EDR-generated threat alerts. TPGs provide compact visualization of multi-stage attacks to analysts, accelerating investigation. To address EDR's false alarm problem, we introduce a threat scoring methodology that assesses risk based on the temporal ordering between individual threat alerts present in the TPG. In contrast to the retention of unwieldy system logs, we maintain a minimally-sufficient skeleton graph that can provide linkability between existing and future threat alerts. We evaluate our system, RapSheet, using the Symantec EDR tool in an enterprise environment. Results show that our approach can rank truly malicious TPGs higher than false alarm TPGs. Moreover, our skeleton graph reduces the long-term burden of log retention by up to 87%.

I. Introduction

Today's system intrusions are remarkably subtle and sophisticated. Exemplified by the "living-off-the-land" attack strategies of *Advanced Persistent Threats* (APTs), adversaries now lurk in the enterprise network for longer periods to extend their reach before initiating a devastating attack. By avoiding actions that would immediately arouse suspicion, the dwell time for such attackers can range from weeks to months, as was the case in numerous data breaches including Target [1], Equifax [2], and the Office of Personnel Management [3].

The canonical enterprise solution for combatting APTs is known as *Endpoint Detection and Response (EDR)*. EDR tools constantly monitor activities on end hosts and raise threat alerts if potentially-malicious behaviors are observed. In contrast to signature scanning or anomaly detection techniques, EDR tools hunt threats by matching system events against a knowledge base of adversarial Tactics, Techniques, and Procedures (TTPs) [4], which are manually-crafted expert rules that describe low-level attack patterns. TTPs are hierarchical, with tactics describing "why" an attacker performs a given action while techniques and procedures describe "how" the action is performed. According to a recent survey, 61% of organizations deploy EDR tools primarily to provide deep visibility into attacker TTPs and facilitate threat investigation [5]. MITRE's ATT&CK [6] is a publicly-available TTP knowledge base which is curated by domain experts based on the analysis of real-world APT attacks, and is one of the most widely used collections of TTPs [7], [8], [9]. In fact, all 10 of the top EDR tools surveyed by Gartner leverage the MITRE ATT&CK knowledge base to detect adversary behavior [10].

While EDR tools are vital for enterprise security, three challenges undermine their usefulness in practice. The first challenge is that TTP knowledge bases are optimized for recall, not precision; that is, TTP curators attempt to describe all procedures that have any possibility of being attack related, even if the same procedures are widely employed for innocuous purposes. An obvious example of this problem can be found in the "File Deletion" Technique [11] in MITRE ATT&CK – while file deletion may indicate the presence of evasive APT tactics, it is also a necessary part of benign user activities. As a result, EDR tools are prone to high volumes of false alarms [12], [13], [14], [15]. In fact, EDR tools are one of the key perpetrators of the "threat alert fatigue" problem[1] that is currently plaguing the industry. A recent study found that the biggest challenge for 35% of security teams is keeping up with the sheer volume of alerts [16]. Consequently, the true attacks detected by EDR tools are at risk of being lost in the noise of false alerts.

The second challenge comes from the dubious nature of EDR-generated threat alerts. After receiving an alert, the first job of a cyber analyst is to determine the alert's veracity. For validation, cyber analysts review the context around the triggered alert by querying the EDR for system logs. Although EDR tools collect a variety of useful contextual information, such as running processes and network connections, the onus is on the cyber analyst to manually piece together the chain of system events. If the alert is deemed truly suspicious, the cyber analyst then attempts to recover and correlate various stages

[1] A phenomenon in which cyber analysts do not respond, or respond inadequately, to threat alerts because they receive so many each day.

of the attack through further review of enormous system logs. Security Indicator & Event Management (SIEM) products are often the interface through which this task is performed (e.g., Splunk [17]), allowing analysts to write long ad-hoc queries to join attack stages, provided that they have the experience and expertise to do so.

Long-term log retention is the third challenge for existing EDR tools. It is still commonplace for EDR tools to delete system logs soon after their capture. Logs are commonly stored in a small FIFO queue that buffers just a few days of audit data [18], [19], such that system events are commonly unavailable when investigating a long-lived attack. Even worse, unless an organization staffs a 24/7 security team, the audit data for an alert that fires over the weekend may be destroyed by Monday. This indicates that despite advancements in the efficiency of causal analysis, long-term retention of system log simply does not scale in large enterprises. Not only does this mean that EDR tools cannot reap the benefits of causal analysis during threat investigation, but it also means that current EDR tools lack the necessary context to understand the interdependencies between related threat alerts.

To aid alert validation and investigation, it would seem that the research community has already arrived at a solution – *data provenance*. Data provenance analysis can be applied to system logs to parse host events into provenance graphs that describe the totality of system execution and facilitate *causal analysis* of system activities. In recent years, significant advancements have been made that improve the fidelity [20], [21], [22], [23], [24], [25], [26], [27], [28] and efficiency [29], [30], [31], [32], [33], [34], [35], [36], [37] of causal analysis, and recent results indicate that causal analysis can even be leveraged to improve alert triage [38], to detect intrusions [39], [40], [41], and to derive alert correlations [42], [43]. Better yet, most causal analysis engines are based on commodity auditing frameworks (e.g., Windows ETW), which analyze the same information stream that is already being used by EDR tools.

Based on data provenance, we introduce a new concept in this paper which we call *Tactical Provenance* that can reason about the causal dependencies between EDR-generated threat alerts. Those causal dependencies are then encoded into a tactical provenance graph (TPG). The key benefit of TPG is that a TPG is more succinct than a classical whole-system provenance graph because it abstracts away the low-level system events for cyber analysts. Moreover, TPGs provide higher-level visualizations of multi-stage APT attacks to the analysts, which help to accelerate the investigation process.

To tackle the threat alert fatigue problem, we present methods of triaging threat alerts based on analysis of the associated TPGs. APT attacks usually conform to a "kill chain" where attackers perform sequential actions to achieve their goals [44], [45]. For instance, if the attacker wants to exfiltrate data, they must first establish a foothold on a host in the enterprise, locate the data of interest (i.e., reconnaissance), collect it, and finally transmit the data out of the enterprise. Our key idea is that *these sequential attack stages seen in APT campaigns can be leveraged to perform risk assessment*. We instantiate this idea in a threat score assignment algorithm that inspects the temporal and causal ordering of threat alerts within the TPG to identify sequences of APT attack actions. Afterward, we assign threat score to that TPG based on the identified sequences and use that threat score to triage TPGs.

To better utilize the limited space available on hosts for long-term log storage, we present a novel log reduction technique that, instead of storing all the system events present in the logs, maintains a minimally-sufficient skeleton graph. This skeleton graph retains just enough context (system events) to not only identify causal links between the existing alerts but also any alerts that may be triggered in the future. Even though skeleton graphs reduce the fidelity of system logs, they still preserve all the information necessary to generate TPGs for threat score assignment, risk assessment, and high-level attack visualization.

In summary, we make the following contributions:

- We propose tactical provenance graphs (TPGs), a new representation of system events that brings the benefits of data provenance into the EDR ecosystem.
- We present a threat scoring algorithm based on TPGs to rank threat alerts.
- We present a novel log reduction scheme that can reduce the storage overhead of system logs while preserving causal links between existing and future threat alerts.
- We integrate our prototype system, RapSheet, into the Symantec EDR tool. We evaluated RapSheet with an enterprise dataset to show that RapSheet can rank truly malicious TPGs higher than false alarm TPGs. Moreover, our skeleton graph reduces the storage overhead of system logs by up to 87% during our experiments.

II. Background & Motivation

A. Data Provenance

Data provenance is a promising approach to investigate cyber attacks [46]. In the context of operating systems, data provenance techniques parse logs generated by system-level auditing frameworks, such as Windows ETW [47] and Linux Audit [48] into a provenance graph. Provenance graphs encode causal dependence relations between system subjects (e.g., processes) and system objects (e.g., files, network sockets). Given a symptom event of an attack, cyber analysts can find the root cause of the attack by issuing a backward tracing query on the provenance graph. After identifying the root cause, cyber analysts can also issue a forward tracing query to understand the ramifications of the same attack. Thus, data provenance is a powerful technique for attack attribution.

B. MITRE ATT&CK and EDR tools

MITRE ATT&CK is a publicly-available knowledge base of adversary tactics and techniques based on real-world observations of cyber attacks. Each tactic contains an array of techniques that have been observed in the wild by malware or threat actor groups. Tactics explain what an attacker is trying to accomplish, while techniques[2] and procedures[3] represent how an adversary achieves these tactical objectives (e.g., How are attackers escalating privileges? or How are adversaries exfiltrating data?) The MITRE ATT&CK Matrix [49] visually arranges all known tactics and techniques into an easy-to-understand format. Attack tactics are shown at the top of the matrix. Individual techniques are listed down each column. A completed attack sequence would be built by moving through the tactic columns from left (Initial Access) to right (Impact) and performing one or more techniques from those columns. Multiple techniques can be used for one tactic. For example, an attacker might try both an attachment (T1193) and a link (T1192) in a spearphishing exploit to achieve the Initial Access tactic. Also, some techniques are listed under multiple tactics since they can be used to achieve different goals.

One common use of MITRE ATT&CK tactics and techniques is in malicious behavior detection by Endpoint Detection and Response (EDR) tools. EDR tools serve four main purposes in enterprises: 1) detection of potential security incidents, 2) scalable log ingestion and management, 3) investigation of security incidents, and 4) providing remediation guidance. To implement those capabilities, EDR tools record detailed, low-level events on each host including process launches and network connections. Typically, this data is stored locally on end hosts. Events that are of potential interest may be pushed to a central database for alerting and further analysis, during which additional events may be pulled from the endpoint to provide forensic context. EDR tools provide a rule matching system that processes the event stream and identifies events that should generate alerts. Major EDR vendors [7], [8], [9] already provide matching rules to detect MITRE ATT&CK TTPs; however, cyber analysts can also add new rules to detect additional TTPs at an enterprise where the EDR tool is deployed.

C. Motivating Example

We now consider a live attack exercise that was conducted by the Symantec's red team over a period of several days; this exercise was designed to replicate the tactics and techniques of the APT29 threat group. APT29 is one of the most sophisticated APT groups documented in the cyber security community [50]. Thought to be a Russian state-sponsored group, APT29 has conducted numerous campaigns with different tactics that distribute advanced, custom malware to targets located around the globe. Discovered attacks attributed to APT29 have been carefully analyzed by MITRE, yielding a known set of tactics and techniques that APT29 commonly use to achieve their goals [51]. In this exercise, different techniques were performed from that known set, ranging from Registry Run Keys (T1060) to Process Injection (T1055). These techniques allowed us to observe different MITRE tactics including persistence, privilege escalation, lateral movement, and defense evasion.

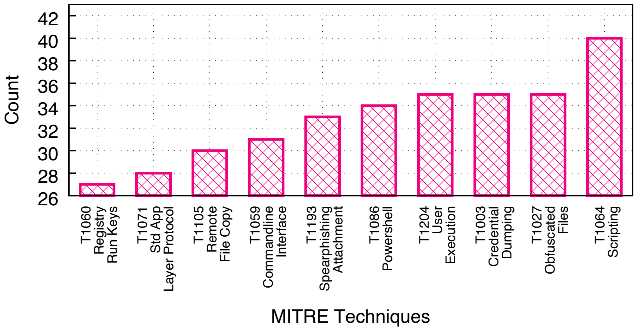


Fig. 1: Top 10 techniques based on the number of times exploited by 93 MITRE-curated APT groups. 6 of these 10 techniques are benign in isolation and occur frequently during normal system execution.

1) Limitations of EDR tools: Existing EDR tools excel at scalably identifying potentially malicious low-level behaviors in real-time. They can monitor hundreds or thousands of hosts for signs of compromise without event congestion. However, they suffer from some major usability and resource issues which we list below.

False-positive Prone. Existing EDR tools are known to generate many false alarms [12], [13], [14] which lead to the threat alert fatigue problem. The main reason for this high false alarm rate is that many MITRE ATT&CK behaviors are only sometimes malicious. For example, MITRE ATT&CK lists a technique called "File Deletion" T1107 under the "Defense Evasion" tactic. Finding this individual behavior and generating an alert is straightforward. But how would the analyst discern whether this file deletion is the result of normal system activity, or an attempt by an attacker to cover his tracks? Alerting on individual MITRE techniques generates false alarms and requires a human in the loop for alert validation.

To further quantify how many techniques from the MITRE ATT&CK knowledge-base can be benign in isolation, we took techniques used by 93 APT attack groups provided by MITRE and identified the most used techniques from these attack groups. Figure 1 shows the top ten most used techniques. After manual inspection, we found that 6 of 10 techniques may be benign in isolation, and in fact occur frequently during typical use. For example, the Powershell technique (T1086) can be triggered during a normal execution of applications like Chrome or Firefox. During our attacks simulation period, the Symantec EDR generated a total of 58,096 alerts on the 34 machines. We analyzed these alerts and found that only 1,104 were related to true attacks from the APT29 exercise and from other attack simulations we describe later. The remaining 56,992 were raised during benign activity, yielding a precision of only 1.9%.

Laborious Context Generation. To investigate and validate the triggered alerts, analyst usually write ad hoc queries using the SIEM or EDR tool's interface to generate context around alerts or to correlate them with previous alerts. Such

[2] Techniques are referenced in ATT&CK as `Txxxx` such as Spearphishing link is T1192 and Remote Access Tools is T1219. Description of these techniques is available at https://attack.mitre.org/techniques/enterprise/

[3] A procedure is a specific instantiation of a technique; in this paper we use the term "technique" to describe both techniques and procedures.

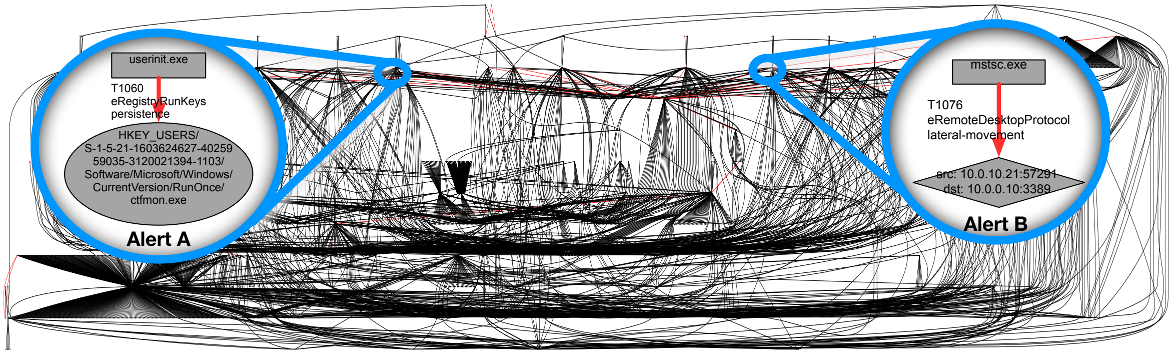


Fig. 2: Part of the APT29 attack provenance graph. We zoomed-in on two threat alerts from this attack, and excluded the network connections and registry operations from this graph for presentation purposes. In the complete graph, there are total 2,342 edges and 1,541 vertices. In this graph, and the rest of the paper, we use boxes to represent processes (count=79), diamonds to represent sockets (count=750), and oval nodes to represent files (count=54), registries (count=132), kernel objects (count=30), and modules (count=496). Edges represent casual relationships between the entity nodes, and red edges represent threat alerts (count=26).

context generation requires a lot of manual effort and time, which can delay investigation and recovery. Even after analysts have generated the context around an alert, it is difficult to understand the progression of the attack campaign by looking at system-level events. Depicting these events in a graph helps to show the causal relationships, but the volume of information is still overwhelming. Note that certain EDR tools, such as CrowdStrike Falcon [52] provide interfaces to only get the chain of process events that led to the triggered alert. These process chains do *not* capture information flow through system objects (e.g., files, registries). As a result, such EDR tools can not aggregate causally related alerts that are associated with system objects, leading to incomplete contexts.

During our exercise, APT29 generated 2,342 system events such as process launches and file creation events. Figure 2 shows a classical whole-system provenance graph for all the events related to APT29. The unwieldy tangle of nodes and edges in the figure demonstrates how daunting it can be for a cyber analyst to explore and validate a potential attack and understand the relationship between alerts.

Storage Inefficiency. EDR tools constantly produce and collect system logs on the end hosts. These system logs can quickly become enormous [31], [34]. In our evaluation dataset, the EDR recorded 400K events per machine per day from total of 34 end hosts, resulting in 35GB worth of system logs with a total of 40M system events. Note that the database used to store the events on hosts performs light compression, resulting in on-disk sizes roughly half this size. Retaining those system logs can become costly and technically challenging over longer periods. Further, for enterprises, it is important to clarify how long logs will be stored for and plan for the resulting financial and operational impact. For example, keeping log data for a week may be inexpensive, but if an attack campaign spans more than a week (which is common [3], [2], [1]), then the company will lose critical log data necessary for forensic investigation.

We surveyed the white papers and manuals of the top 5 EDR tools curated by Gartner [10]. In these white papers, we specifically looked for techniques used by these EDR tools for log retention. We found that *no* EDR tool currently describes any meaningful log retention techniques that can best utilize the limited storage for the investigation of long-lived APTs. Instead, those EDR tools use a FIFO queue that depending on the EDR vendor's retention policies buffers only a few days of system logs. For example, by default, Symantec's EDR allocates 1GB of space on each host which is sufficient for a couple of days or perhaps a week's worth of logs. The oldest logs are purged when this limit is reached. Events that are pushed to the server are also purged, with the oldest 10% of events deleted when used storage capacity reaches 85% [18].

III. System Overview

A. Threat Model

This work considers an enterprise environment comprised of thousands of machines that is the target of a sophisticated remote attacker. The attacker follows the strategy of *low* – primarily utilizing techniques that are unlikely to draw significant attention, and *slow* – often spanning weeks to months in duration. Moreover, we consider APT-style attacks that are highly *disruptive* [53], creating significant business disruption. We make the following assumptions about the environment. First, we assume that an EDR tool is collecting system logs on each end host in the enterprise. Next, we assume that APT attacks begin after the EDR has started monitoring the victim host. We assume that the underlying EDR tool is not compromised and that the system logs are correct (not tampered with by the attacker) at the time of the investigation. However, tamper-evident logging solutions [54], [55] can help alleviate log integrity assumption. Finally, we do not consider hardware trojans, side-channels, and backdoor attacks in this paper.

B. Design Goals

We set out to design a system that will bring the best of provenance-based solutions to solve the shortcomings of EDR tools. The following are the design goals of our system:

G1 Multi-stage Attack Explanations. The system should provide a compact visualization to describe different high-level stages of attack campaigns.

G2 Causal Alert Triage. The system should triage threat alerts based on their severity.

G3 Long-Term Log Retention. Our techniques for investigation and triage must be possible for even prolonged attack campaigns without sacrificing accuracy.

G4 Broadly Applicable. The techniques we develop for alert triage and log management should comply with EDR tool use cases. Our techniques should work with generic system logs collected already by most EDR tools.

G5 Minimally Invasive. The system should be able to work with any commodity host without requiring changes to the underlying OS or the EDR tool.

G6 Extensible. Our algorithms should be able to work with any adversarial TTP knowledge base as long as those TTPs are detected by the underlying EDR tool.

C. Our Approach

A high-level overview of our system, RapSheet, is shown in Figure 3. Full details will be given in the next section, but we overview the approach here. First, RapSheet performs rule matching on system logs to identify the events that match MITRE ATT&CK behaviors. In our APT29 exercise, we were able to match techniques T1060, T1086, T1085, T1055, T1082, T1078, T1076, T1040 against logs. Each rule match signifies an alert of a possible threat behavior. Next, we generate a provenance graph database from the logs. During the graph generation, we annotate the edges (events) that match the MITRE ATT&CK techniques in the previous step. Figure 2 shows the provenance graph for the APT29 engagement.

Once the construction of the provenance graph with alert annotations is done, we generate a tactical provenance graph (TPG) which is a graph derived from the provenance graph that shows how causally related alerts are sequenced. To generate a TPG, we first identify the *initial infection point* (IIP) vertex, i.e., the first vertex in the timeline that generated a threat alert. Then we find all the alerts in the progeny of the IIP vertex using forward tracing. Finally, extraneous system events are removed from this progeny graph (Goal **G1**), forming what we call the *IIP graph*. Figure 4a shows the IIP graph for the APT29 attack. After that, we perform threat score assignment.

The key idea behind our threat score assignment algorithm is to *use temporal ordering between all the causally related alerts (i.e., all the alerts in the IIP graph) to rank the alerts that conform to the MITRE ATT&CK kill chain higher than the alerts that appear in an arbitrary order*. However, ordering information for alerts on different paths is not immediately apparent in the IIP graph. To remedy this, we perform a happens-before analysis to find temporal orderings between the different alerts present in the IIP graph which gives us a TPG. Figure 4b shows the TPG for the APT29 attack scenario. After that our threat score assignment algorithm finds ordered subsequences of alerts from the TPG that conform to the MITRE kill chain and uses these to assign a severity score for alert prioritization (Goal **G2**). Note that our evaluation and implementation are based on an offline analysis similar to prior causal analysis work (e.g., [38], [31]). We discuss how to adapt our system to online settings in Section IX.

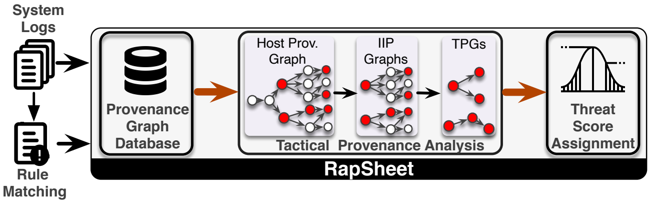


Fig. 3: Overview of RapSheet architecture (Section III-C)

IV. System Design

A. Log Collection

EDR tools collect system logs on each host in the enterprise. For Linux hosts, our underlying EDR uses Linux Audit framework [48] while for Windows it uses ETW [47] as well as custom system call hooking. This is standard for most EDR tools [56], [57]. System logs contain low-level system events including process launches and file operations. Those system events capture causal relationships between different system entities. For example, in Linux the causal relationship between a parent process creating a child process is represented by an event generated by capturing calls to `sys_clone()`. Once those system logs are collected on each host they are processed into a JSON format.

We note that we supplemented the events collected by our underlying EDR with logs of Asynchronous Local Procedure Call (ALPC) messages which we collected separately on Windows hosts. ALPC is the mechanism that Windows components use for inter-process communication (IPC) [58]. After running real-world attack scenarios on Windows machines, we realized that many of the attacks manifest in part through system activities that are initiated using ALPC messages. Missing those causal links can undermine the forensic investigation, as the provenance graph becomes disconnected without them. Note that previous papers [42], [25], [43], [22], [38] on Windows provenance do not capture ALPC messages, resulting in disconnected provenance chains.

B. Rule Matching

Generating alerts for individual MITRE techniques is a feature of most EDR tools, including the one that we use in our experiments. Because of RapSheet's novel use of TPGs for grouping, scoring, and triaging alerts, we are able to include even the most false-positive-prone MITRE techniques as alerts without overwhelming an analyst. In our experiments, we use a default set of MITRE rules that was provided by the Symantec EDR tool, and we supplemented these with additional rules for MITRE techniques that were not already covered. Users can easily extend our system by adding new rules for additional TTPs (Goal **G6**). Moreover, to ensure Goal **G4** our rule matching only relies on events that are commonly collected by EDR tools or readily available from commodity auditing frameworks.

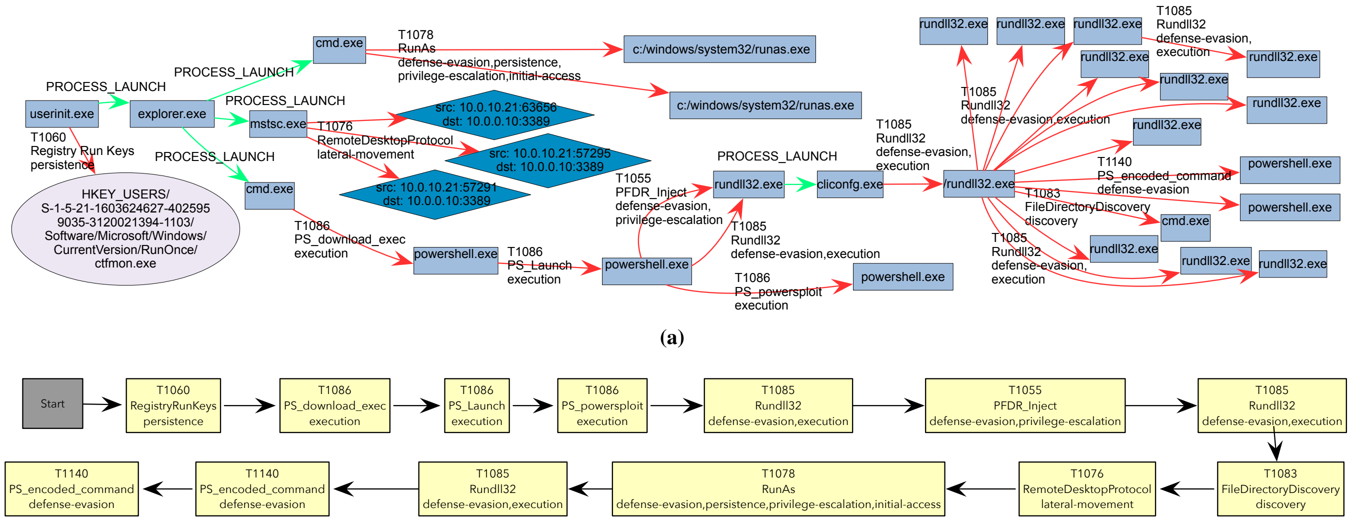


Fig. 4: APT29 attack scenario. **(a)** IIP Vertex graph generated by RapSheet. Threat alert edges are annotated with the MITRE technique ID, technique name, and tactic name. "PS" stands for PowerShell. **(b)** Tactical Provenance Graph (TPG) for APT29 attack after applying readability pass. RapSheet generated TPG is 2 orders of magnitude smaller than the classical provenance graph shown in Figure 2

As is described next, the low-level system events will form edges in a provenance graph. In RapSheet, we annotate the edges that triggered an alert with the alert information (e.g., the MITRE technique ID). Some rules provided by the EDR vendor generate alerts for behaviors not covered by the MITRE ATT&CK, which we ignore for the purposes of this work. For our example attack scenario described in Section II, the threat alert annotated as **Alert B** in Figure 2 matched the following rule (syntax simplified for clarity):

Listing 1: Example MITRE technique matching rule.

```
IF EXISTS E WHERE E.tgtType = 'network' AND
E.action = 'connect' AND E.dstPort = 3389
THEN ALERT(E.actorProc, 'T1076')
```

C. Provenance Graph Database

The system logs on each host are parsed into a graph structure called a provenance graph. The provenance graph generated by RapSheet is similar to previous work on provenance graphs [26], [21], [22], [23], [27] with some new additions to reason about MITRE ATT&CK tactics. Our provenance graph data model is shown in Figure 5. We have two types of vertices: process vertex type and object vertex type which includes files, registry, etc. The edges that connect these vertices are labeled with an event type that describes the relationship between the connected entities and the timestamp of event occurrence. Moreover, process vertices are marked with start and terminate time which allows us to check if a process is still alive during our analysis.

We also implemented a summarization technique from previous work, *causality-preserved reduction* [31], [34] in our provenance graph database. This technique merges the edges between two vertices that have the same operation and keeps only one edge with the latest timestamp. For example, most operating systems and many EDRs produce several system-level events for a single file operation. RapSheet aggregates those events into a single edge in the provenance graph. This technique has been shown to reduce the size of the provenance graph while still preserving the correctness of causal analysis.

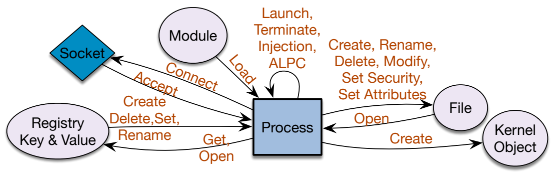


Fig. 5: Data model of our provenance graph database. Vertices represent the system entities (actors and objects) while the edges represent the causal dependency. Edges are annotated with the timestamp of event occurrence and event type.

D. Tactical Provenance Analysis

Given a list of triggered alerts and host provenance graphs, we find all the *initial infection point* (IIP) vertices in the graphs. We define an IIP to be a vertex that meets two conditions: (i) it corresponds to a process that generated an alert event e_a, and (ii) a backward trace from e_a in the provenance graph contains no other alert events. Note that there can be multiple IIP vertices in a given provenance graph. Intuitively, we are finding the earliest point that potentially suspicious behavior occurred on a given provenance chain. The IIP represents the process that exhibited this behavior. If it turns out that e_a was the first step in a multistage attack, then the remainder of the attack will be captured by future alerts generated by this process and its progeny. This gives us an effective way to group correlated alerts. For each IIP vertex, we generate a graph that is rooted at the IIP. We call this an IIP graph and define it as follows:

Def. 1. ***IIP Graph*** *Given a provenance graph $G < V, E >$ and alert event e_a incident on IIP Vertex v_a, the IIP Graph $G' < V', E' >$ is a graph rooted at V_a where $e \in E'$ iff e is causally dependent on e_a and e is either an alert event or an event that leads to an alert event.*

We generate the IIP graph by issuing a forward tracing query from the IIP vertex, producing a progeny provenance graph containing only events which happened after that first alert event incident on the IIP vertex. We then perform a pruning step on this subgraph, removing all provenance paths originating from the IIP that do not traverse an alert edge. Each path in the resulting, pruned graph contains at least one alert event. In Algorithm 1, Lines 1-16 show the IIP graph generation process. For our attack scenario example from Section II, the pruned progeny graph rooted at the IIP is shown in Figure 4a.

This IIP graph based approach is a key differentiating factor that sets RapSheet apart from the path-based approach to alert triage in NoDoze [38] and the full graph approach in Holmes [43]. A path-based approach fails to correlate alerts that are causally related but appear on different ancestry paths. For example, after initial compromise, an attacker can launch several child processes, with each child generating its own, separate path. Even though all child paths are causally related, the path-based approach will fail to correlate alerts on the separate paths. On the other hand, Holmes' full graph approach requires a normal behavior database and other heuristics to reduce false alarms from benign activities before threat score assignment. RapSheet does not require a normal behavior database, rather we rely on extracting certain subgraphs (the IIP graphs) and assigning scores based on well-known attacker behaviors, which alleviates the problem of false alarms (further discussed in Section V).

The IIP graph captures the temporal ordering between events on the same path. However, when reasoning about the overall attack campaign, we are not concerned with, e.g., which attacker-controlled process takes a given action. Instead, we want to capture the temporal order of all alerts contained in the IIP graph, which better reflects attacker intent. Because this graph may consist of multiple paths, we need a way to capture ordering between edges on different paths. To achieve this goal, we transform the IIP graph into a new graph in which each vertex is an alert event and edges indicate the temporal ordering between alerts based on a happens-before relationship [59]. We call these edges sequence edges, and they are defined as follows:

Def. 2. ***Sequence Edge.*** *A sequence edge (e_a, e_b) exists between two alerts e_a and e_b iff any of the following hold:*
(a) e_a and e_b are alerts on the same host and on the same provenance path and e_a causally preceded e_b; or
(b) e_a and e_b are alerts on the same host and the vertex timestamp of e_a is less than the vertex timestamp of e_b or
(c) e_a had an outgoing `Connect` event edge on one host, while e_b has the corresponding `Accept` edge on the receiving host.

In other words, for events that happen on the same machine, we can use the event timestamps to generate sequence edges. For events on different machines, we can use communication between the machines to generate the happens-before relationship (events before a packet was sent on one machine definitely happened before events that happened after the packet was received on the other machine). In the end, we generate a graph (Algorithm 1 Lines 17-30) which we call a tactical provenance graph whose formal definition is as follows:

Def. 3. ***Tactical Provenance Graph.*** *A tactical provenance graph TPG can be defined as a pair (V, E), where V is a set of threat alert events and E is a set of sequence edges between the vertices.*

As defined above, the TPG is already useful for analysts to visualize multi-stage APT campaigns because it shows temporally ordered and causally related stages of an attack without getting bogged down in low-level system events. However, the tactical provenance graph may not be as succinct as the analyst would like, since MITRE techniques may be matched repeatedly on similar events, such as a process writing to multiple sensitive files or a process sending network messages to multiple malicious IP addresses. This can add redundant alert event vertices in the tactical provenance graph. To declutter the TPG, we perform a post-processing step where we aggregate the alert vertices ascribing the same technique if they were triggered by the same process. Note that for events on a single host, without cross-machine links, the TPG is a single chain. An illustration of this post-processing step is given in Figure 4a. While the IIP shows `mstsc.exe` triggering three lateral movement alerts, the TPG in Figure 4b only has one lateral movement vertex.

V. Threat Score Assignment

A key goal of RapSheet is to group alerts and assign them a threat score that can be used to triage those contextualized alerts. Because some alerts are more suspicious than others, we pursued a scoring mechanism that incorporated a risk score of the individual alerts. Where available, we used information published by MITRE to assign those scores to individual alerts.

Many of the MITRE ATT&CK technique descriptions include a metadata reference to a pattern in the Common Attack Pattern Enumeration and Classification (CAPEC) [60] knowledge base. The CAPEC pattern entries sometimes include two metrics for risk assessment: "Likelihood of Attack" and "Typical Severity". Each of these is rated on a five category scale of `Very Low`, `Low`, `Medium`, `High`, `Very High`. The first metric captures how likely a particular attack pattern is to be successful, taking into account factors such as the attack prerequisites, the required attacker resources, and the effectiveness of countermeasures that are likely to be implemented. The second metric aims to capture how severe the consequences of a successful implementation of the attack would be. This information is available on MITRE's website, as well as in a repository of JSON files [61] from which we programmatically extracted the scores.

Algorithm 1 Tactical Provenance Analysis

```
Inputs:
    Raw provenance graph $G(V, E)$; Alert Events $AE$
Output:
    List of Tactical Provenance Graphs $List_{TPG}$

$AE' \leftarrow \{ae : time(ae)\}, ae \in AE$, sort by timestamp in asc. order
$Seen \leftarrow \emptyset$, set of seen alert events
$List_{IIP} \leftarrow \emptyset$, List of IIP Vertex Graphs
for all $ae : AE', ae \notin Seen$ do
    $Seen \leftarrow Seen \cup \{ae\}$
    // return all forward tracing paths from input event using DFS
    $Paths \leftarrow \text{ForwardPaths}(ae)$
    $IIPG \leftarrow \emptyset$ , IIP graph
    for all $path : Paths$ do
        // return all alert events in the input provenance path
        $alerts \leftarrow \text{GetAlertEvents}(path)$
        // keep only those paths in IIP graph with at least one alert
        if $alerts \neq \emptyset$ then
            $IIPG \leftarrow IIPG \cup path$
            $Seen \leftarrow Seen \cup alerts$
    $List_{IIP} \leftarrow List_{IIP} \cup IIPG$
$List_{TPG} \leftarrow \emptyset$, List of TPGs to return
for all $IIPG : List_{IIP}$ do
    $TPG \leftarrow \emptyset$ , tactical provenance graph
    $alerts \leftarrow \text{GetAlertEvents}(IIPG)$
    // sort alerts according to Happens Before rules
    $alerts_{hb} \leftarrow \{a : time(a)\}, a \in alerts$
    // Loop over sorted alerts, two at a time
    for all $ae_1, ae_2 : alerts_{hb}$ do
        $V \leftarrow ae_1$
        $V' \leftarrow ae_2$
        $TPG \leftarrow TPG \cup (V, V')$ // add sequence edge
    // Post process the TPG for readability
    $TPG \leftarrow \text{ReadabilityPass}(TPG)$
    $List_{TPG} \leftarrow List_{TPG} \cup TPG$
```

For some MITRE techniques, no CAPEC reference is provided, or the provided CAPEC reference has no likelihood and severity scores. In these cases, we fall back on a separate severity score that was provided by the EDR vendor, normalized to our fifteen point scale. We converted the descriptive values for each metric into a numeric scale of one to five, and combined the two metrics together. We give the severity score a higher weight than the likelihood score since we are defending against advanced adversaries that have many resources at their disposal to effectively execute techniques that might be considered unlikely due to their difficulty or cost. The resulting threat score for each individual alert is:

$$TS(\text{technique}) = (2 * SeverityScore) + LikelihoodScore \quad (1)$$

For example, the MITRE technique called *Registry Run Keys / Startup Folder* (T1060) [62] refers to the attack pattern called *Modification of Registry Run Keys* (CAPEC-270) [63] which assigns a likelihood of attack of "medium" and a severity of "medium". Thus, we assign an alert that detects technique T1060 a score of nine out of a possible fifteen ($TS(\text{T1060}) = 2 * 3 + 3 = 9$).

Next, we explain different schemes that we used to combine individual alert scores into an overall threat score.

A. Limitations of Path-Based Scoring Schemes

To aggregate scores, we first tried an approach based on grouping and scoring alerts using a single, non-branching provenance path as was proposed by Hassan et al. in [38]. For each alert, we generated the backward tracing path and then aggregated the scores that occurred on that path. We tried different aggregation schemes such as adding the individual alert scores or multiplying them, with and without technique or tactic deduplication. Unfortunately, we realized during our experiments that the path-based approach was not capturing the entire context of the attacks in some situations. This led us to explore another approach to grouping and scoring alerts.

B. Graph-Based Scoring Schemes

To capture the broader context of a candidate alert, we generate the TPG for the candidate alert which is derived from the subgraph rooted at the shallowest alert in the candidate's backward tracing provenance path as described in Section IV.

The key insight behind our proposed scheme is that we would like to maximize the threat score for TPGs where the alerts are consistent with an attacker proceeding through the ordered phases of the tactical kill chain defined by MITRE. We formalize this intuition in a scoring algorithm as follows. The sequence edges in the TPG form a temporally ordered sequence of the graph's constituent alerts. We find the longest (not necessarily consecutive) subsequence of these ordered alerts that is consistent with the phase order of MITRE's tactical kill chain. We then multiply the scores of the individual alerts in this subsequence to give an overall score to the TPG. If there are multiple longest subsequences, we choose the one that yields the highest overall score. More formally:

$$TS(\text{TPG}) = \max_{\mathbf{T^i} \in \mathfrak{T}} \prod_{T^i_j \in \mathbf{T^i}} TS(T^i_j) \quad (2)$$

In Equation 2, $\mathfrak{T}$ is the set of all longest subsequences in TPG consistent with both temporal and kill-chain phase ordering. Note that an attacker cannot evade detection by introducing out-of-order actions from earlier, already completed stages of the attack. RapSheet's scoring approach will simply ignore these actions as noise when finding the longest subsequence of alerts from the TPG, which need not be consecutive.

VI. Graph Reduction

System logs enable two key capabilities of EDR tools: 1) threat alert triage based on alert correlation and 2) after-the-fact attack investigation using attack campaign visualization. Thus, EDR tools need to retain these logs long enough to provide these capabilities. However, system logs can become enormous quickly in large enterprises, making long-term retention practically prohibitive. As mentioned in Section II, most EDR tools store logs in a limited FIFO buffer, destroying old logs to make space for new logs. Unfortunately, this naive log retention strategy can lose critical information from older logs. So, it is important to use this limited memory efficiently.

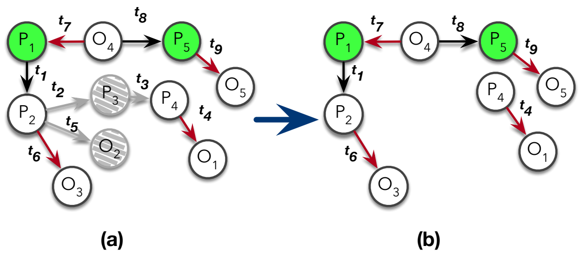


Fig. 6: Graph reduction example. After every configurable time interval, RapSheet runs graph reduction and store only skeleton graph which preserves the linkability between current and future tactics.

We propose a novel technique to reduce the fidelity of logs while still providing the two key EDR capabilities. To provide these key capabilities, we need to ensure that we can generate the TPG from the pruned graph. Once we have the TPG, we can derive correlations between alerts, assign threat scores to correlated alerts and provide high-level visual summaries of attacks to the cyber analyst.

For our graph reduction algorithm, we assume the properties of the provenance graph and backward tracing graph described in Section IV-C. We also assume all the alert events in the provenance graph are incident to at least one process vertex. Based on these properties, we propose the following two rules to prune the provenance graph at any point in time while preserving TPG-based alert correlation.

Rule#1: Remove object vertex O iff there are no alert events in the backward tracing graph of O and there are no alert event edges directly connected to O.

This rule ensures that O is not currently part of any IIP graph derived from the current provenance graph. If it were, then it either would be directly involved in an alert (i.e., there would be an alert edge incident to O), or it would be on a path from some IIP vertex to some alert edge, which entails that the alert incident to that IIP vertex would be in O's backward tracing graph. Note that even if there is a live process vertex in the ancestry of object O, and that process generates an alert event E_1 in the future, this new alert event will have a timestamp later than the edges currently leading to O. Hence, O would not be part of the IIP graph containing E_1.

To explain our graph reduction algorithm we use an example provenance graph shown in Figure 6(**a**). Vertices labeled with a P represent processes while those with an O represent object vertices. The red edges indicate alerts, green vertices show live processes at the time of reduction, and edges are marked with ordered timestamps t_1 to t_9. Gray vertices and edges show candidates for removal according to Rule#1 and Rule#2.

The only candidate for object vertex reduction is O_2 since it satisfies all the conditions of Rule#1. The backward tracing graph of O_2 consists of vertices $\{P_2, P_1\}$ and the edges with timestamps $\{t_5, t_1\}$, which do not have any alert events. Thus, we can safely remove O_2 and the edge with timestamp t_5 from the graph without losing any connectivity information for current or future alerts. Note that the edge with timestamp t_7 will not be included in the backward tracing graph because it happened after t_5. After graph reduction, if some process vertex reads or writes to the object O_2, then vertex O_2 will reappear in the provenance graph. Next, we discuss how to prune process vertices from the graph.

Rule#2: Remove process vertex P iff: i) there are no alert events in the backward tracing graph of P, ii) there are no alert event edges directly connected to P and iii) process P is terminated.

The first two conditions of Rule#2 have the same reasoning as Rule#1. In addition, we have to ensure that process P is terminated so that it does not generate new alerts which will become part of an IIP graph. In the example shown in Figure 6(**a**), process P_3 is terminated, has no alert event in its backward tracing graph, and does not have any incident edges that are alert events. Thus, we can safely remove the process vertex P_3 from the graph along with the edges that have timestamp $\{t_2, t_3\}$.

By applying these two reduction rules to a given provenance graph, RapSheet generates a space-efficient skeleton graph which can still identify all the causal dependencies between alerts and can generate exactly the same set of TPGs (procedure described in Section IV-D) as from the classical provenance graph. Figure 6(**b**) shows the skeleton graph for our example graph. We describe an efficient way to generate the skeleton graph, which does not require performing a backward trace for every vertex of a given provenance graph, in Appendix B.

Properties. A skeleton graph generated by RapSheet will not have any false positives, that is, TPGs generated from the skeleton graph will not have alert correlations that were not present in the original provenance graph. This is clear since RapSheet does not add any new edges or vertices during the reduction process. Furthermore, a skeleton graph generated by RapSheet will not have any false negatives, meaning it will capture all alert correlations that were present in the original provenance graph. This follows from the properties of provenance and our backward tracing graphs. The reduction rules ensure that, at the time of reduction, the removed nodes and edges are not part of any IIP graph. And since our backward traces include only events that happened before a given event, they would not be part of any future IIP graph.

Retention Policy. To provide log reduction and prevent storage requirements from growing indefinitely, enterprises can run the graph reduction algorithm at a configurable retention time interval. This configuration value must be long enough for alert rule matching to complete. The retention policy can be easily refined or replaced according to enterprise needs. The configured retention interval controls how long we store high-fidelity log data (i.e., the unpruned graph). RapSheet's backward tracing and forward tracing works seamlessly over the combined current high-fidelity graph and the skeleton graph that remains from prior pruning intervals.

VII. Evaluation

In this section, we focus on evaluating the efficacy of RapSheet as a threat investigation system in an enterprise setting. In particular, we investigated the following research questions (RQs):

RQ1 How effective is RapSheet as an alert triage system?

RQ2 How fast can RapSheet generate TPGs and assign threat scores to TPGs?

RQ3 How much log reduction is possible when using skeleton graphs?

RQ4 How well does RapSheet perform against realistic attack campaigns?

A. *Implementation*

We used Apache Tinkerpop [64] graph computing framework for our provenance graph database. Tinkerpop is an in-memory transactional graph database and provides robust graph traversal capabilities. We implemented the three RapSheet components (tactical graph generation, threat score assignment, and graph reduction) in 6K lines of Java code. We use a single thread for all our analyses. We generate our provenance graphs in GraphViz (dot) format which can be easily visualized in any browser. Our implementation interfaces with Symantec EDR. Symantec EDR is capable of collecting system logs, matching events against attack behaviors, and generating threat alerts.

B. *Experiment Setup & Dataset*

We collected system logs and threat alerts from 34 hosts running within Symantec. The logs and alerts were generated by Symantec EDR which was configured with 67 alert generating rules that encode techniques from the MITRE ATT&CK knowledge-base. In our experiments, we turned off other EDR rules that did not relate to MITRE ATT&CK. During all experiments, RapSheet was run on a server with an 8-core AMD EPYC 7571 processor and 64 GB memory running Ubuntu 18.04.2 LTS.

Our data was collected over the period of one week from hosts that were regularly used by members of a product development team. Tasks performed on those hosts included web browsing, software coding and compilation, quality assurance testing, and other routine business tasks. Due to variations in usage, some machines were used for only one day while others logged events every day during data collection week. In total, 35GB worth of (lightly compressed) logs with around 40M system events were collected. On average, each host produced 400K events per machine per day. We describe further characteristics of our dataset in Appendix A.

During the experimental period, we injected attack behaviors into three different hosts. The attack behaviors correspond to three different attack campaigns, two based on real-world APT threat groups (APT3 and APT29) and one custom-built data theft attack. These simulated attacks were crafted by an expert security red-team. The underlying EDR generated 58,096 alerts during the experiment period. We manually examined the alerts from the machines which were targeted by the simulated attacks to determine that 1,104 alerts were related to simulated attacker activity. The remaining alerts were not associated with any of the simulated attacks and we consider them to be false positives.

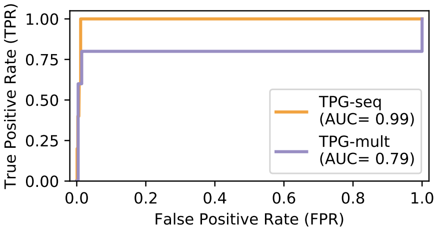


Fig. 7: ROC curve for our experiments. We tried two different schemes to rank TPGs. TPG-Seq means sequence-based scoring while TPG-mult means strawman approach of score multiplication.

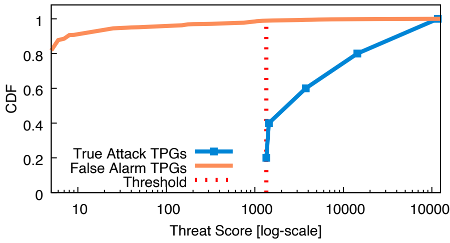


Fig. 8: CDF of threat scores for false alarm and true attack TPGs.

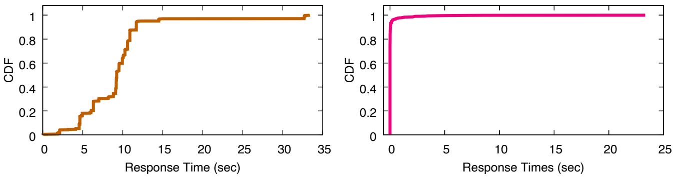
C. *Effectiveness*

The first research question of our evaluation is how effective RapSheet is as an alert triage tool. In our experiment, we used the EDR tool to monitor hosts for MITRE ATT&CK behaviors and generate alerts. We then manually labeled these alerts as true positives and false positives based on whether the log events that generated the alert were related to simulated attacker activity. This labeled set is used as the ground truth in our evaluation. Then, we used RapSheet to automatically correlate these alerts, generate TPGs, and assign threat scores to TPGs.

Of the 1,104 true alerts and 56,992 false alarms generated during our experiments, RapSheet correlated these alerts into 681 TPGs. Of these, 5 were comprised of true alerts and 676 contained only false alarms.[4] We then calculated threat scores for these TPGs and sorted them according to their score. We tried two different scoring schemes. For the first scheme, we assigned scores to each TPG using a strawman approach of multiplying the threat scores of all alerts present in the TPG. However, since TPGs may contain duplicate alerts, we normalize the score by combining alerts which have the same MITRE technique, process, and object vertex. For the second scheme, we used the scoring methodology described in Section V.

Different true positive rates (TPRs) and false positive rates (FPRs) for the scoring schemes above are shown in the ROC graph in Figure 7. Our sequence-based scoring scheme was

[4] Three out of five truly malicious TPGs were related to the APT29 simulation, which the red team performed three times during the week with slight variations. The other two attack campaigns resulted in one TPG each.



(a) Response times to generate provenance graphs for all alerts.

(b) Response times to generate TPGs with their threat scores.

Fig. 9: CDF of response times to run RapSheet analysis.

more effective than the other scheme. Figure 8 shows the cumulative distribution function for ranked true attack and false alarm TPGs based on threat scores. When we set a threshold (shown with a vertical red line) that captures 100% of true positives, we can remove 97.8% of false TPGs since all true attack TPGs are scored significantly higher than most false alert TPGs. At this threshold, RapSheet has a 2.2% FPR. Note that the goal of RapSheet is not to eliminate false TPGs from consideration, but to prioritize TPG investigation based on their threat score. The threshold is a configurable parameter and can be set more conservatively or aggressively based on the goals of a particular enterprise security team. A ranked list of the TPGs with the highest threat scores in our evaluation is presented in Appendix C.

D. Response Times

To answer RQ2, we measured the TPG generation query response (turn-around) time for all the alerts in our evaluation dataset. We divided the response time of TPG generation queries into two parts. First, we measured how long RapSheet takes to generate the provenance graph for each alert in our 58,096 alerts dataset. These provenance graphs are generated by performing backward and forward tracing queries for each alert, which reads the provenance graph database from disk. Figure 9a shows the cumulative distribution function (CDF) of response times for all the alerts. The results show that for 80% of alerts, RapSheet generates the provenance graph in less than 10 secs. Note that most of this time was spent in disk reads, which we can likely speed up using existing main-memory graph databases [65], [66].

Second, we measured the response time for performing tactical provenance analysis, which includes first extracting the IIP graph from the provenance graph of each alert, transforming this IIP vertex graph into a TPG, and finally assigning threat score to the TPG. For this response time, we assume that the provenance graph of the alert (from Figure 9a) is already in the main memory. Figure 9b shows that RapSheet was able to perform tactical provenance analysis and calculate threat scores on 95% of all the alerts in less than 1 ms.

E. Graph Reduction

To answer RQ3, we measured the graph size reduction from applying the technique discussed in Section VI. Figure 10 shows the percentage reduction in the number of edges for the 34 hosts in our evaluation, one bar for each host. On average, RapSheet reduces the graph size by 63%, increasing log buffer capacities by 2.7 times. Note that we saw a similar reduction in the number of vertices. In other words, the same end host can store 2.7 times more data without affecting storage capacity provided by EDR and data processing efficiency. This shows that skeleton graphs can effectively reduce log overhead.

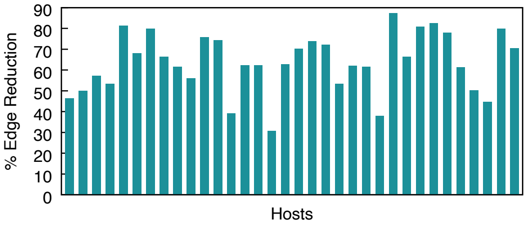


Fig. 10: Percentage of edges removed from each host's provenance graph after applying our graph reduction algorithm.

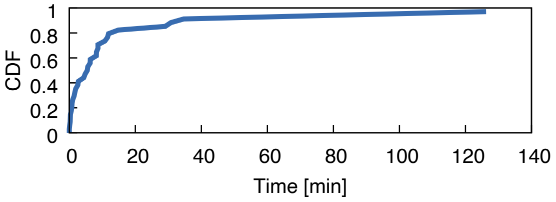


Fig. 11: CDF of running graph reduction algorithm on each of the hosts' provenance graph.

Since currently RapSheet does not support cross-machine provenance tracking, our graph reduction algorithm is limited to ensure the correctness of causality analysis. Recall that our reduction algorithm does not remove a provenance path if it leads to some alert. So in our implementation we conservatively assume all the network connections made to hosts within our enterprise can lead to an alert and thus do not remove such network connections during the reduction process (Line 21 in Algorithm 16). We expect to see a further reduction in graph size once we incorporate cross-machine provenance analysis using the methodology described in Section IX and remove our assumption.

We also measured the cost of running our graph reduction algorithm on the full provenance graphs for the full duration of our data collection for each machine. The results are shown in Figure 11. As we can see, graph reduction finished in under 15 minutes on 80% of the hosts. In the worst case, one host took around two hours to finish. Upon further investigation, we found that this host has the highest number of edges in our dataset with 1.5M edges while the average is 370K edges. This overhead, which can be scheduled at times when machines are not busy, is acceptable for enterprises since the benefit of extra storage space from pruning graph (Section II) while maintaining alert scoring and correlation outweighs the cost of running the graph reduction algorithm.

F. APT Attack Campaign Case Studies

For our evaluation, we analyzed APT attacks from two well-known threat groups (APT3 and APT29) and one custom-designed attack executed using the MITRE CALDERA frame-

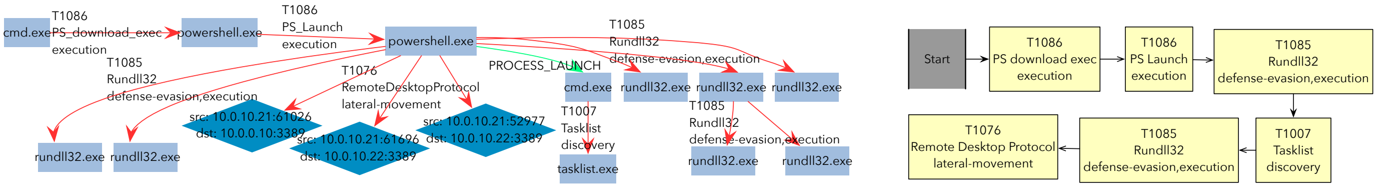


Fig. 12: APT3 Attack Scenario. **(a)** IIP Vertex graph generated by RapSheet. **(b)** Tactical Provenance Graph for APT3 attack after applying readability post-processing pass. TPG is three orders of magnitude smaller than classical provenance graph. RapSheet will choose the maximum ordered tactic sequence from this TPG for the final threat score assignment.

work [67]. We already presented the APT29 attack scenario as a motivating example in Section II. Details of the attack using CALDERA, as well as further statistics about the provenance graphs and TPGs for all three attacks are included in Appendix D. We now describe the APT3 attack scenario.

APT3 is a China-based threat group that researchers have attributed to China's Ministry of State Security. This group is responsible for the campaigns known as Operation Clandestine Fox, Operation Clandestine Wolf, and Operation Double Tap [68]. Similar to APT29, APT3 has been well studied. ATP3's goals have been modeled using MITRE tactics and techniques. In our attack scenario, we performed various techniques from this known set ranging from System Service Discovery (T1007) to Remote Desktop Protocol (T1076). These techniques allowed us to achieve several of the MITRE tactics including execution, lateral movement, and defense evasion on the victim host. Figure 12a shows the IIP graph for the APT3 attack scenario, while Figure 12b shows the TPG extracted from this IIP graph. Our threat scoring algorithm ranked this TPG at number 15 out of 681, higher than the vast majority of the 676 false TPGs. To score this TPG, RapSheet found the following temporally ordered sequence of tactics: execution, defense-evasion, discovery, and lateral-movement.

VIII. Related Work

This work joins a growing body of literature seeking to bridge the gap between causal analysis and threat detection. Holmes [43] is the first system to demonstrate that event-matching techniques can be applied to data provenance, and also includes a method for threat score assignment. However, several factors may complicate the deployment of Holmes on top of commercial EDR tools. First, Holmes assumes 100% log retention in perpetuity to assign threat scores and identify alert correlations. In practice, EDR tools have limited log buffers making such an approach practically prohibitive, a limitation addressed in RapSheet through the introduction of skeleton graphs. Second, Holmes assumes a normal behavior database to reduce false alarms from benign activities, creating a risk of adversarial poisoning of normal behavior due to concept drift as benign usage changes; in contrast, RapSheet makes no such assumption instead mitigates false alarms through the construction of IIP graphs and sequence-based threat scoring scheme. Finally, Holmes is evaluated based on 16 author-created TTP matching rules, whereas RapSheet makes use of 67 TTP rules written in an actual EDR tool. We believe this distinction is significant – 16 rules is insufficient to encode all tactics in the MITRE ATT&CK knowledge base, which means that Holmes would encounter *more* false negatives and *less* false positives than an EDR tool. As a result, while Holmes demonstrates the feasibility of EDR-like approaches on provenance graphs, the original study cannot be easily compared to EDR tools, which are optimized for recall.

NoDoze [38] is an anomaly-based alert triage that uses historical information to assign threat scores to alerts. Like Holmes, NoDoze assumes the availability of an accurate normal behavior database. Unlike RapSheet, NoDoze uses a path-based threat scoring scheme; as we described in Section V, this approach can miss attack-related events lie on different graph paths. Further, both Holmes and NoDoze consider only UNIX-like system call events when constructing provenance graphs. As a result they do not track ALPC messages (extensively used in Windows environment) which in practice would create disconnected provenance graphs and admit more error into causal analysis.

An important component of RapSheet is the log reduction algorithm, which is a topic that is well-studied in recent literature [30], [29], [34], [37], [36]. In the early stages of this study, we realized that existing log reduction techniques were inapplicable to our design because they did not preserve the necessary connectivity between EDR generated alerts. For example, LogGC [30] removes *unreachable* events, and thus would not be able to correlate alerts that were related through garbage-collected paths. Similarly, Hossain et al.'s dependence-preserving data compaction technique [37] does not consider that some edges are alert events and must, therefore, be preserved. Alternately, Winnower [29] and Process-centric Causality Approximation [34] both reduce log size by over-approximating causal relations, introducing new sources of false alerts. Other techniques, while similarly motivated, are orthogonal to the present study.

In the absence of provenance-based causality, alert correlation is another technique to assist analysts by correlating similar alerts. Existing systems use statistical-, heuristic-, and probabilistic-based alert correlation [69], [70], [71], [72], [73] to correlate alerts. Similar approaches are used in industry for building SIEMs [74], [75]. These techniques are based on feature correlations that do not establish causality. In contrast, RapSheet can establish actual system-layer dependencies between events. BotHunter [73] searches for a specific pattern of events in IDS logs to detect successful infections caused by botnets. This approach relies on network-level communication to identify the stages of a botnet infection. RapSheet, on the other hand, uses host-level provenance graphs to chain together different APT attack stages.

Elsewhere in the literature, several provenance-based tools have been proposed for network debugging and troubleshooting [76], [77], [78], [79], [80]. Chen et al. [78] introduced the concept of differential provenance to perform precise root-cause analysis by reasoning about differences between provenance trees. Zeno [77] proposed temporal provenance to diagnose timing-related faults in networked systems. Using sequencing edges Zeno was able to explain why the event occurred at a particular time. RapSheet also uses the sequencing edges but to reason about dependencies between different attack tactics. Zhou et al. [55] designed SNooPy a provenance-based forensic system for distributed systems that can work under adversarial settings. RapSheet can use tamper-evident logging from SNooPy to defend against anti-forensic techniques. DTaP [81] introduced a distribute time-aware provenance system. RapSheet can leverage DTaP's efficient distributed storage and query system to improve its query response times.

IX. Discussion & Limitations

Cross-Machine Analysis. In our experiments and implementation, we exclusively considered each host in isolation, i.e., cross-machine provenance was not analyzed. That said, our method of extracting TPGs retains sufficient information to connect provenance graphs across machines through network vertices in the same way as has been observed by previous papers [31], [82]. Afterward, our score assignment algorithm would work the same as in the single-machine scenario.

Online Analysis. Our implementation and experiments are based on offline analysis. As the offline implementation is able to process alerts in roughly 10 seconds, it is already possible for RapSheet to provide real-time intelligence to analysts. Adapting RapSheet to an online setting poses new challenges, but such online solution is attainable. In an online setting, RapSheet would need to be extended with a data structure that tracks the threat score of the current TPG and can check if new events need to be added to the TPG. Further, threat scoring (Eq. 2) is monotonic, which means that it permits incremental updates to the score without having to fully recalculate as the TPG updates. We leave such extensions to future work.

Adaptive Attacks. When considering APT detection, it is essential that the problem of adaptive attack behaviors be considered. As RapSheet analyzes alerts based on the MITRE ATT&CK kill-chain, an adaptive strategy would be for an attacker to employ tactics in an order that violates the expected sequence in an attempt to lower their behaviors' threat score. While it may be feasible to somewhat reduce a threat score through careful attack sequencing, it is not straightforward since in many cases one MITRE tactic cannot be performed before another tactic has been completed. For example, in order to perform the "Credential Access" tactic, the attacker must first successfully perform "Privilege Escalation" to have the permissions necessary to open credential files. As another example, the "Discovery" tactic, which identifies other hosts in the victim environment, is a necessary prerequisite to "Lateral Movement". An even more sophisticated scoring algorithm could encode the partial order defined by strict dependencies between certain MITRE phases in order to reduce the effectiveness of this already difficult evasion technique. Note that an attacker is certainly able to inject out-of-order tactics that act as noise between the necessarily sequenced stages of their attack. But this strategy would not reduce the final threat score assigned by RapSheet, since we extract the longest, not-necessarily-consecutive subsequence of tactics from the IIP graph that is consistent with the MITRE kill-chain ordering. The injected noise will simply be ignored.

Limitations of APT Exercises. For obvious reasons, our experiments are based on simulated APT behaviors, not actual APT campaigns. Those simulations were written by expert analysts at Symantec through analysis of APT malware samples. One limitation of these simulations is that the threat actors did not add innocuous events in between different stages of the APT attacks, which is less realistic. That said, such activity would not affect the threat scores assigned by RapSheet in any way – the alerts associated with the malicious activities would still appear in the same order in the TPG.

Missing Alerts. RapSheet's log reduction algorithm assumes that all the threat alerts are detected by the underlying EDR tool. As we have seen in Section II, it is not unrealistic to assume that most of the attack's constituent events will generate alerts since EDR tools are designed to optimize recall, and hence generate alerts even when they detect low severity, potentially suspicious activity. However, if an alert was not caught by the underlying EDR tool, then our log reduction may remove edges and vertices from the provenance graph and break the linkability between existing and future alerts. In other words, if some attack behavior does not cause the underlying EDR to generate an alert, our log reduction algorithm *cannot* necessarily preserve the ability to generate accurate TPGs from the skeleton graph for future alerts.

X. Conclusion

In this work, we propose a viable solution for incorporating data provenance into commercial EDR tools. We use the notion of tactical provenance to reason about causally related threat alerts, and then encode those related alerts into a tactical provenance graph (TPG). We leverage the TPG for risk assessment of the EDR-generated threat alerts and for system log reduction. We incorporated our prototype system, RapSheet, into the Symantec EDR tool. Our evaluation results over an enterprise dataset show that RapSheet improves the threat detection accuracy of the Symantec EDR. Moreover, our log reduction technique dramatically reduces the overhead associated with long-term system log storage while preserving causal links between existing and future alerts.

Acknowledgment

We thank our shepherd, Guofei Gu, and the anonymous reviewers for their comments and suggestions. We also thank Akul Goyal, Riccardo Paccagnella, and Ben Ujcich for feedback on early drafts of this paper, as well as all members of the NortonLifeLock Research Group. Wajih Ul Hassan was partially supported by the Symantec Graduate Fellowship. This work was supported in part by the NSF under contracts CNS-16-57534 and CNS-17-50024. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[1] "Target Missed Warnings in Epic Hack of Credit Card Data," https://bloom.bg/2KjElxM, 2019.

[2] "Equifax Says Cyberattack May Have Affected 143 Million in the U.S." https://www.nytimes.com/2017/09/07/business/equifax-cyberattack.html, 2017.

[3] "Inside the Cyberattack That Shocked the US Government," https://www.wired.com/2016/10/inside-cyberattack-shocked-us-government/, 2016.

[4] "Whats in a name? TTPs in Info Sec," https://posts.specterops.io/whats-in-a-name-ttps-in-info-sec-14f24480ddcc, 2019.

[5] "The Critical Role of Endpoint Detection and Response," https://bit.ly/39NrNwo, 2019.

[6] "MITRE ATT&CK," https://attack.mitre.org, 2019.

[7] "Why MITRE ATT&CK Matters," https://symantec-blogs.broadcom.com/blogs/expert-perspectives/why-mitre-attck-matters.

[8] "Experts advocate for ATT&CK," https://www.cyberscoop.com/mitre-attck-framework-experts-advocate/.

[9] "ATT&CK Evaluations," https://attackevals.mitre.org/.

[10] "Endpoint Detection and Response Solutions Market," https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions, 2019.

[11] "File Deletion," https://attack.mitre.org/techniques/T1107/, 2019.

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

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

[14] G. P. Spathoulas and S. K. Katsikas, "Using a fuzzy inference system to reduce false positives in intrusion detection," in *International Conference on Systems, Signals and Image Processing*, 2009.

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

[16] "An ESG Research Insights Report," http://pages.siemplify.co/rs/182-SXA-457/images/ESG-Research-Report.pdf.

[17] "Splunk," https://www.splunk.com.

[18] "About purging reports," https://support.symantec.com/us/en/article.howto129116.html, 2019.

[19] "Evaluating Endpoint Products," https://redcanary.com/blog/evaluating-endpoint-products-in-a-crowded-confusing-market/, 2018.

[20] A. Bates, W. U. Hassan, K. Butler, A. Dobra, B. Reaves, P. Cable, T. Moyer, and N. Schear, "Transparent web service auditing via network provenance functions," in *WWW*, 2017.

[21] A. Bates, D. Tian, K. R. B. Butler, and T. Moyer, "Trustworthy whole-system provenance for the Linux kernel," in *USENIX Security*, 2015.

[22] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. D. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *USENIX Security*, 2017.

[23] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie *et al.*, "MCI: Modeling-based causality inference in audit logging for attack investigation," in *NDSS*, 2018.

[24] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition," in *NDSS*, 2013.

[25] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for Windows," in *ACSAC*. ACM, 2015.

[26] 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 *USENIX Security*, 2017.

[27] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis," in *NDSS*, 2020.

[28] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *CCS*, 2019.

[29] W. U. Hassan, M. Lemay, N. Aguse, A. Bates, and T. Moyer, "Towards scalable cluster auditing through grammatical inference over provenance graphs," in *NDSS*, 2018.

[30] K. H. Lee, X. Zhang, and D. Xu, "LogGC: Garbage collecting audit log," in *CCS*, 2013.

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

[32] S. Ma, X. Zhang, and D. Xu, "ProTracer: Towards practical provenance tracing by alternating between logging and tainting," in *NDSS*, 2016.

[33] T. Pasquier, X. Han, T. Moyer, A. Bates, O. Hermant, D. Eyers, J. Bacon, and M. Seltzer, "Runtime analysis of whole-system provenance," in *CCS*. ACM, 2018.

[34] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *CCS*, 2016.

[35] S. Ma, J. Zhai, Y. Kwon, K. H. Lee, X. Zhang, G. Ciocarlie, A. Gehani, V. Yegneswaran, D. Xu, and S. Jha, "Kernel-supported cost-effective audit logging for causality tracking," in *USENIX ATC*, 2018.

[36] Y. Tang, D. Li, Z. Li, M. Zhang, K. Jee, X. Xiao, Z. Wu, J. Rhee, F. Xu, and Q. Li, "Nodemerge: Template based efficient data reduction for big-data causality analysis," in *CCS*. ACM, 2018.

[37] M. N. Hossain, J. Wang, R. Sekar, and S. D. Stoller, "Dependence-preserving data compaction for scalable forensic analysis," in *USENIX Security Symposium*, 2018.

[38] 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 *NDSS*, 2019.

[39] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. Gunter, and H. Chen, "You are what you do: Hunting stealthy malware via data provenance analysis," 2020.

[40] X. Han, T. Pasquier, A. Bates, J. Mickens, and M. Seltzer, "UNICORN: Runtime provenance-based detector for advanced persistent threats," in *NDSS*, 2020.

[41] A. Bates and W. U. Hassan, "Can data provenance put an end to the data breach?" *IEEE Security Privacy*, vol. 17, no. 4, pp. 88–93, July 2019.

[42] 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 *ACSAC*. ACM, 2016.

[43] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "HOLMES: Real-time APT detection through correlation of suspicious information flows," in *IEEE S&P*, 2019.

[44] "Threat-based Defense," https://www.mitre.org/capabilities/cybersecurity/threat-based-defense, 2019.

[45] E. M. Hutchins, M. J. Cloppert, and R. M. Amin, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains," *Leading Issues in Information Warfare & Security Research*, vol. 1, no. 1, p. 80, 2011.

[46] S. T. King and P. M. Chen, "Backtracking intrusions," in *SOSP*. ACM, 2003.

[47] "Windows Event Tracing," https://docs.microsoft.com/en-us/windows/desktop/ETW/event-tracing-portal.

[48] "The Linux audit daemon," https://linux.die.net/man/8/auditd.

[49] "MITRE Matrix," https://attack.mitre.org/matrices/enterprise/.

[50] "APT 29 - Put up your Dukes," https://www.anomali.com/blog/apt-29-put-up-your-dukes, 2019.

[51] "APT29," https://attack.mitre.org/groups/G0016/, 2019.

[52] "CrowdStrike," https://www.crowdstrike.com/.

[53] Airbus Cyber Security, "APT Kill Chain," https://airbus-cyber-security.com/apt-kill-chain-part-2-global-view/, 2018.

[54] R. Paccagnella, P. Datta, W. U. Hassan, C. W. Fletcher, A. Bates, A. Miller, and D. Tian, "Custos: Practical tamper-evident auditing of operating systems using trusted execution," in *NDSS*, 2020.

[55] W. Zhou, Q. Fei, A. Narayan, A. Haeberlen, B. T. Loo, and M. Sherr, "Secure Network Provenance," in *SOSP*, 2011.
[56] "Endgame - Endpoint Protection," https://www.endgame.com/sites/default/files/architecturesolutionbrief.pdf, 2019.
[57] "Endpoint Security in Todays Threat Environment," https://ziften.com/wp-content/uploads/2016/12/UserMode_Whitepaper.pdf, 2019.
[58] "Monitoring ALPC Messages," http://blogs.microsoft.co.il/pavely/2017/02/12/monitoring-alpc-messages/, 2017.
[59] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978. [Online]. Available: http://doi.acm.org/10.1145/359545.359563
[60] "Common Attack Pattern Enumeration and Classification," https://capec.mitre.org, 2019.
[61] MITRE, "Cyber Threat Intelligence Repository," https://github.com/mitre/cti.
[62] "Registry Run Keys / Startup Folder," https://attack.mitre.org/techniques/T1060/, 2019.
[63] "CAPEC-270: Modification of Registry Run Keys," https://capec.mitre.org/data/definitions/163.html, 2019.
[64] "Apache TinkerPop," http://tinkerpop.apache.org/, 2019.
[65] "RedisGraph - a graph database module for Redis," https://oss.redislabs.com/redisgraph/, 2019.
[66] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky, "Mica: A holistic approach to fast in-memory key-value storage." USENIX, 2014.
[67] MITRE, "Technology Transfer: CALDERA," https://www.mitre.org/research/technology-transfer/open-source-software/caldera.
[68] "APT3," https://attack.mitre.org/groups/G0022/, 2019.
[69] A. Valdes and K. Skinner, "Probabilistic alert correlation," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 54–68.
[70] W. Wang and T. E. Daniels, "A graph based approach toward network forensics analysis," *TISSEC*, 2008.
[71] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2001, pp. 85–103.
[72] Y. Shen and G. Stringhini, "Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks," in *USENIX Security*, 2019.
[73] G. Gu, P. Porras, V. Yegneswaran, and M. Fong, "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *USENIX Security Symposium*, 2007.
[74] "Endpoint Monitoring & Security," https://logrhythm.com/solutions/security/endpoint-threat-detection/, 2019.
[75] "What is SIEM?" https://logz.io/blog/what-is-siem/, 2019.
[76] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN Be Your Eyes: Secure Forensics in Data Center Networks," in *SENT*, 2014.
[77] Y. Wu, A. Chen, and L. T. X. Phan, "Zeno: Diagnosing performance problems with temporal provenance," in *NSDI*, 2019.
[78] A. Chen, Y. Wu, A. Haeberlen, W. Zhou, and B. T. Loo, "The good, the bad, and the differences: Better network diagnostics with differential provenance," in *ACM SIGCOMM*, 2016.
[79] Y. Wu, M. Zhao, A. Haeberlen, W. Zhou, and B. T. Loo, "Diagnosing missing events in distributed systems with negative provenance," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 383–394, 2014.
[80] W. Zhou, M. Sherr, T. Tao, X. Li, B. T. Loo, and Y. Mao, "Efficient querying and maintenance of network provenance at internet-scale," in *ACM SIGMOD*, 2010.
[81] W. Zhou, S. Mapara, Y. Ren, Y. Li, A. Haeberlen, Z. Ives, B. T. Loo, and M. Sherr, "Distributed time-aware provenance," *Proceedings of the VLDB Endowment*, pp. 49–60, 2012.
[82] A. Gehani and D. Tariq, "SPADE: Support for provenance auditing in distributed environments," in *Middleware*, 2012.

Appendix A
Dataset Characterization

In this section, we characterize dataset that we used in our evaluation. We collected 40M system monitoring event from 34 hosts in a real-world enterprise environment. These host machines were used by employees daily for web browsing, software coding and compilation, quality assurance testing, project management, and other routine business tasks. We used 67 total alert rules to detect various MITRE ATT&CK techniques in our experiments. Of these rules, some were written by us, while the other were included by default in the Symantec EDR software.

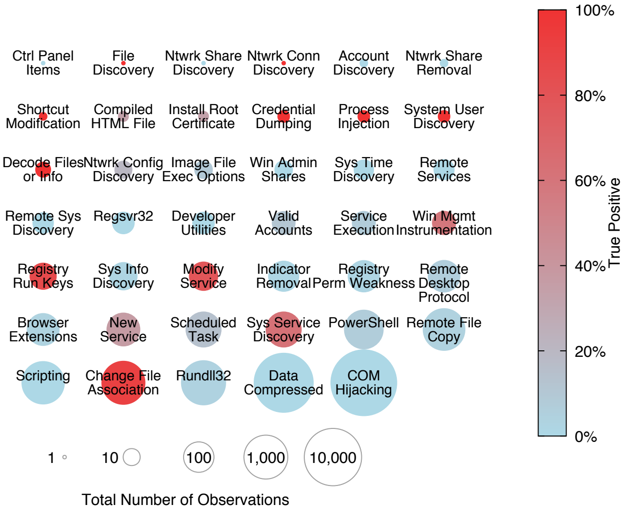


Fig. 13: Number of matched MITRE ATT&CK techniques during our evaluation with their true positive rates.

First, we look at how often the various MITRE ATT&CK technique and tactic rules caused alerts on the hosts in our experiment. Figure 13 shows which MITRE ATT&CK techniques were matched, how many times, and what proportion of the alerts for each technique were related to a true attack. We can see from the figure that rules for techniques like RunDLL32 (T1085) and Scripting (T1064) generated many alerts, but have very low true positive rates since these techniques are commonly used for benign purposes. On the other hand, techniques like "Change File Association" (T1042) and "System Service Discovery" (T1007) were triggered many times and have high true positive rate because these techniques usually only happen during malicious activity. Thus, these techniques can be strong indication of an attack campaign.

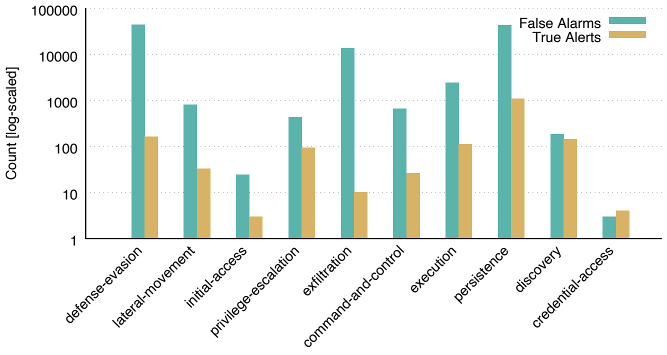


Fig. 14: Number of matched MITRE ATT&CK tactics during our evaluation.

Figure 14 shows the tactics to which the alerting techniques in our data belong. During evaluation, we observed 10 out of the 12 tactics defined by MITRE ATT&CK. As is evident from the graph, there are certain tactics, such as "Exfiltration" and "Defense Evasion", are more false-positive prone. Others, such as "Discovery", still have many false alarms, but have a more balanced distribution.

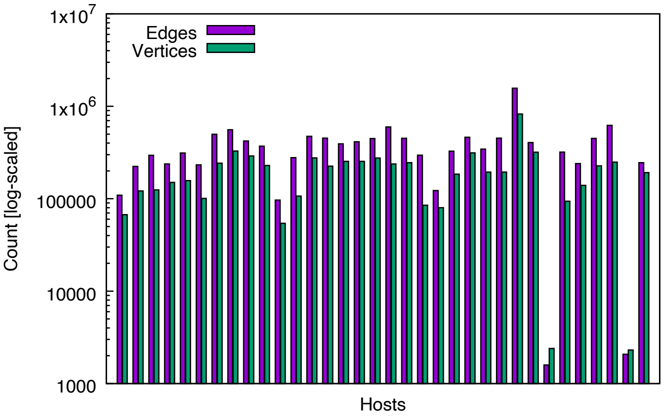


Fig. 15: Number of vertices and edges in the provenance graph for each of 34 hosts in our evaluation.

Figure 15 shows the number of vertices and edges in provenance graph database for each of 34 hosts in our evaluation. We see that all hosts have a similar number of edges and vertices except for two hosts. From these two hosts, we had only few hours worth of logs.

Appendix B
Graph Reduction Algorithm

We describe two rules for alert correlation preserving graph reduction in Section VI. Here we present our efficient graph reduction algorithm that removes edges and vertices from the graph according to those two rules.

Function ReduceGraph in Figure 16 is the main function that takes the input provenance graph G and deletes edges and vertices from G. We loop over each edge e in G and extract actor vertex v_{proc} which is always a process vertex and target vertex v_{target} which can be either a process or an object vertex. First of all we check that if edge e is an alert event. If e is an alert event then we ignore this edge. Then, we check that if v_{target} is a registry type because we do not delete registry events during our reduction as we discussed in Section VI.

Then, we check that if there are any outgoing or ingoing edges of v_{target} that are alert events (Function CheckAlertInOut in Figure 16). If there is an alert event then we continue to next edge because this v_{target} with edge e cannot be deleted to preserve alert correlations.

After that we check that if v_{target} is a process type vertex. In which case, we apply Rule#2 described in Section VI where we check that if v_{target} is terminated. If it is not terminated we do not delete v_{target} vertex and corresponding edge e and continue to next edge from G. After that we check if v_{target} is a registry type vertex then we do not remove that vertex. Then, we check that v_{target} is a network type vertex and the outgoing IP address is an internal IP address that belongs to other host within the enterprise. If that is the case then we do not remove such vertex. The reason why we do not remove such network vertices is described in Section VII.

For our underlying EDR, there are certain events that will not be part of any TPG because these events do not generate information flow between alerts. For example, our underlying EDR only tracks Module read operation for performance purposes. Since, it does not capture write operation there will be no information flow from one process to another process using module vertex. Similarly, connections made outside the enterprise network also do not generate information flow to another machine in the enterprise network. We simply remove these events during our graph generation without losing any alert connectivity information. Function CheckSafeToRemove in Figure 16 handles these cases.

After we have performed above-mentioned checks, we add current edge in a Hashmap $ProcMap$. In this hashmap, key is an actor process vertex v_{proc} and value is a priority queue of edges that are directly connected to the v_{proc}. The priority is based on descending order of timestamps present on edges. We use this hashmap to reduce the number of calls on backward tracing on target vertices of edges. The key idea is that we poll an edge e_{latest} from priority queue which will return the latest happened edge (event) from the buffer. We then call backward tracing function on target vertex v_{target} from that edge e_{latest}. If we do not find any alert in the backward trace of v_{target} then we can safely remove all the edges in the priority queue of key vertex v_{proc} since their backward trace will also not have any alerts. Function CheckAlertBackTrace and Function BackwardDFS in Figure 16 performs backward tracing on input vertex.

One precondition to performing above optimization is that we need to ensure that target vertices in the buffer have incoming edges from the same key vertex v_{proc} (Function $\mathrm{CheckAllInSame}$ in Figure 16). If this condition is not true we have to perform backward tracing on that target vertex because there can be another path besides v_{proc} which can lead to alert. After deleting the edges from graph, we delete all the vertices that now have no incoming or outgoing edges using Function $\mathrm{DeleteIsolatedVertices}$.

Appendix C
Additional Experimental Results

Table I summarizes the ranking of top 16 threat scoring TPGs out of total 681 TPGs in our evaluation. This list contains all the 5 truly malicious TPGs that are present in our evaluation. In this table, the first column represents the root vertex ID given by RapSheet. Recall that TPG is identified by the IIP root vertex. The second column shows where the TPG was a false alarm or truly malicious. The third column shows the threat score given by RapSheet to the TPG. The fourth column shows the number of threat alerts present in the corresponding TPG. Finally, the fifth column represents the longest ordered sub-sequence extracted by RapSheet from the TPG that gave the highest threat score.

```
function ReduceGraph($G < V, E >$)
    $ProcMap \leftarrow \{\}$, hashmap from vertex to timestamp-based priority queue of edges connected to vertex
    for all $e : E$ do
        $type \leftarrow \text{GetType}(e)$
        $v_{proc} \leftarrow \text{GetProcVertex}(e)$, get process vertex
        $v_{target} \leftarrow \text{GetTargetVertex}(e)$, get target vertex
        $t \leftarrow \text{GetTime}(e)$, return timestamp on edge
        // From Rule No. 1 & 2
        if $type$ is an alert event then
            $continue$
        // From Rule No. 1 & 2
        if CheckAlertInOut($v_{target}$) then
            $continue$
        // From Rule No. 2
        if $v_{target}$ is a process and $v_{target}$ not terminated then
            $continue$
        // Exception that we described in Section VI
        if $v_{target} \in registry$ then
            $continue$
        // Do not remove connections made to hosts within enterprise
        if $v_{target} \in network$ and has an internal IP address then
            $continue$
        if CheckSafeToRemove($v_{target}$, $type$) then
            $\text{DeleteEdge}(G, e)$
            $continue$
        $ProcMap.put(v_{proc}, queue.push(e))$
    for all $v_{proc}, queue \leftarrow ProcMap.entries$ do
        $f_{same} \leftarrow False$, flag to show edges coming in from same process
        $f_{flush} \leftarrow False$, flag to remove rest of queue without backtracking
        while $queue \neq \emptyset$ do
            $e_{latest} \leftarrow queue.pop()$
            $v_{target} \leftarrow \text{GetTargetVertex}(e_{latest})$
            $f_{same} \leftarrow \text{CheckAllInSame}(v_{proc}, v_{target})$
            if $f_{same}$ then
                if not $f_{flush}$ and CheckAlertBackTrace($v_{target}$, $t$) then
                    $continue$
                $f_{flush} \leftarrow True$
                $\text{DeleteEdge}(G, e_{latest})$
            else
                if CheckAlertBackTrace($v_{target}$, $t$) then
                    $continue$
                $\text{DeleteEdge}(G, e_{latest})$
        // Deletes vertices from G that have no incoming or outgoing edges
        $\text{DeleteIsolatedVertices}(G)$
        // Return skeleton graph
        return $G$

function CheckSafeToRemove($v_{target}$, $type$)
    if $v_{target} \in module$ or $kernel$ then
        return $True$
    if $v_{target} \in network$ and has an external IP address then
        return $True$
    if $type \in FileDelete$ then
        return $True$

function CheckAlertInOut($v_{target}$)
    for all $e_{inout}$ : $\text{getInOutEdges}(v_{target})$ do
        if $e_{inout}$ is an alert event then
            return $True$
    return $False$

function CheckAlertBackTrace($v$,$t$)
    $Seen \leftarrow \emptyset$, set of seen vertices during traversal
    $d \leftarrow 0$, current depth in DFS traversal
    return BackwardDFS($v$, $d$, $t$, $seen$)

function BackwardDFS($v$,$d$,$t$,$seen$)
    $flag = False$, return flag true if we see an alert during traversal
    if $d \geq maxdepth$ then
        return $flag$
    // returns edges going out of the given vertex
    $edges_{out} \leftarrow \text{getOutEdges}(v)$
    // filter edges which happened before given time
    $edges_{out} \leftarrow \text{FilterAfter}(out_edges, t)$
    for all $edge_{out} : edges_{out}$ do
        if $edge_{out}$ is an alert event then
            return $True$
    // returns edges going in the given vertex
    $edges_{in} \leftarrow \text{getInEdges}(v)$
    // filter edges which happened before given time
    $edges_{in} \leftarrow \text{FilterAfter}(in_edges, t)$
    for all $edge_{in} : edges_{in}$ do
        if $edge_{in}$ is an alert event then
            return $True$
    for all $edge_{in} : edges_{in}$ do
        $t_{in} \leftarrow \text{GetTime}(edge_{in})$
        // Returns tail of directed edge
        $v_{out} \leftarrow \text{GetOutVertex}(edge_{in})$
        if $v_{out} \notin seen$ then
            // Recursive call
            $flag \leftarrow \text{BackwardDFS}(v_{out}, d + 1, t_{in}, seen)$
            if $flag$ then
                $break$
    return $flag$
```

Fig. 16: Efficient algorithm to perform graph reduction on given provenance graph G and generate skeleton graph. We set maximum depth ($maxdepth$) of backward DFS traversal in our experiments to 6.

TABLE II: Summary of observed space overheads for our attack simulations. "#E" and "#V" mean the number of edges and vertices respectively.

	Raw Prov. Graph		IIP Graph		TPG	
Scenario	#E	#V	#E	#V	#E	#V
APT29	2,342	1,541	30	31	12	13
APT3	22,645	15,979	14	15	5	6
CALDERA	1,029	247	49	50	19	20

Note that the highest threat score was given to a benign or false alarm TPG. This is an interesting case which shows the limitation of our approach that a high threat score is given to a sequence of actions that match MITRE kill chain even if such sequence is performed for a benign or legitimate purpose. In this case, we found that a company employee compressed a bunch of sensitives files on different hosts and then transferred them to the company's external data-hosting website. Even though no attack was performed in this case, this whole course of actions by employee matched, in order, to different tactics from MITRE ATT&CK which led to a higher threat score.

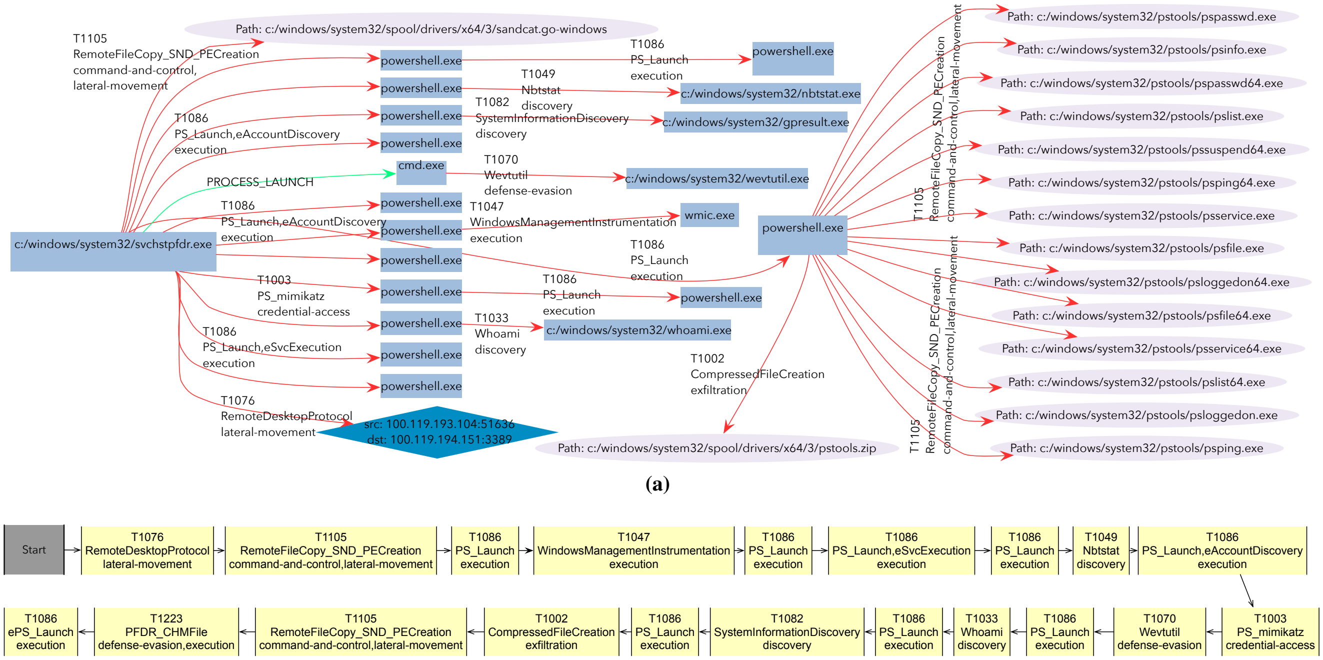
Appendix D
Additional Case Study

The size of the raw provenance graphs, IIP graphs, and TPGs in terms of total number of vertices and edges for all three APT attack cases we used in our evaluation is summarized in Table II. TPG size is shown after applying our readability pass.

In addition to the two simulations of actual advanced adversary groups that were performed by red teams, we performed our own third campaign using a configurable, automated attack emulation framework called CALDERA [67] which is maintained by MITRE. CALDERA provides a client "malware" agent and a command-and-control server that agents can communicate with to receive commands to execute on the infected machines.

TABLE I: Top 16 threat scoring TPGs out of total 681 TPGs.

TPG ID	Category	Threat Scores	Threat Alerts	Longest Ordered Subsequence of Tactics
052c89	False Alarm	125000	54	execution, persistence, privilege-escalation, defense-evasion, discovery, lateral-movement, command-and-control
e431ac	True Attack	116640	992	execution, persistence, privilege-escalation, defense-evasion, credential-access, discovery
69f88c	False Alarm	25000	30	execution, persistence, privilege-escalation, defense-evasion, discovery, lateral-movement
2e91b2	False Alarm	15625	52403	execution, persistence, privilege-escalation, defense-evasion, lateral-movement, exfiltration
c17d94	True Attack	14400	26	persistence, privilege-escalation, defense-evasion, discovery, lateral-movement
9b1f4a	False Alarm	7350	12	execution, persistence, privilege-escalation, defense-evasion, discovery
3f3fa5	False Alarm	5250	26	persistence, privilege-escalation, defense-evasion, discovery, exfiltration
08f25f	False Alarm	4375	45	execution, persistence, privilege-escalation, lateral-movement, exfiltration
ba6b01	False Alarm	3750	11	execution, persistence, privilege-escalation, defense-evasion, discovery
b464e4	True Attack	3750	77	persistence, privilege-escalation, defense-evasion, discovery, exfiltration
8d88e3	False Alarm	3125	127	execution, persistence, defense-evasion, discovery, exfiltration
d68b64	False Alarm	3125	16	initial-access, execution, persistence, defense-evasion, exfiltration
3fb85e	False Alarm	1600	44	execution, persistence, lateral-movement, exfiltration
ae5f39	False Alarm	1600	64	execution, persistence, lateral-movement, command-and-control
e448f1	True Attack	1440	13	execution, defense-evasion, discovery, lateral-movement
0c1d5e	True Attack	1350	48	execution, defense-evasion, discovery, lateral-movement


Path: c:/windows/system32/spool/drivers/x64/3/sandcat.go-windows
T1105
RemoteFileCopy_SND_PECreation
command-and-control,
lateral-movement
powershell.exe
T1086
PS_Launch
execution
T1049
Nbtstat
discovery
c:/windows/system32/nbtstat.exe
T1082
SystemInformationDiscovery
discovery
c:/windows/system32/gpresult.exe
T1086
PS_Launch,eAccountDiscovery
execution
PROCESS_LAUNCH
cmd.exe
T1070
Wevtutil
defense-evasion
c:/windows/system32/wevtutil.exe
T1047
WindowsManagementInstrumentation
execution
wmic.exe
c:/windows/system32/svchstpfdr.exe
T1003
PS_mimikatz
credential-access
T1033
Whoami
discovery
c:/windows/system32/whoami.exe
T1086
PS_Launch,eSvcExecution
execution
T1076
RemoteDesktopProtocol
lateral-movement
src: 100.119.193.104:51636
dst: 100.119.194.151:3389
T1002
CompressedFileCreation
exfiltration
Path: c:/windows/system32/spool/drivers/x64/3/pstools.zip
T1105
RemoteFileCopy_SND_PECreation
command-and-control,lateral-movement
Path: c:/windows/system32/pstools/pspasswd.exe
Path: c:/windows/system32/pstools/psinfo.exe
Path: c:/windows/system32/pstools/pspasswd64.exe
Path: c:/windows/system32/pstools/pslist.exe
Path: c:/windows/system32/pstools/pssuspend64.exe
Path: c:/windows/system32/pstools/psping64.exe
Path: c:/windows/system32/pstools/psservice.exe
Path: c:/windows/system32/pstools/psfile.exe
Path: c:/windows/system32/pstools/psloggedon64.exe
Path: c:/windows/system32/pstools/psfile64.exe
Path: c:/windows/system32/pstools/psservice64.exe
Path: c:/windows/system32/pstools/pslist64.exe
Path: c:/windows/system32/pstools/psloggedon.exe
Path: c:/windows/system32/pstools/psping.exe
(a)
Start
T1076 RemoteDesktopProtocol lateral-movement
T1105 RemoteFileCopy_SND_PECreation command-and-control,lateral-movement
T1086 PS_Launch execution
T1047 WindowsManagementInstrumentation execution
T1086 PS_Launch execution
T1086 PS_Launch,eSvcExecution execution
T1086 PS_Launch execution
T1049 Nbtstat discovery
T1086 PS_Launch,eAccountDiscovery execution
T1003 PS_mimikatz credential-access
T1070 Wevtutil defense-evasion
T1086 PS_Launch execution
T1033 Whoami discovery
T1086 PS_Launch execution
T1082 SystemInformationDiscovery discovery
T1086 PS_Launch execution
T1002 CompressedFileCreation exfiltration
T1105 RemoteFileCopy_SND_PECreation command-and-control,lateral-movement
T1223 PFDR_CHMFile defense-evasion,execution
T1086 ePS_Launch execution

(b)

Fig. 17: Caldera attack scenario. **(a)** IIP Vertex graph generated by RapSheet. We have omitted some of the edges and vertices from the graph for presentation. Complete IIP graph consists of 49 edges and 50 vertices. **(b)** Tactical Provenance Graph for Caldera attack after applying readability pass. RapSheet will choose the maximum ordered tactic sequence from this TPG for the final threat score assignment.

As a first step, we manually installed the client agent on one machine. This is a realistic scenario, since the initial infection stage is often missed by deployed defenses, either because a zero-day vulnerability is exploited or because the agent is installed by an unsuspecting, legitimate user. We then configured the command-and-control server to issue commands to discover other machines on the network, attempt to log in to those machines using stolen credential, copy the agent to any machines it successfully logged into, search for document files on all infected machines, zip up any found files and exfiltrate the files by sending the stolen file archives to the command and control server.

This covers a variety of ATT&CK techniques, from System User Discovery (T1033) to Remote File Copy (T1105), and several tactics including credential access, lateral movement, and exfiltration on the victim hosts. The IIP vertex graph and tactical provenance graph for one of the victim machines are shown in Figure 17.