ATLAS: A Sequence-based Learning Approach for Attack Investigation

Abdulellah Alsaheel[*1], Yuhong Nan[*1], Shiqing Ma[2], Le Yu[1], Gregory Walkup[1], Z. Berkay Celik[1], Xiangyu Zhang[1], and Dongyan Xu[1]

[1]Purdue University, *{aalsahee, nan1, yu759, gwalkup, zcelik, xyzhang, dxu}@purdue.edu*
[2]Rutgers University, *shiqing.ma@rutgers.edu*

Abstract

Advanced Persistent Threats (APT) involve multiple attack steps over a long period, and their investigation requires analysis of myriad logs to identify their attack steps, which are a set of activities undertaken to run an APT attack. However, on a daily basis in an enterprise, intrusion detection systems generate many threat alerts of suspicious events (attack symptoms). Cyber analysts must investigate such events to determine whether an event is a part of an attack. With many alerts to investigate, cyber analysts often end up with alert fatigue, causing them to ignore a large number of alerts and miss true attack events. In this paper, we present ATLAS, a framework that constructs an end-to-end attack story from off-the-shelf audit logs. Our key observation is that different attacks may share similar abstract attack strategies, regardless of the vulnerabilities exploited and payloads executed. ATLAS leverages a novel combination of causality analysis, natural language processing, and machine learning techniques to build a sequence-based model, which establishes key patterns of attack and non-attack behaviors from a causal graph. At inference time, given a threat alert event, an attack symptom node in a causal graph is identified. ATLAS then constructs a set of candidate sequences associated with the symptom node, uses the sequence-based model to identify nodes in a sequence that contribute to the attack, and unifies the identified attack nodes to construct an attack story. We evaluated ATLAS with ten real-world APT attacks executed in a realistic virtual environment. ATLAS recovers attack steps and construct attack stories with an average of 91.06% precision, 97.29% recall, and 93.76% F1-score. Through this effort, we provide security investigators with a new means of identifying the attack events that make up the attack story.

[*] The authors contributed equally.

1 Introduction

Forensic analysis approaches collect diverse audit logs from multiple hosts, applications, and network interfaces. The massive volumes of logs are often analyzed offline or monitored in real-time to debug system failures and identify sophisticated threats and vulnerabilities. For instance, recent works construct causal dependency graphs from audit logs [21, 27] and use query systems to locate key attack phases (e.g., a compromised process or malicious payload) [16, 31]. Several research systems aimed to extend machine learning (ML) techniques to extract features/sequences from logs to automate intrusion and failure detection [8, 36], while others built techniques to discover associations among disparate log events through event correlation [44]. Yet, while security investigators desire to identify the attack steps, which are the specific activities undertaken to conduct an attack, these approaches are largely unable to precisely locate the critical attack steps which can efficiently highlight the end-to-end attack story.

In this paper, we aim at identifying the key entities (nodes) from audit logs that help cyber analysts construct the critical steps of an APT attack. We introduce ATLAS, a framework for attack story recovery that integrates natural language processing (NLP) and deep learning techniques into data provenance analysis to identify attack and non-attack sequences. ATLAS operates in three phases: (a) it processes the system logs and builds its own optimized causal dependency graph, (b) it constructs semantically-augmented sequences–timestamped events–from the causal graph through NLP techniques, and (c) it learns a sequence-based model that represents the attack semantics, which helps to recover key attack entities describing the attack story at inference time. These phases do not impose additional overhead on a running system, and different audit logs can be easily integrated into the ATLAS log parser to construct causal graphs and obtain precise sequences and models. During attack investigation, ATLAS enables cyber analysts to identify those key attack steps through an attack symptom event (alert), based on those sequences that share semantically similar attack patterns to the ones it had previously

learned. Such knowledge helps cyber analysts substantially save time when investigating large causal graphs, and aids them in constructing the attack story from a limited number of attack symptoms.

Our approach is based on the insight that crucial steps of different attacks in a causal dependency graph may share similar patterns. The patterns, transformed into sequences through NLP techniques (i.e., lemmatization [37] and word embedding [30]) that group together various inflected forms of relations between attack and non-attack entities. Such a sequence-based representation naturally fits the training of a model, which equips the model with deeper memories with different causal relations, and in turn, improves the sequence-model accuracy in identifying attack steps from unknown audit logs. However, there are three key challenges to this approach: (a) the causal graph is often large and complex, which makes sequence construction difficult, (b) it requires a means to precisely construct the sequences to model legitimate and suspicious activities effectively, and (c) an automated approach is needed to identify the attack events from a given attack symptom. To address these issues, ATLAS uses customized graph-optimization algorithms to reduce the graph complexity, implements a novel technique to extract the sequences of attack patterns from events, and performs attack investigation through an attack symptom to recover attack events that help comprehensively build the attack story.

We implemented and deployed ATLAS to investigate real-world attacks in a controlled environment. We developed four single-host and six multi-host attacks through their detailed APT campaign reports [17, 18, 22, 28, 34, 39]. We collected around 6.7 GB of audit log data, including over 196K entities (e.g., unique files, processes, and IP addresses) and 2.5 million events spanning 24 hours for attack investigation. Our evaluation results demonstrate that ATLAS achieves an average 91.06% precision and 97.29% recall in identifying attack entities.[2] In this work, we make the following contributions:

- We introduce ATLAS, a framework for attack story recovery, which leverages natural language processing and sequence-based model learning techniques to help cyber analysts recover attack steps from audit logs.
- We present a novel sequence representation that abstracts the attack and non-attack semantic patterns through lemmatization and word embeddings. The sequences allow ATLAS to build an effective sequence-based model to identify attack events that make up the attack story.
- We validate ATLAS on ten realistic APT attacks developed through their real-world reports in a controlled environment. The results show that ATLAS identifies the key attack entries for an attack story with high accuracy and minimal overhead.

[2] ATLAS and the audit logs used in our evaluations are available at https://github.com/purseclab/ATLAS.

2 Motivation and Definitions

Motivating Example. We describe a real-world APT attack [18] that we use throughout the paper. An attacker sends a malicious Microsoft Word file (`contract.doc`) by email to a targeted user in an enterprise. The user is deceived into downloading and opening the Word file from Gmail using Firefox. The document contains a piece of malicious code that exploits a vulnerable Microsoft Word (`winword.exe`) and issues HTTPS requests to download a malicious Microsoft HTA script (`template.hta`). This script executes a malicious Visual Basic script (`maintenance.vbs`) that includes PowerShell commands installing a backdoor to exfiltrate sensitive files. Lastly, the attacker laterally moves to other hosts.

Attack Investigation. The attack investigation often begins by collecting data about the attack from the audit logs, such as system events, DNS queries, and browser events. Attack investigation tools often represent the audit logs in the form of a causal graph (or provenance graph) that serves as a forensic tool, allowing security investigators to perform root cause analysis, and better understand the nature of an attack. Most prior research (e.g., [11, 50]) recovers the attack story from the causal graph as a sub-graph, where nodes and edges in this graph have causality relations with the attack symptom(s) for starting attack investigation. Figure 1 (a) shows a causal graph of our example attack scenario generated by those tools. The red dashed arrow represents the alert event (α, a suspicious network connection) that the attack investigation is started from and the red dashed rectangular area illustrates the recovered attack subgraph.

As detailed by a number of recent works [10, 16, 31], such graphs are, however, still very large and difficult to interpret in practice even with different graph-optimization techniques applied. These works largely rely on heuristics or hard-coded rules, which are time-consuming to develop and maintain. Thus, a domain-knowledge expert is required to constantly update those rules to cover newly developed attacks. ATLAS however, only requires more attack training data to learn new attack patterns. Others proposed anomaly-based approaches [8, 9, 11, 12, 50] that learn user behavior and identify any behavior deviates from it as an anomaly. While anomaly-based approaches can identify unknown attacks, they can have many false positives as the user behavior changes through time. To address this issue, ATLAS aims to learn both attack patterns and user behavior to identify the similarities and differences between the two. Similar to ATLAS, learning-based approaches [36, 42, 43] use ML algorithms to model attack events from logs. While these approaches can effectively reduce the number of log entries, a significant amount of manual effort is still required to find a high-level view of the attack events. To address this issue, ATLAS investigation aims to identify attack key entities (nodes), which enables it to automatically identify a subset of associated attack events.

ATLAS Approach. ATLAS is motivated by the observation

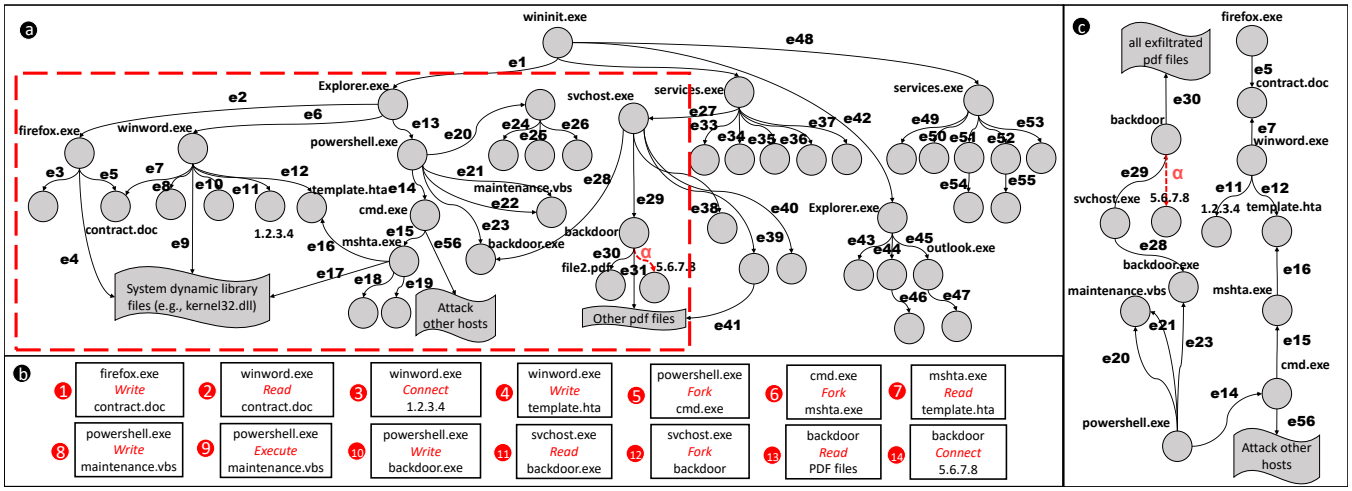

Figure 1: A real-world APT attack scenario reported by FireEye [18]. (**a**) shows a causal graph generated by prior approaches [11, 50], the red dashed area represents the attack activities reported by those approaches (some nodes and edges were omitted for brevity). (**b**) shows the attack story recovered by ATLAS as a temporal sequence of the attack steps, and (**c**) shows a concise causal graph generated by ATLAS that describes the complete attack details.

that an APT attack can be summarized as a temporal sequence of attack phases obtained from audit logs, such as the steps 1-14 illustrated in Figure 1 (b) similar to the attack steps described in natural language. These attack steps often fit in specific contexts as unique sequences representing the semantics of an attack, which can be differentiated from those normal activities in audit logs. Such a sequence-based attack representation naturally fits the training of a model to identify similar attack steps across different APT instances as they often share similar patterns regardless of their log-level details. ATLAS, given an attack symptom node (a malicious IP address that alert event α includes) at inference time, extracts a set of candidate sequences associated with symptom node, and uses a sequence-based model to identify which of those nodes in the sequences contribute to the attack. Thereafter, it uses identified attack nodes to construct the attack story, which includes events of the identified attack nodes, thus making attack investigation more concise and easier to interpret by investigators. Figure 1 (c) illustrates the attack story recovered by ATLAS for the motivating example, which includes the complete key attack steps of the example attack. This process significantly reduces manual efforts for attack investigation from large causal graphs, which excludes events that do not contribute to the attack and reduce the time needed to investigate large causal graphs.

2.1 Definitions

We formally define key terms used throughout (see Figure 2) and present the threat model.

Causal Graph. A Causal Graph `G` is a data structure extracted from audit logs and often used in provenance tracking, indicating the causality relations among subjects (e.g., processes) and objects (e.g., files or connections). The causal graph consists of nodes, which represent subjects and objects, connected with edges, which represent actions (e.g., read or connect) between subjects and objects. We consider here a directed cyclic causal graph, and its edges point from a subject to an object.

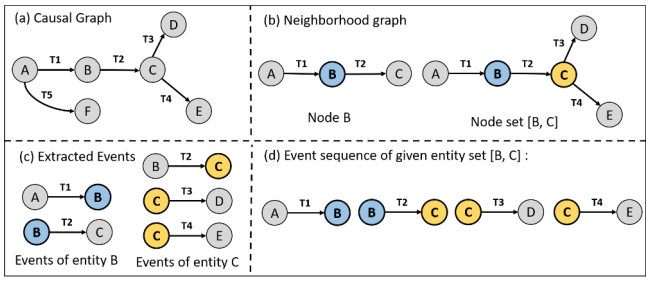

Figure 2: Illustration of causal graph, neighborhood graph, events, and sequences.

Entity. An entity `e` is a unique system subject or object extracted from the causal graph where it is represented as a node [3]. The entities we consider include processes, files, and network connections (i.e., IP addresses and domain names). For instance, `winword.exe_21` is a subject that represents a process instance of MS Word application with a process name and ID, and `192.10.0.1:80` is an object that represents an IP address with a port number.

Neighborhood Graph. Given a causal graph, two nodes `u` and `v` are said to be neighbors if they are connected by an edge. The neighborhood of a node `n` is the subgraph of `G` composed of the node `n` and edges connecting neighbor nodes with the node `n`. Similarly, given a set of nodes $\{n_1, n_2, \ldots, n_n\}$, we

[3] We use "entity" and "node" interchangeably across the paper. The term "node" is specifically used when we explain the structure of the causal graph.

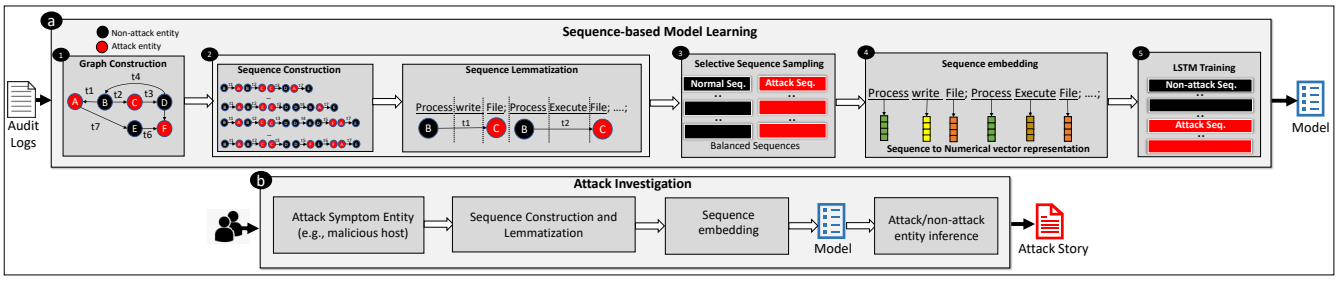

Figure 3: Overview of ATLAS architecture.

extract one unified neighborhood graph that includes all nodes and edges connecting them to their neighbors.

Event. An event ε is a quartet $(\texttt{src}, \texttt{action}, \texttt{dest}, \texttt{t})$, the source (`src`) and destination (`dest`) are two entities connected with an `action`. The `t` is the event timestamp that shows when an event occurred. Given an entity `e`, its events can be extracted from `e` neighborhood graph, which includes all actions associated with `e`'s neighbors. For example, given an entity `Firefox.exe` and a neighborhood graph that includes an action `open` and timestamp `t` from node `Firefox.exe` to node `Word.doc`, then $(\texttt{Firefox.exe}, \texttt{open}, \texttt{Word.doc}, \texttt{t})$ is an event where a Firefox process opens a Word file at time `t`.

Sequence. Given an entity `e`, a sequence `S` can be extracted from a causal graph. The sequence `S` includes all events of entity `e`'s neighborhood graph in a temporal order, such that $\texttt{S}_{\{\texttt{e}\}} := \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\}$. Similarly, if a set of entities are given, we can extract a sequence that includes all events from their unified neighborhood graph.

Figure 2 (a) illustrates a causal graph with six entities $\{\texttt{e}_\texttt{A}, \texttt{e}_\texttt{B}, \ldots, \texttt{e}_\texttt{F}\}$. Figure 2 (b) shows the neighborhood graph of $\texttt{e}_\texttt{B}$ that includes node `B`, neighbor nodes $\{\texttt{A}, \texttt{C}\}$ and their connecting edges $\{\texttt{E}_\texttt{AB}, \texttt{E}_\texttt{BC}\}$. Similarly, the neighborhood graph of entities set $\{\texttt{e}_\texttt{B}, \texttt{e}_\texttt{C}\}$ includes the nodes $\{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \texttt{E}\}$ and edges $\{\texttt{E}_\texttt{AB}, \texttt{E}_\texttt{BC}, \texttt{E}_\texttt{CD}, \texttt{E}_\texttt{CE}\}$ shown in Figure 2 (b). The events of entity $\texttt{e}_\texttt{B}$ is $\varepsilon_{\texttt{AB}} =< \texttt{e}_\texttt{A}, \texttt{a}_1, \texttt{e}_\texttt{B}, \texttt{t}_1 >$ and $\varepsilon_{\texttt{BC}} =< \texttt{e}_\texttt{B}, \texttt{a}_2, \texttt{e}_\texttt{C}, \texttt{t}_2 >$ shown in Figure 2 (c). The event sequence from the entity set $\{\texttt{e}_\texttt{B}, \texttt{e}_\texttt{C}\}$ is shown in Figure 2 (d).

Threat Model and Assumptions. We assume the underlying OS and the auditing applications are part of the trusted computing base (TCB) similar to prior research on provenance tracking [2, 35]. Hence, the audit logs used to construct the causal graph are tamperproof. We consider that the system is benign at the outset, and the attack origin is external to the enterprise, where the attacker uses remote network access to infiltrate the systems. The attack goal is to exfiltrate sensitive data via a set of actions such as information gathering, user manipulation, vulnerable software exploitation, injecting malicious payloads, installing backdoors, and laterally moving to other hosts to perform similar attack actions.

3 Approach Overview

ATLAS, an attack investigation tool, integrates natural language processing and deep learning techniques into data provenance analysis to model sequence-based attack and non-attack behavior. Figure 3 gives an overview of the ATLAS architecture. It mainly consists of two components: sequence-based model learning (a), and attack investigation (b).

During sequence-based model learning (a), ATLAS processes system and application (e.g., browser) logs and builds a causal graph (❶). Here, we implement a set of pre-processing optimizations to reduce the burden of obtaining complex sequences. These optimizations do not affect ATLAS's sequence semantics logic and improve the efficacy of sequence extraction from large-scale causal graphs. ATLAS then constructs sequences of different attack (suspicious or malicious) and non-attack (normal) activities from the optimized causal graph in order to model their behaviors (❷). The constructed non-attack sequences are then undersampled and attack sequences are oversampled as training data to balance the ratio between attack and non-attack sequences (❸). Lastly, ATLAS uses word embedding to map the lemmatized sequences to vectors of real numbers, which capture the context of an entity in a sequence and the relation with other entities (❹). Through the steps above, the extracted sequences enforce the memory of attack patterns through different causal relations, which helps to build an accurate model that can identify the attack traces across different instances (e.g., similar attacks implemented on different victims). Such a sequence-based representation naturally fits the training of a learning model (e.g., LSTMs), similar to models for machine translation and audio, to identify potential future attacks (❺). The learning process is effective because the key steps of different attacks often share similar patterns (semantics) at the entity and action level. More specifically, different attack instances share a generalized pattern regardless of their log-level details, and temporal orders of a sequence effectively separate normal behavior from suspicious behavior.

During attack investigation (b), learning a model from sequences allows a cyber analyst to reason about future attacks. A cyber analyst starts an attack investigation from unknown audit logs with an identified attack symptom entity such as a

suspicious hostname. Here, ATLAS aims at identifying entities among many unknown entities that are involved in attack phases together with the attack symptom entity. To do so, ATLAS uses the attack symptom entity together with each unknown entity, constructs sequences, and uses the trained model to identify whether a sequence is attack or non-attack. If a sequence is classified as an attack, ATLAS infers that the unknown entity is an attack entity. This process helps reduce the time needed to investigate large causal graphs and accelerates the attack investigation by identifying the key attack entities that make up the attack story.

Design Challenges. Developing ATLAS for effective and scalable attack investigation raises a set of unique challenges. Below we present these challenges and how we address each.

The first challenge concerns *constructing sequences to model legitimate and suspicious activities*. We aim at finding sequences that can better separate benign and malicious activities, and generalize sequences extraction across different audit log types. In traditional sequence problems [15], this poses two challenges to obtain the sequences from audit logs. First, there exists a huge number of unique entities in audit logs, such as different processes with multiple instances, and each entity set (i.e., combination) maps to a different arbitrary length sequence. Second, the same attack patterns occurring in different process executions lead to different sequences with the same or highly similar sequence contexts. These may lead to long and repeating entity-based sequences affecting the model convergence and precision in learning (e.g., vanishing and exploding gradients [14]). To address these issues, ATLAS applies a customized graph-optimization to reduce graph complexity (see Sec. 4.1). As a result, short yet appropriate length sequences are obtained. Additionally, ATLAS implements a novel technique to extract and learn sequences that properly represent attack patterns (see Sec. 4.2).

A second challenge concerns the *model learning from sequences*. Attack investigation is historically similar to "finding needles in a haystack", where many activities are monitored, and only a few of them signal a true attack. This results in imbalanced datasets consisting of under-represented attack sequences and over-represented non-attack sequences. At attack investigation, the curse of imbalanced sequences substantially undermines the learning process [38] and tends to bias the model towards non-attack sequences, leaving a number of attack sequences undetected. To address this issue, ATLAS implements under-sampling to reduce the number of non-attack sequences and over-sampling to generate extra attack sequences, obtaining an appropriate balancing ratio between attack and non-attack sequences (see Sec. 4.2.3).

The third challenge is the *automated attack investigation* using the trained sequence-based model. Though ATLAS supports querying arbitrary sequences on the model and reports whether the sequence is attack or non-attack, the generation of such sequences by investigators is ad-hoc and may require the finding of many sequences with candidate attack entities.

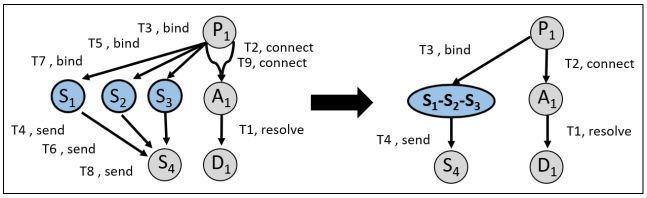

Figure 4: Illustration of graph optimization in ATLAS. P: Process, S: Session, A: IP Address, D: Domain name.

To address this issue, ATLAS includes an attack investigation phase, which thoroughly analyzes entities in audit logs to identify attack entities that form an attack sequence when paired with an attack symptom entity. Thus, it is able to comprehensively recover those attack entities that help build the attack story more accurately and efficiently (see Sec. 4.3).

4 ATLAS

In this section, we detail the ATLAS architecture introduced in Figure 3. We start with an *audit log pre-processing* phase that constructs and optimizes the causal graph for scalable analysis (Sec. 4.1). We then present a *sequence construction and learning* phase that constructs attack and non-attack sequences for model learning (Sec. 4.2). Lastly, we present an *attack investigation* phase that uses the model to identify attack entities, which helps build the attack story (Sec. 4.3).

4.1 Audit Log Pre-processing

For model learning and attack investigation, ATLAS starts by transforming audit logs into a platform-independent causal graph to extract sequences. Here, we build an optimized causal graph that reduces logs complexity (i.e., reducing the number of nodes and edges) without sacrificing key semantics for attack investigation. A less complex graph leads to shorter sequences, a crucial metric that guarantees the efficacy and precision of the sequence-based model learning. ATLAS uses three techniques for causal graph optimization. First, ATLAS eliminates all nodes and edges which are not reachable from the attack nodes (in model learning) or the attack symptom node (in attack investigation). Second, ATLAS constructs the causal graph from the audit logs with non-repeating edges, thus, we drop all repeated edges except the edge of the first occurrence of an action (e.g., read or write) between a subject and an object entity, regardless of how many times an action is repeated. As shown in Figure 4, for nodes $\texttt{P}_1$ and $\texttt{A}_1$, among the two events $(\texttt{P}_1, \texttt{connect}, \texttt{A}_1, \texttt{T2})$ and $(\texttt{P}_1, \texttt{connect}, \texttt{A}_1, \texttt{T9})$ which have the same action (connect), ATLAS only considers the event with the earliest timestamp (T2) for constructing the causal graph. Third, ATLAS combines certain nodes and edges if they refer to the same type of events. Turning to Figure 4, the session nodes $\texttt{S}_1$, $\texttt{S}_2$ and $\texttt{S}_3$ are combined into one node $\texttt{S}_1 - \texttt{S}_2 - \texttt{S}_3$, as they share the same incoming-edges (bind)

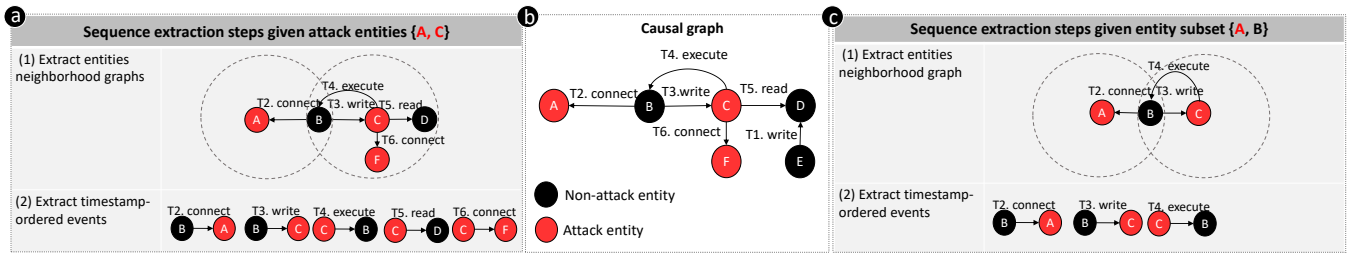

Figure 5: (Middle) An example causal graph to illustrate sequence construction process. (Left) Attack sequence extraction steps. (Right) Non-attack sequence extraction steps.

and outgoing-edges (`send`). During this process, ATLAS assigns the earliest timestamp of their original edges to the new edge. While this might break the original temporal order of events when building the sequence, it does not affect the identification of expected attack patterns, as the temporal order of events in constructed sequences are consistent between the model learning and attack investigation phases. Through this process, ATLAS achieves on average an 81.81% reduction in terms of the number of the entities, compared to the original causal graph (see Sec. 6.3).

4.2 Sequence Construction and Learning

ATLAS transforms the causal graph into sequences labeled either "attack" or "non-attack" (Sec. 4.2.1), and extends lemmatization and selective sampling into the sequence construction to effectively abstract attack and non-attack patterns (Sec. 4.2.2-4.2.3). Lastly, it uses word embedding to convert sequences into vectors of real numbers and learns a sequence-based model through LSTM (Sec. 4.2.4).

4.2.1 Attack and Non-attack Sequence Extraction

ATLAS uses attack entities as ground-truths to extract attack and non-attack sequences for model training. The entities such as a malicious host-name and payload known to us at attack execution are labeled "attack" and other entities are labeled "non-attack". The attack entities here are the ones that can only be associated with attack events. We use this criteria to distinguish them from non-attack entities. We detail the sequence extraction process below.

Attack Sequences. The attack sequences include temporally ordered events of attack entities. ATLAS first obtains a set of all attack entities from a causal graph and constructs their entity subsets that include two or more entities. For example, Figure 5 (Middle) shows three attack entities $\{\mathtt{A},\mathtt{C},\mathtt{F}\}$ in a causal graph, which have the attack subsets of $\{\mathtt{A},\mathtt{C}\}$, $\{\mathtt{A},\mathtt{F}\}$, $\{\mathtt{C},\mathtt{F}\}$ and $\{\mathtt{A},\mathtt{C},\mathtt{F}\}$ that include two or more entities. Formally, if a causal graph includes $\mathtt{k}$ attack entities, the number of attack entity subsets is $\mathtt{m_a} = \sum_{\mathtt{i}=2}^{\mathtt{k}} \mathtt{C}_{\mathtt{k}}^{\mathtt{i}}$, where $\mathtt{C}_{\mathtt{k}}^{\mathtt{i}}$ is all possible subsets of choosing $\mathtt{i}$ attack entities from $\mathtt{k}$. We note that the number of attack entity subsets can be exponential when $\mathtt{k}$ (the number of attack entities) is large. However, in practice, the number of attack entities are usually not large (e.g., less than 40) as attackers normally try to hide and minimize the traces of their activities. For instance, it is in an attacker's best interest of remaining stealthy to drop one backdoor (represented as one attack entity) instead of dropping $\mathtt{n}$ number of backdoors (represented as $\mathtt{n}$ entities). For each attack entity subset, ATLAS extracts an attack sequence from the optimized causal graph through the following steps. First, for each entity in the attack entity subset, ATLAS extracts its neighborhood graph (see its definition in Sec. 2). This step enables ATLAS to capture all entities which have causal relations with an attack entity. To illustrate, given an attack entity subset $\{\mathtt{A},\mathtt{C}\}$, Figure 5 (Left) Step (1) shows neighborhood graphs of `A` and `C` entities in dashed circles. Second, ATLAS obtains the attack events ordered by timestamps from the constructed neighborhood graph. An event is labeled attack if the source or destination node represents an attack entity. For instance, the extracted attack events for the subset $\{\mathtt{A},\mathtt{C}\}$ are shown in Figure 5 (Left) Step (2), where attack events represent timestamp-ordered nodes connected by edges extracted from the neighborhood graph of the attack entities `A` and `C`. Lastly, ATLAS converts the extracted timestamp-ordered attack events to a sequence, and labels it as attack if (a) it only consists of attack events, and (b) it includes all the attack events of the entity subset. For example, the extracted sequence for the subset $\{\mathtt{A},\mathtt{C}\}$ is labeled attack, since it consists of all the attack events that contain the attack entities `A` or `C`.

Non-attack Sequences. A naive approach to identify non-attack sequences would be similar to constructing attack sequences. That is, obtaining all non-attack entities in a causal graph and extracting their sequences by following the steps above. However, this process is complicated due to the exponential number of non-attack entities. We note that ATLAS does not attempt to learn or identify any benign activity (i.e., non-attack sequences). Instead, it aims to accurately learn and identify the boundary between malicious and non-malicious activities. To this end, ATLAS adds a non-attack entity to each attack subset to extract a non-attack sequence. The added non-attack entity can potentially add non-attack events into the sequence, which enables ATLAS to extract attack-sequence deviations (i.e., non-attack sequences), and

Table 1: Abstracted vocabulary set for lemmatization

Type	Vocabulary
process	system_process, lib_process, programs_process, user_process
file	system_file, lib_file, programs_file, user_file, combined_files
network	ip_address, domain, url, connection, session
actions	read, write, delete, execute, invoke, fork, request, refer, bind receive, send, connect, ip_connect, session_connect, resolve

to precisely learn the similarities and differences between attack and non-attack sequences. Formally, if a causal graph includes $\mathtt{k}$ attack entities and $\mathtt{k}'$ non-attack entities, the number of non-attack entity subsets is $\mathtt{n_a} = \sum_{\mathtt{i}=1}^{\mathtt{k}} \mathtt{C}_{\mathtt{k}}^{\mathtt{i}}.\mathtt{k}'$, where $\mathtt{C}_{\mathtt{k}}^{\mathtt{i}}$ is all possible subsets of choosing $\mathtt{i}$ attack entities from $\mathtt{k}$.

Figure 5 (Middle) shows three attack entities $\{\mathtt{A},\mathtt{C},\mathtt{F}\}$ used to extract all possible attack subsets $\{\mathtt{A}\}, \ldots, \{\mathtt{A},\mathtt{C},\mathtt{F}\}$ that include one or more attack entities. To generate non-attack entity subsets, ATLAS appends one entity at a time from the three non-attack entities $\{\mathtt{B},\mathtt{D},\mathtt{E}\}$ to the extracted attack entity subsets. For each non-attack entity subset, ATLAS then extracts non-attack sequences from the causal graph similar to attack-sequences through the following steps. First, for each entity in the subset, ATLAS extract the neighborhood graph for the entity node. For example, for the non-attack entity subset $\{\mathtt{A},\mathtt{B}\}$, ATLAS extracts the neighborhood graph for entities A and B as shown in Figure 5 (Right) Step (1). Second, ATLAS extracts the ordered events from the neighborhood graph. Figure 5 (Right) Step (2) shows the extracted events for the non-attack entity subset $\{\mathtt{A},\mathtt{B}\}$, which includes ordered events represented by edges extracted from the neighborhood graph for entities A and B. Lastly, ATLAS labels a sequence as non-attack if it does not match any extracted attack sequence, otherwise, the processed sequence is discarded. For example, the extracted sequence for the subset $\{\mathtt{A},\mathtt{B}\}$ is labeled as a non-attack because it does not match any attack sequence.

Sequence Length and Number of Sequences. The sequence length is the total number of entities and actions in a sequence. The sequence construction process of ATLAS does not lead to fixed-length sequences as each sequence may consist of different number of events obtained from a causal graph. Further, the number of attack and non-attack sequences extracted from a casual graph depends on the size of the causal graph, which can include different numbers of entities and events associated with the attack and non-attack entities. Therefore, ATLAS can extract varying lengths and numbers of attack and non-attack sequences from a given causal graph.

4.2.2 Sequence Lemmatization

ATLAS uses lemmatization to transform the sequences into a generalized text representing the sequence patterns for semantic interpretation. Lemmatization is often applied in natural language processing to group differently inflected forms of a word as a single term [37]. This process retains the original semantics of the complete sequences and is conducive to sequence-based model learning. Table 1 shows the four different vocabulary types and the vocabulary in each type that ATLAS uses to abstract entities and actions in a sequence. The vocabulary includes a total of 30 words, which reduces inflectional forms and derivationally related forms of words to a common base form. The vocabulary is grouped into four different types based on fine-grained semantics of the words: process, file, network, and actions. The process, file and network types are used to lemmatize entities. These types are sufficient to capture the context of entities in a causal graph, semantic and syntactic similarity and relation with other words. ATLAS parses each sequence, finds the entities and map each of them to a corresponding vocabulary. For example, `</system/process/malicious.exe read /user/secret.pdf>` is transformed to `<system_process read user_file>`. Overall, the sequences after lemmatization process are transformed into a "sentence-like" intermediate representation which contains the full semantics of generalized sequence patterns. We note that undesired repeating of attack and non-attack sequences may occur after lemmatizing the sequences. To train the model with non-repeating sequences, we discard all non-attack sequences that overlap with an attack sequence before they are passed to the selective sequence sampling, detailed next.

4.2.3 Selective Sequence Sampling

The number of attack and non-attack sequences constructed can be imbalanced. The reason is that there are generally fewer attack entities than non-attack entities in the log entries. For example, we found in our evaluation by analyzing audit logs that the average number of attack entities is 61, while the average number of non-attack entities is around 21K. Training the classifier using such an extremely imbalanced dataset would make it either biased in favor of the majority (non-attack) class or unable to learn the minority (attack) class [14]. To balance the training dataset, ATLAS first undersamples non-attack sequences with a certain similarity threshold. Then, it uses the oversampling mechanism to randomly mutate those attack sequences, until their total number reaches the same number of non-attack sequences. A naive technique to balance a training dataset would be to either duplicate the sequences in the minority attack sequences or randomly remove a subset of the sequences in the majority non-attack sequences. Unfortunately, our initial prototype showed that this leads to a model that over-fit to specific attack patterns or miss many important non-attack patterns. To address these issues, ATLAS uses two mechanisms detailed below.

Undersampling. ATLAS reduces the number of non-attack sequences through Levenshtein Distance [3] to compute the similarity among lemmatized sequences. it then filters out sequences when their similarities exceed an identified threshold. While Levenshtein Distance is often applied in NLP to find the similarity between sentences, ATLAS computes the number of editing steps such as adding or deleting vocabu-

lary words in a sequence to transform a sequence to another lemmatized sequence. The complexity of this process for all sequences in a training set is $O(n^2)$. For each sequence, ATLAS removes the sequences when their similarity exceeds a certain threshold. Particularly, through our experiments, we found that a threshold of 80% similarity between sequences yields a good undersampling ratio that sufficiently filters out highly similar and redundant sequences.

Oversampling. ATLAS employs a mutation-based oversampling mechanism to include a larger variety of attack sequences to the training sequences. Recall that ATLAS defines different vocabulary words that represent different processes and file types (e.g., `system_process` and `program_process`). Here, for each extracted attack sequence after lemmatization, ATLAS randomly mutates one vocabulary word type to another vocabulary word of the same type. This process does not fundamentally change the mutated sequence. However, it increases the number of similar sequences not triggered in the attacks used for model training yet may still occur in other attacks due to contextual differences.

4.2.4 Sequence Embedding and Model Learning

ATLAS uses word-representations embedding [30] to transform the lemmatized sequences into a generalized text representing the sequence patterns for semantic interpretation. This process retains the original semantics of the complete sequence and is conducive to sequence-based model learning.

Sequence Embedding. ATLAS integrates word embedding into model learning to transform the lemmatized sequences into numerical vectors. Word embeddings such as word-representations [30] and word2vec [29] have been widely used in NLP for text representations, since they precisely infer the semantic relations between different words. These vectors define a domain-specific semantic relationship between the vocabularies and help in highlighting the patterns of different sequences for model training. The corpus used for training the word embeddings includes all the lemmatized attack and non-attack sequences from the audit logs. The embedded sequences improve model learning compared to other widely used approaches such as one-hot-encoding [40]. We will present their detailed comparison in Sec. 6.3.

Sequence-based Model Learning. ATLAS uses the Long Short-term Memory (LSTM) [15] network, a subtype of Recurrent Neural Network (RNN) [41] to learn a model from attack or non-attack sequences. LSTM is widely applied and proven to be effective for sequence-based learning in different tasks, such as machine translation [45] and sentimental analysis [52]. The LSTM enables ATLAS to automatically learn a model that differentiate reflected patterns in attack and non-attack sequences. The model also includes a Convolutional Neural Network (CNN) [52], which helps ATLAS capture the stealthy and dynamic nature of APT attacks. Specifically, the learning model uses (1) a Dropout layer for regularization to reduce overfitting and improve generalization error, (2) a Conv1D layer with Max Pooling to process lemmatized sequences, (3) a dense, fully-connected layer with sigmoid activation to predict the attack-relevancy probability of the sequences. This model yields better accuracy compared to other architectures we have experimented with. We detail full architecture of the LSTM model in Appendix A, and compare its classification performance with traditional machine learning models (e.g., Support Vector Machines) in Sec. 6.3.

4.3 Attack Investigation

We describe how ATLAS helps a security investigator conduct attack investigation after a sequence-based model is trained. The investigation often starts from one or more attack symptom entities. For instance, an attack symptom might be a malicious website or an IP address identified by a security analyst or reported by network monitoring systems like Nagios [33] as threat alerts. Here, ATLAS helps automatically discover more attack entities through the given attack symptom entities by querying the sequence-based learning model to find out the entities related to the attack symptom. We detail this process below.

Attack Entity Identification. The goal of ATLAS's investigation phase is to recover all the attack entities related to a given attack symptom entity. Here, ATLAS enumerates all unknown entities and identifies whether an entity in a causal graph is an attack or non-attack entity. This process has a time complexity of $O(n)$ for traversing all unknown entities (n) in the causal graph. We note that ATLAS is able to start an investigation with a varying number of (one or more) attack symptom entities since it exhaustively trains the model with a varying number of attack entities (see Sec. 4.2.1).

To illustrate, Figure 5 (Middle) shows three graph nodes that represent attack entities $\{A, C, F\}$ in a causal graph. One or more of these entities can be given as known attack symptom entities during the investigation, and the rest of the entities, whether they are attack or non-attack, are unknown. To identify the unknown attack entities, ATLAS first obtains a set of all unknown entities from a causal graph and constructs its subsets that include one unknown entity. ATLAS then appends the attack symptom entities to each subset; thus, each subset contains all the known attack symptom entities and only one unknown entity. For example, given the attack symptom entity A in Figure 5 (Middle), ATLAS constructs its subsets $\{A, B\}$, ..., $\{A, F\}$. ATLAS uses these subsets to extract sequences from the causal graph as detailed in Sec. 4.2.1. The LSTM model is then used to predict whether each sequence is attack or non-attack through a prediction score. This process identifies whether the unknown entity is closely relevant to the attack symptom entity by inspecting whether the temporal-ordered events of these two entities form an attack pattern that the model previously learned. An identified attack sequence indicates the unknown entity is a part of the attack entities. To

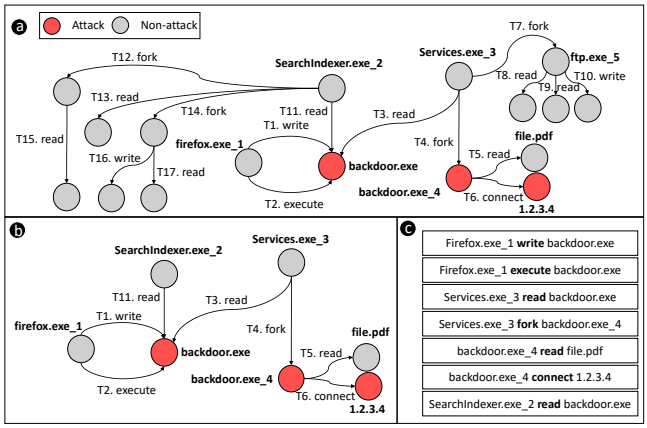

Figure 6: Illustration of an attack story recovery process.

illustrate, Figure 5 (Left) shows an example of sequence construction for the subset $\{A, C\}$ where A is an attack symptom entity and C is an unknown entity. To extract the sequence for A and C, ATLAS first extracts the neighborhood graph to find their related events and reforms the neighborhood graph nodes and edges into a sequence of timestamp-ordered events. This process is applied to all the entity subsets, which results in a set of different sequences with varying lengths. ATLAS then lemmatizes the sequence and passes its word embeddings to the model. If the sequence is classified as an attack sequence, ATLAS infers that the unknown entity in the subset (i.e., C) is an attack entity.

Attack Story Recovery. The goal of ATLAS attack story recovery is to identify attack events associated with the identified attack entities from the attack investigation phase. ATLAS extracts the neighborhood graph of the identified attack entities and obtains all included events as attack events. These events are further ordered by their timestamps as the recovered attack story. We note that the mapping between identified attack entities to attack events is highly dependent on the attack under investigation. For example, if ATLAS recovers 30 attack entities at attack investigation, there can be a varying number of events associated with those 30 entities depending on the number of attack actions (e.g., read or write file). Figure 6 (b)-(c) illustrates the steps that ATLAS constructs an attack story from the causal graph illustrated in Figure 6 (a). We consider that during the attack investigation phase ATLAS has successfully recovered the attack entities {`backdoor.exe` (backdoor file), `backdoor.exe_4` (backdoor process) and `1.2.3.4` (malicious host)}. ATLAS uses these attack entities to extract their neighborhood graph in Figure 6 (b), which includes the attack events. This mapping between the attack entities and events allows ATLAS to automatically extract those related attack events without the need for the cyber analyst to perform any manual investigation. For instance, the non-attack entity `SearchIndexer.exe_2` (the Windows NT program) that continuously enumerates and reads files metadata, make a normal read to the `backdoor.exe` file. We note that ATLAS includes this as an attack event in the neighborhood graph in Figure 6 (b) since it includes the attack entity `backdoor.exe`. In general, if a process reads a malicious file, the process likely becomes a part of the attack, and it can be used by the attacker to launch further attack actions. Yet, ATLAS does not include other events (e.g., `(SearchIndexer.exe_2, fork, ε, T12)`) which originate from the process `SearchIndexer.exe_2`, even if they occur after the attack event `(SearchIndexer.exe_2, read, backdoor.exe, T11)`. Lastly, ATLAS reports the attack events ordered by their timestamps from the constructed neighborhood graph as shown in Figure 6 (c).

Handling Multi-host Attacks. To investigate an attack targeting multiple hosts, a cyber analyst often starts from one host and includes more hosts as the investigation progresses. Thus, the attack entities recovered from a host are indicative of including more hosts for cross-host attack investigation. Consider an investigation of a compromised web server that has a malicious program `backdoor.exe` in its web directory. When ATLAS identifies the attack entity of `backdoor.exe`, it uses this entity as a new attack symptom entity to investigate other hosts that have downloaded `backdoor.exe`. This enables ATLAS to naturally support multi-host attack scenarios in a scalable manner. As a result, ATLAS investigation does not require association of the causal graph among different hosts, which is often necessary for provenance tracking techniques [19]. We show in our evaluation that the effectiveness of ATLAS is not affected by the attacks performed across multiple hosts, and it only needs to perform analysis on audit logs from individual hosts to discover all attack entities (see Sec. 6.2). To construct a multi-host attack story, ATLAS merges the audit logs from the compromised hosts and constructs a *unified* optimized causal graph (as detailed in Sec. 4.1) representing the logs of compromised hosts. ATLAS then uses the identified attack entities from those hosts to extract a neighborhood graph that includes all the attack events in the causal graph. Lastly, ATLAS constructs a sequence that details a temporal order of the attack events across multiple hosts (an example case study is presented in detail in Sec. 6.5).

5 Implementation

We implemented ATLAS in Python 3.7.7, with around 3,000 lines of code (LoC) for all its components. Our prototype processes Windows security events for system logs (with Sysmon enabled to log files operations and network connections), Firefox logs for visited webpages, and TShark for DNS logs. ATLAS uses the LSTM model as implemented in the Keras library [6] with the TensorFlow [1] back-end. The LSTM model is tuned with the parameters through grid search for better generalization, and to prevent overfitting on the training data (see Appendix A for model architecture details).

To ensure the efficiency of the LSTM model, ATLAS shortens sequences to a predefined threshold at training as this leads to the vanishing gradients problem [14]. We have found a sequence length of 400 yields similar or better accuracy on the test data compared to other sequence lengths, as the majority of the extracted sequences are shorter than 400 words. We note that the exclusion of such sequences does not lead to losing the key semantics of the attack patterns. Specifically, (1) compared to those normal activities which frequently appeared in audit logs, most attacks are highly targeted to specific goals and hence tend to form shorter sequences, (2) the graph optimization (Sec. 4.1) shortens long sequences, and more importantly, (3) the long sequences of attack steps are often covered by their sub-sequences with shorter lengths, which are extracted through subsets of attack entities.

6 Evaluation

We begin by describing our experimental settings (Sec. 6.1). We then present the effectiveness of ATLAS (Sec. 6.2), efficiency of each component (Sec. 6.3) and the run-time overhead of attack identification (Sec. 6.4). Lastly, we demonstrate a case study to illustrate the use of ATLAS for attack investigation in practice (Sec. 6.5).

6.1 Experimental Settings

Dataset. The lack of publicly available attack datasets and system logs is a common challenge in forensic analysis. For example, the data released by DARPA's Transparent Computing program do not include audit logs generated during evaluation engagements [47]. To address these, we have implemented ten attacks based on their detailed reports on real-world APT campaigns and generated the audit logs in a controlled testbed environment. Additionally, similar to previous works that construct benign system events [20, 26, 36], we emulate diverse normal user activities on the same machine during each attack execution in a best-effort. More specifically, we manually generated various benign user activities including browsing different websites, executing different applications (e.g., reading emails, downloading attachments), and connecting to other hosts. Similar to a typical work day environment, such activities are randomly performed within an 8-hour-window during the daytime. More details about normal user behaviors and log statistics collected can be found in Appendix B. Table 2 details each attack that exploits different vulnerabilities (i.e., CVEs). These attacks are selected to include different malware tactics such as phishing links, email attachments, intermediate processes, and lateral movements such as leaking sensitive data. The attacks `S-1` to `S-4` were performed on single hosts and `M-1` to `M-6` were performed on multiple hosts. For each multi-host attack, the emulation was performed on two hosts where the second host was used as the target for lateral movement. All attacks were developed and executed on Windows 7 32-bit virtual machines and took about an hour to complete. After the attacks were completed, we collected the audit logs within a 24-hour-window. Table 2 column "Size (MB)" details the size of the collected logs, and the column "Log Type" shows the total percentages of different types of events in the audit logs. Overall, the 24-hour emulation generated an average of 20,088 unique entities with 249K events for each attack.

Evaluation Setup. We have the ground-truth of attack entities for each attack, known to us at attack execution. For instance, a malicious URL set by us in an attack to upload sensitive data is an attack entity. Other entities in a causal graph are labeled as non-attack. These entities are used to construct the events and sequences, following the procedures as we have elaborated in Sec. 4.2 and Sec. 4.3. Table 3 presents the number of entities, events, sequences, and balanced sequences for each attack. For example, `S-1` includes 22 attack and 7,445 non-attack entities. These entities are associated with 4,598 and 90,467 attack and non-attack events. These events are used to compose 42 attack and 14,243 non-attack lemmatized sequences. Lastly, 1,388 balanced attack and non-attack sequences are obtained through the selective sequence sampling process and used for model training. As detailed in Table 3, similar to realistic attack investigating scenarios, the malicious activities only constitute a vanishingly small percentage of the log data (less than 0.14% attack entities in the whole audit logs). Hence, we believe our dataset can reasonably reflect ATLAS's true efficacy for real-world attack investigation.

We evaluate the effectiveness of ATLAS for each implemented attack based on the model trained on other attacks. For example, if we aim at identifying the multi-host attack `M-1`, we use a model trained on audit logs of the multi-host attacks `M-2,...,M-6` excluding `M-1`. We separated single-host and multi-host attacks in the training phase because both types of attacks were implemented based on the same APT reports (e.g., both `S-1` and `M-1` are implemented based on [17]). This setting ensures that training and testing data do not overlap with each other. Overall, ATLAS trains ten separate models to evaluate each attack.

After the models are trained, the attack investigation is performed in two steps as detailed in Sec. 4.3. First, we generate sequences by randomly selecting a single attack symptom entity from the ground-truth attack entities. These identified attack symptom entities naturally represent real-world cases where a security analyst often starts from (see Table 4- Column "Symptom Entity"). Second, we pass the sequences that are generated by combining each unknown entity in a causal graph with the symptom entity and check whether each constructed sequence is identified as attack or non-attack. This enables us to find unknown entities that are indeed relevant to the attack (as detailed in Sec. 4.3). Since ATLAS investigation is entity-based, we present the attack investigation results in terms of entities. Additionally, we present the attack identification results in terms of events similar to other attack

Table 2: Overview of implemented APT attacks for ATLAS evaluation.

Attack ID	APT Campaign	Exploiting CVE by attack	Attack Features†							Size (MB)	Log Type (%)			Total	
			PL	PA	INJ	IG	BD	LM	DE		System	Web	DNS	# entity	# event
S-1	Strategic web compromise [17]	2015-5122	√		√	√	√		√	381	97.11%	2.24%	0.65%	7,468	95.0K
S-2	Malvertising dominate [22]	2015-3105	√		√	√	√		√	990	98.58%	1.09%	0.33%	34,021	397.9K
S-3	Spam campaign [39]	2017-11882		√	√	√	√		√	521	96.82%	2.43%	0.75%	8,998	128.3K
S-4	Pony campaign [18]	2017-0199		√	√	√	√		√	448	97.08%	2.24%	0.68%	13,037	125.6K
M-1	Strategic web compromise [17]	2015-5122	√		√	√	√	√	√	851.3	96.89%	1.32%	1.32%	17,599	251.6K
M-2	Targeted GOV phishing [34]	2015-5119	√		√	√	√	√	√	819.9	97.39%	1.36%	1.25%	24,496	284.3K
M-3	Malvertising dominate [22]	2015-3105	√		√	√	√	√	√	496.7	99.11%	0.52%	0.37%	24,481	334.1K
M-4	Monero miner by Rig [28]	2018-8174		√	√	√	√	√	√	653.6	98.14%	1.24%	0.62%	15,409	258.7K
M-5	Pony campaign [18]	2017-0199	√		√	√	√	√	√	878	98.14%	1.24%	0.62%	35,709	258.7K
M-6	Spam campaign [39]	2017-11882		√	√	√	√	√	√	725	98.31%	0.96%	0.73%	19,666	354.0K
Avg.	-	-	-	-	-	-	-	-	-	**676.5**	**97.76%**	**1.46%**	**0.73%**	**20,088**	**249K**

† **PL**: Phishing email link. **PA** : Phishing email attachment. **INJ**: Injection. **IG**: information gathering. **BD**: backdoor. **LM**: Lateral movement. **DE**: Data ex-filtration.

Table 3: Ground-truth information of each implemented attack, including the number of entities, events, sequences and balanced sequences.

Attack ID	#Attack Entity	#Non-attack Entity	#Attack Event	#Non-attack Event	#Attack Seq.	#Non-attack Seq.	#Balanced Seq.*
S-1	22	7,445	4,598	90,467	42	14,243	1,388
S-2	12	34,008	15,073	382,879	43	13,388	1,386
S-3	26	8,972	5,165	123,152	21	8,600	2,598
S-4	21	13,016	18,062	107,551	32	12,238	1,244
M-1	28	17,565	8,168	243,507	83	26,764	2,682
M-2	36	24,450	34,956	249,365	82	27,041	2,748
M-3	36	24,424	34,979	299,157	81	27,525	2,710
M-4	28	15,378	8,236	250,512	79	27,076	2,746
M-5	30	35,671	34,175	667,337	78	25,915	2,540
M-6	42	19,580	9,994	344,034	70	23,473	2,598
Avg.	**28**	**20,051**	**17,341**	**275,796**	**61**	**20,626**	**2,264**

* The sampled number of attack and non-attack sequences are identical.

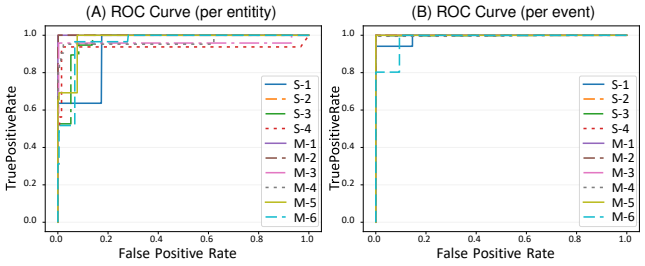

Figure 7: ROC curves at entity (left) and event (right) level.

investigation works [11, 25]. We generate the events-based results by using the identified attack entities. We iterate through all events in audit logs, and if an event's subject or object matches one of the identified attack entities, then we label that event as an attack. Lastly, we compare the number of classified attack and non-attack entities and events with their ground-truth labels and report classification metrics.

6.2 Effectiveness

This section presents the effectiveness of ATLAS at identifying attack entities and events for each attack (Sec. 6.2.1), and details its individual components (Sec. 6.2.2).

6.2.1 Attack Investigation Results

We report the effectiveness of ATLAS at identifying attack entities and events for each attack in Table 4. For example, the first row shows the investigation results for $\mathtt{S-1}$ given a malicious host as an attack symptom entity. Table 4, Column "Entity-based Investigation Results" shows that ATLAS correctly identifies attack entities with an average 91.06% precision and 97.29% recall. This means that most key entities of the APT attacks are successfully recovered with a very limited number of false positives. For instance, from an investigator's perspective, given 100 attack entities, ATLAS recovered around 91 true attack entities, with the other nine being false positives. Similarly, 97.29% recall means that ATLAS recovered around 97 attack entities, with three attack entities remaining undiscovered. We also report the identification results in terms of events in Table 4 Column "Event-based investigation Results". Clearly, since the huge number of normal entities results in many more normal events compared to the less-frequently occurring attack events, the precision of event-level results (99.88%) is much higher than the entity-level results (91.06%). This means that ATLAS helps significantly reduce the number of suspicious events that need to be manually checked by an investigator.

Lastly, to show the overall effectiveness of our models, we present the ROC curves for identifying each attack in Figure 7. Here, ATLAS achieves on average 97.01% Area Under Curve (AUC) at the entity-level, and 99.67% AUC at the event-level. The high precision and recall results indicate that different attacks share a high-level similarity in terms of attack steps and sequences generated by those attack entities. For example, many attacks tend to open a malicious webpage that exploits a vulnerable browser, then drop and execute backdoors and extract data of interest. In contrast, normal program executions in uncompromised hosts rarely perform such activities, which is clearly reflected in the causal sequences generated by any combinations of their entities. Since each entity could be associated with multiple events in the audit logs, the number of false positives and negatives for the event-based results are much higher than the entity-based results. However, we note that even in this case, the number of reported false positives and false negatives identified by ATLAS are very small compared to the number of true positives and true negatives.

Analysis of False Positives. False positives are the number of non-attack (normal) entities and events that ATLAS incorrectly classified as attack (see Table 4, Column 5 and 12). ATLAS yields on average a 0.01% false positive rate for both

Table 4: Entity-based and event-based investigation results.

ID	Symptom entity	Entity-based Investigation Results							Event-based Investigation Results						
		TP	TN	FP	FN	Precision %	Recall %	F1-score %	TP	TN	FP	FN	# Precision %	# Recall %	F1-score %
S-1	malicious host	22	7,445	0	0	100.00%	100.00%	100.00%	4,598	90,467	0	0	100.00%	100.00%	100.00%
S-2	leaked file	12	34,008	2	0	85.71%	100.00%	92.31%	15,073	382,876	3	0	99.98%	100.00%	99.99%
S-3	malicious host	24	8,972	0	2	100.00%	92.31%	96.00%	5,155	123,152	0	10	100.00%	99.81%	99.90%
S-4	leaked file	21	13,011	5	0	80.77%	100.00%	89.36%	18,062	107,506	45	0	99.75%	100.00%	99.88%
M-1	leaked file	28	17,562	3	0	90.32%	100.00%	94.92%	8,168	243,504	3	0	99.96%	100.00%	99.98%
M-2	leaked file	36	24,445	5	0	87.80%	100.00%	93.51%	34,956	249,316	49	0	99.86%	100.00%	99.93%
M-3	malicious file	35	24,423	1	1	97.22%	97.22%	97.22%	34,978	299,147	10	1	99.97%	100.00%	99.98%
M-4	malicious file	24	15,378	0	4	100.00%	85.71%	92.31%	8,161	250,512	0	75	100.00%	99.09%	99.54%
M-5	malicious host	30	35,665	6	0	83.33%	100.00%	90.91%	34,175	667,329	8	0	99.98%	100.00%	99.99%
M-6	malicious host	41	19,573	7	1	85.42%	97.62%	91.11%	9,993	343,959	75	1	99.26%	99.99%	99.62%
Avg.	-	**27**	**20,048**	**3**	**1**	**91.06%**	**97.29%**	**93.76%**	**17,332**	**275,777**	**19**	**9**	**99.88%**	**99.89%**	**99.88%**

TP and **TN** stands for correctly reported attack and non-attack (normal) entities/events. **FP** and **FN** stands for incorrectly labeled attack and non-attack (normal) entities/events.

entity-level (3 out of 20,075 entities) and event-level (19 out of 293,109 events) analyses. These results show that a cyber analyst requires less manual labor to validate the causes of a true attack needed for the attack story. We found that most false positives are due to the misclassification of the IP addresses. For instance, most false positives in `M-5` and `M-6` attacks were due to the benign IP addresses, which were active during the same time as those malicious IP addresses of the Command and Control (C&C) servers. However, the security investigators can easily identify the IP addresses by checking their traffic content and registration information to filter out such false positives.

Analysis of False Negatives. False negatives are the number of attack entities and events that ATLAS incorrectly classified as non-attack (see Table 4, Column 6 and 13). ATLAS yields on average a 2.71% false-negative rate at the entity-level and a 0.11% false-negative rate at the event-level. We found that even when ATLAS misidentifies an attack entity, ATLAS can still identify attack entities which were caused by such a misidentified attack entity. For example, the false negatives in attack `M-4` are due to misidentifying a malicious Microsoft Word file (`evil.rtf`) that is used to download a malicious payload; however, ATLAS was able to identify the malicious payload entity (`payload.exe`) which was caused by the misidentified word file. False negatives in the attacks `M-6` and `S-3` are caused by missing some scripts downloaded from a stealthy network I/O trace performed by the attacker. Here, the attacker uses the Multiple UNC Provider (MUP) [7], a specific component in the Windows NT operating system, to access shared folders across the network. We note that the false negatives can be alleviated by training ATLAS with more attack types sharing similar patterns with these cases.

6.2.2 Individual Component Analysis

The effectiveness of ATLAS lies in a set of optimization techniques integrated into its components. Here we elaborate on how these components contribute to its effectiveness.

Causal Graph Optimization. As detailed in Sec. 4.1, we developed our customized optimization algorithms to construct the causal graph which helps reduce the graph complexity and in turn improves the sequence construction. Figure 8 shows

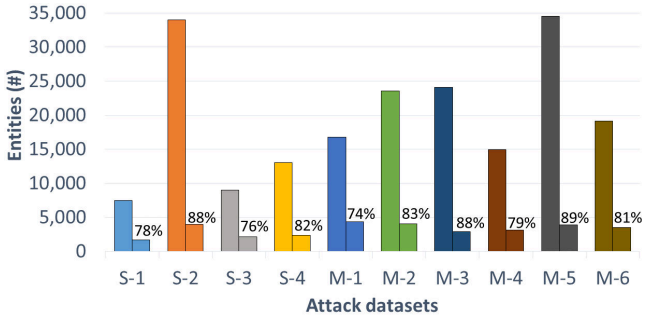

Figure 8: Effectiveness of causal graph optimization of given audit logs for attack investigation. The percentages on the bars show the percentage of the logs reduction.

the number of entities before and after graph optimization. ATLAS reduces the number of entities in a causal graph on average 81.81% for audit logs of each attack compared their original graph size. The reduction removes the redundant or unrelated events from the huge volume of logs that do not contribute any semantics to model different attack sequences. Hence, the further extracted sequences are more representative and efficient as input for the model training.

Selective Sequence Sampling. The selective sequence sampling mechanism of ATLAS is the key step to building a precise model from a balanced dataset. As illustrated in Table 3 (Column 6 and 8), ATLAS oversamples the attack sequences with an average 37x increase, from 61 to 2,264, and undersamples non-attack sequences with an average reduction of 9x, from 20,626 to 2,264. Our evaluation shows that this process reduces the training time on average 87% (from 3h:37min to 0h:28min for training each model). Overall, this mechanism extracts an average of 22% of the initial sequences as a highly representative training set and uses them for model training, which significantly improves the model accuracy.

6.3 Comparison Analysis

We have implemented a set of state-of-the-art approaches that can be used in lieu of ATLAS components and compare their performances with ATLAS in attack identification. We note that the comparison is conducted through event-based attack investigation results as previous provenance tracking

Table 5: Comparison of ATLAS with the baseline approaches.

Method	Precision	Recall	F1-score
Graph-traversal	17.82%	100.00%	30.26%
Non-optimized causal graph	87.58%	41.55%	56.36%
Oversampling-only model	97.85%	79.64%	87.81%
One-hot encoding	99.60%	80.75%	89.19%
Support Vector Machine (SVM)	87.12%	90.42%	88.74%
ATLAS	**99.88%**	**99.89%**	**99.88%**

approaches (e.g., [11, 16, 50]) provide event-based attack results. Table 5 summarizes our results.

Graph Traversal. To compare ATLAS in attack investigation with the state-of-art approaches [11, 16, 50], we have implemented a baseline approach that performs backward and forward tracing on the provenance graph, a directed acyclic graph built based on the W3C provenance data model specification [48]. We note that since none of the previous approaches are publicly available, we are not able to perform a direct comparison with them. This approach proceeds in two steps. First, the backward tracing starts from an attack symptom event and stops at the root causes of the attack (e.g., a phishing link in malicious email). Second, the forward tracing starts from the identified root causes of the attack and stops at the final attack effects (e.g., leaked files). Finally, we include all traversed nodes and edges along the paths as the recovered attack story, and compare the result with the ground-truth of each attack in Table 3 (Column 4 and 5).

Table 5 (first row) presents the results of graph-traversal baseline in identifying attack events. The baseline yields 100% recall (i.e., recovers all attack events) yet it results in an average precision of 17.82%. The main reason for the low precision is the well-known dependency explosion problem [11, 31] that introduces a significant amount of non-attack events as false provenance when traversing the causal graph. For example, the backward and forward analysis can identify a long-lived process that forks many other system processes as attack, and this can add a large number of false attack dependencies. In our experiments, we found that many recovered attacks include the process entity `services.exe` and its corresponding events, where `services.exe` forks every service in the Windows system. ATLAS does not rely on traversing the graph to identify attack events and yields a significantly higher precision without sacrificing the recall.

Non-optimized Causal Graph. Table 5 (second row) presents the attack investigation results of ATLAS with and without graph optimization. We observe that the non-optimized causal graph reduces the precision, recall and F-1 score by 12.30%, 58.34% and 43.52%, respectively. This is because the graph optimization removes redundant or unrelated events from the huge volume of logs that do not contribute semantics and temporal relationship to model different attack sequences, and prevents model overfitting. Overall, the graph optimization process helps ATLAS extract shorter attack/non-attack sequences and improves the model generalization.

Table 6: Average time (hh:mm:ss) to train the model and investigate individual attacks.

Phase	Graph construction	Sequences processing	Model learning/inference	Total
Training	0:04:11	0:26:12	0:28:26	0:58:49
Investigation		0:00:04	0:00:01	0:04:16

Oversampling-only Model. The process of oversampling attack sequences balances the limited number of attack sequences with the vast number of non-attack sequences. If the oversampling was not used, then the imbalanced dataset biases the sequence-model towards the more common non-attack sequences, which yields high non-attack prediction scores for all sequences. To evaluate the benefit of undersampling non-attack sequences, we compare ATLAS with an oversampling-only baseline. Table 5 (third row) shows the oversampling-only model reduces the precision, recall and F1-score by 2.03%, 20.25% and 12.07% respectively. This is because, without undersampling, non-attack sequences tend to bring more amplified noise data to the classifier. Instead, our similarity-based undersampling approach helps reduce such noisy data while retaining the key patterns in sequences.

One-hot Encoding. We compare ATLAS with a simplified baseline by replacing the word embedding with one-hot-encoding to show the effectiveness of using word embeddings in attack investigation. One-hot-encoding is mainly used to convert the categorical variables to numerical vectors that could be inputted to the ML algorithms [40]. Table 5 (fourth row) presents results of ATLAS's word embedding with the one-hot-encoding, which reduces precision, recall and F1-score by 0.28%, 19.14% and 10.69% respectively. The main reason is that one-hot-encoding ignores the semantic relations between different words. In contrast, the word embedding helps better to differentiate those fine-grained behaviors between attack and non-attack sequences.

Support Vector Machine (SVM). To evaluate the effectiveness of the LSTM classifier, we compare it with the SVM [46], an alternative simpler classifier which is widely used for binary classification tasks. In addition to SVM, we also experimented with the Random Forest classifier [23], which gives less accurate classification results than SVM. We have evaluated the SVM classifier using the same training data for each attack. We used a grid search to tune the parameters to improve classifier accuracy, a linear kernel with `C=1.0` and `gamma="auto"`. The SVM reduces the precision, recall and F1-score by 12.76%, 9.47%, and 11.14% respectively (see Table 5 (fifth row)). The main limitation of SVM is that it is unable to model the temporal relations among different entities of a sequence, one of the critical features that reflects the attack patterns.

6.4 Performance Overhead

ATLAS is trained on attack and non-attack sequences offline; thus, it only introduces overhead on the order of seconds at in-

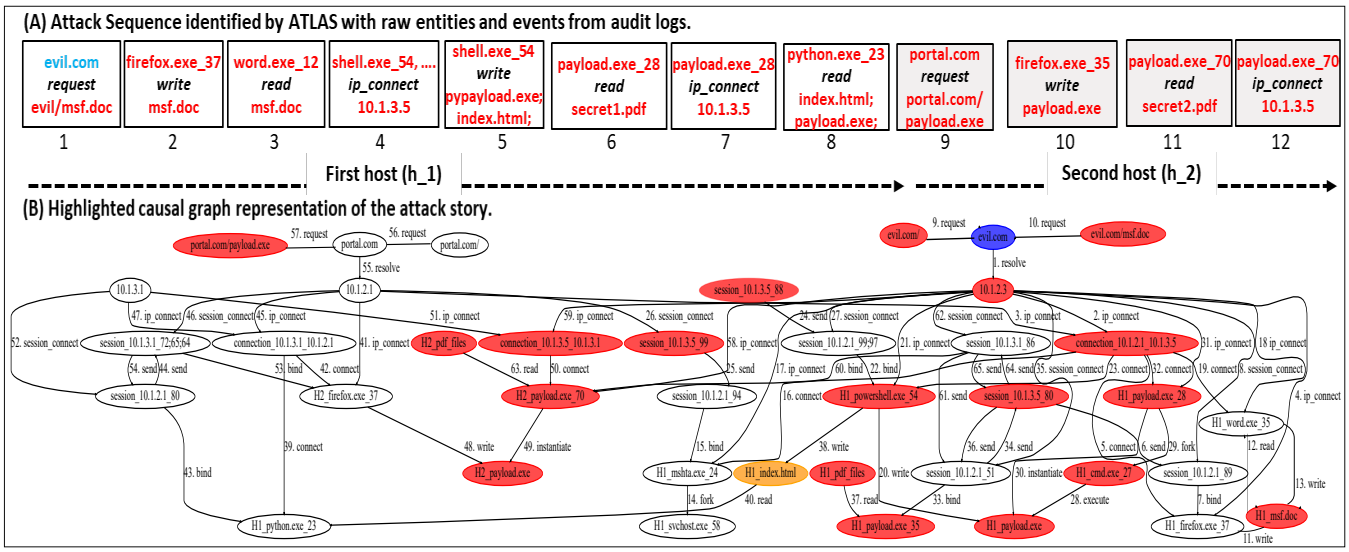

Figure 9: Recovered sequences and causal graph of the "Pony campaign" attack (M-5).

ference time to identify sequences as attack or non-attack. We note that forensics tools often rely on a system or application-level instrumentation for inference [21, 27], which often requires more time compared to ATLAS. We evaluated ATLAS's performance at model training and attack identification phases. Table 6 presents the time used for graph construction, sequences processing and model learning and inference. ATLAS takes on average four minutes to process each 24-hour audit logs to construct the causal graph with an average size of 676.5 MB audit logs. Further, the model training phase takes an average of 26 minutes to construct all attack and non-attack sequences and to balance the sequences, with an additional 28 minutes to train the LSTM model. In total, the training process for each attack takes less than one hour.

We note that training ATLAS is a one-time effort, and new attacks can be incrementally added to the LSTM model without requiring re-training the previously learned attacks. In addition, our experiment was performed on a laptop PC, meaning the training time could be significantly reduced with more powerful machines in the production environment (e.g., multi-core and large-memory servers). Although ATLAS takes time to build the causal graph for starting the attack investigation, it only takes around four seconds to extract the sequences. In addition, it only takes on average one second to recover the attack story (with 1-day audit logs) by going through the complete list of unknown entities in the causal graph.

6.5 Case Study

We illustrate how ATLAS can be deployed and benefit cyber analysts for attack investigation through a case study (i.e., the attack `M-5` which we used in evaluation). We use numbers in Figure 9-A to illustrate the key steps of this attack. Here, a user downloads a malicious document that compromises the Word process on the victim machine (❶ – ❸). Thereafter, an injected shellcode forks other processes and grants additional capabilities to the attacker, including information gathering and downloading files to the victim system (❹ and ❺). It also executes a backdoor that the attacker uses to leak a secret file to a C&C server (❻ and ❼). Additionally, the attacker identifies that the compromised host acts as the company portal's web server. For lateral movement, the attacker uploads the backdoor code to this web server, and adds a piece of code to the portal main webpage `portal.com/index.html` (orange node in the causal graph) to prompt a message asking users to update their machines (❽). After users download and install the backdoor (❾), more secret files are leaked to the C&C server (❿ and later).

Figure 9-B illustrates the causal graph constructed for this attack. We note that though we simplified the causal graph for ease of presentation, the figure still includes many non-attack entities (the white nodes in graph), which can be difficult for the analysts to manually analyze for attack investigation. The attack investigation starts from a malicious hostname `evil.com` (the blue node). ATLAS first identifies a set of attack entities (red nodes), and a set of non-attack entities (white nodes) through the learning model. Second, ATLAS reports the identified events in temporal order as a sequence, which helps an investigator to reason about the attack story. For this attack, ATLAS only reports six false positives in terms of entities and recovers the attack story similar to Figure 9-A.

7 Limitations and Discussion

The preceding analysis of ATLAS shows that it can precisely recover the key attack steps from the attack symptoms, and help security investigators obtain the attack story. Although we focused our work on Windows platform logs, ATLAS can be

easily extended to other platforms such as Linux and FreeBSD. This is because our approach starts its analysis from any log entities and builds a customized platform-independent causal graph. We plan to extend our framework to diverse types of audit logs in the future. We note that a list of manually designated attack entities is required for ATLAS training. However, labeling such data is a one-time-effort for investigating future attacks. Another limitation of ATLAS is that it cannot detect attacks that use a similar sequence of normal events to hide its behavior, such as the mimicry attacks [4, 5, 49]. However, we note that following the behaviors of normal events will significantly limit the capability of any attack investigation techniques [31, 36]. Besides, ATLAS requires the analyst to start an investigation with a true attack-symptom entity. Using a false positive entity as an attack-symptom will only discover non-attack sequences since their subset entities include a non-attack entity. Lastly, the correctness of the sequence-based model highly depends on the quality of the collected training log entries. Hence, more representative temporal relations among attacks will enable ATLAS to learn more precise models. This can be easily alleviated by introducing more types of attacks to the training set.

8 Related Work

ATLAS is mainly related to three sub-topics that support provenance tracking with audit logs, including causality analysis over the provenance graph, anomaly-based analysis, and application of ML techniques for attack investigation.

Causality Analysis. Much prior work has been done on causality analysis over audit logs for attack investigation, including optimizing the provenance graph and reporting a concise attack story [20, 21, 27]. These approaches require system modifications via source-code instrumentation, static binary-level instrumentation, or dynamic program instrumentation at runtime. Unfortunately, source-code level instrumentation is not applicable for proprietary software due to software licenses, while static and dynamic instrumentation incur additional overhead on the user-system. Recent works proposed instrumentation-free approaches [10, 13, 16, 31, 51] that do not require any changes to the user-system for provenance tracking. However, most of these approaches are heuristic- or rule-based, which require non-trivial effort to develop and maintain the rules or heuristics. HOLMES [31] and RapSheet [10] rely on a knowledge base of adversarial Tactics, Techniques, and Procedures (TTPs) [32]. In contrast, ATLAS only requires attack training data to learn the co-occurrence of attack steps through temporal-ordered sequences.

Anomaly-based Analysis. Anomaly-based approaches [11, 12, 25, 50] learn the normal system behavior to identify anomalous behavior. Unfortunately, while anomaly-based approaches can effectively detect unknown attacks, they are notoriously prone to false positives due to user behavior change over time and lack of sufficient training data. For instance, a host-based intrusion detection framework Unicorn [9] learns a model from normal provenance graphs to detect anomalies. PrioTracker [25] ranks node importance with statistical information to more accurately report real attack events. NoDoze [11] reduces false alarms by computing and propagating anomaly scores within a dependency graph. Winnower [12] provides threat alerts for cluster auditing by noticing the difference between multiple instances of clusters. ProvDetector [50] identifies stealthy malware through learning the sequences of normal execution paths of applications from a provenance graph. Deeplog [8] models existing audit logs as natural language sequences and detects abnormal events. Lastly, Log2vec [24] proposes a clustering framework to identify unseen abnormal sequences from system logs. Unlike anomaly-based approaches that only learn user behaviors, ATLAS learns both attack and non-attack (user) sequences and exploits their temporal and causal relations to reduce false positives and false negatives.

Learning-based Analysis. Learning-based attack investigation approaches [36, 42, 43] use machine learning techniques to model attack events in the logs. HERCULE [36] uses a community detection algorithm to correlate attack events. Similar to ATLAS, a number of recent works [42, 43] employ word embeddings to transform the textual information (i.e., sequences) into vectors to facilitate its learning process. However, these approaches are limited to identifying and reporting individual attack events in logs. In contrast to these approaches, ATLAS aims to locate attack entities and construct an attack story through associating each entity with its events.

9 Conclusion

We have presented ATLAS, a framework to identify and reconstruct end-to-end cyber attack stories from unmodified system and software audit logs. ATLAS employs a novel combination of causality analysis, natural language processing, and machine learning techniques that model and recognize high-level patterns of different attacks through a sequence-based analysis. Evaluation results over 10 real-world APT attack scenarios showed that ATLAS successfully recovered key attack steps which constitute the attack story with both high precision and efficiency.

Acknowledgments

We thank our shepherd, Adam Bates, and the anonymous reviewers for their valuable comments and suggestions. This work was supported in part by ONR through a subcontract from the Intelligent Automation, Inc., an LDRD Grant from Sandia National Labs, and a gift from Cisco. Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[2] Adam Bates, Dave Jing Tian, Kevin RB Butler, and Thomas Moyer. Trustworthy whole-system provenance for the linux kernel. In *USENIX Security Symposium*, 2015.

[3] Karin Beijering, Charlotte Gooskens, and Wilbert Heeringa. Predicting intelligibility and perceived linguistic distance by means of the levenshtein algorithm. *Linguistics in the Netherlands*, 15:13–24, 2008.

[4] Z Berkay Celik, Patrick McDaniel, Rauf Izmailov, Nicolas Papernot, Ryan Sheatsley, Raquel Alvarez, and Ananthram Swami. Detection under privileged information. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 199–206, 2018.

[5] Z Berkay Celik, Robert J Walls, Patrick McDaniel, and Ananthram Swami. Malware traffic detection using tamper resistant features. In *IEEE Military Communications Conference*, 2015.

[6] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015. Accessed: 2020-06-06.

[7] Microsoft Corporation. Support for unc naming and mup. https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/support-for-unc-naming-and-mup, 2017. Accessed: 2020-06-06.

[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[9] Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. Unicorn: Runtime provenance-based detector for advanced persistent threats. *arXiv preprint arXiv:2001.01525*, 2020.

[10] Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2020.

[11] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *Network and Distributed Systems Security Symposium*, 2019.

[12] Wajih Ul Hassan, Mark Lemay, Nuraini Aguse, Adam Bates, and Thomas Moyer. Towards scalable cluster auditing through grammatical inference over provenance graphs. In *Network and Distributed Systems Security Symposium*, 2018.

[13] Wajih Ul Hassan, Mohammad A Noureddine, Pubali Datta, and Adam Bates. Omega-log: High-fidelity attack investigation via transparent multi-layer log analysis. In *Network and Distributed Systems Security Symposium*, 2020.

[14] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Md Nahid Hossain, Sanaz Sheikhi, and R Sekar. Combating dependence explosion in forensic analysis using alternative tag propagation semantics. In *IEEE S&P*, 2020.

[17] FireEye Threat Intelligence. Second adobe flash zero-day cve-2015-5122 from hackingteam exploited in strategic web compromise targeting japanese victims. https://www.fireeye.com/blog/threat-research/2015/07/second_adobe_flashz0.html, 2015. Accessed: 2020-06-06.

[18] Genwei Jiang, Rahul Mohandas, Jonathan Leathery, Alex Berry, and Lennard Galang. CVE-2017-0199: In the Wild Attacks Leveraging HTA Handler. https://www.fireeye.com/blog/threat-research/2017/04/cve-2017-0199-hta-handler.html, 2017. Accessed: 2020-06-06.

[19] Samuel T King, Zhuoqing Morley Mao, Dominic G Lucchetti, and Peter M Chen. Enriching intrusion alerts through multi-host causality. In *Network and Distributed Systems Security Symposium*, 2005.

[20] Yonghwi Kwon, Fei Wang, Weihang Wang, Kyu Hyung Lee, Wen-Chuan Lee, Shiqing Ma, Xiangyu Zhang, Dongyan Xu, Somesh Jha, Gabriela Ciocarlie, et al. Mci: Modeling-based causality inference in audit logging for attack investigation. In *Network and Distributed Systems Security Symposium*, 2018.

[21] Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. High accuracy attack provenance via binary-based execution partition. In *Network and Distributed Systems Security Symposium*, 2013.

[22] Brooks Li and Joseph C. Chen. Exploit kits in 2015: Flash bugs, compromised sites, malvertising dominate. https://blog.trendmicro.com/trendlabs-security-intelligence/exploit-kits-2015-flash-bugs-compromised-sites-malvertising-dominate/, 2016. Accessed: 2020-06-06.

[23] Andy Liaw, Matthew Wiener, et al. Classification and regression by random forest. *R news*, 2(3):18–22, 2002.

[24] Fucheng Liu, Yu Wen, Dongxue Zhang, Xihe Jiang, Xinyu Xing, and Dan Meng. Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[25] Yushan Liu, Mu Zhang, Ding Li, Kangkook Jee, Zhichun Li, Zhenyu Wu, Junghwan Rhee, and Prateek Mittal. Towards a timely causality analysis for enterprise security. In *Network and Distributed Systems Security Symposium*, 2018.

[26] Shiqing Ma, Juan Zhai, Fei Wang, Kyu Hyung Lee, Xiangyu Zhang, and Dongyan Xu. Mpi: Multiple perspective attack investigation with semantics aware execution partitioning. In *USENIX Security Symposium*, 2017.

[27] Shiqing Ma, Xiangyu Zhang, and Dongyan Xu. Protracer: Towards practical provenance tracing by alternating between logging and tainting. In *Network and Distributed Systems Security Symposium*, 2016.

[28] Trend Micro. Rig exploit kit now using cve-2018-8174 to deliver monero miner. https://blog.trendmicro.com/trendlabs-security-intelligence/rig-exploit-kit-now-using-cve-2018-8174-to-deliver-monero-miner/, 2018. Accessed: 2020-06-06.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[31] Sadegh M Milajerdi, Rigel Gjomemo, Birhanu Eshete, R Sekar, and VN Venkatakrishnan. Holmes: real-time apt detection through correlation of suspicious information flows. In *IEEE Symposium on Security and Privacy*, 2019.

[32] MITRE. Mitre att&ck. https://attack.mitre.org/, 2020. Accessed: 2020-06-06.

[33] Nagios. Network, server and log monitoring software. https://www.nagios.com, 2019. Accessed: 2020-06-06.

[34] Pierluigi Paganini. Phishing campaigns target us government agencies exploiting hacking team flaw cve-2015-5119. https://securityaffairs.co/wordpress/38707/cyber-crime/phishing-cve-2015-5119.html, 2015. Accessed: 2020-06-06.

[35] Thomas Pasquier, Xueyuan Han, Mark Goldstein, Thomas Moyer, David Eyers, Margo Seltzer, and Jean Bacon. Practical whole-system provenance capture. In *ACM Symposium on Cloud Computing*, 2017.

[36] Kexin Pei, Zhongshu Gu, Brendan Saltaformaggio, Shiqing Ma, Fei Wang, Zhiwei Zhang, Luo Si, Xiangyu Zhang, and Dongyan Xu. Hercule: Attack story reconstruction via community discovery on correlated log graph. In *The 32nd Annual Conference on Computer Security Applications*, 2016.

[37] Joël Plisson, Nada Lavrac, Dunja Mladenic, et al. A rule based approach to word lemmatization. In *Proceedings of IS*, 2004.

[38] Enislay Ramentol, Yailé Caballero, Rafael Bello, and Francisco Herrera. Smote-rsb*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory. *Knowledge and information systems*, 33(2):245–265, 2012.

[39] Cedrick Ramos. Spam campaigns with malware exploiting cve-2017-11882 spread in australia and japan. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/spam/3655/spam-campaigns-with-malware-exploiting-cve201711882-spread-in-australia-and-japan, 2017.

[40] Pau Rodríguez, Miguel A Bautista, Jordi Gonzalez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018.

[41] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[42] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. Tiresias: Predicting security events through deep learning. In *ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[43] Yun Shen and Gianluca Stringhini. Attack2vec: Leveraging temporal word embeddings to understand the evolution of cyberattacks. In *USENIX Security Symposium*, 2019.

[44] Riyanat Shittu, Alex Healing, Robert Ghanea-Hercock, Robin Bloomfield, and Muttukrishnan Rajarajan. Intrusion alert prioritisation and attack detection using post-correlation analysis. *Computers & Security*, 50:1–15, 2015.

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[46] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[47] Jacob Torrey. Dapra transparent computing. https://www.darpa.mil/program/transparent-computing, 2014. Accessed: 2020-06-06.

[48] W3C. Prov-dm data model. https://www.w3.org/TR/2013/REC-prov-dm-20130430/, 2013. Accessed: 2020-06-06.

[49] David Wagner and Paolo Soto. Mimicry attacks on host-based intrusion detection systems. In *ACM SIGSAC Conference on Computer and Communications Security*, 2002.

[50] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, C Gunter, et al. You are what you do: Hunting stealthy malware via data provenance analysis. In *Network and Distributed Systems Security Symposium*, 2020.

[51] Runqing Yang, Shiqing Ma, Haitao Xu, Xiangyu Zhang, and Yan Chen. Uiscope: Accurate, instrumentation-free, and visible attack investigation for gui applications. In *Network and Distributed Systems Symposium*, 2020.

[52] Alec Yenter and Abhishek Verma. Deep cnn-lstm with combined kernels from multiple branches for imdb review sentiment analysis. In *IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference*, 2017.

Appendix

A LSTM Model Details

As shown in Table 7, we detail model architecture and parameters to train the LSTM model. Below we present each layer of the model, what parameters represent and how we specify their values. We refer interested readers to a relevant research [52] for more details about the model.

The embedding layer transforms the network index number of each word to an embedding vector. The "Input Maximum Features" represents how many words ATLAS model can learn. Since our vocabulary contains 30 words, we set it to 31 to accommodate the in-vocabulary words and include an additional word for sequences padding. The "Embedding Size"

Table 7: Architecture and parameters of LSTM model.

	Model Architecture		
Embedding	**Input Maximum Features** 31	**Embedding Size** 128	**Maximum Input Length** 400
Convolution (1-dimensional)	**Filters** 64	**Kernel Size** 5	**Activation** ReLU
Max Pooling (1-dimensional)	**Pool Size** 8		
Dropout	**Rate** 0.2		
LSTM	**Output Size** 256		
Dense	**Output Size** 1	**Activation** Sigmoid	
Compiled Model	**Loss Function** Binary Cross Entropy	**Optimizer** Adam	**Metrics** Accuracy
Batch	**(# samples/batch)** 1		
Epoch	**(# training iterations)** 8		
Probability Threshold	**(Classify as attack if equal/greater)** 0.5		

represents the output vector size for each input word. We found that 128 yields a better result than other values. The "Maximum Input Length" represents the maximum length (i.e., number of words) in a sequence that ATLAS can process. We set this parameter to 400 since we found that processing longer sequences lead to the vanishing gradients problem [14].

The 1-dimensional convolution layer is effective in learning spatial features, such as learning adjacent words. The "Filters" parameter represents the convolution output filter size. "Kernel Size" specifies how many embedded words are contained in each convolution branch. We found that setting "Filters" to 64 and "Kernel Size" to 5 yields a better result than other values. The convolution "Activation" is set to the Rectified Linear Unit (ReLU), which replaces negative output values with zeroes, and leads to a better model generalization than other activation functions [52]. Max pooling layer reduces input dimensionality to minimize training time and to mitigate the training overfitting problem [14]. The "Pool Size" specifies the output size; we found that setting this parameter to the value 8 yields a good result. Dropout layer is used to reduce the model overfitting by the factor specified in the parameter "Rate", which we set to 0.20 as we found this value yields a better model in our dataset.

The LSTM layer is used to learn from sequential data effectively. The LSTM output size parameter is set to 256 since we found that the model is more effective when this value is used. Dense layer input is a merged (i.e., concatenated) from the LSTM output and is transformed into a single array. The "Output Size" parameter for the dense layer specifies the overall model output size. We set it to 1 because we seek to find a scalar value representing the sequence class predicted probability. The dense "Activation" is set to Sigmoid to represent the predicted probability as a single value between 0 and 1. The model "Loss Function" parameter is set to a binary

Table 8: Statistics of the simulated normal user behaviors in audit logs for each attack.

Attack ID	# Processes		# Files		# Domain names		# IP Addresses		# Socket send/recv		# Web Requests	
	# U.	# I.	# U.	# I.	# U.	# I.	# U.	# I.	# U.	# I.	# U.	# I.
S-1	46	67,338	3,847	57,684	89	610	120	4,031	920	920	530	1,372
S-2	49	376,315	82,200	310,229	300	1,323	450	35,366	7,685	7,685	1,106	3,065
S-3	25	113,933	5,030	78,899	91	972	143	5,946	1,353	1,353	723	2,161
S-4	39	99,770	4,782	68,998	186	843	177	4,684	1,289	1,289	753	1,968
M-1	78	217,010	8,450	154,549	592	3,319	758	15,520	3,318	3,318	1,131	3,121
M-2	52	206,992	7,948	157,001	573	3,537	736	18,188	3,671	3,671	1,010	2,606
M-3	85	285,859	11,366	197,404	158	1,220	278	8,501	1,782	1,782	425	1,082
M-4	72	236,405	8,856	162,751	188	1,610	309	10,256	2,247	2,247	729	2,169
M-5	79	585,524	21,500	432,745	636	3,096	753	15,071	3,328	3,328	841	2,165
M-6	85	328,490	12,505	224,471	206	2,550	392	13,371	2,740	2,740	762	2,293
Avg.	**61**	**251,764**	**16,648**	**184,473**	**302**	**1,908**	**412**	**13,093**	**2,833**	**2,833**	**801**	**2,200**

* **U.** means *Unique Objects* and **I.** means *Instances*.

entropy loss function, which is an effective and standard loss function for binary classification network architectures. The model "Optimizer" parameter specifies what optimizer we use to optimize the model training accuracy using a loss function feedback. We set this parameter to Adam optimizer since we found that it yields a better classification result. The model "Metric function" parameter specifies what metric the model uses to measure the model performance during the training phase. We set this parameter to the Accuracy metric function as we found that it leads to a more effective model learning.

The "Batch" parameter specifies how many sequences the model can process at a time. We set this parameter to 1 since we found that that model yields a better precision when it processes the sequences one by one. The "Epoch" parameter specifies how many times the model iterates over each sequence during the training phase. We set this value to 8 since we found that this value leads to a more effective model.

The "Probability Threshold" parameter specifies ATLAS classifier threshold at attack investigation, such that if a predicted probability value is greater or equal to the specified threshold, ATLAS then classifies the sequence as an attack; otherwise ATLAS classify the sequence as non-attack. Since ATLAS is trained with balanced datasets using sampling (detailed in Sec. 4.2.3), the classification is no longer biased towards one of the two classes; for this reason, we have chosen the value 0.5 as the probability threshold.

B Attack Simulation

Table 8 presents the details of a user behavior within our collected audit logs, including various activities such as running processes, accessing files, browsing the web, and downloading files. We compute the statistics of different activities during the deployment of each attack. For each type of activity (e.g., the number of running processes), column *U.* shows how many *unique objects* were accessed, and column *I.* shows how many times these objects were accessed (i.e., object instances) during the simulation.