Forensic event reconstruction with process mining

Rida Adila, Hudan Studiawan, Frank Breitinger

Angaben zur Veröffentlichung / Publication details:

Adila, Rida, Hudan Studiawan, and Frank Breitinger. 2026. "Forensic event reconstruction with process mining." *IEEE Access* 14: 18964–85.
https://doi.org/10.1109/access.2026.3660165.

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 26 November 2025, accepted 9 January 2026, date of publication 2 February 2026, date of current version 6 February 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3660165

RESEARCH ARTICLE

Forensic Event Reconstruction With Process Mining

RIDA ADILA[1], HUDAN STUDIAWAN[1], (Member, IEEE), AND FRANK BREITINGER[2]
[1]Department of Informatics, Institut Teknologi Sepuluh Nopember, Surabaya 60111, Indonesia
[2]Chair for Cybersecurity, University of Augsburg, 86159 Augsburg, Germany

Corresponding author: Frank Breitinger (frank.breitinger@uni-a.de)

ABSTRACT Event reconstruction is a fundamental aspect of the investigative process in digital forensics. During this process, one systematically analyzes and organizes evidence to formulate a hypothesis regarding past events. The starting point is often the raw data from forensic timelines (e.g., a table including all parsed events), which may include millions of timeline entries. Various tools and techniques have been proposed to create and analyze timelines. However, the feasibility of applying process mining solutions remains unexplored. Process mining, with its ability to uncover patterns, deviations, and process flows from event data, can offer valuable insights into forensic event reconstruction. In this study, we explore the utilization of episode mining to generate case identifiers and provide event sequences, visualizations, and evaluation metrics from process models generated by process mining algorithms. As a result, we developed an open-source, web-based prototype application. Experiments and case studies conclude that the proposed method can reconstruct events and provide intuitive results to forensic investigators.

INDEX TERMS Event reconstruction, forensic timeline, event abstraction, episode mining, process mining.

I. INTRODUCTION

Event reconstruction is central to digital forensics, providing investigators with a chronological view of (user) activities. To accomplish this, tools parse forensic disk images and generate *forensic timelines*, i.e., comprehensive views that merge logs, metadata, files, and other artifacts into a single sequence [1], [2]. Timelines may contain millions of entries and require additional processing, such as converting low-level events into high-level ones [3], [4]: A *low-level event* is a raw, timestamped action (e.g., file creation, browser history, registry updates), whereas a *high-level event* is a human-readable abstraction formed by combining such entries. An example would be 'detecting a USB connection' which involves recognizing a sequence of low-level events [3]. Common challenges in event reconstruction and timelines are discussed in [5].

As stated, timelines can be used to reconstruct events, i.e., activities a user performed or happened to/on a system. Currently, reconstruction primarily employs approaches such as process algebra [6], formal semantic representation [7], directed graphs [8], or finite-state machines [9] with raw event logs as the data source. Most of these studies do not include evaluation metrics that can show whether the resulting reconstruction model truly represents actual events [10].

In comparison, *process mining* is a research field that aims to understand how processes function by analyzing event logs collected by digital systems [11]. The outcome of process mining is a *process model*, such as a Petri net, which represents the sequence and logic of real-world process execution, extracted from logs [12]. The aim is to automatically discover, monitor, and improve business processes based on the information extracted from event log files [13]. Process mining is widely used, and various tools exist, e.g., ProM,[1] Fluxicon,[2] and Disco.[3]

The question this article tries to answer is: *To what extent can we apply process mining to digital forensic event*

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandra De Benedictis.

[1]https://promtools.org/
[2]https://fluxicon.com/
[3]https://fluxicon.com/disco/

reconstruction? Unfortunately, existing process mining tools cannot be applied directly to forensic timelines, as these timelines lack elements such as a case identifier (case ID), which are required by process mining to analyze process flows [14]. Although case IDs can be added manually as metadata, forensic timelines consist of sequential events from multiple sources without predefined grouping into distinct cases. Therefore, preprocessing is required.

Contribution. This paper offers three key contributions. First, we introduce a novel approach that uses episode mining to automatically derive case identifiers from forensic timelines, addressing the limited attention given to automated segmentation in existing work. Second, we demonstrate how the resulting cases can be combined with process mining techniques to reconstruct events from forensic timelines, thereby eliminating the need for manual preprocessing by investigators. Finally, we release an open-source, fully reproducible workflow that demonstrates the practical value of applying process mining to forensic timelines, providing a basis for further research in this area.

Outline. The remainder of this paper is organized as follows: Section II reviews related work on forensic event reconstruction. Section IV presents the proposed method, which details how process mining and episode mining are utilized for event reconstruction and to generate case identifiers, respectively. Section V reports the experimental results, including dataset details, evaluation metrics, and analysis of the method performance. Section VI concludes the paper and discusses future research directions for improving forensic event reconstruction.

II. RELATED WORK

Event reconstruction in digital forensics has a long-standing history. Gladyshev and Patel [9] present a reconstruction system based on a finite-state machine (FSM) evaluated on a synthetic dataset. Their findings indicate that FSMs perform effectively in low-complexity environments; however, the study does not demonstrate whether the reconstructed sequence faithfully reflects the underlying event data. Similarly, James et al. [15] apply an FSM-based approach to synthetic datasets and report comparable outcomes, noting strong performance on systems of limited complexity. Yet, as with the earlier work, no quantitative evaluation is provided to verify that the reconstructed events accurately correspond to the original data. Instead, the evaluation focuses solely on testing the validity of witness statements against the system model.

Soltani and Seno [6] employ the mCRL2 process algebra to reconstruct events from a FAT file system, demonstrating that their method is can produce plausible reconstruction. However, the study lacks an evaluation metric to verify that the reconstructed sequence accurately reflects the events present in the dataset. Another work proposes a novel digital forensic event reconstruction method using sufficient and necessary evidence, analyzed through linear temporal logic (LTL) and NuSMV model checking [16]. This approach requires checking the final state of the system and is more efficient than previous methods. The results show that this method is fast and accurate, but relies on a complete and accurate formalized system model. The concept of time anchors is introduced in [4] to verify the accuracy of the system time when reconstructing events. Through two experiments, Google search and file creation, the method proves to be effective in detecting time deviations. In terms of the challenges of timestamp manipulation in digital systems, Vanini et al. [17] discuss this issue through a qualitative study involving ten students who were asked to directly swap the order of two events on a live Windows 10 system. The results highlight that tampering is not easy because of limitations in tools, technical knowledge, and the creation of new traces resulting from the manipulation itself.

Several previous studies have used process mining in the fields of network and information security. For instance, Lagraa and State [18] apply process mining to detect suspicious login activities. The underlying dataset contains login activity logs, with the alpha algorithm for the process discovery stage. However, the results of the study show only the visualization of the model without presenting a quantitative evaluation. Another work uses process mining to support security detection in IoT devices using controller area network datasets [19]. In the process discovery stage, the inductive miner algorithm is used. The results indicate that the process mining technique combined with the clustering method provides better performance than machine learning methods, such as random forest, support vector machines, elliptic envelope, and local outlier factor. Zhang et al. [20] introduced a novel approach for assessing file-processing security in multi-cloud environments using process-mining techniques. Their case study demonstrates that the method is effective in detecting intrusion attempts, providing an additional layer of protection for sensitive data in multi-cloud settings. Konsta et al. [21] introduced an automated method for generating attack trees from logs using process mining techniques. The method produces a process model in the form of a tree that captures attacker behavior as reflected in the log data. This process tree then serves as the basis for transforming the model into an attack tree suitable for security risk analysis.

Based on these studies, the following research gap has been identified: No methods have been proposed for reconstructing events from forensic timeline data using process mining. Moreover, no prior work has combined event abstraction with episode mining during the preprocessing stage to generate case identifiers for forensic event reconstruction.

Previous work, such as FSM-based reconstruction, provides structured and visual models, but typically relies on predefined representations and lacks standardized evaluation metrics. In contrast, process mining derives models directly from data and offers established measures, such as fitness and precision. These metrics support a transparent and quantitative assessment of reconstruction quality.

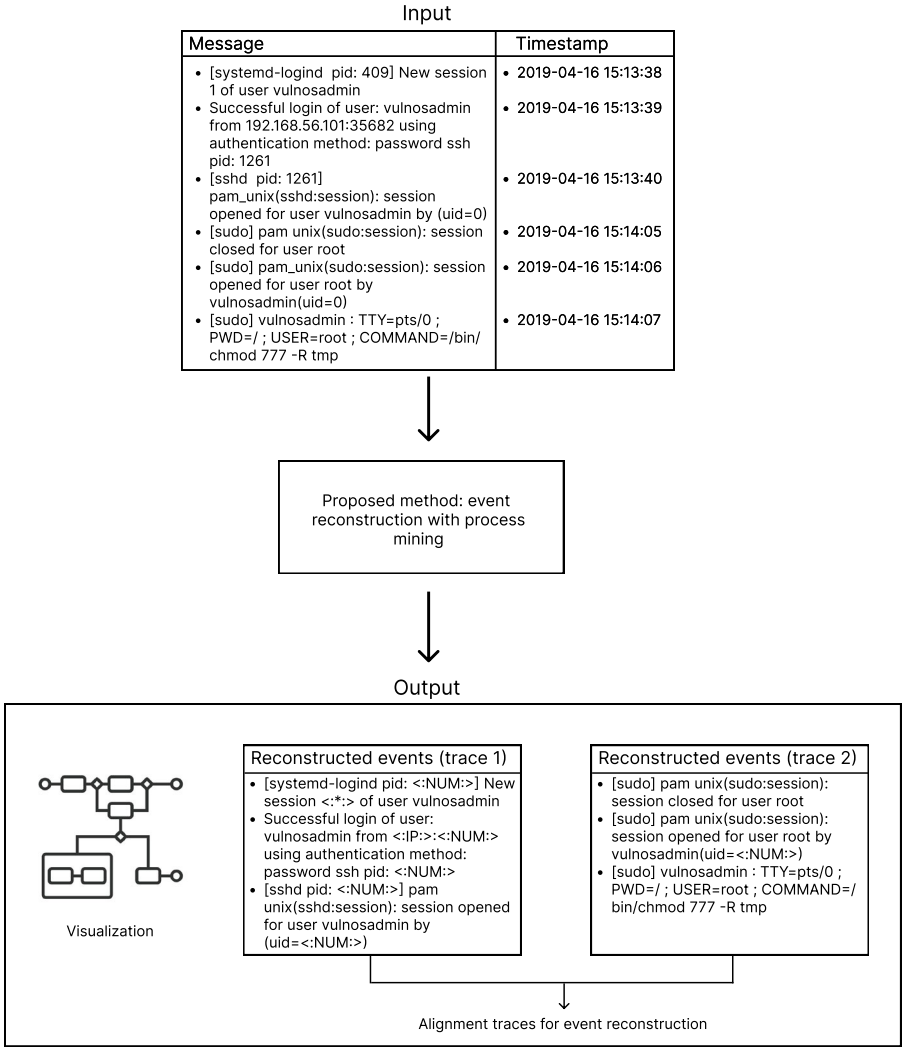

FIGURE 1. Illustrative example of a forensic timeline and the expected reconstructed outcome.

TERMINOLOGY

This section defines the key terms and concepts used throughout this paper to provide clarity and consistency. A *forensic timeline* is a comprehensive timeline that results from parsing entire systems and combining all[4] information from log files and other artifacts into a single timeline [1], e.g., a CSV file. *Event log files* are structured digital records that chronologically capture sequences of events or actions generated by applications or operating systems [22].

Episode mining is a data mining technique used to identify frequently occurring sequential patterns of activities across traces in an event log, where each trace represents the life cycle of a particular case [23]. A *Petri net* is a mathematical modeling method developed by Carl Adam Petri [24], which is commonly used to represent and analyze workflows and system behaviors. *Event abstraction* refers to the process of simplifying large volumes of log data by identifying and extracting essential events to help investigators uncover patterns, relationships, and irregularities that could indicate suspicious or harmful activities [8].

[4]All here means parsable by the application.

III. ILLUSTRATIVE EXAMPLE

This section illustrates an example (see Figure 1), where a forensic timeline is generated from a disk image using *log2timeline/plaso*. The timeline then serves as an input for the proposed method, which produces two outputs: (1) A visualization of the process model and (2) alignment traces.

The process model transforms scattered timeline entries into a structured representation of the underlying process and provides a high-level overview of the system's activity flow. Meanwhile, alignment traces reveal the sequence of events as recorded in the timeline. Together, these outputs enable investigators to reconstruct the incident chronology and to develop a comprehensive understanding of the case. The proposed method clarifies how events relate to one another and reveals how the incident unfolded.

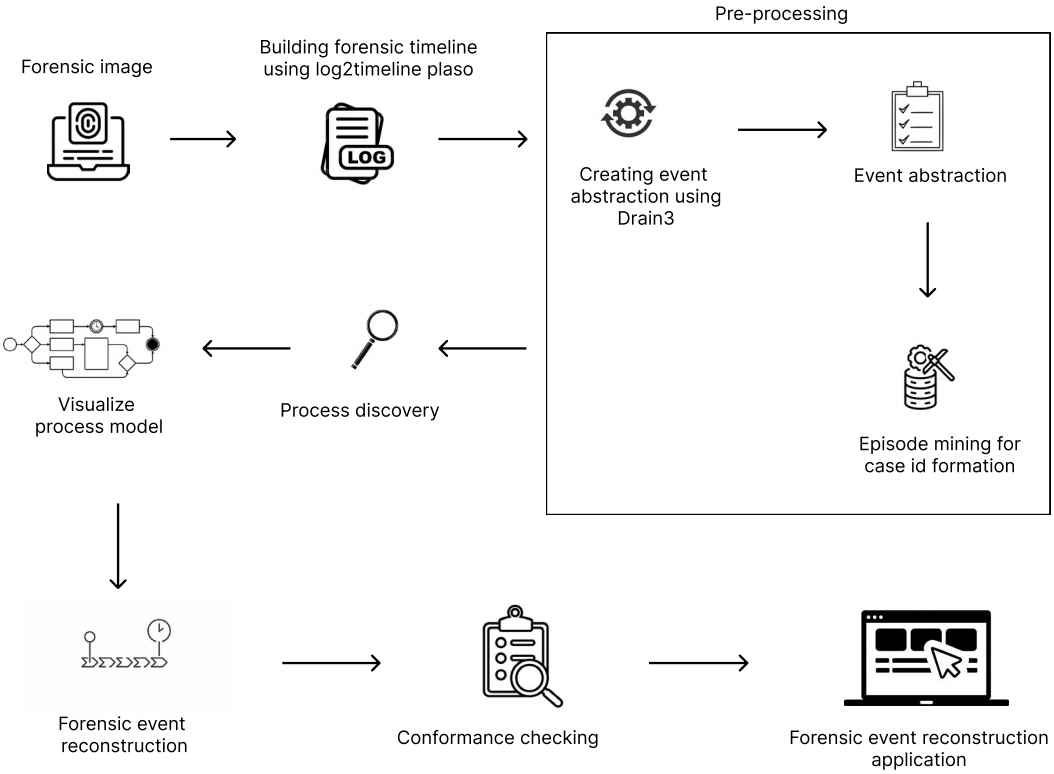

FIGURE 2. The proposed method for forensic event reconstruction with process mining.

IV. PROPOSED METHOD

The research is carried out in several stages, as depicted in Figure 2. The first stage (top left corner) involves obtaining forensic images and building a forensic timeline using log2timeline a.k.a. Plaso.[5] The timeline is essentially a CSV file with eight columns. For this research, only two columns are required for the subsequent stage: the *timestamp* and *message* columns. Both columns are utilized for the event abstraction process to generate *abstract messages*, i.e., common patterns from a set of similar message events. An example is shown in Figure 3, specifically in the box labeled ''grouping event messages based on word length''.

The process and techniques are summarized below; details are provided in the upcoming sections. We decided to use the Drain method [25] for event abstraction to structure the forensic log data. The event abstraction process produces a *cluster identifier* (cluster ID) and an *abstract message*. The cluster ID serves as the *event identifier*, which, together with the *abstract message*, is then mapped back to the original row in the timeline data as the input for the next stage. Subsequently, a *case identifier* is created using the episode mining method.

Process mining includes two main tasks [26]:

1) *Process discovery* generates a process model from data, such as a forensic timeline, without requiring prior knowledge of the underlying process [27]. A *process model* is a diagrammatic representation that captures the key behavioral structure of a process [28].
2) *Conformance checking* compares event logs against a predefined process model to determine whether the recorded executions align with the expected behavior [29]. A key technique used in conformance checking is *alignment*, which reconstructs the execution trace by mapping each event in the log to the most appropriate step in the process model [29].

Although events are present in the forensic timeline, their underlying process structure and the sequence of activities to which they correspond are often implicit. To address this, we provide an open-source web-based tool that performs trace reconstruction by aligning each event with the closest matching step in a predefined process model, such as a Petri net. The visualization generated through process mining techniques helps to make the process structure and the sequence of activities it corresponds to explicit. This allows for better identification of patterns [13]. The proposed tool also offers flexibility by allowing users to specify a time range and view reconstruction results within that period. Subsequently, it helps analysts interpret and visualize the flow of events in forensic timelines.

Once each event in the timeline has a *case identifier* and an *abstract message*, process discovery is performed to derive a process model that represents the chronological

[5]https://github.com/log2timeline/plaso

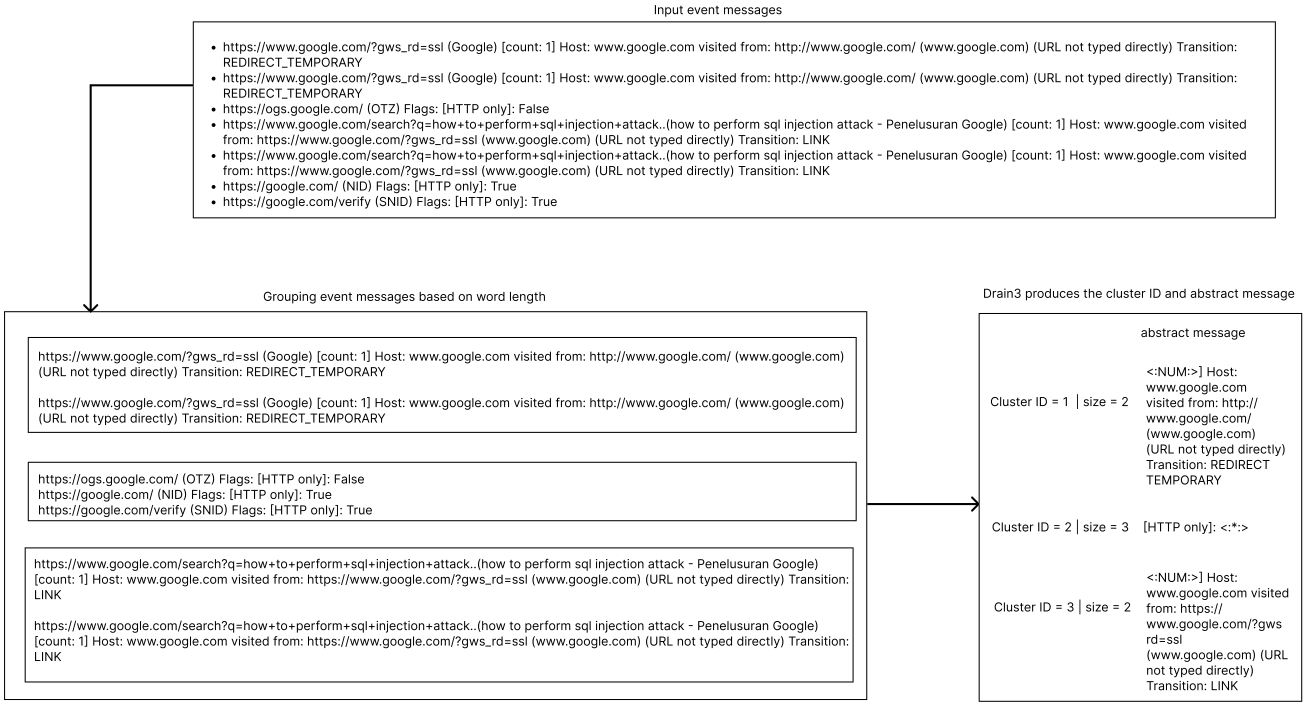

FIGURE 3. Example of abstraction results from event messages using drain3.

sequence of events. The process model is then evaluated and validated by conformance checking to ensure its accuracy in representing forensic event sequences. Finally, the process model is visualized as a Petri net to provide a representation of the event flow. After completing all steps, the research progresses by developing a web-based application using the Python Flask framework, which enables users to input the required data, generate the evaluation metrics, and visualize the resulting Petri net.[6]

A. BUILDING FORENSIC TIMELINE USING LOG2TIMELINE/PLASO

Having the forensic disk image, the next step is to generate a forensic timeline using log2timeline/Plaso. The tool produces eight columns: *datetime*, *timestamp_desc*, *source*, *source_long*, *message*, *parser*, *display_name*, and *tag*. As mentioned earlier, only the two columns are utilized: The *timestamp* column represents the time of an event, while the *message* column contains important information about the type and details of the event.

B. CREATING EVENT ABSTRACTION USING THE DRAIN METHOD

As the resulting forensic timeline has many entries (rows), it is necessary to perform an event abstraction process as shown in Figure 3 to obtain a template message a.k.a. general pattern for each event. In this research, the Drain method [25] is applied. The outcome of this stage is an *abstract message* in the form of a template message and a *cluster identifier* as shown in the bottom right box of Figure 3. The *cluster identifier* can be considered as a unique identification for each type of event or can be referred to as an *event identifier*. The obtained *abstract messages* and *event identifier* are then mapped back to the initial timeline data, while the *timestamp* column remains unchanged.

Drain is designed to parse logs efficiently by overcoming the limitations of traditional methods, which often require offline processing and are limited in terms of handling large log volumes quickly. The process starts by collecting (unstructured) raw log messages and applying a preprocessing step using regular expressions based on domain knowledge to mask dynamic tokens, such as IP addresses, block IDs, port numbers, or other parameters. In Drain, this masking is performed immediately upon log arrival, before the log is processed through the parsing tree. This early masking ensures that the fixed structure of the log message (the log template) can be recognized and analyzed accurately, without being affected by constantly changing values [25].

A list of masks and the corresponding regular expressions used in the Drain implementation is presented in Table 1. The predefined masks listed in Table 1 are obtained directly from the original configuration file (drain.ini)[7] and from the previous study [8].

[6]https://flask.palletsprojects.com/en/stable/

[7]https://github.com/logpai/Drain3

TABLE 1. Predefined regular expressions for masking from the original Drain3 configuration file (drain.ini).

No	Mask	Regex Pattern
1	CMD	`(?<=executed cmd )(".*?")`
2	NUM	`((?<=[^A-Za-z0-9])|^)([-+]?\d+)((?=[^A-Za-z0-9])|$)`
3	HEX	`((?<=[^A-Za-z0-9])|^)(0x[a-f0-9A-F]+)((?=[^A-Za-z0-9])|$)`
4	SEQ	`((?<=[^A-Za-z0-9])|^)([0-9A-F]{4} ?){4,}((?=[^A-Za-z0-9])|$)`
5	SEQ	`((?<=[^A-Za-z0-9])|^)([0-9a-f]{6,} ?){3,}((?=[^A-Za-z0-9])|$)`
6	IP	`((?<=[^A-Za-z0-9])|^)(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})((?=[^A-Za-z0-9])|$)`
7	ID	`((?<=[^A-Za-z0-9])|^)(([0-9a-f]{2,}:){3,}([0-9a-f]{2,}))((?=[^A-Za-z0-9])|$)`

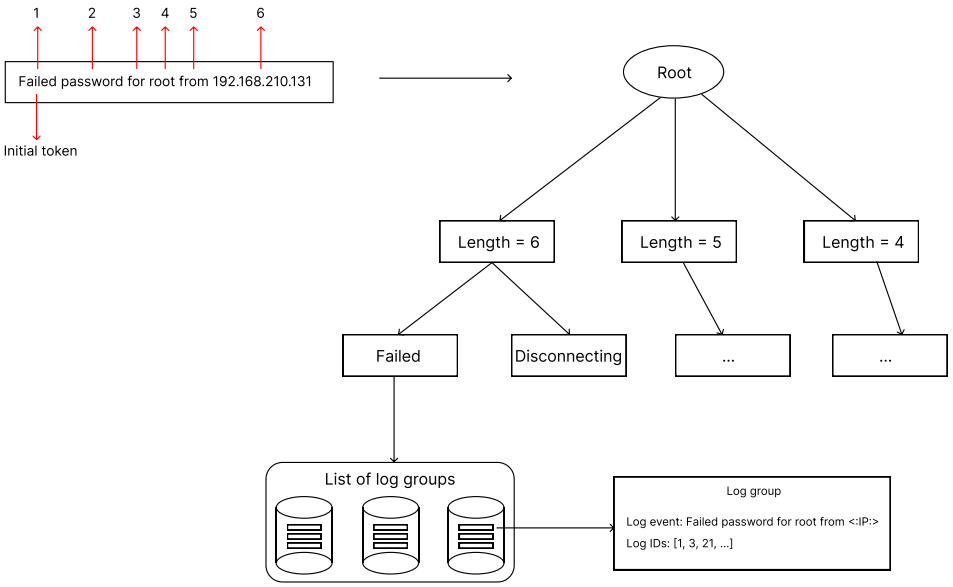

FIGURE 4. Illustration of the Drain method for generating event abstractions.

The first mask, "CMD", is used to hide commands executed within the log. The regex pattern replaces the content enclosed in quotation marks following the phrase "executed cmd" with the label CMD. Next, the "NUM" mask is designed to identify numerical values, both positive and negative, and replace them with the label NUM. The "HEX" mask targets hexadecimal values.

There are also two different patterns, both masked with the label "SEQ", used to identify long hexadecimal sequences. The first pattern, `[0-9A-F]{4} ?){4,}`, is used for detecting sequences written in uppercase letters, while the second pattern `[0-9a-f]{6,} ?){3,}` is for sequences written in lowercase. The "IP" mask is used to detect IP addresses. Finally, the "ID" mask is applied to detect addresses such as MAC addresses or similar identifiers separated by colons [25]. These are replaced with the label ID. All masked labels are enclosed with the prefix "<:" and the suffix ":>". For example, a masked number would be displayed as <:NUM:>.

Drain utilizes a data structure known as a fixed-depth parse tree to expedite the real-time (streaming) parsing of logs. The tree has a fixed depth and is designed to guide the search for log groups. Each node in the tree contains specially designed parsing rules, and each path from the root to a leaf node corresponds to a specific log group. Leaf nodes store a list of relevant log groups [25].

When a new log entry arrives, Drain searches the parse tree to find the most suitable log group, or creates a new group if there is no match. The search process is performed in several steps. First, Drain navigates to internal nodes based on the length of the processed log entries. This procedure assumes that log entries with the same event tend to have the same length. After reaching the node, Drain continues the search to the next leaf node, considering the tokens in the initial position of the log entry, as these tokens are more likely to be a fixed part of the message. If Drain finds a suitable log group, the log entry is matched with the event logs stored in that group. If no matching group is found, Drain creates a

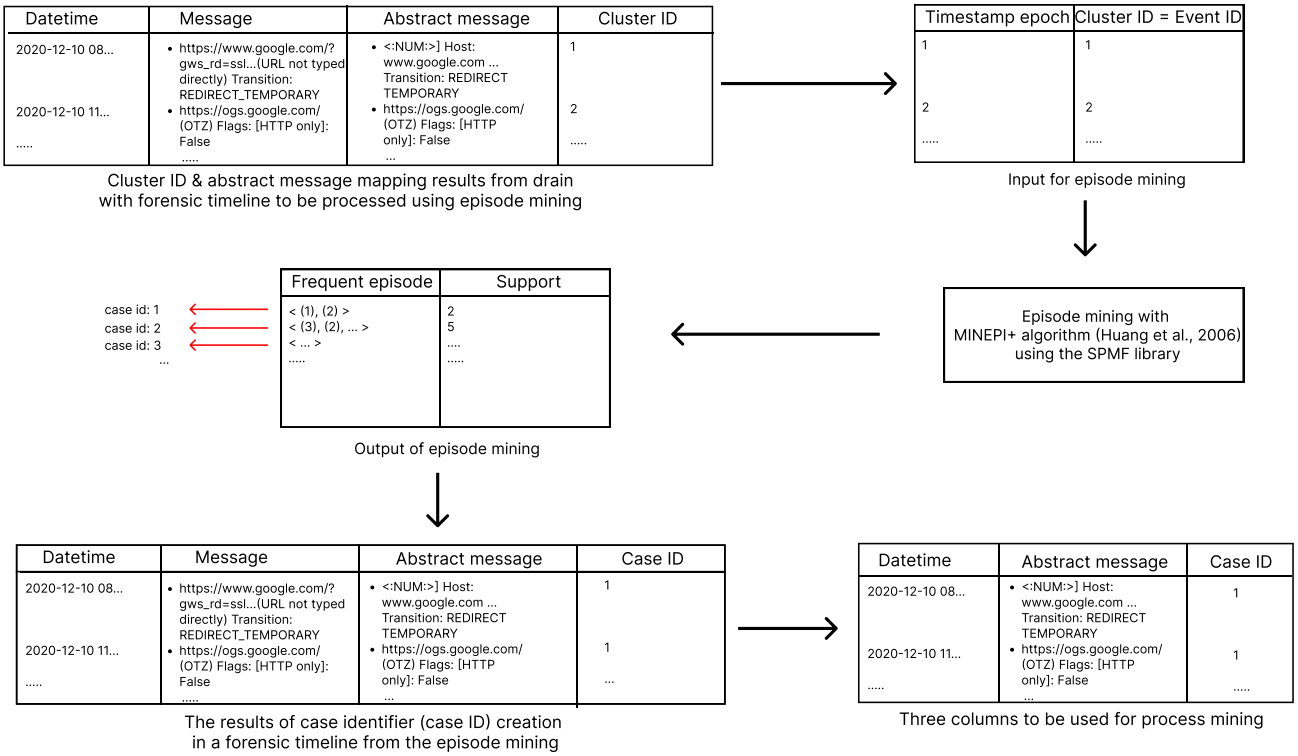

FIGURE 5. Process flow of case identifier creation using episode mining.

new group based on the log entry [25]. An illustration of this process is depicted in Figure 4.

For our prototype, we use Drain3,[8] an extended version of Drain that introduces incremental learning and state persistence, thus making it more suitable for evolving log patterns. Unlike the original Drain, which relies on static log templates, Drain3 continuously updates its model as new log events emerge.

C. EPISODE MINING FOR CASE ID FORMATION

Episode mining is a research area in data mining that focuses on identifying relevant episodes, which are subsequences of events within a longer sequence of events. The most common task in episode mining is frequent episode mining, which aims to identify episodes that frequently occur in the sequence of events. However, this task has been further developed and expanded in various ways. Episode mining is a descriptive data mining technique used to identify meaningful relationships between events in long event sequences [30]. It has also been applied in the security domain, particularly for detecting attack patterns and identifying malicious online activities [31], [32].

We apply the MINEPI+ algorithm to perform episode mining [33]. It was chosen for its ability to efficiently extract frequent episodes from complex sequences while maintaining relatively low memory usage and a competitive execution time compared to alternative approaches. The columns used from the event abstraction results are *timestamp* and *cluster identifier*, where a *cluster identifier* acts as an *event identifier*. The reason for using these two columns is that the timestamp is used to determine the time sequence of the event, while the cluster identifier represents the type of event. By focusing on these two fields, episode mining can detect relevant patterns in the sequence of events. The process of this stage is shown in Figure 5. The *timestamp* column from the event abstraction is encoded as integer values that reflect the temporal order of events before applying episode mining, as the SPMF library[9] only accepts integer data types for timestamps. MINEPI+ is an algorithm to identify sequential event patterns in complex data, extending the MINEPI algorithm, which was originally designed to detect significant event patterns in time-based data [34]. MINEPI detects minimal occurrences of time intervals that contain an episode without smaller subintervals supporting the same pattern, but struggles with complex sequences where multiple events occur simultaneously. To address this, MINEPI+ introduces equal join, which merges simultaneous events, and temporal join, which extends sequential patterns within a defined time window [33]. Furthermore, MINEPI+ adopts a depth-first search for more efficient enumeration, reducing

[8]https://github.com/logpai/Drain3

[9]https://www.philippe-fournier-viger.com/spmf/

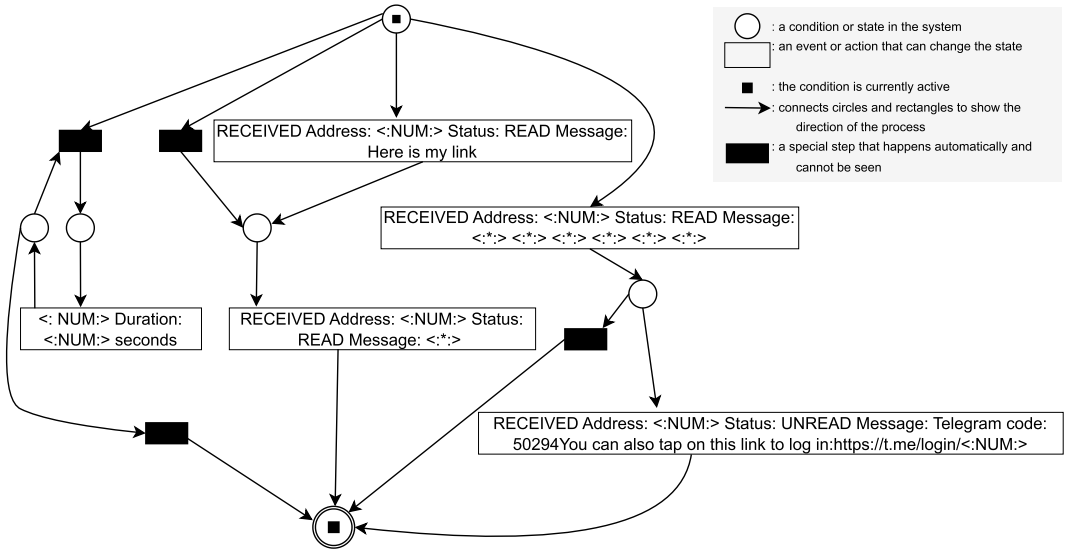

FIGURE 6. Visualization of receiving the telegram code on an android device.

memory usage compared to MINEPI's breadth-first search approach.

We use the SPMF library (Footnote 9) for episode mining. The library is Java-based but may be accessed via Python. Important parameters that must be provided are `winlen` (window length) and `minsup` (minimum support). `Winlen` defines the time limit used to detect episodes in the event log sequence. It is measured in discrete units (time points) and determines how long a pattern can appear before being considered an episode [30]. An episode is defined as an ordered sequence of one or more sets of events (simultaneous event sets), where each set contains events that occur at the same time. The sets themselves are ordered chronologically within the sequence [35].

For example, if `winlen` is set to 10 seconds, only events occurring within that 10-second window will be considered as part of the episode. Events occurring outside this time frame will not be included in the same episode. `minsup` (minimum support) is a threshold that defines the minimum frequency required for an episode to be considered significant in the event log. It is measured based on the number of times an episode appears within a given time window. The support of an episode refers to its actual occurrence count in the dataset. If the support of an episode meets or exceeds the `minsup` threshold, the episode is considered frequent. Otherwise, it is filtered out as insignificant [30].

In this context, a *sequential event pattern* refers to a series of events that occur in a specific order within a user-defined time limit. The algorithm explores possible patterns using a depth-first approach and applies event combination techniques based on time to determine the frequency of pattern occurrences in the data. By employing the depth-first approach, MINEPI+ optimizes memory usage and processing efficiency. Furthermore, the algorithm can analyze complex data, including scenarios in which multiple events occur simultaneously [33].

D. PROCESS DISCOVERY

Process discovery is a technique that extracts a process model from event logs or timeline data. It identifies the sequence of activities and their dependencies without prior knowledge. This facilitates the visualization of workflows or event sequences and supports a better understanding of the overall process. A sample Petri net generated through process discovery from the forensic timeline is shown in Figure 6. It consists of place (circle) which represents conditions or states, transition (rectangle) which represents events that change states, arc (arrows) which connect places to transitions to show the direction of the process, token (dots inside places) which indicate that the condition is currently active, and silent transition (black rectangle) which are special transitions that happen automatically and cannot be seen. In this stage, process discovery is carried out from the episode mining data to produce a process model from the event data. The inputs are *timestamp* and *abstract message*. The applied algorithms are the heuristics miner and the inductive miner, with the help of the PM4PY library.[10] Before the discovery process begins, the abstract message in the dataset is cleaned of unreadable characters so that it can be visualized with PM4PY. Our prototype applies four process discovery algorithms: the Alpha Miner, the Heuristic Miner, the Inductive Miner, and the Integer Linear Programming Miner. To maintain readability, only the conceptual overviews are presented in this section, while

[10]https://github.com/process-intelligence-solutions/pm4py

mathematical formulations and algorithmic representations are provided in Appendices A–D.

The *Alpha Miner* is a process mining algorithm designed to discover causal relationships between activities from a set of event logs. It analyzes sequential patterns of activities, such as determining whether one activity consistently follows another [36]. Introduced by van der Aalst in 2004, the alpha algorithm is recognized as the earliest process discovery algorithm in process mining [37]. It operates by scanning the event log L to identify specific ordering patterns between activities. For example, if activity a is followed by activity b but b is never followed by a, this indicates a causal dependency from a to b. This dependency is represented in the Petri net by adding a place between a and b [37].

The *Heuristic Miner* algorithm uses dependency graphs, where the model is built by considering the frequency and sequence of events. This algorithm performs the discovery process by analyzing causal dependencies. Specifically, if an activity is always followed by another activity, there is a possibility of a dependency relationship between the two activities [38].

The *Inductive Miner* is a process discovery algorithm based on process trees [23]. A process tree represents a process model in the form of a tree, which illustrates the sequence of activities and the relationships between them. Each node in the process tree corresponds to a specific activity or operation (such as sequence, parallelism, selection, or looping), while the branches indicate how these activities are connected [39]. This algorithm operates by breaking down log data into smaller components based on patterns and then reconstructing them into a complete model that accurately reflects the process or workflow.

The *Integer Linear Programming (ILP) Miner* is a process discovery algorithm that constructs a Petri net from event logs by formulating and solving an ILP problem [40].

E. FORENSIC EVENT RECONSTRUCTION

Conformance checking in process mining compares an event log with a predefined process model to assess whether the execution follows the expected sequence of activities. It identifies deviations such as missing, extra, or incorrectly ordered events, and quantifies the level of compliance. To reconstruct the sequence of forensic events, we apply the alignment technique from conformance checking [37]. Alignment is used to compare the trace (sequence of activities) in the timeline data with the process model generated from process discovery in the previous stage. This technique allows us to determine the extent to which an event in the timeline conforms or does not conform to the defined process model [41], and is available in the `pm4py.algo.conformance.alignments` class. The output of alignments provides a list of traces (activity sequences) from both the log and the model. Since our objective is event reconstruction, we adopt this technique by taking the trace event log produced by the alignment as the reconstructed sequence of events. However, the perspective of the process model can be leveraged for further forensic analysis, particularly in identifying anomalies or deviations from the expected process.

F. EVENT VISUALIZATION

After the process model is acquired or if the conformance checking results have shown good conformity, the next step is to visualize the process model generated from the process discovery stage. This visualization is presented using a Petri net [37]. An example of the Petri net visualization obtained through process discovery from the forensic timeline is shown in Figure 6.

V. EXPERIMENTAL RESULTS AND ANALYSIS

The experiments are designed to evaluate the process mining approach in the context of forensic event reconstruction. Several metrics, such as replay fitness and precision, are used to assess the quality of process models derived from the timeline, which are discussed in the subsequent section. Thereafter, we present three case studies, followed by a discussion of the challenges and limitations encountered during our experiments.

A. EVALUATION METRICS

Once process discovery is complete, the next step is conformance checking, which assesses whether the resulting process model aligns with the forensic timeline. This stage serves as an evaluation of the accuracy and validity of the generated model.

1) REPLAY FITNESS

Replay fitness is a method to measure the accuracy with which a process model can replay the sequence of events recorded in the event log [42]. Fitness indicates whether the model can replay or follow the trail of activities in the forensic timeline. If the model can explain most of the activities in the timeline, then fitness has a high value. This fitness is calculated as the percentage of activity in the timeline that the model can accommodate [41].

Fitness is the ratio between the number of traces (sequences of activities or events) in the process model that can be found or matched with the data in the timeline and the number of traces in the event log used as input data in process mining [43]. Replay fitness can be calculated as follows:

$$\text{Fitness} = \frac{\text{num of captured traces}}{\text{num of traces in event log}} \tag{1}$$

where 'num of captured traces' is the number of traces in the process model that can be found or matched with the data in the event log, and 'num of traces in event log' indicates the total number of traces in the event log that are used as input data in process mining.

2) PRECISION

Precision ensures that the model does not show behavior that is significantly different from the data in the timeline. A model with low precision is an underfitting model, which means that the model allows behavior that is much different from the data in the log [42]. Precision attempts to detect models that are too general or too flexible and penalizes them if they allow undesirable behavior or do not fit the data [43]. It is calculated by

$$\text{Precision} = \frac{\text{num of captured traces}}{\text{num of traces in model}} \tag{2}$$

where 'num of captured traces' is the number of traces (sequences of activities or events) in the process model that can be found or matched with data in the timeline, and 'num of traces in the model' indicates the total number of traces enabled by the model, both those that match the timeline and those that do not.

3) GENERALIZATION

Generalization measures the extent to which the model can capture the general pattern of the timeline. This concept is related to overfitting, which is when the model focuses too much on the details in the timeline, so that it cannot understand other patterns outside the timeline. A good model should be able to generalize, that is, it should have the ability to support or accommodate variations in events that have not appeared in the existing data but may occur in the future [13]. Generalization is calculated using:

$$\text{Generalization} = 1 - \frac{\sum_{i=1}^{\#\text{no}} \sqrt{(\#\text{execution})^{-1}}}{\#\text{no}} \tag{3}$$

If #execution < 100, then the formula:

$$\text{Generalization} = \frac{\#\text{execution}}{\text{trace in event log}} \times 100 \tag{4}$$

where *#no* is the number of operators in the process model that can be used to create process flow variations (activities, branching conditions, or transitions). Subsequently, *#execution* represents the number of operators executed in the trace event log set, and *trace in event log* indicates the number of traces in the timeline.

4) SIMPLICITY

Simplicity indicates how complex or simple a process model is. The simpler the model, the easier it is for people to read and understand it. Simplicity ensures that the model is as simple as possible. This principle is related to Occam's Razor, which suggests that models should avoid unnecessary details or complexity. In process discovery, the resulting models are often too complicated to understand [44]. Simplicity can be calculated by

$$\text{Simplicity} = 1 - \frac{\#da + \#mo}{\#no} \tag{5}$$

where *#da* is the number of duplicate activities, *#mo* is the number of operators executed in the trace event log set, and *#no* is the number of operators in the process model that do not correspond to the actual sequence of events in the event log.

B. RESULTS AND DISCUSSION

We provide an open-source implementation of the proposed method, which can be downloaded from our GitHub repository.[11]

To ensure a consistent and systematic evaluation, the parameter tuning of the episode mining stage was carried out independently for each case study. Multiple `winlen` and `minsup` values were explored to assess their effects on episode generation and the resulting process mining outcomes. The rationale behind the final parameter choices is described within each case study. In general, increasing `winlen` enables the discovery of longer and more comprehensive event sequences. However, it may combine events that are not strictly sequential in the original forensic timeline. In contrast, higher `minsup` values lead to fewer but more reliable frequent episodes. This systematic tuning approach ensured that the selected parameters achieved an appropriate balance between pattern richness and model interpretability in all cases.

Since forensic timelines often contain a large number of events, directly processing the entire dataset may result in unstructured and overly complex process models. To counteract, a time-range filtering approach was applied in each case study, focusing on events of interest. This strategy is consistent with prior findings that range-based filtering can help reduce model complexity and improve the overall clarity and interpretability of the resulting process models [45]. Defining a specific timeframe of interest is likewise a common practice in real-world investigations to appropriately narrow the scope.

1) CASE STUDY 1 - PRIVILEGE ESCALATION ATTACK

For the first case study, we downloaded a publicly available dataset consisting of disk images from a compromised Linux-based web server.[12] The events of interest occurred within the time range of 2019-10-05 13:00:00 to 2019-10-05 13:10:00. This study specifically focuses on reconstructing the privilege escalation attack, analyzing how the attacker gained elevated privileges within the system. Our investigation examines the initial access vector, subsequent system modifications, and persistence mechanisms such as creating new user accounts, installing backdoors, or configuration changes. The findings presented in this paper are reconstructed and discussed based on the selected timeframe of the incident.

We used several values of `winlen`, specifically 5, 10, 20, 30, 40, and 50, to analyze their impact on episode mining and subsequent process mining results. Since `winlen` is a user-defined parameter and no universally optimal method

[11] https://github.com/ridaadila/forensic-event-reconstruction-app
[12] https://www.ashemery.com/dfir.html

TABLE 2. Experimental results from privilege escalation attack scenario (`minsup` = 1).

`winlen`	Frequent episodes	Cases	Algorithm	Fitness	Precision	Generalization	Simplicity
5	187	7	Alpha miner	0.276	1.000	0.100	0.714
	187	7	Heuristics miner	0.931	1.000	0.109	0.778
	187	7	Inductive miner	1.000	0.852	0.156	0.760
	187	7	ILP miner	1.000	0.774	0.158	0.714
10	187	7	Alpha miner	0.276	1.000	0.100	0.714
	187	7	Heuristics miner	0.931	1.000	0.109	0.778
	187	7	Inductive miner	1.000	0.852	0.156	0.760
	187	7	ILP miner	1.000	0.774	0.158	0.714
20	280	6	Alpha miner	0.071	1.000	0.100	0.750
	280	6	Heuristics miner	0.929	1.000	0.115	0.810
	280	6	Inductive miner	1.000	0.736	0.167	0.733
	280	6	ILP miner	1.000	0.779	0.154	0.756
30	280	6	Alpha miner	0.071	1.000	0.100	0.750
	280	6	Heuristics miner	0.929	1.000	0.115	0.810
	280	6	Inductive miner	1.000	0.736	0.167	0.733
	280	6	ILP miner	1.000	0.779	0.154	0.756
40	311	6	Alpha miner	-	-	-	-
	311	6	Heuristics miner	1.000	0.984	0.099	0.760
	311	6	Inductive	1.000	0.840	0.146	0.769
	311	6	ILP miner	1.000	0.778	0.154	0.471
50	407	5	Alpha miner	-	-	-	-
	407	5	Heuristics miner	**1.000**	**0.981**	**0.104**	**0.787**
	407	5	Inductive miner	1.000	0.714	0.125	0.724
	407	5	ILP miner	1.000	0.778	0.150	0.559

TABLE 3. Sample results of one alignment trace from a privilege escalation attack scenario using the heuristics miner.

No	Reconstructed events
1	[chsh pid: <:NUM:>] changed user 'mail' shell to '/bin/bash'
2	[CRON pid: <:NUM:>] pam unix(cron:session): session closed for user <:*:>
3	[CRON pid: <:NUM:>] pam unix(cron:session): session opened for user <:*:> by (uid=<:NUM:>)
4	[-x /usr/lib/php5/maxlifetime] && [-x /usr/lib/php5/sessionclean] && [-d /var/lib/php5] && /usr/lib/php5/sessionclean /var/lib/php5 $(/usr/lib/php5/maxlifetime) for user: root pid: <:NUM:>
5	[chpasswd pid: <:NUM:>] pam smb-pass(chpasswd:chauthtok): Failed to find entry for user mail
6	[chpasswd pid: <:NUM:>] pam unix(chpasswd:chauthtok): password changed for mail
7	[usermod pid: <:NUM:>] add <:*:> to group 'sudo'
8	[usermod pid: <:NUM:>] add <:*:> to shadow group 'sudo'

exists for determining its value, the choice of setting can significantly affect the patterns discovered [30]. Prior research suggests that testing multiple values is necessary to identify the most suitable setting for a given context [46]. Therefore, we conducted experiments using each value and assessed the resulting event segmentation with four process mining algorithms: Alpha Miner, Heuristics Miner, Inductive Miner, and ILP Miner. The quality of the resulting process models was then evaluated using process mining metrics (see Section V-A).

Episode mining lacks a standardized method for selecting an optimal `minsup` value, thus requiring heuristic adjustments [35]. We performed iterative trials with `minsup` values greater than or equal to 1 to assess the quality of the resulting patterns. Empirical analysis showed that thresholds greater than 1 tended to produce episodes consisting only of single-event sets without sequential ordering, thereby reducing their informational value. In the end, a `minsup` value of 1 was empirically validated as the most suitable configuration for this case study, as it preserves multi-event sequential patterns for subsequent process mining stages.

After conducting the episode mining process with a `minsup` value of 1 and various `winlen` configurations, as shown in Table 2, the resulting frequent episodes were used to construct case IDs in the forensic timeline. The best results were obtained using the Heuristics Miner with a `winlen` configuration of 50, as indicated by high metric values: fitness (1.000), precision (0.981), generalization (0.104), and simplicity (0.787). The Heuristics Miner employs a frequency-based statistical approach that considers the relative co-occurrence of activities, rather than relying solely on strict sequential relations. This makes it more robust in identifying process behaviors that include parallelism, loops, and noise. In this case study, the Heuristics Miner achieved the best evaluation performance due to the relatively low complexity of the dataset, evidenced by the limited number of frequent episodes generated (only a few hundred). With fewer frequent episodes, the transformation into case identifiers is straightforward and yields clearer traces, which suits the strengths of the Heuristics Miner. These observations are consistent with findings from previous benchmark studies, which show that Heuristics Miner tends to generate clearer and more stable models than other algorithms, particularly when the underlying event log is relatively structured [47].

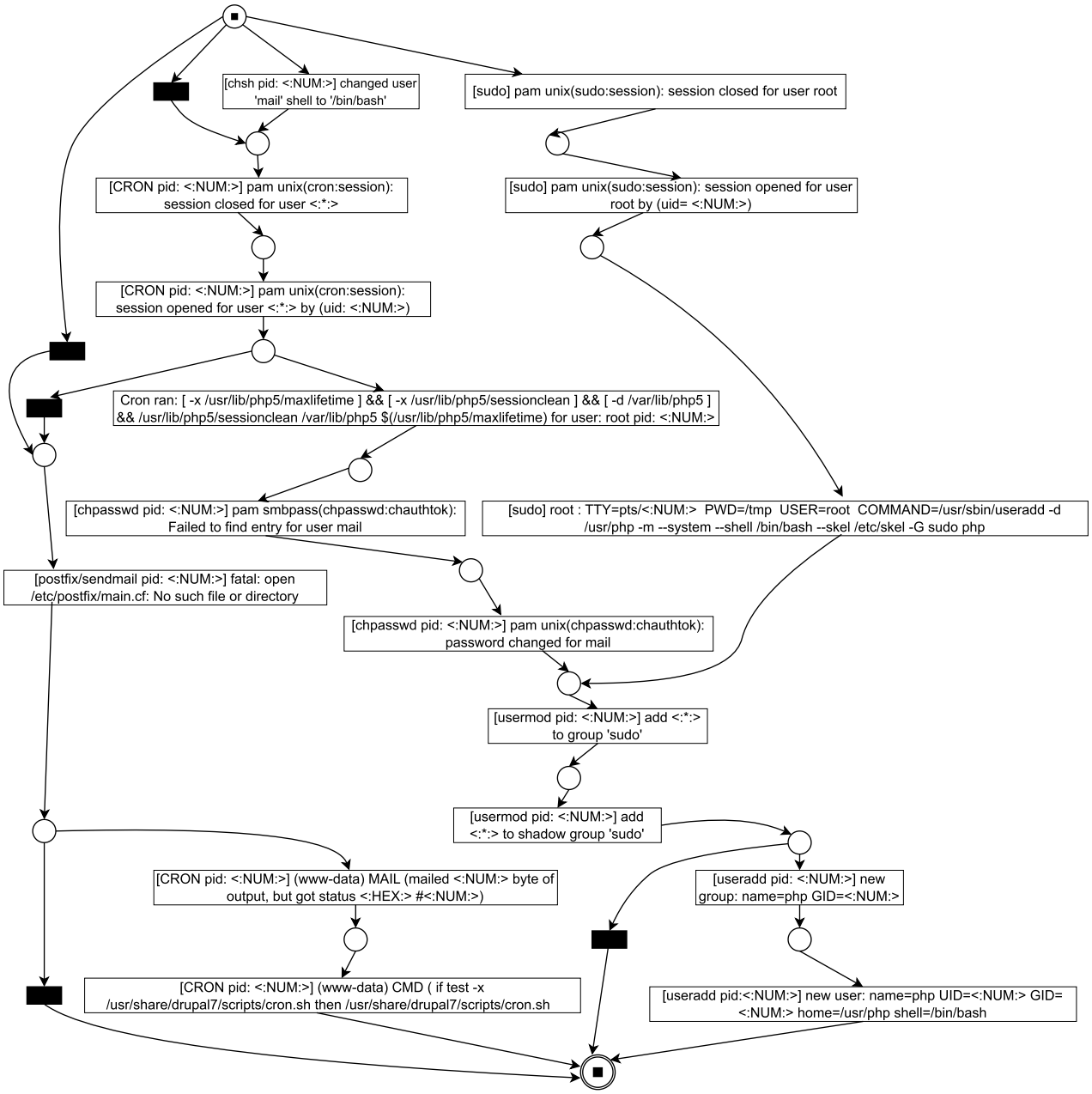

FIGURE 7. **Visualization of privilege escalation attack scenario using Heuristics miner with `minsup` = 1 `winlen` = 50.**

Fitness is the primary metric, as it measures how well the model can replay or reconstruct the sequences of events recorded in the log. A high fitness value indicates that most or all event sequences in the log can be reconstructed. Precision is the second priority, as it evaluates the extent to which the model restricts itself only to sequences that actually occurred. In this context, a high precision value is desirable to prevent the model from allowing non-existent event sequences, which could lead to misinterpretation. While precision is generally associated with avoiding overfitting, in forensic event reconstruction, overfitting is not a concern, since the goal is to replicate the log sequences for further investigation.

Although generalization and simplicity are also considered, their roles differ in this context. Generalization measures the ability to handle unseen data, but here the focus is on reconstructing events already present in the log rather than predicting future events. Simplicity ensures the model remains interpretable for stakeholders, such as forensic investigators, without compromising its representational capability. However, an overly simplistic model may fail to represent the actual event sequences in the log.

Conversely, the worst results were obtained using the Alpha Miner with the `winlen` of 40 and 50, where all evaluation metrics were unavailable (marked with the

"-" symbol). When the evaluation process was conducted using the PM4PY library in Python, the system returned the following error message: "Exception: trying to apply alignments on a Petri net that is not an easy sound net." This message indicates that the alpha miner model does not form a sound workflow net, and it is impossible to evaluate using alignment-based metrics. This highlights the structural limitations of the resulting model. As a result, the analysis could not continue for these configurations, and the affected models were excluded from quantitative evaluation. Instead, they were discussed qualitatively to highlight the structural limitations of the Alpha Miner when applied with higher `winlen` values.

The error seems to occur because increasing the `winlen` parameter changes how events are grouped into cases. With a larger `winlen`, the episode mining process captures more events within each frequent episode, which in turn produces fewer cases. As a result, each case contains a broader range of events that are not necessarily sequential in the original forensic timeline, leading to inconsistent directly-follows relations and causing Alpha Miner to generate a less sound process model.

In particular, with `winlen` of 20 and 30, Alpha Miner was still able to produce valid models, although the number of cases (6) was the same as in `winlen` of 40. This suggests that failure at higher `winlen` values is not caused by the number of cases, but by the increasing variation and concurrency within each case. As a result, the discovered Petri nets exhibit structural inconsistencies such as deadlocks or disconnected transitions, rendering the models unsound.

The larger `winlen` configurations do not necessarily degrade performance for all algorithms. For example, the Heuristics Miner achieved its best results with `winlen` of 50, showing that algorithms designed to handle concurrent and noisy events are more capable of utilizing the richer event information produced by larger `winlen`.

To provide an overview of computational efficiency, the runtime of each process discovery algorithm was measured using `winlen` of 5. The Alpha Miner demonstrated the fastest execution time (0.046 seconds), followed by the Inductive Miner (0.076 seconds) and the Heuristics Miner (0.104 seconds). The ILP Miner was the slowest, requiring 1.169 seconds. This difference reflects the underlying algorithmic complexity. While Alpha, Heuristics, and Inductive Miners are computationally efficient, the ILP Miner relies on integer linear programming, which results in higher computational cost.

Figure 7 visualizes this case study using the best experimental configuration, obtained with the heuristics miner at a `minsup` of 1 and a `winlen` of 50. The resulting process model provides a holistic view of the activity flow in the forensic data, which is useful for understanding the overall structure of the incident and identifying potential areas for deviation. Although the model captures the general behavior across multiple traces, a more detailed reconstruction requires examining the specific sequence of events as they were recorded. For this purpose, the conformance-checking alignment process generated five alignment traces, aligning the event log with the process model. However, in the context of event reconstruction, we adopted the event log perspective from these alignments, as it directly reflects the chronological order observed in the forensic data. One such alignment trace, representing the reconstructed sequence of events, is shown in Table 3, which provides a detailed illustration of the typical sequence of activities in the scenario of a privilege-escalation attack.

The reconstructed events in Table 3 illustrate a typical sequence observed during a privilege escalation attack. The trace begins with a change in the shell for the user mail to `/bin/bash`, enabling interactive shell access. This is immediately followed by a record indicating that a cron job initiated a session for the mail user. Subsequently, the cron service opens a session for the root user to execute scheduled tasks, involving PHP session cleanup scripts located in `/usr/lib/php5/sessionclean`. The next step shows an attempt by the mail user to run the `smbpasswd` utility, which fails due to the absence of an entry for the user in the Samba password database. Following this, the mail user's password is changed using `passwd`. The attack concludes with the mail user being added to both the sudo group and the shadow group sudo, granting administrative privileges and enabling the execution of commands with elevated rights.

In summary, the reconstructed events from the alignment trace provide a picture of how the attack unfolded. The sequence of actions, such as changing the user's shell, starting a cron-based session, modifying the password, and adding the user to administrative groups, shows a deliberate and structured process of privilege escalation. These events can be verified through related system artefacts, such as `/etc/passwd` and `/etc/shadow` for shell and password changes, crontab entries or files under `/var/spool/cron` for scheduled tasks, the PHP script path in the trace for signs of execution, samba password databases for the failed smbpasswd attempt, and `/etc/group` or sudo logs for group additions and privilege use. With high fitness and precision scores, the reconstructed events serve as a reliable representation of the activities recorded in the logs, although their interpretation should still be validated using supporting artefacts and other relevant systems.

2) CASE STUDY 2 - BRUTE FORCE LOGIN ATTEMPT

The second case study utilizes the same dataset but focuses on the time range 2019-10-05 12:52:00 to 2019-10-05 13:00:00, during which a brute-force login attempt incident occurred. Again, we tested several `winlen` (20, 30, and 40) similar to the first case study. Since `winlen` is a user-defined parameter and there is currently no established method to determine its optimal value, we adopted an empirical approach through experimentation. In preliminary tests, we found that using values smaller than 20 resulted in episodes that contained too few event sequences, which limited their informativeness. Therefore, we began our

TABLE 4. Experimental results from brute force login attempt scenario.

minsup	winlen	Frequent episodes	Cases	Algorithm	Fitness	Precision	Generalization	Simplicity
8	20	621	63	Alpha miner	0.792	1.000	0.498	0.419
		621	63	Heuristics miner	0.390	0.952	0.567	0.647
		621	63	Inductive miner	1.000	0.962	0.603	0.600
		621	63	ILP miner	1.000	0.783	0.555	0.351
	30	4595	60	Alpha miner	0.769	1.000	0.498	0.419
		4595	60	Heuristics miner	0.449	0.878	0.568	0.600
		4595	60	Inductive miner	1.000	0.951	0.604	0.600
		4595	60	ILP miner	1.000	0.762	0.555	0.351
	40	4595	60	Alpha miner	0.769	1.000	0.498	0.419
		4595	60	Heuristics miner	0.449	0.878	0.568	0.600
		4595	60	Inductive miner	1.000	0.951	0.604	0.600
		4595	60	ILP miner	1.000	0.762	0.555	0.351
9	20	183	62	Alpha miner	0.785	1.000	0.498	0.419
		183	62	Heuristics miner	0.411	0.956	0.566	0.647
		183	62	Inductive miner	1.000	0.961	0.604	0.589
		183	62	ILP miner	1.000	0.772	0.555	0.351
	30	825	61	Alpha miner	0.777	1.000	0.498	0.419
		825	61	Heuristics miner	0.401	0.963	0.600	0.647
		825	61	Inductive miner	**1.000**	**0.963**	**0.592**	**0.589**
		825	61	ILP miner	1.000	0.768	0.555	0.351
	40	825	61	Alpha miner	0.777	1.000	0.498	0.419
		825	61	Heuristics miner	0.401	0.963	0.600	0.647
		825	61	Inductive miner	**1.000**	**0.963**	**0.592**	**0.589**
		825	61	ILP miner	1.000	0.768	0.555	0.351

TABLE 5. Sample of one alignment trace from brute force login attempt scenario.

No	Reconstructed events
1	[sshd pid: <:NUM:>] pam unix(sshd:auth): authentication failure; logname= uid=:NUM:> euid=:NUM:> tty=ssh ruser= rhost=:IP:> user=root
2	[sshd pid: <:NUM:>] Connection closed by :IP:> [preauth]
3	[sshd pid: <:NUM:>] Failed password for root from :IP:> port :NUM:> ssh2
4	[sshd pid: <:NUM:>] Connection closed by :IP:> [preauth]
5	[sshd pid: <:NUM:>] Failed password for root from :IP:> port :NUM:> ssh2

experiments with `winlen` of 20 and explored larger values to evaluate their impact on episode mining as well as the quality of the resulting process models.

Values of 8 and 9 were used for `minsup`, selected empirically because no standard method exists for determining the optimal value (as in Case Study 1). During our experiments, we observed that using `minsup` values below 8, especially when combined with larger `winlen` values (greater than 20), led to a significant increase in the number of candidate episodes, resulting in substantially longer processing times. This made the analysis inefficient and impractical. Consequently, `minsup` values of 8 and 9 balance computational feasibility with effective pattern discovery. Experimental results, as shown in Table 4, indicate that the Inductive Miner with a `minsup` of 9 and a `winlen` of 30 or 40 delivers the best performance across almost all evaluation metrics, namely fitness (1.000), precision (0.963), generalization (0.592), and simplicity (0.589), with consistent results. Inductive Miner builds process models by breaking the event log into smaller parts based on activity patterns, then combining them into a complete and correct model. This makes it reliable, especially when handling noisy or incomplete data. These observations are consistent with findings from prior benchmark studies, which report that the Inductive Miner generally achieves high fitness and precision while maintaining robustness to noise and log incompleteness [48].

Figure 8 illustrates the sequence of events in a brute-force login attempt targeting the SSH service, generated using the Inductive Miner algorithm with parameters `minsup` of 9 and `winlen` of 30. The resulting visualization reveals multiple traces or paths, each representing a different sequence of events that occurred during the attack process. A total of 61 alignment traces were generated through conformance checking, where each trace aligns the sequence of events recorded in the log with those in the process model. In the context of this event reconstruction, however, only the sequences derived from the event log were used.

For illustrative purposes, only one representative trace is presented in Table 5 to demonstrate the typical pattern observed during a brute force login attempt scenario. Table 5 presents a reconstructed sequence of events representing a brute force login attempt on an SSH service. The trace begins with an authentication failure for the root account, where the SSH daemon logs details such as the process ID, login name, user ID, and the originating IP address. This is followed by a connection closure initiated by the remote host before completing the authentication process. Subsequent entries repeat the pattern of failed password attempts for the root

TABLE 6. Experimental results from keylogger activity.

minsup	winlen	Frequent episodes	Cases	Algorithm	Fitness	Precision	Generalization	Simplicity
1	20	4113	21	Alpha miner	0.048	1.000	0.314	0.405
		4113	21	Heuristics miner	1.000	0.974	0.193	0.623
		4113	21	Inductive miner	1.000	0.845	0.384	0.618
		4113	21	ILP miner	1.000	0.765	0.361	0.105
	30	20493	19	Alpha miner	0.053	1.000	0.314	0.391
		20493	19	Heuristics miner	1.000	0.966	0.180	0.628
		20493	19	Inductive miner	1.000	0.816	0.429	0.636
		20493	19	ILP miner	1.000	0.749	0.359	0.081
	40	20493	19	Alpha miner	0.053	1.000	0.314	0.391
		20493	19	Heuristics miner	1.000	0.966	0.180	0.628
		20493	19	Inductive miner	1.000	0.816	0.429	0.636
		20493	19	ILP miner	1.000	0.749	0.359	0.081
2	20	2049	19	Alpha miner	0.064	0.783	0.314	0.415
		2049	19	Heuristics miner	0.947	0.968	0.178	0.636
		2049	19	Inductive miner	**1.000**	**0.913**	**0.378**	**0.644**
		2049	19	ILP miner	1.000	0.808	0.359	0.091
	30	2049	19	Alpha miner	0.064	0.783	0.314	0.415
		2049	19	Heuristics miner	0.947	0.968	0.178	0.636
		2049	19	Inductive miner	**1.000**	**0.913**	**0.378**	**0.644**
		2049	19	ILP miner	1.000	0.808	0.359	0.091
	40	4096	19	Alpha miner	0.098	0.738	0.314	0.402
		4096	19	Heuristics miner	0.947	0.966	0.184	0.636
		4096	19	Inductive miner	1.000	0.896	0.411	0.655
		4096	19	ILP miner	1.000	0.811	0.359	0.091

user, specifying the port used, interspersed with connection closures. This repetitive sequence reflects the characteristic behavior of automated brute-force attacks, where multiple login attempts are executed rapidly to gain unauthorized access.

In summary, the reconstructed events from the alignment trace capture the pattern of a brute-force login attempt targeting the SSH service. The repeated authentication failures, followed by rapid connection closures, point to the use of an automated script that cycles through multiple password guesses. From a forensic perspective, these events can be confirmed by reviewing artefacts such as `/var/log/auth.log`, `/var/log/secure`, or intrusion detection logs that record repeated SSH connection attempts from the same IP. The close timing of these attempts also helps determine the attack window and distinguish automated behavior from normal user activity. With high fitness and precision values, the reconstructed events provide a reliable basis for interpreting the attack activity and correlating it with the supporting system.

3) CASE STUDY 3 - KEYLOGGER ACTIVITY

Lastly, a disk image of a Linux-based operating system was downloaded from NIST's Computer Forensic Reference Data Sets (CFReDS. The dataset was filtered based on the events that occurred within the time frame of 2023-05-04 21:19:20 to 2023-05-04 21:31:02, which corresponds to the incident of a keylogger[13] activity executed via a cron job.

[13]A keylogger is software or hardware used to record every keystroke made by a user on a computer keyboard or other digital devices, typically without the user's knowledge.

Again, an iterative process was used to identify `minsup`. Based on the experiments, the `minsup` values chosen and presented in Table 6 are 1 and 2. This decision was made because, when tested with a `minsup` of 3, the results were similar to those obtained with a value of 2. Meanwhile, at values of 4 and above, the number of frequent episodes generated was small, resulting in only a single case.

For `winlen`, 5, 10, 20, 30, and 40 were tested. Based on initial experiments, the values 5 and 10 produced evaluation metrics similar to those of 20, while values above 40 produced evaluation results that were similar to those obtained with `winlen` of 30. For efficiency, only the `winlen` values of 20, 30, and 40 are presented in Table 6.

The best results were achieved by the Inductive Miner with `minsup` of 2 and `winlen` of 20 and 30. These configurations produced the same values across all evaluation metrics: fitness (1.000), precision (0.913), generalization (0.378), and simplicity (0.644). Although `minsup` of 2 and `winlen` of 40 resulted in the same number of cases as `winlen` of 20 and 30, this did not guarantee similar evaluation results. The difference occurred because the number of frequent episodes generated at `winlen` of 40 was 4096, while at `winlen` of 20 and 30, it was only 2049. Since the frequent episodes were different, the cases formed from them also varied, which led to differences in the evaluation metrics.

Inductive Miner's good results are likely because it is designed to discover structured, sound process models that closely fit the behavior recorded in the event log. Its ability to achieve perfect fitness while maintaining high precision and reasonable generalization makes it highly effective for

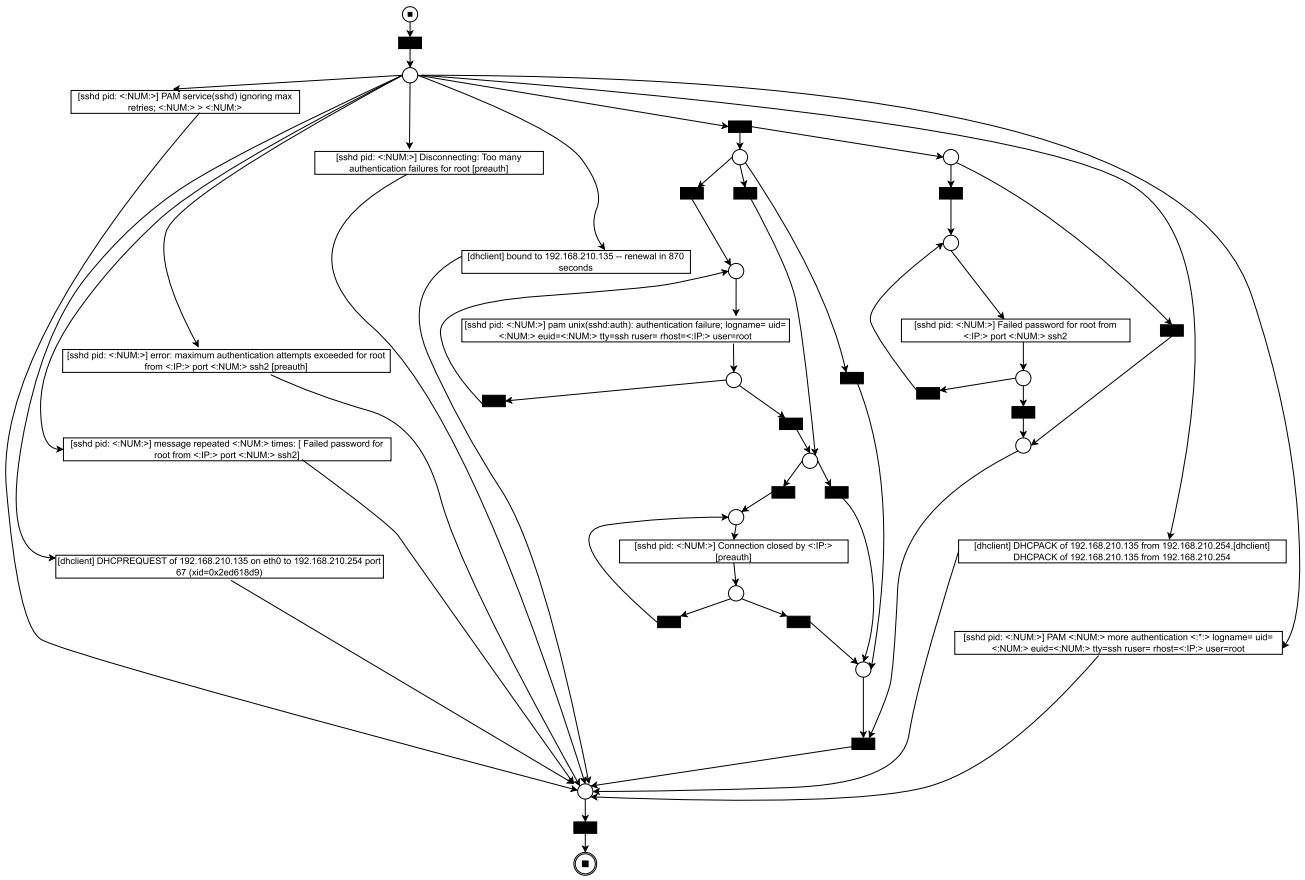

FIGURE 8. **Visualization of brute force login attempt scenario using inductive miner with `minsup` = 9 `winlen` = 30.**

this kind of data. In addition, the simplicity of the resulting models enhances their interpretability without sacrificing accuracy. These findings are consistent with prior work showing that the Inductive Miner produces sound process models by construction, as it derives a process tree whose structure reflects the actual behavior recorded in the event log [49].

Figure 9 presents a visualization using the best experimental result, i.e., Inductive Miner, `minsup` of 2, and `winlen` of 5. A total of 19 traces were produced from the alignment traces of the Inductive Miner algorithm, where each trace represents a sequence of events contained in the resulting process model. This section presents two example alignment traces, namely the 16th trace in Table 7 and the 7th trace in Table 8.

Table 7 shows two consecutive events originating from the cron command executed by the root user, specifically to run a keylogger using nohup so that the process continues to run in the background even after the session is terminated. The executed command was `python3 keylogger.py` from the `/root/Keylogger/linux/` directory, indicating activity related to monitoring user keyboard input. Subsequently, there are two events from the postfix service, involving the cleanup and qmgr modules, which suggest email or message processing and removal activities within the system. This sequence of events is suspected to indicate that the system has been scheduled to execute keylogger malware via cron automatically.

Furthermore, Table 8 presents a sample alignment trace from the keylogger activity scenario (trace-7), consisting of a sequence of nine reconstructed events. The first three events originate from the ntpd (Network Time Protocol Daemon) service, indicating that the system is synchronizing its local time with an external time server. These entries, including ‘‘freq mode’’ and ‘‘clock step’’ suggest adjustments to the system clock, typically occurring during startup or when a significant time discrepancy is detected. This is followed by a log from systemd indicating that the system time has been changed, which confirms the time synchronization process. Subsequently, another ntpd event appears, showing the message ‘‘no sys peer’’ meaning that no suitable server could be found for time synchronization, possibly due to a network or configuration issue. Events 6 to 8 represent interactions with crontab, specifically an edit operation performed by the root user. This strongly suggests the scheduling of a task, likely to automate the execution of a keylogger script.

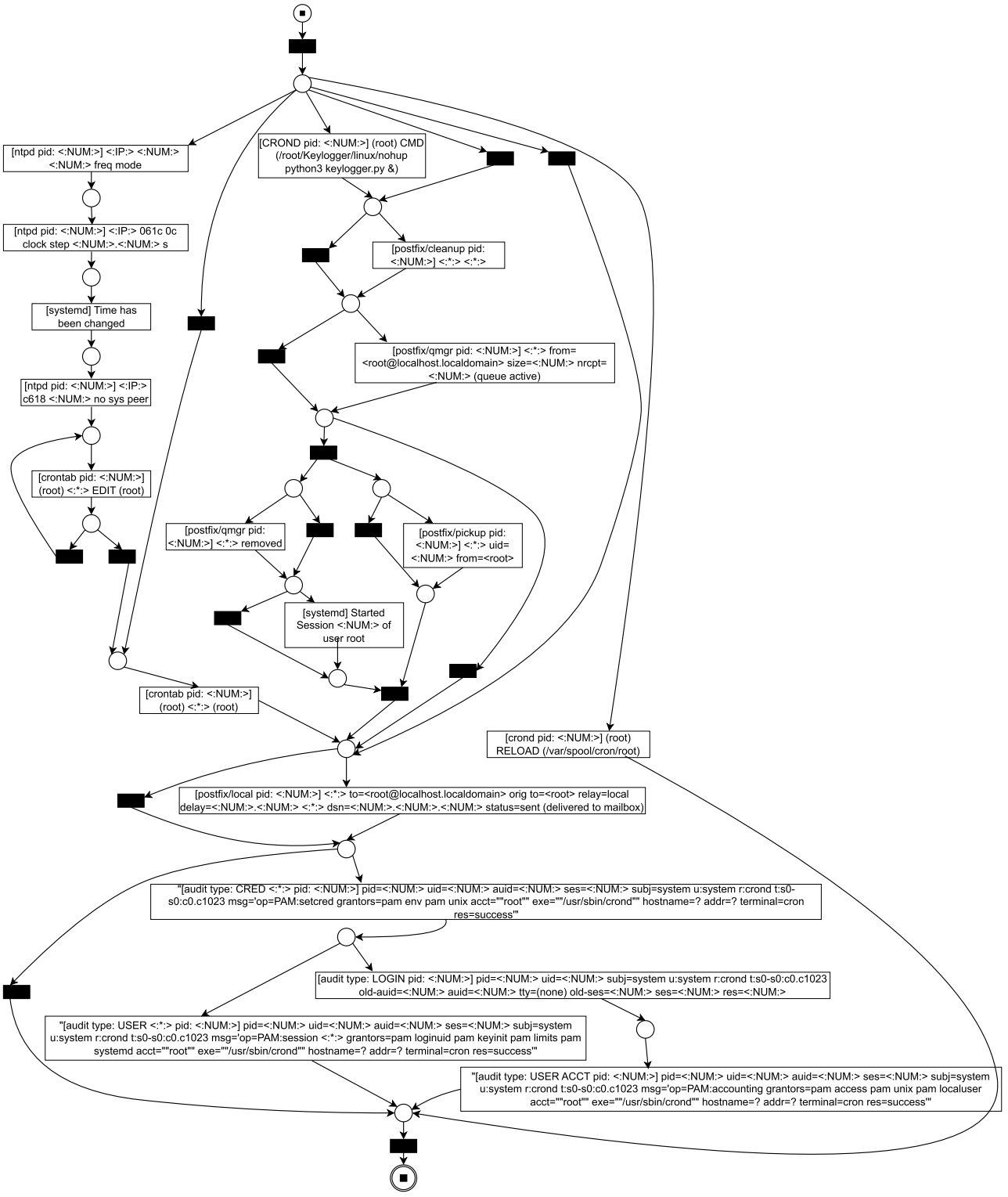

FIGURE 9. **Visualization of keylogger activity scenario using inductive miner with `minsup` = 2 `winlen` = 20.**

The final event shows a message from the postfix service, indicating that a local email has been sent to the root user. This could imply that the system was configured to automatically send the results or logs of the keylogger activity via email. Together, these events may indicate a sequence of malicious or unauthorized behavior where a keylogger is scheduled to run periodically and potentially exfiltrates data through the local mail system.

TABLE 7. Sample of one alignment trace from keylogger activity scenario (trace-16).

No	Reconstructed events
1	[CROND pid: <:NUM:>] (root) CMD (/root/Keylogger/linux/nohup python3 keylogger.py &)
2	[CROND pid: <:NUM:>] (root) CMD (/root/Keylogger/linux/nohup python3 keylogger.py &)
3	[postfix/cleanup pid: <:NUM:>] <:*:> <:*:>
4	[postfix/qmgr pid: <:NUM:>] <:*:> removed

TABLE 8. Sample of one alignment trace from keylogger activity scenario (trace-7).

No	Reconstructed events
1	[ntpd pid: <:NUM:>] <:IP:> <:NUM:> <:NUM:> freq mode
2	[ntpd pid: <:NUM:>] <:IP:> <:NUM:> <:NUM:> freq mode
3	[ntpd pid: <:NUM:>] <:IP:> 061c 0c clock step <:NUM:> <:NUM:> s
4	[systemd] Time has been changed
5	[ntpd pid: <:NUM:>] <:IP:> c618 <:NUM:> no sys peer
6	[crontab pid: <:NUM:>] (root) <:*:> EDIT (root)
7	[crontab pid: <:NUM:>] (root) <:*:> (root)
8	[crontab pid: <:NUM:>] (root) <:*:> (root)
9	[postfix/local pid: <:NUM:>] <:*:> to <root@localhost.localdomain> orig to=<root> relay=local delay=<:NUM:>.<:NUM:> <:*:> dsn=<:NUM:>.<:NUM:>.<:NUM:> status=sent (delivered to mailbox)

In summary, the reconstructed events from the alignment traces reveal how the attacker maintained persistence on the compromised system by automating the execution of a keylogger. The sequence of actions shows a systematic process, including editing the crontab, launching the keylogger script, and using the postfix service to send local emails, which together suggest an attempt to collect and possibly exfiltrate sensitive data. These activities can be verified through related artefacts such as `/var/spool/cron`, `/etc/cron*` files, and mail logs under `/var/log/maillog` or `/var/log/mail.log`, as well as by inspecting the presence and modification times of the `keylogger.py` script. With high fitness and precision scores, the reconstructed events provide a reliable reconstruction of the actual sequence of events, supporting further forensic validation using system evidence.

C. CHALLENGES AND LIMITATIONS

During the experimental phase, several challenges and limitations were identified.

An issue arises in the episode mining stage, where some frequent episodes contain only a few events or show limited variation. When this happens, the resulting cases tend to capture less meaningful behavioral information, which can affect the quality and interpretability of the process models. This limitation may be mitigated by adjusting `winlen` and `minsup` to explore a broader range of configurations, allowing for richer and more representative frequent episode sets. In this work, the MINEPI+ algorithm was selected for episode mining because previous studies have shown that it can efficiently extract frequent episodes from complex event sequences while keeping memory usage and execution time relatively low compared to alternative methods. Future work could explore other episode mining algorithms and continue systematic parameter tuning to find configurations that better fit different datasets.

Another challenge was encountered in the event abstraction stage using the Drain method. In some cases, the clustering process produced highly similar event groups, making it harder to distinguish between abstracted event types. In this study, Drain was applied with its default parameters. When such similarity occurs, adjusting these parameters may help improve the distinctiveness of the clusters.

The proposed approach still relies on manual parameter tuning, particularly when defining appropriate values of `winlen` and `minsup` in the episode mining step. Since there is no established method for determining these parameters automatically, empirical testing remains essential to achieve a reasonable balance between pattern richness and interpretability. The experiments also showed that changes in these parameters could noticeably affect the number and structure of the discovered episodes, indicating that the method is sensitive to parameter settings. Consequently, careful tuning is important to ensure stable and meaningful process models.

VI. CONCLUSION AND FUTURE WORK

This study proposed a novel method for reconstructing forensic events using process mining. It comprises key steps such as building the forensic timeline, creating event abstractions, episode mining to identify case identifiers, process discovery, and forensic event reconstruction. By automatically deriving case identifiers and using them as the basis for process mining, our approach reduces the amount of manual preprocessing typically required by investigators. The open-source and reproducible workflow developed in this work demonstrates the practical potential of this method and provides a foundation for further research in forensic event reconstruction. The experimental results demonstrated the performance of our proposed event reconstruction technique in terms of replay fitness and precision. We also demonstrate its applicability to forensic scenarios. The case studies further validated the capability of the proposed method to reconstruct high-level forensic events and bridge the gap between raw timeline data and actionable insights for investigators.

Future work should focus on scalability improvements, such as optimizing the framework to handle larger and more complex datasets efficiently, particularly in scenarios involving distributed systems or cloud environments. Automated event abstraction can be enhanced by using machine learning techniques to reduce the dependency on manual input. Furthermore, the integration of emerging technologies

offers exciting opportunities. For example, adapting the method to incorporate advancements in AI-driven anomaly detection could further improve its robustness. Cross-domain applicability is another promising direction, extending the framework to address forensic challenges in domains such as IoT, UAVs, and industrial control systems. Finally, we will compare the proposed method with other forensic reconstruction methods, such as the finite state machine-based reconstruction approach.

Although this study is presented as a proof of concept, the proposed framework has demonstrated stable and reproducible results. This showcased its potential to support practical forensic analysis. Moreover, the availability of an open-source implementation further strengthens its suitability for real-world experimentation and application.

At the same time, several components remain open for future research, particularly the sensitivity of the episode mining stage to parameter settings and the reliance on default configurations in the event abstraction process using Drain. The framework would benefit from automated parameter selection and more adaptive abstraction mechanisms before large-scale deployment.

APPENDIX A
ALPHA MINER

The Alpha Miner defines four types of event ordering relations to describe possible dependencies between activities in an event log [37]:

- *Directly follows* ($a >_L b$): there exists a trace $\sigma = \langle t_1, t_2, \ldots, t_n \rangle \in L$ and an index $i \in \{1, \ldots, n-1\}$ such that $t_i = a$ and $t_{i+1} = b$; in other words, a occurs immediately before b in at least one trace.
- *Causal relation* ($a \rightarrow_L b$): $a >_L b$ holds and $b >_L a$ does not hold, indicating a one-way dependency from a to b.
- *Non-causal relation* ($a \#_L b$): $a >_L b$ does not hold, and $b >_L a$ does not hold, meaning there is no direct succession between a and b in either direction.
- *Parallel relation* ($a \|_L b$): both $a >_L b$ and $b >_L a$ hold, typically indicating concurrency between a and b.

APPENDIX B
HEURISTICS MINER

In the Heuristic Miner algorithm, the frequency of the "directly follows" relation, which represents immediate sequential dependencies between activities, is identified and calculated using (6). This relation indicates a condition in which one activity occurs immediately after another in an event log. From the equation (6), let L be an event log over $\mathcal{A}$ and $a, b \in \mathcal{A}$. $|a >_L b|$ is the number of times a is directly followed by b in L [37].

$$|a >_L b| = \sum_{\sigma \in L} L(\sigma) \times \left| \left\{ 1 \leq i <| \sigma | \; \middle| \; \sigma(i) = a \wedge \sigma(i+1) = b \right\} \right| \tag{6}$$

Then, the Heuristic Miner algorithm calculates the "dependency relation" value using (7) and (8). $|a \rightarrow_L b|$ is the value of the dependency relation between a and b [37].

If $a \neq b$:

$$|a \Rightarrow_L b| = \frac{|a >_l b| - |b >_l a|}{|a >_l b| + |b >_l a| + 1} \tag{7}$$

If $a = b$:

$$|a \Rightarrow_L b| = \frac{|a >_l a|}{|a >_l a| + 1} \tag{8}$$

The measure $|a \Rightarrow_L b|$ produces values in the range from -1 to 1. A value close to 1 indicates a strong positive dependency, meaning that a is often directly followed by b, while b is rarely directly followed by a. Such a value occurs when a frequently appears immediately before b in the sequence. In contrast, a value close to -1 reflects a strong negative dependency, where b is often directly followed by a [37].

APPENDIX C
INDUCTIVE MINER

The Inductive Miner algorithm works by splitting the initial event log into smaller sublogs. The techniques for splitting the initial event log include the exclusive choice cut (9), sequence cut (10), parallel cut (11), and redo loop cut (12) [37].

An *exclusive-choice cut* of the directly-follows graph $G(L)$ is a cut $(\times, A_1, A_2, \ldots, A_n)$ such that [50]:

$$\forall i, j \in \{1, \ldots, n\} \; \forall a \in A_i \; \forall b \in A_j \quad i \neq j \Rightarrow a \not\rightarrow_L^+ b \tag{9}$$

where:

- $A_1, A_2, \ldots, A_n$ form a partition of the set of activities in the log.
- i, j are indices that refer to different subsets A_i and A_j.
- $a \in A_i$ means activity a belongs to the subset A_i.
- $b \in A_j$ means activity b belongs to the subset A_j.
- $a \not\rightarrow_L^+ b$ means that in the directly-follows graph of L, there is no path from activity a to activity b (not directly and not through other activities).
- $\forall$ denotes "for all", and $\Rightarrow$ denotes the logical implication.

This condition states that if two activities belong to different subsets ($i \neq j$), then no activity in one subset is ever directly followed by any activity in the other subset in the event log [50].

A *sequence cut* of the directly-follows graph $G(L)$ is a cut $(\rightarrow, A_1, A_2, \ldots, A_n)$ such that [50]:

$$\forall i, j \in \{1, \ldots, n\} \; \forall a \in A_i \; \forall b \in A_j \quad i < j \Rightarrow (a \mapsto_L^+ b \wedge b \not\mapsto_L^+ a) \tag{10}$$

where:

- $A_1, A_2, \ldots, A_n$ are mutually disjoint subsets of activities in the log.
- i, j are indices that refer to different subsets.
- $i < j$ enforces a strict ordering between subsets.
- $a \in A_i$ means activity a belongs to the subset A_i.

- $b \in A_j$ means activity b belongs to the subset A_j.
- $a \rightarrow_L^+ b$ means there exists a path of one or more directly-follows relations from activity a to activity b in the log (transitive closure of the directly-follows relation).
- $b \not\rightarrow_L^+ a$ means there is no path from b back to a in the transitive closure of the directly-follows relation.
- $\forall$ denotes "for all", $\Rightarrow$ denotes logical implication, and $\wedge$ denotes logical AND.

This condition states that if a subset A_i precedes another subset A_j ($i < j$), then every activity in A_i must be able to reach every activity in A_j through the directly-follows relation, and there must be no possible reverse path from A_j back to A_i. This represents a strict sequential execution order between the subsets, corresponding to a pure sequence in the process model [50].

A *parallel cut* of the directly-follows graph $G(L)$ is a cut $(\wedge, A_1, A_2, \ldots, A_n)$ such that [50]:

$$\begin{aligned} &- \forall i \in \{1, \ldots, n\} \quad A_i \cap A_L^{start} \neq \emptyset \wedge A_i \cap A_L^{end} \neq \emptyset \\ &- \forall i, j \in \{1, \ldots, n\}\ \forall a \in A_i\ \forall b \in A_j \quad i \neq j \Rightarrow a \mapsto_L b \end{aligned} \tag{11}$$

where:

- $A_1, A_2, \ldots, A_n$ are mutually disjoint subsets of activities.
- A_L^{start} is the set of start activities in the log L.
- A_L^{end} is the set of end activities in the log L.
- $A_i \cap A_L^{start} \neq \emptyset$ means subset A_i contains at least one start activity.
- $A_i \cap A_L^{end} \neq \emptyset$ means subset A_i contains at least one end activity.
- $a \mapsto_L b$ means that in the directly-follows graph, activity a is directly followed by activity b.
- $\forall$ denotes "for all", $\neq \emptyset$ means the intersection is non-empty, $\wedge$ denotes logical AND, and $\Rightarrow$ denotes logical implication.

This condition enforces that each subset A_i starts with at least one start activity and ends with at least one end activity, and that for any two different subsets A_i and A_j, activities from one subset can directly follow activities from the other. This represents a parallel execution pattern (AND-split) in the process model, where activities from different subsets can occur in any order and possibly overlap in time [50].

A *redo-loop cut* of the directly-follows graph $G(L)$ is a cut $(\circlearrowleft, A_1, A_2, \ldots, A_n)$ such that [50]:

$$\begin{aligned} &- n \geq 2, \\ &- A_L^{start} \cup A_L^{end} \subseteq A_1, \\ &- \{a \in A_1 \mid \exists i \in \{2, \ldots, n\} \exists b \in A_i \quad a \mapsto_L b\} \subseteq A_L^{end}, \\ &- \{a \in A_1 \mid \exists i \in \{2, \ldots, n\} \exists b \in A_i \quad b \mapsto_L a\} \subseteq A_L^{start}, \\ &- \forall i, j \in \{2, \ldots, n\}\ \forall a \in A_i\ \forall b \in A_j \quad i \neq j \Rightarrow a \not\mapsto_L b, \\ &- \forall i \in \{2, \ldots, n\}\ \forall b \in A_i \exists a \in A_L^{end} \\ &\quad b \mapsto_L a \Rightarrow\ \forall a' \in A_L^{start} b \mapsto_L a' \end{aligned} \tag{12}$$

where:

- A_1 is the *do-part* of the loop, containing all start and end activities of the log.
- $A_2, \ldots, A_n$ are the *redo-parts*.
- n is the total number of parts in the cut. $n \geq 2$ ensures there is at least one redo-part.
- A_L^{start} is the set of start activities in log L.
- A_L^{end} is the set of end activities in log L.
- $a \mapsto_L b$ means activity a is directly followed by activity b in the directly-follows graph of L.
- $\subseteq$ denotes subset, $\cup$ denotes union.
- $\forall$ means "for all", $\exists$ means "there exists," and $\Rightarrow$ denotes logical implication.

These conditions describe a redo loop structure where A_1 (the *do-part*) contains all the start and end activities of the log. The sets $A_2, \ldots, A_n$ (the *redo-parts*) represent alternative branches that bring the process back to A_1 for repetition. Transitions between different redo-parts are not allowed. Moreover, connections between redo-parts and the start/end activities must be consistent, every redo-part must be reachable from all end activities in A_1 and must be able to return to all start activities in A_1. This ensures that the loop can repeat any number of times before the process ends [50].

APPENDIX D
ILP MINER

The ILP problem for discovering a place is defined as follows [40]:

$$\begin{aligned} \text{Minimize} \quad & c + \mathbf{1}^\top A(\mathbf{x} - \mathbf{y}) \\ \text{such that} \quad & c + A'\mathbf{x} - A\mathbf{y} \geq 0, \\ & \mathbf{1}^\top \mathbf{x} + \mathbf{1}^\top \mathbf{y} \geq 1, \\ & \mathbf{x} \in \{0,1\}^{|T|}, \quad \mathbf{y} \in \{0,1\}^{|T|}, \quad c \in \{0,1\} \end{aligned} \tag{13}$$

where:

- T is the set of activities or transitions in the Petri net.
- A and A' are two $|\mathcal{L}| \times |\mathcal{T}|$ matrices with $A(w,t) = w(t)$, and $A'(w,t) = w'(t)$, with $w = w'a$.
- $\mathbf{x} \in \{0,1\}^{|T|}$ is a binary vector indicating the transitions that serve as inputs to the place being discovered.
- $\mathbf{y} \in \{0,1\}^{|T|}$ is a binary vector indicating the transitions that serve as outputs from the place being discovered.
- $c \in \{0,1\}$ is a binary variable representing the initial marking (number of initial tokens) of the discovered place.
- $\mathbf{1}$ is a column vector of ones, used to sum the elements of a vector.

REFERENCES

[1] M. Debinski, F. Breitinger, and P. Mohan, "Timeline2GUI: A Log2Timeline CSV parser and training scenarios," *Digit. Invest.*, vol. 28, pp. 34–43, Mar. 2019.

[2] H. Studiawan, F. Sohel, and C. Payne, "Automatic event log abstraction to support forensic investigation," in *Proc. Australas. Comput. Sci. Week Multiconference*, Feb. 2020, pp. 1–9.

[3] C. Hargreaves and J. Patterson, "An automated timeline reconstruction approach for digital forensic investigations," *Digit. Invest.*, vol. 9, pp. S69–S79, Aug. 2012.
[4] C. Vanini, C. J. Hargreaves, H. van Beek, and F. Breitinger, "Was the clock correct? Exploring timestamp interpretation through time anchors for digital forensic event reconstruction," *Forensic Sci. Int., Digit. Invest.*, vol. 49, Jul. 2024, Art. no. 301759.
[5] F. Breitinger, H. Studiawan, and C. Hargreaves, "SoK: Timeline based event reconstruction for digital forensics: Terminology, methodology, and current challenges," *Forensic Sci. Int., Digit. Invest.*, vol. 53, Jul. 2025, Art. no. 301932.
[6] S. Soltani and S. A. H. Seno, "A formal model for event reconstruction in digital forensic investigation," *Digit. Invest.*, vol. 30, pp. 148–160, Sep. 2019.
[7] Y. Chabot, A. Bertaux, C. Nicolle, and M.-T. Kechadi, "A complete formalized knowledge representation model for advanced digital forensics timeline analysis," *Digit. Invest.*, vol. 11, pp. S95–S105, Aug. 2014.
[8] H. Studiawan, "Event abstration in a forensic timeline," in *Proc. Int. Conf. Inf. Commun. Technol.*, 2023, pp. 119–129.
[9] P. Gladyshev and A. Patel, "Finite state machine approach to digital event reconstruction," *Digit. Invest.*, vol. 1, no. 2, pp. 130–149, Jun. 2004.
[10] Y. Chabot, A. Bertaux, T. Kechadi, and C. Nicolle, "Event reconstruction: A state of the art," in *Handbook of Research on Digital Crime, Cyberspace Security, and Information Assurance*. Hershey, PA, USA: IGI Global, 2015, pp. 231–245.
[11] G. Park, M. Cho, and J. Lee, "Leveraging machine learning for automatic topic discovery and forecasting of process mining research: A literature review," *Expert Syst. Appl.*, vol. 239, Apr. 2024, Art. no. 122435.
[12] M. Xavier, V. Dubinin, S. Patil, and V. Vyatkin, "A framework for the generation of monitor and plant model from event logs using process mining for formal verification of event-driven systems," *IEEE Open J. Ind. Electron. Soc.*, vol. 5, pp. 517–534, 2024.
[13] W. M. P. V. D. Aalst et al., "Process mining manifesto," in *Proc. Bus. Process Manage. Workshops, BPM Int. Workshops*, 2012, pp. 169–194.
[14] E. Brzychczy, A. Żuber, and W. V. D. Aalst, "Process mining of mining processes: Analyzing longwall coal excavation using event data," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 54, no. 5, pp. 2723–2734, May 2024.
[15] J. James, P. Gladyshev, M. T. Abdullah, and Y. Zhu, "Analysis of evidence using formal event reconstruction," in *Proc. Digit. Forensics Cyber Crime. ICDF2C*, 2010, pp. 85–98.
[16] J. Gruber, M. Humml, L. Schröder, and F. C. Freiling, "Formal verification of necessary and sufficient evidence in forensic event reconstruction," in *Proc. Digit. Forensics Res. Conf. (DFRWS) Europe*, 2023, pp. 1–11.
[17] C. Vanini, J. Gruber, C. Hargreaves, Z. Benenson, F. Freiling, and F. Breitinger, "Understanding strategies and challenges of timestamp tampering for improved digital forensic event reconstruction," in *Proc. Digit. Forensics Doctoral Symp.*, Apr. 2025, pp. 1–8.
[18] S. Lagraa and R. State, "Process mining-based approach for investigating malicious login events," in *Proc. NOMS-IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2020, pp. 1–5.
[19] A. Hemmer, R. Badonnel, and I. Chrisment, "A process mining approach for supporting IoT predictive security," in *Proc. NOMS-IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2020, pp. 1–9.
[20] X. Zhang, L. Cui, W. Shen, J. Zeng, L. Du, H. He, and L. Cheng, "File processing security detection in multi-cloud environments: A process mining approach," *J. Cloud Comput.*, vol. 12, no. 1, p. 100, Jul. 2023.
[21] A.-M. Konsta, G. D. Federico, A. L. Lafuente, and A. Burattin, "Attack tree generation via process mining," in *Proc. Int. Symp. Leveraging Appl. Formal Methods*, 2024, pp. 356–372.
[22] H. Studiawan, F. Sohel, and C. Payne, "A survey on forensic investigation of operating system logs," *Digit. Invest.*, vol. 29, pp. 1–20, Jun. 2019.
[23] M. Leemans and W. M. P. V. D. Aalst, "Discovery of frequent episodes in event logs," in *Proc. 4th Int. Symp. Data-Driven Process Discovery Anal.*, 2015, pp. 1–31.
[24] Y. Yang, C. Zhong, X. Liu, and W. Lu, "Modeling and analysis of cyber-physical systems based on Petri net," *Int. J. Control, Autom. Syst.*, vol. 21, no. 9, pp. 2980–2994, Sep. 2023.
[25] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 33–40.
[26] M. Savino, G. Chiloiro, C. Masciocchi, N. D. Capocchiano, J. Lenkowicz, B. Gottardelli, M. A. Gambacorta, V. Valentini, and A. Damiani, "A process mining approach for clinical guidelines compliance: Real-world application in rectal cancer," *Frontiers Oncol.*, vol. 13, May 2023, Art. no. 1090076.
[27] J.-R. Rehse, S. J. J. Leemans, P. Fettke, and J. M. E. M. Van Der Werf, "On process discovery experimentation: Addressing the need for research methodology in process discovery," *ACM Trans. Softw. Eng. Methodology*, vol. 34, no. 1, pp. 1–29, Jan. 2025.
[28] D. Chapela-Campa and M. Dumas, "From process mining to augmented process execution," *Softw. Syst. Model.*, vol. 22, no. 6, pp. 1977–1986, Dec. 2023.
[29] J. Casas-Ramos, M. Mucientes, and M. Lama, "REACH: Researching efficient alignment-based conformance checking," *Expert Syst. Appl.*, vol. 241, May 2024, Art. no. 122467.
[30] O. Ouarem, F. Nouioua, and P. Fournier-Viger, "A survey of episode mining," *WIREs Data Mining Knowl. Discovery*, vol. 14, no. 2, p. 1524, Mar. 2024.
[31] M.-Y. Su, "Discovery and prevention of attack episodes by frequent episodes mining and finite state machines," *J. Netw. Comput. Appl.*, vol. 33, no. 2, pp. 156–167, Mar. 2010.
[32] M.-Y. Su, "Applying episode mining and pruning to identify malicious online attacks," *Comput. Electr. Eng.*, vol. 59, pp. 180–188, Apr. 2017.
[33] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex sequences," *Inf. Syst.*, vol. 33, no. 1, pp. 96–114, Mar. 2008.
[34] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 259–289, Sep. 1997.
[35] P. Fournier-Viger, Y. Yang, P. Yang, J. C.-W. Lin, and U. Yun, "TKE: Mining top-K frequent episodes," in *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices*. Cham, Switzerland: Springer, 2020, pp. 832–845.
[36] S. Weerapong, P. Porouhan, and W. Premchaiswadi, "Process mining using α-algorithm as a tool (a case study of student registration)," in *Proc. 10th Int. Conf. ICT Knowl. Eng.*, Nov. 2012, pp. 213–220.
[37] W. Van Der Aalst, *Process Mining: Data Science in Action*. Cham, Switzerland: Springer, 2016.
[38] A. J. Weijters, W. M. van Der Aalst, and A. A. De Medeiros, *Process Mining With the HeuristicsMiner Algorithm*. Eindhoven, The Netherlands: Technische Universiteit Eindhoven, 2006.
[39] A. Bogarín, R. Cerezo, and C. Romero, "Discovering learning processes using inductive miner: A case study with learning management systems (LMSs)," *Psicothema*, vol. 3, no. 30, pp. 322–329, Aug. 2018.
[40] J. M. E. M. V. D. Werf, B. F. V. Dongen, C. A. J. Hurkens, and A. Serebrenik, "Process discovery using integer linear programming," in *Proc. Int. Conf. Appl. Theory Petri Nets*, 2009, pp. 387–412.
[41] B. F. van Dongen, J. De Smedt, C. Di Ciccio, and J. Mendling, "Conformance checking of mixed-paradigm process models," *Inf. Syst.*, vol. 102, Dec. 2021, Art. no. 101685.
[42] W. Van Der Aalst, A. Adriansyah, and B. Van Dongen, "Replaying history on process models for conformance checking and performance analysis," *WIREs Data Mining Knowl. Discovery*, vol. 2, no. 2, pp. 182–192, Mar. 2012.
[43] K. R. Sungkono, R. Sarno, F. D. M. Kinanggit, I. D. Shubhi, and K. Nurlaela, "Graph-based process mining for measuring quality of business process model," in *Proc. Int. Conf. Informat. Electr. Electron. (ICIEE)*, Oct. 2022, pp. 1–6.
[44] J. C. A. M. Buijs, B. F. V. Dongen, and W. M. P. V. D. Aalst, "On the role of fitness, precision, generalization and simplicity in process discovery," in *Proc. OTM Confederated Int. Conf. Move Meaningful Internet Syst.*, 2012, pp. 305–322.
[45] U. Singh, Y. Hoskere, N. Tapas, R. Vyas, and O. P. Vyas, "Evaluating the effectiveness of multi-filtering techniques on comprehensibility improvement of spaghetti models," in *Proc. 14th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Jul. 2023, pp. 1–6.
[46] J. Lijffijt, P. Papapetrou, and K. Puolamäki, "Size matters: Choosing the most informative set of window lengths for mining patterns in event sequences," *Data Mining Knowl. Discovery*, vol. 29, no. 6, pp. 1838–1864, Nov. 2015.
[47] R. Andreswari, I. Syahputra, and M. Lubis, "Performance analysis of heuristic miner and genetics algorithm in process cube: A case study," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 11, no. 1, pp. 393–399, Feb. 2021.

[48] H. Boisaubert, C. Grivaud, A. Bouchet, C. Lejus-Bourdeau, and C. Sinoquet, "Expertise versus data: Comparison of expertise based process models to process-mining models of surgery under general anesthesia," in *Proc. 18th Int. Joint Conf. Biomed. Eng. Syst. Technol.*, 2025, pp. 742–749.
[49] M. I. A. Pradana, A. P. Kurniati, and G. A. A. Wisudiawan, "Inductive miner implementation to improve healthcare efficiency on Indonesia national health insurance data," in *Proc. Int. Conf. Data Sci. Appl. (ICoDSA)*, Jul. 2022, pp. 239–244.
[50] V. Der Aalst. (2021). *Rwth Process Mining Lecture 10: Inductive Mining*. Accessed: Aug. 8, 2025. [Online]. Available: https://www.youtube.com/watch?v=ZPsMaFBSk10&t=4823s

RIDA ADILA received the bachelor's and master's degrees in informatics from Institut Teknologi Sepuluh Nopember (ITS), Surabaya, Indonesia, in 2022 and 2025, respectively. She is currently a Web Developer with the Directorate of Technology and Information Systems Development, ITS.

HUDAN STUDIAWAN (Member, IEEE) received the bachelor's and master's degrees from Institut Teknologi Sepuluh Nopember, Indonesia, in 2009 and 2011, respectively, and the Ph.D. degree from Murdoch University, Australia, in 2021. He is currently an Associate Professor with Institut Teknologi Sepuluh Nopember. His current research interests include digital forensics and natural language processing.

FRANK BREITINGER was an Associate Professor of digital forensic science with the University of Lausanne and an Assistant Professor at the Hilti Chair for Data and Application Security, Liechtenstein University, and the University of New Haven, CT, USA, where he also acted as the Co-Director of the University of New Haven Cyber Forensics Research and Education Group (UNHcFREG, http://www.unhcfreg.com). He is currently with the Chair for Cybersecurity, University of Augsburg, Germany. His teaching and research interests include cybersecurity and digital forensics. Additional information about him and his work is on his website (https://www.fbreitinger.de)