Sequential Data Mining in Digital Forensic Analysis

Kiara Koščová[1,†], Rastislav Krivoš-Belluš[1,*,†]

[1]Institute of Computer Science, Faculty of Science, Pavol Jozef Šafárik University in Košice, Jesenná 5, 040 01 Košice, Slovakia

Abstract

Digital forensic analysis of Windows Event Logs plays an important role in investigating cybersecurity incidents. However, the continuously increasing volume of system-generated events makes the identification of forensic evidence challenging, as most records represent routine system activity. Existing rule-based approaches are limited to detecting known threats, while machine learning methods often lack the interpretability required in forensic investigations.

The aim of this paper is to design and experimentally evaluate a methodology based on sequential pattern mining for the automatic identification of routine event sequences in Windows Event Logs and the subsequent reduction of data requiring manual forensic inspection. The proposed pipeline transforms raw events logs into a sequential representation through tokenization and sequence construction, followed by the application of the PrefixSpan, GSP and SPADE algorithms.

The methodology was evaluated using an annotated dataset derived from the The Stolen Szechuan Sauce CTF scenario containing 40,917 Windows Event Log records, including 27 attack-related events. The experimental results demonstrate that the proposed approach significantly reduces the amount of data requiring manual analysis while preserving most of the forensic evidence. The obtained results indicate that sequential pattern mining represents a promising and interpretable approach for supporting digital forensic analysis of large-scale Windows Event Logs.

Keywords

digital forensics, Windows Event Logs, sequential patterns, sequential pattern mining

1. Introduction

Digital forensic analysis in the Windows operating system environment relies primarily on system event records, known as Windows Event Logs. Every interaction with the system leaves a trace that, in line with Locard's principle, may be crucial for reconstructing a security incident [1]. In practice, however, analysis is complicated by the extreme volume of these records: normal system operation generates tens of thousands of events per day, and the vast majority of them reflect routine system activity with no forensic relevance [1]. For the investigator, these records create informational noise that obscures truly relevant traces.

Existing approaches to this problem can be divided into two categories, each with its own limitations. Static detection rules can efficiently identify known attack patterns, but they are inherently blind to new or modified threats for which no rule yet exists. Machine learning models can detect unknown anomalies as well, but often at the cost of low interpretability, which is a serious obstacle in the legal and security context of forensic investigation.

This study examines sequential pattern mining as an alternative to both existing categories. Unlike static rules, it does not require prior knowledge of a specific attack, but instead relies solely on the statistical properties of the data itself. Unlike machine learning models, it produces results in the form of explicit sequential patterns that are directly interpretable. Each pattern represents a concrete sequence of system events identified as routine. The key assumption of this approach is that malicious activity forms a frequency minority in Windows Event Logs, so records covered by frequent patterns are likely noise and can be automatically removed before manual inspection.

ITAT'26: Information Technologies – Applications and Theory, September 25–29, 2026, Biele Karpaty, Slovakia

*Corresponding author.

†These authors contributed equally.

kiara.koscova@student.upjs.sk (K. Koščová); rastislav.krivos-bellus@upjs.sk (R. Krivoš-Belluš)

0009-0007-0650-5180 (R. Krivoš-Belluš)

© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

published 2026-11-09

CEUR-WS.org/Vol-4323/paper45.pdf

The main goal of this paper is to design, implement, and experimentally validate a methodology that uses sequential mining algorithms to automatically reduce the volume of Windows Event Logs while preserving forensically relevant attack-related records. Specifically, the work addresses the question of to what extent sequential mining algorithms can reduce the amount of forensically irrelevant records in Windows Event Logs while preserving attack-related records.

2. Methodology

Network threat detection and security event analysis have been widely studied, with recent surveys covering mathematical detection models, intrusion detection and prevention systems, machine learning, and pattern mining. Most existing approaches focus on detecting specific attacks in network traffic [2].

In contrast, this work analyses Windows Event Logs at the system level. Instead of direct attack detection, it aims to identify normal behaviour patterns that can reduce the volume of forensically irrelevant records and simplify subsequent forensic analysis.

The present era is characterized by the exponential growth of data volumes. Technological progress has made it possible to store and reprocess data with ease, resulting in large repositories that are difficult to examine. In this context, the field known as knowledge discovery in databases (KDD) comes to the forefront, originally defined as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data." This definition [3] was later refined: KDD is the nontrivial process of identifying valid, novel, potentially useful, and understandable patterns in data. The central step in this process is data mining. In this phase, computational techniques are applied to data in order to uncover patterns. The remaining stages of KDD focus on data preparation and the evaluation of discovered patterns.

Pattern mining is one of the key areas of data mining, aiming to discover statistically significant structures in data. Depending on the nature of the data and the relationships being sought, several approaches are distinguished, including frequent itemset mining, sequential pattern mining, graph pattern mining, and high-utility pattern mining. Sequential pattern mining takes the order of item occurrence into account [4]. It focuses on identifying frequent sequences of events across multiple transactions within a record and is used, for example, in user behaviour analysis.

A sequential pattern is a subsequence that appears in a sequence database with a frequency not lower than the selected minimum support value [5]. Such subsequences are referred to as frequent sequences. The goal of sequential mining algorithms is to identify all frequent sequential patterns that satisfy the specified minimum support threshold.

In the context of this paper, these concepts are used to formally represent system events and analyse them afterward. One token (item) represents a combination of the Channel, Provider, and EventId attributes, which uniquely identifies a type of system event. Sequences are then formed from chronologically ordered tokens belonging to the same user and device, and these sequences serve as input to the sequential mining algorithms. Frequent sequential patterns are subsequently searched for within these sequences.

When analyzing sequential data, it is important not only to identify frequent sequential patterns but also to search for them efficiently so that large volumes of data can be processed in a reasonable time. For this reason, various sequential mining algorithms have been developed, differing in how they explore the space of possible patterns and in the optimization techniques used during mining.

2.1. Generalized Sequential Patterns

Generalized Sequential Patterns (GSP) is a sequential pattern mining algorithm based on the *candidate generation* approach. It iteratively generates candidate sequences and verifies their occurrence in the database. The algorithm operates in a level-wise manner, extending frequent sequences of length k to candidates of length $k + 1$ at each step. Its input is a sequence database, where each sequence consists of an ordered list of transactions and each transaction contains a set of items. GSP searches for patterns in which items appear in the correct order, but not necessarily consecutively.

The GSP algorithm [6] works as follows:

1. Scan the database and identify frequent sequences of length 1.
2. Generate candidate sequences of length $k+1$ by joining pairs of frequent sequences of length k, where the prefix of one matches the suffix of the other.
3. Remove candidate sequences whose any subsequence is not frequent, based on the Apriori property.
4. Compute the support of the remaining candidate sequences by scanning the database and remove sequences with support lower than `min_sup`.
5. Repeat steps (2) to (4) for progressively increasing sequence lengths until no new frequent sequences are found.

2.2. PrefixSpan

PrefixSpan belongs to the class of pattern-growth methods and, unlike GSP, avoids candidate generation. Its key idea is to recursively reduce the search space using database projections.

For each frequent prefix α, an α-projected database is constructed as the subset of sequence suffixes that contain the prefix α. The algorithm then searches for frequent extensions exclusively within this reduced database, thereby shrinking the search space at each recursive step. This approach corresponds to depth-first search, in contrast to GSP, which performs breadth-first search [6].

The PrefixSpan algorithm [7] proceeds as follows:

1. Scan the database and identify frequent items of length 1.
2. Remove items with support lower than `min_sup` and construct the initial prefixes.
3. Build an α-projected database for each prefix α.
4. Search the projected database and extend the prefix by each item satisfying `min_sup`.
5. Recursively repeat steps (3) and (4) for each new prefix until no further extension is possible.

Compared with GSP, which explores the pattern space breadth-first and generates candidates explicitly, PrefixSpan works depth-first and does not generate candidates; instead, it builds each pattern directly from the projected database. The main disadvantage of PrefixSpan is the memory overhead associated with constructing a large number of projected databases, whereas GSP may be more advantageous when the pattern space is dense [6].

2.3. Sequential PAttern Discovery using Equivalence classes

Sequential PAttern Discovery using Equivalence classes (SPADE) is a sequential pattern mining algorithm that operates on a vertical data format. Unlike GSP and PrefixSpan, which scan the database horizontally, SPADE represents each item as a list of pairs $(\text{sequence ID}, \text{timestamp})$. With this representation, the support of candidates can be computed by simple list intersections without repeatedly scanning the entire database [8].

The SPADE algorithm [8] works as follows:

1. Convert the input database into a vertical representation, where each item contains a list of pairs $(\text{sequence ID}, \text{timestamp})$ for all of its occurrences in sequences.
2. Partition the pattern space into equivalence classes according to a common prefix, with each single-item sequence forming its own class.
3. Within each equivalence class, generate candidate sequences of length $k+1$ by joining pairs of frequent sequences of length k sharing the same prefix.
4. Compute candidate support by intersecting vertical databases, and remove sequences with support lower than `min_sup`.
5. Repeat steps (3) and (4) for each new equivalence class until no new frequent sequences are generated.

A disadvantage of SPADE is that it mines all frequent sequences, including redundant subsequences, which can lead to an enormous number of output patterns when the minimum support threshold is low.

2.4. Detection of Standard Behavior Patterns

In this work, a standard behavior pattern is understood as a sequential pattern that occurs with high frequency in the analyzed dataset. This approach is based on the assumption that legitimate and routine operating system activity continuously generates recurring sequences of events. These legitimate operations form the dominant share of records in system logs, while targeted user interaction or covert malicious activity leaves only a much smaller number of traces in the data.

However, this analytical assumption has natural limitations. Certain types of cyberattacks, such as brute-force attacks or network scanning, may by their nature generate a large volume of records in a very short time. In such cases, frequency analysis could paradoxically classify them among the most common sequences. Such attack behavior could then be identified directly from the most frequent patterns, since it would exhibit a structure clearly atypical for normal system operation. A systematic verification of this phenomenon lies beyond the scope of this work, as the primary objective is to design a method for detecting hidden and less obvious attacks whose manifestations in system logs are not frequency-dominant.

To define the boundary between standard and potentially forensically relevant behavior, an 80/20 ratio inspired by the Pareto principle was chosen as a practical heuristic. The goal of the proposed filtering is to remove approximately 80% of records representing normal system activity and retain the remaining 20% for manual inspection, as they represent deviations from routine behavior. This ratio was not selected as a fixed rule, but rather as a starting point for parameter setting in the algorithms. The exact level of reduction depends on the chosen algorithm and its parameters and is evaluated experimentally in Chapter 4.

2.5. Data Transformation into Sequences and Analysis Pipeline

Sequential mining algorithms require data to be preprocessed in the form of sequences. A sequence consists of an ordered list of itemsets, where each itemset may contain one or more tokens. The basic building block of sequences is a token, which represents one type of system event and is created by concatenating the values of the `Channel`, `Provider`, and `EventId` attributes in the form `Channel` | `Provider` | `EventId`. These three attributes were selected because, together, they uniquely characterize the event type. The `EventId` attribute alone would be ambiguous without the provider context, since different providers may use the same numeric identifiers for different events. The `Channel` attribute further distinguishes the Windows subsystem in which the event originated, such as `Security`, `System`, or `Application`. The resulting token compactly encodes the semantic identity of the event and serves as the basic unit for sequence construction and subsequent pattern matching.

Two approaches to sequence construction were experimentally evaluated. The first uses session segmentation with preprocessing, while the second applies a sliding window method. In the first approach, data were aggregated into sessions, and each group of events belonging to the same `Computer` and `User` pair was assigned a unique identifier in the form `Computer` | `User` | `SessionId`. A sequence was split into separate sessions if more than 15 minutes elapsed since the last token or if the session reached a limit of 200 tokens. This limit was chosen empirically with respect to computational constraints and the high volume of Windows Event Logs. The session-based approach included burst detection, in which repeated tokens occurring three or more times consecutively were marked with the suffix `!BURST`, deduplication of consecutive identical tokens, and final filtering of sequences shorter than 2 tokens or longer than 200 tokens. Each token retained a reference to the original row in the dataset, enabling traceability for forensic evaluation.

The second approach uses the sliding window method. In this variant, the chronologically ordered dataset is directly divided into partially overlapping segments of fixed length w, shifted by a defined step size `step`. Unlike the session-based approach, it does not apply deduplication or cleaning mechanisms.

The analytical process is organized as a modular pipeline that is independent of the sequence-construction method. It begins with loading and chronologically sorting the raw dataset, followed by token construction and sequence construction using one of the two methods described above. The

resulting sequences are then passed to a sequential mining algorithm to identify frequent patterns. Finally, records whose tokens are covered by discovered frequent patterns are removed as routine system activity, while uncovered tokens remain in the reduced dataset as anomalies intended for detailed forensic analysis.

3. Experiment

3.1. Dataset

The data used in this work come from the publicly available CTF scenario *The Stolen Szechuan Sauce* [9], which simulates a real security incident in a Windows operating system environment. The version of the dataset used in this work was prepared within the ADFIR project [10], with logs extracted using tools developed by Eric Zimmerman and subsequently transformed into a structured format suitable for processing in Python.

This work uses the subset of the dataset labeled *Desktop-EventLogs*, which contains Windows Server 2012 R2 (64-bit) system events. The dataset provides a chronological record of system activity that includes traces of both legitimate and malicious behavior. For the purposes of analysis, the dataset was extended with a `Label` attribute, which serves as a binary annotation indicating whether a particular record is related to an attack (1) or not (missing value). These annotations were taken from the original scenario and serve as reference labels for evaluating the analysis results.

The dataset contains a total of 40 917 records and 28 attributes. In terms of data types, it includes 19 text attributes, supplemented by 7 numeric attributes, 1 binary attribute, and the `Label` attribute used for record annotation, which takes the value `1` or remains missing. From the perspective of the classification task, the dataset is highly imbalanced:

- 0.066% of records (27) are labeled as attacks,
- 99.934% of records (40 890) represent normal behavior.

This imbalance reflects real-world conditions, in which malicious activity is much less frequent than routine behavior.

The dataset contains several types of attributes:

- **identification:** `EventRecordId`, `RecordNumber`,
- **temporal:** `TimeCreated`,
- **process-related:** `ProcessId`, `ThreadId`,
- **system:** `Provider`, `Channel`, `Level`,
- **user-related:** `UserId`, `UserName`,
- **content (payload):** `PayloadData1–6`, `Payload`.

The analyzed dataset contains a significant number of missing values, especially in payload-related and user-related attributes. Some attributes, such as `RemoteHost` or `PayloadData6`, contain more than 95% missing values. However, this is not due to data collection errors, but rather to a natural characteristic of Windows Event Logs. Individual event types record only the attributes relevant to a specific event, which leads to a large number of empty values in the dataset.

The `EventId` attribute represents the type of event generated by a specific system component (`Provider`). In the Windows operating system environment, it serves as an identifier of the event category. The distribution of `EventId` values is strongly uneven, which is a natural consequence of system behavior. Routine system operations generate some event types much more frequently than specific or rare events. This uneven distribution has a significant impact on the pattern mining process, since frequent sequential patterns are formed primarily by event types that recur often.

3.2. Implementation

The proposed approach was implemented in the Python programming language using the PyCharm development environment, with additional use of Jupyter Notebook for prototyping and analysis of intermediate results. To mine frequent sequential patterns, algorithms representing different approaches to sequential data mining were used. PrefixSpan was applied through the `prefixspan` library, GSP was implemented as a custom solution respecting the Apriori principle, and SPADE was executed through the `SPMF` library implemented in Java.

The experiments were designed to compare the individual algorithms in terms of their suitability for reducing the volume of Windows Event Logs while preserving forensically relevant records. All algorithms were tested on the same dataset and on an identical sequence representation created according to the methodology described in the previous chapter. The only variable changed was the sequential mining algorithm used, which made it possible to compare their properties objectively.

The following characteristics were evaluated in particular:

- dataset reduction rate,
- recall with respect to annotated attack records,
- algorithm execution time.

The reduction rate expresses the proportion of records identified as part of frequent patterns representing routine system behavior and subsequently removed from further analysis. Recall represents the proportion of correctly preserved attack records relative to their total number in the dataset. Execution time characterizes the computational complexity of each algorithm.

The experiments were carried out on a computer equipped with an Intel Core i7-12650H processor, 16 GB of RAM, and the Windows 11 operating system.

Next subsections present the results of three sequential mining algorithms – PrefixSpan, GSP, and SPADE – together with their mutual comparison. The evaluated criteria were their ability to reduce the volume of analyzed data while preserving forensically relevant records, the number of preserved forensically relevant rows in the reduced dataset, and the execution time of the algorithms. Since all three algorithms share the same evaluation framework, the parameters and metrics are defined uniformly in this introductory part and are not repeated in the following sections.

The processing result is produced in two separate phases, and each parameter affects only one of them. In the mining phase, the algorithm searches the sequences obtained during sequence construction for frequent subsequences representing normal system behavior. This phase is governed by minimum support, window size, step, and maximum pattern length. In the coverage and removal phase, the discovered frequent patterns are applied to the original chronologically ordered dataset, and each row covered by at least one pattern is removed from the dataset. This phase is governed by the maximum pattern gap.

The parameters examined were:

- **Minimum support (`min_sup`)** – the minimum number of sequences in which a subsequence must occur to be considered a frequent pattern. Lowering it expands the set of patterns classified as routine, increasing coverage and reduction, but also sharply increasing the number of patterns and execution time. At very low values, filter contamination and recall loss may occur.
- **Window size (`win`)** – the length of input sequences in the sliding-window construction. A larger window contains combinatorially more subsequences and produces more and longer patterns, but its computational scaling is the least favorable.
- **Step / overlap (`step`)** – the number of events by which the window is shifted, and thus the degree of overlap between neighboring windows.
- **Maximum pattern length (`maxlen`)** – the upper bound on the length of generated patterns. It affects the mining phase and typically exhibits saturation.
- **Maximum pattern gap (`MAX_GAP`)** – the number of foreign tokens that may lie between two consecutive elements of a pattern for the occurrence to still be counted during the coverage phase. It acts exclusively in the coverage and removal phase.

The PrefixSpan variant with preprocessing introduces an additional preprocessing phase before mining, with two specific parameters: SESSION_GAP_MIN (the time interval delimiting sessions) and BURST_THRESHOLD (the threshold for clustering burst activity). These parameters are specific to this variant only and must not be confused with the maximum pattern gap used in the coverage phase.

Metrics. Reduction expresses the proportion of removed records out of the total 40 917. Recall expresses the proportion of preserved attack records out of the 27 annotated attacks. Execution time characterizes computational complexity.

Methodology. The influence of the parameters was examined using the OFAT (one factor at a time) method: in each experiment, one parameter was changed while all others remained fixed at the values stated in the description of each table.

3.3. GSP

The GSP algorithm uses candidate generation family and is a direct sequential extension of Apriori: it generates candidate sequences of length $k + 1$ level by level through joining, removes unpromising candidates based on anti-monotonicity, and verifies the support of the remaining ones by scanning the database. This choice was motivated by the hypothesis that, for the structure of Windows Event Logs, in which routine activity generates frequent and relatively short patterns, explicit candidate generation may be effective.

When the minimum support is reduced, the number of candidates grows combinatorially. In GSP this is caused by the fact that even less frequent event sequences are considered frequent and the algorithm tries to further join them into longer chains. This leads to a nonlinear increase in computational cost, since verifying each new candidate requires a separate database scan.

Table 1
Effect of minimum support (win = 20, step = 10, max_k = 10)

Support	Min. support (abs.)	Runtime	Number of patterns	Removed	Reduction	Recall
0.5%	20	2096.57 s	292 809	33 135	81.02%	77.78%
1.0%	40	302.25 s	64 172	33 064	80.81%	85.19%
2.0%	81	27.43 s	10 799	29 636	72.47%	92.59%
5.0%	204	4.07 s	893	22 362	54.68%	92.59%

The window size determines the depth of local context from which GSP generates candidates and proved to be the most critical parameter for scalability. A larger window dramatically increases the number of tested subsequences, resulting in an extreme growth in the number of database scans. In terms of filtering, a larger window captures more complex and longer routine sequences, slightly increasing the reduction rate, but long patterns may also accidentally cover isolated attack rows surrounded by routine activity, which causes recall loss. Smaller windows, in contrast, provide a finer approximation of routine behavior, are more computationally efficient, and thanks to shorter patterns better preserve the forensic integrity of attack records.

Table 2
Effect of window size (step = 10, support = 1.0%)

Window	Runtime	Number of patterns	Removed	Reduction	Recall
10	6.20 s	1 392	30 606	74.84%	88.89%
15	38.74 s	10 082	32 707	79.98%	85.19%
20	302.25 s	64 172	33 064	80.81%	85.19%
25	4649.32 s	450 053	33 388	81.64%	85.19%

Reducing the step size, and thus increasing overlap, causes the same log segment to appear in multiple windows, which artificially strengthens the statistical significance of routine motifs. This mechanism ensures that GSP captures frequent patterns even at window boundaries, which would otherwise be broken and lost in a non-overlapping configuration. For filtering, higher overlap has a clearly positive

effect in the form of higher dataset reduction, since it enables a more robust pattern base that covers routine activity more thoroughly without affecting attack records.

Table 3
Effect of step size (`win` = 20, support = 1.0%)

Step	Overlap	Runtime	Removed	Reduction	Recall
5	75%	287.59 s	33 573	82.10%	85.19%
8	60%	307.68 s	33 176	81.12%	85.19%
10	50%	302.25 s	33 064	80.85%	85.19%
15	25%	137.18 s	32 392	79.21%	85.19%
20	0%	130.54 s	31 375	76.72%	85.19%

At low values, the algorithm quickly exhausts short candidates, but as their length increases, combinatorial growth becomes apparent. A saturation point was observed in the experiments, where with increasing pattern length the support monotonically fell below the threshold, and the generation of new candidates naturally stopped. Mining up to medium lengths $k = 5$ to 7 significantly increases routine coverage, but further extension brings only a negligible gain in reduction at the cost of disproportionately high execution time for additional candidate-generation levels. For log filtering, medium-length patterns that capture the structure of normal activity therefore appear to provide a reasonable trade-off, while extremely long patterns are inefficient for dataset reduction.

Table 4
Effect of maximum pattern length (`win` = 20, `step` = 10, support = 1.0%)

`max_k`	Runtime	Number of patterns	Removed	Reduction	Recall
3	21.06 s	3 310	27 214	66.54%	85.19%
5	53.26 s	18 061	30 910	75.58%	85.19%
8	146.91 s	52 293	33 225	81.24%	85.19%
10	302.25 s	64 172	33 064	80.81%	85.19%
15	197.30 s	71 020	33 064	80.85%	85.19%

The parameter analysis confirmed that the time complexity of GSP is not driven by short-pattern mining, but by the generation of long candidates, which contribute only marginally to reduction. The strongest factor affecting computational cost is window size, while the most effective way to increase filtering without loss of recall is to decrease the step size and thus increase overlap. For efficient log filtering, the results suggest that a suitable choice is a smaller window combined with higher overlap and a limited mining depth of `max_k` = 5 to 7, which makes it possible to achieve a comparable reduction rate in significantly shorter time while preserving recall at the level of 85.19%.

3.4. PrefixSpan

The PrefixSpan belongs to pattern-growth techniques, meaning that instead of explicitly generating and testing candidates, it recursively builds projected databases according to prefixes. Since the mining itself proved lossless in terms of preserving forensically relevant rows, the decisive factor affecting filtering quality was the way input sequences were prepared and segmented. PrefixSpan was therefore examined in three variants that differ precisely in data preparation: PrefixSpan with preprocessing, iterative PrefixSpan, and PrefixSpan with a sliding window.

`PrefixSpan with Preprocessing` introduces a three-step preparation phase before mining: deduplication and burst collapsing, removal of out-of-range records, and construction of time-bounded sessions. Mining is then performed only on these prepared sessions.

A key finding is that the recall of this variant remains constant at 55.56% (15 of 27) across all tested parameter settings. Of the twelve lost attack records, eleven were removed by deduplication and one by the out-of-range filter. Mining itself did not remove any attack. After subtracting the effect of deduplication, filtering achieves a recall of 93.75%. The loss of recall therefore does not occur during

mining, but already in preprocessing, especially in deduplication and burst collapsing, which merge attack records with visually identical but forensically distinct events.

Table 5
Effect of minimum support (`minlen` = 2, `maxlen` = 8, SESSION_GAP_MIN = 15, BURST_THRESHOLD = 3)

Support	Number of patterns	Removed	Reduction	Recall	Runtime
50	12	18 284	44.69%	55.56%	0.6 s
40	923	20 169	49.29%	55.56%	1.1 s
30	97 442	23 392	57.17%	55.56%	28.4 s
20	3 324 716	26 428	64.59%	55.56%	657.0 s

When support decreases from 50 to 20, reduction increases from 44.69% to 64.59%, but the number of patterns grows from 12 to more than 3.3 million and runtime rises from 0.6 s to 657.0 s. Minimum support is therefore the strongest factor influencing computational cost: it improves reduction only moderately, while increasing the number of patterns and runtime by orders of magnitude. Recall remains unchanged, since attack records are rare and unique and never reach the support threshold to become removable patterns.

Table 6
Effect of maximum pattern length (`support` = 20, SESSION_GAP_MIN = 15, BURST_THRESHOLD = 3)

`maxlen`	Number of patterns	Removed	Reduction	Recall	Runtime
4	21 019	25 239	61.68%	55.56%	3.8 s
6	377 706	26 135	63.87%	55.56%	65.4 s
8	3 324 716	26 428	64.59%	55.56%	657.0 s

Extending the maximum pattern length increases reduction only marginally, from 61.7% to 64.6%, while the number of patterns and runtime grow by orders of magnitude. The parameter therefore saturates.

Table 7
Effect of session gap (`support` = 20, `maxlen` = 8, BURST_THRESHOLD = 3)

Gap	Number of sessions	Number of patterns	Removed	Reduction	Recall	Runtime
5	407	3 306 173	26 695	65.24%	55.56%	630.8 s
10	375	3 324 716	26 430	64.59%	55.56%	624.9 s
15	359	3 324 716	26 428	64.59%	55.56%	657.0 s
100	301	3 492 415	26 647	65.12%	55.56%	640.4 s

The session gap changes the number of sessions, but has practically no effect on reduction or recall. It is the weakest of the tested parameters.

Table 8
Effect of burst threshold (`support` = 20, `maxlen` = 8, SESSION_GAP_MIN = 15)

Burst	Number of patterns	Removed	Reduction	Recall	Runtime
2	2 262 540	25 955	63.43%	55.56%	443.9 s
3	3 324 716	26 428	64.59%	55.56%	657.0 s
5	3 781 999	26 190	64.01%	55.56%	691.6 s

The burst threshold primarily affects the number of patterns and runtime, while reduction stays in the narrow range of 63–65% and recall remains unchanged. Burst is one of the steps responsible for recall loss, but within mining it has only a minor influence on filtering quality.

PrefixSpan with preprocessing achieves the highest reduction among the variants, 64.6%, but is fundamentally limited by a recall ceiling of 55.56% caused by deduplication. For forensic deployment, where preserving all traces is the priority, it is therefore unsuitable. Its value is mainly illustrative: aggressive preprocessing damages forensic integrity before mining even starts.

`Iterative PrefixSpan` omits preprocessing and runs in three passes with decreasing support thresholds. In each pass, frequent patterns are mined, sessions covered by them are removed, and only the active remainder enters the next pass with a lower threshold. This first filters the most common routine and then progressively less frequent behavior.

The recall of this variant is constant at 92.59% (25 of 27) across all configurations. Two attack records are consistently lost in the second pass, when they become part of a covered session. Because no deduplication is performed, recall is much higher than in the preprocessing variant.

Table 9
Effect of maximum session length (passes 10% → 5% → 2%, `SESSION_GAP_MIN` = 15)

Session length	Sessions	Patterns (1st pass)	Removed	Reduction	Recall	Runtime
50	952	598	20 597	50.34%	92.59%	1.1 s
100	551	13 449	21 671	52.96%	92.59%	7.5 s
200	359	72 897	17 276	42.22%	92.59%	37.5 s
250	323	119 950	17 267	42.20%	92.59%	171.2 s

Maximum session length proved to be the most influential factor. Shorter sessions, meaning a higher number of sessions, bring both higher reduction and dramatically lower runtime, because they produce more numerous but simpler sequences that are mined much faster.

Table 10
Effect of support thresholds (`SESSION_GAP_MIN` = 15, `maxlen` = 8)

Threshold configuration	Removed	Reduction	Recall	Runtime
10% → 5% → 2% (`min_abs` = 10)	21 355	52.19%	92.59%	38.7 s
10% → 5% → 3% (`min_abs` = 10)	21 355	52.22%	92.59%	39.3 s
15% → 8% → 4% (`min_abs` = 10)	18 452	45.10%	92.59%	68.6 s
20% → 10% → 5% (`min_abs` = 10)	15 395	37.62%	92.59%	14.5 s

Table 11
Effect of maximum pattern length (`passes` 10% → 5% → 2%, `SESSION_GAP_MIN` = 15, number of sessions = 359)

`maxlen`	Removed	Reduction	Recall	Runtime
5	15 417	37.68%	92.59%	4.5 s
8	17 276	42.22%	92.59%	37.5 s
10	18 095	44.22%	92.59%	137.2 s
15	18 883	46.15%	92.59%	537.0 s

Increasing pattern length raises reduction from 37.7% to 46.2%, but runtime increases from 4.5 s to 537.0 s, i.e. by more than two orders of magnitude, while reduction grows by only about 8 percentage points. Recall remains unchanged, since the loss of two attacks always occurs in the second pass regardless of pattern length. The gain in reduction is therefore strongly diminishing, and `maxlen` = 8 appears to be a reasonable operational setting; higher values bring only marginal filtering improvement at much higher cost.

Iterative PrefixSpan is the best compromise among the variants: it achieves up to 52.2% reduction at 92.59% recall and runtime in the order of seconds. The loss of two attacks is systematic and independent of parameter settings, which makes this variant predictable and robust.

`PrefixSpan with Sliding Window` segments the dataset using overlapping windows of fixed length. Frequent patterns are mined over all windows, and the records covered by them are removed.

The recall of this variant is constant at 100% (27 of 27) across all configurations. The reason is that overlapping windows artificially increase the support of every token, since the same record belongs to multiple windows, so rare attack tokens never reach the support threshold and never become part of a frequent pattern. The price for preservation of all annotated attack records in the evaluated dataset is a lower reduction rate.

Table 12
Effect of minimum support (`win` = 50, `step` = 10)

Support	Number of patterns	Removed	Reduction	Recall	Runtime
10%	1 242	16 481	40.28%	100.00%	3.8 s
15%	97	11 191	27.35%	100.00%	0.9 s
20%	11	7 557	18.47%	100.00%	0.5 s
30%	1	1 080	2.64%	100.00%	0.4 s

Table 13
Effect of window size (`step` = 10, support = 10%)

Window	Number of patterns	Removed	Reduction	Recall	Runtime
20	53	10 681	26.10%	100.00%	0.5 s
30	221	13 575	33.18%	100.00%	0.8 s
50	1 242	16 481	40.28%	100.00%	3.8 s
75	16 977	17 724	43.32%	100.00%	53.8 s

Table 14
Effect of step size / overlap (`win` = 50, support = 10%)

Step	Overlap	Number of patterns	Removed	Reduction	Recall	Runtime
5	90%	1 298	17 944	43.85%	100.00%	8.4 s
10	80%	1 242	16 481	40.28%	100.00%	3.8 s
25	50%	1 176	13 389	32.72%	100.00%	1.6 s
50	0%	1 070	9 154	22.37%	100.00%	0.9 s

Table 15
Effect of maximum pattern length (`win` = 50, `step` = 10, support = 10%, `minlen` = 2)

`maxlen`	Number of patterns	Removed	Reduction	Recall	Runtime
5	650	15 063	36.81%	100.00%	2.0 s
7	1 002	16 104	39.36%	100.00%	3.2 s
10	2 167	16 928	41.37%	100.00%	6.2 s
15	4 615	16 931	41.38%	100.00%	13.7 s

Reduction increases with lower support, larger windows, and higher overlap, meaning a smaller step. Maximum pattern length saturates; the last two configurations achieve practically identical reduction, while runtime doubles.

PrefixSpan with a sliding window achieves 100% recall on the evaluated dataset, which is a desirable forensic property. The best setting (`win` = 50, `step` = 5, support = 10%) achieves 43.85% reduction in 8.4 s. This variant is ideal where no trace may be lost, even at the cost of lower reduction.

Table 16
Comparison of PrefixSpan variants

Variant	Reduction	Recall	Runtime
With preprocessing	64.59%	55.56%	657.0 s
Iterative	52.19%	92.59%	38.7 s
Sliding window	43.85%	100.00%	8.4 s

The three variants illustrate a fundamental trade-off between reduction and forensic integrity. Preprocessing maximizes reduction, but deduplication lowers recall to 55.6%. The sliding-window variant achieves 100% recall on the evaluated dataset, but with the lowest reduction. The iterative variant is the most balanced. The common conclusion is that PrefixSpan mining itself is lossless; the preservation of attack records depends exclusively on sequence preparation and segmentation, not on the mining phase.

3.5. SPADE

The SPADE algorithm was included as the third experimentally evaluated method in order to test a different data representation approach, namely a vertical database. Unlike GSP and PrefixSpan, which scan the database horizontally, SPADE represents each item as a list of pairs (sequence ID, occurrence time). In this vertical format, candidate support can be computed by simple intersections of occurrence lists without repeatedly scanning the entire database.

Decreasing this parameter causes an exponential growth in the number of discovered patterns, which is a direct consequence of the anti-monotonic property of support. While a lower threshold increases the filtering power, aggressively low values below 8% already threaten recall due to accidental overlap between routine patterns and attack records. A threshold in the range of 8–10% appears to be the best compromise, removing approximately 40% of the log while preserving 100% recall. The target reduction of 80% was not achieved in any test, and cumulative coverage saturates at approximately 58%. This limit is not determined by parameter settings, but by the data structure itself: after high-frequency loops are filtered out, a large number of heterogeneous and unique tokens remain that cannot be captured by sequential patterns.

Table 17
Effect of minimum support on filtering results (`win` = 20, `step` = 10, `MAX_GAP` = 10)

Support	Number of patterns	Removed	Reduction	Recall	Runtime
5%	929	23 908	58.43%	96.30%	12.8 s
8%	182	17 393	42.51%	100.00%	4.8 s
10%	81	16 195	39.58%	100.00%	3.6 s
15%	13	6 115	14.94%	100.00%	2.9 s
20%	6	5 014	12.25%	100.00%	2.8 s

Increasing the sliding window size causes a strong combinatorial expansion in the number of discovered patterns and acts through two opposing mechanisms. Since minimum support is computed as a percentage of the number of sequences, a larger window reduces the sequence base, thereby lowering the absolute support threshold and allowing longer chains to appear in the results. At the same time, the longer input vector contains orders of magnitude more event combinations, which leads to a massive growth in the number of candidates and thus in computational cost.

Table 18
Effect of window size (`step` = `win`/2, support = 10%, `MAX_GAP` = 10)

Window/Step	Number of sequences	Number of patterns	Removed	Reduction	Recall	Runtime
10/5	8 182	16	9 913	24.23%	100.00%	6.9 s
20/10	4 090	76	16 195	39.58%	100.00%	5.7 s
30/15	2 726	350	18 281	44.68%	100.00%	8.3 s
50/25	1 635	5 533	23 963	58.56%	96.30%	112.8 s

The experiments showed that the degree of window overlap has no effect on filtering quality: all three configurations with 75%, 50%, and 0% overlap achieved identical reduction of 39.58% at 100% recall. The reason is twofold. First, the definition of minimum support means that the “artificial” inflation of absolute occurrences caused by duplicated events does not change the core set of frequent patterns. Second, the separation of phases matters: mining is performed on the duplicated dataset, but coverage and removal are evaluated on the original chronologically ordered dataset, which is invariant with respect to the step parameter.

The maximum gap determines how many foreign tokens may lie between two consecutive pattern elements for the occurrence to still be counted during the coverage phase. Unlike the previous parameters, it does not affect mining, but only the removal phase.

The number of discovered patterns remains unchanged across the entire series, confirming that the maximum gap does not affect mining, but only how the already discovered patterns are mapped to

Table 19
Effect of step and overlap on filtering results (win = 20, sup = 10%, gap = 10)

Step	Overlap	Number of sequences	Number of patterns	Removed	Reduction	Recall	Runtime
5	75%	8 180	79	16 195	39.58%	100.00%	10.3 s
10	50%	4 090	76	16 195	39.58%	100.00%	5.7 s
20	0%	2 045	81	16 195	39.58%	100.00%	3.6 s

Table 20
Effect of maximum gap on filtering results (win = 20, step = 10, sup = 10%)

Gap	Number of patterns	Removed rows	Reduction	Recall	Runtime
0	76	13 947	34.09%	100.00%	5.8 s
2	76	15 534	37.96%	100.00%	5.7 s
5	76	15 974	39.04%	100.00%	5.6 s
10	76	16 195	39.58%	100.00%	5.7 s
20	76	16 379	40.03%	100.00%	5.6 s

the log. With zero gap, only strictly contiguous occurrences are accepted, while larger gaps make the matching more permissive and cover more rows. However, the gain is diminishing and saturates at about 40%, since the dominant high-frequency loops are already mostly contiguous and a larger gap only captures a few additional interleaved occurrences.

Minimum support is the strongest factor influencing the amount removed and, together with window size at extreme settings, the only parameter that can affect recall. Window size is the second strongest factor, but it encounters unfavorable runtime scaling. Maximum gap is a fine-grained, monotonic, saturating, and recall-safe parameter. Step size has no impact on detection quality and affects only computational time. The configuration that maximizes cleaning without sacrificing detection completeness corresponds to the baseline setting (win = 20, step = 10, support = 8–10%, MAX_GAP = 10), with approximately 40–42% of rows removed at 100% recall. The target threshold of 80% is structurally unattainable for this dataset; the observed upper bound of removability around 58% is determined by the nature of the data, not by parameter choice.

3.6. Results

GSP achieved the highest reduction rate, removing 80.81% of records in the baseline configuration at 85.19% recall. SPADE, in contrast, provided the highest forensic reliability; at a support threshold of 8–10%, it removed approximately 40% of the log while preserving 100% recall. PrefixSpan proved to be variable depending on sequence preparation: its iterative variant offers the most balanced compromise, with 52.19% reduction, 92.59% recall, and runtime below 40 s, whereas the sliding-window variant achieves 100% recall on the evaluated dataset at the cost of lower reduction, and the preprocessing variant maximizes reduction to 64.59% but lowers recall to 55.56%.

Table 21
Comparison of algorithms at representative settings

Algorithm (setting)	Reduction	Recall	Runtime
GSP (win = 20, step = 10, sup = 1%, max_k = 10)	80.81%	85.19%	302.3 s
GSP (win = 10, step = 10, sup = 1%)	74.84%	88.89%	6.2 s
PrefixSpan – iterative (last threshold sup $\geq$ 10)	52.19%	92.59%	38.7 s
PrefixSpan – sliding window (win = 50, step = 5, sup = 10%)	43.85%	100.00%	8.4 s
SPADE (win = 20, step = 10, sup = 10%, MAX_GAP = 10)	39.58%	100.00%	3.6 s
SPADE (win = 50/25, sup = 10%)	58.56%	96.30%	112.8 s

Several conclusions follow from the results. First, mining frequent patterns itself does not endanger attack records. The attacks are minor and unique in the log, never reach the support threshold, and therefore do not become removable patterns. Recall loss therefore does not arise during mining, but

either during preprocessing (deduplication in PrefixSpan) or through accidental overlap between a short pattern and an attack during the coverage phase (SPADE at aggressively low support). The practical implication is clear: for forensic deployment, it is safer to avoid aggressive preprocessing and sacrifice part of the reduction than to compromise forensic integrity.

Second, separating the mining phase from the coverage phase explains the seemingly surprising behavior of the parameters. Mining parameters (`support`, `win`, pattern length) change both the number of patterns and the reduction, whereas the coverage parameter (`MAX_GAP`) leaves the number of patterns unchanged and only fine-tunes the amount removed. A special case is the window step: in SPADE it is completely invariant with respect to reduction, because removal is performed on the original, non-duplicated log, while in GSP the step is the most effective tool for increasing reduction without loss of recall. This shows that the effect of the same parameter cannot be generalized across algorithms and must be interpreted in the context of their mechanism.

Third, across all algorithms, a strongly diminishing return was confirmed when increasing the window size and maximum pattern length. The number of patterns and runtime grow by orders of magnitude, while reduction improves only marginally. The lowest computational cost while preserving quality is therefore achieved with smaller windows, limited mining depth, and higher overlap.

Fourth, the central trade-off of the whole methodology is between reduction rate and detection completeness. No algorithm achieved both high reduction and 100% recall. GSP was the only method to exceed the indicative 80% removal threshold inspired by the Pareto principle, but at the cost of losing four attacks and incurring high computational overhead, while the algorithms with 100% recall (SPADE and PrefixSpan sliding window) remained below 45%. The analysis of SPADE showed that the 80% target is structurally unattainable for this dataset. Cumulative coverage saturates at around 58%, because the normal activity in the examined log is not repetitive enough and a substantial part consists of heterogeneous, unique tokens that no sequential pattern can cover. The upper bound of removability is therefore determined by the nature of the data, not by parameter choice, and the Pareto 80/20 ratio should be understood as a heuristic starting point rather than an attainable goal for any log.

4. Conclusion

This study designed and experimentally validated a methodology that uses sequential pattern mining to identify routine event sequences in Windows Event Logs and reduce the volume of records requiring manual forensic analysis. The proposed pipeline tokenizes raw logs into sequence form, mines frequent patterns representing normal system behaviour, and removes records covered by those patterns. Evaluated on the annotated CTF dataset The Stolen Szechuan Sauce using algorithms PrefixSpan, GSP, and SPADE. The method revealed a clear trade-off between reduction and forensic completeness: GSP achieved the highest reduction, while SPADE provided the best forensic reliability, and PrefixSpan offered the most balanced compromise. Mining frequent patterns itself did not endanger attack records; recall loss arose mainly from preprocessing or from accidental overlap between short patterns and attacks during coverage, and the approach remains highly interpretable because each removed record can be justified by a specific frequent pattern.

Future work includes validation on larger and more diverse datasets, enrichment of event tokens with additional attributes, integration of frequency-based filtering with detection of high-volume attacks, and comparison with closed or maximal pattern mining methods to reduce redundancy while preserving coverage.

Acknowledgments

This paper was supported by the Slovak Research and Development Agency under contract No. APVV-23-0137.

Declaration on Generative AI

During the preparation of this work, the authors used Perplexity in order to: text translation and citation management. After using these service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] N. Moustafa, Digital Forensics in the Era of Artificial Intelligence, 1 ed., CRC Press, Boca Raton, 2022. doi:10.1201/9781003278962.

[2] M. Lefoane, I. Ghafir, S. Kabir, I. U. Awan, Sequential pattern mining: A proposed approach for intrusion detection systems, in: Proceedings of the 7th International Conference on Future Networks and Distributed Systems (ICFNDS '23), ACM, New York, 2024. doi:10.1145/3644713.3644803.

[3] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Magazine 17 (1996) 37–54.

[4] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, R. Thomas, A survey of sequential pattern mining, Data Science and Pattern Recognition 1 (2017) 54–77.

[5] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3 ed., Morgan Kaufmann, Waltham, 2011.

[6] C. Antunes, A. L. Oliveira, Generalization of pattern-growth methods for sequential pattern mining with gap constraints, in: Machine Learning and Data Mining in Pattern Recognition (MLDM 2004), Springer, Berlin, 2004, pp. 239–251.

[7] K. Zhang, Design of a prefixspan algorithm based on prefix position form, in: ITM Web of Conferences: CSCNS2022, volume 47, 2022. doi:10.1051/itmconf/20224501006.

[8] M. J. Zaki, Spade: An efficient algorithm for mining frequent sequences, Machine Learning 42 (2001) 31–60. doi:10.1023/A:1007652502315.

[9] DFIR Madness, Case 001 – the stolen szechuan sauce, https://dfirmadness.com/the-stolen-szechuan-sauce/, 2020. URL: https://dfirmadness.com/the-stolen-szechuan-sauce/.

[10] P. Sokol, M. Šafranko, S. P. Krišáková, L. Kollárovič, Z. Hennelová, ADFIR forensic dataset, Mendeley Data, 2026. doi:10.17632/65g9gm8zrd.1.