

**ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ**



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

Π.Μ.Σ. ΑΣΦΑΛΕΙΑ ΚΑΙ ΑΝΑΠΤΥΞΗ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Διπλωματική Εργασία

“Process-Aware Insider Threat Detection Using Windows Event Logs.”

Αλέξανδρος Μανδηλαράς

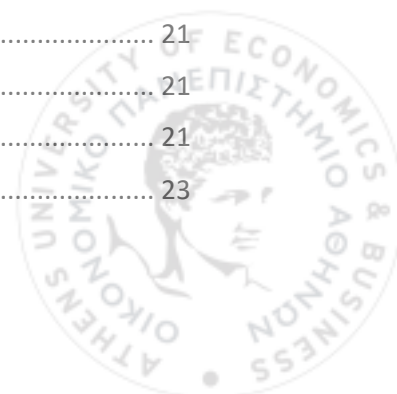
ΑΜ: p3312310

Athens, June 2025

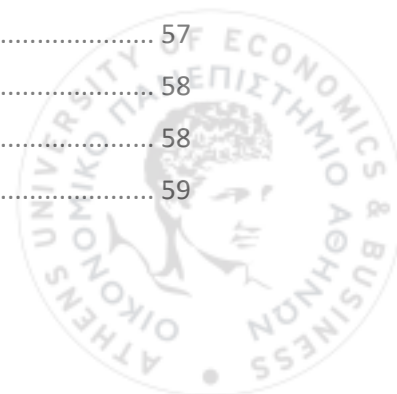


Table of Contents

Περίληψη.....	1
Abstract	2
1. Introduction.....	3
2. Log Analysis	5
2.1 Windows Event Logs.....	5
I. Navigation Panel.....	6
II. Log Panel	7
III. Log View	8
2.2 Sysmon	8
2.3 Audit Policies	10
2.4 Windows Event Forwarding (WEF)	11
3. Insider threat.....	12
3.1 Early Indicators	12
3.2 Process mining influence.....	12
4. Process Mining	14
4.1 Tools.....	14
I. ProM (Process Mining Workbench)	14
II. PM4Py	15
4.2 Methods	15
III. Alpha/Alpha++ Miner.....	15
IV. Inductive Miner	16
V. Heuristic Miner.....	16
VI. Direct-Follows Graph.....	16
5. Methodology	17
5.1 Case study.....	17
5.2 Testing workflow.....	18
5.3 Selected PM techniques	19
5.4 Pre-Implementation Strategy	19
6. Case Study Setup	21
6.1 Systems Setup.....	21
I. Sysmon	21
II. Audit Policies.....	23



6.2 Server Setup	24
I. WEF server	25
6.3 Data Description	25
I. Dataset Preparation	26
II. Dataset Construction	28
III. Noise Cleaning.....	33
6.4 ProM	33
I. Installing ProM	34
II. Using ProM.....	35
III. XES Conversion.....	37
7. Implementation	40
7.1 Code Listing	41
I. Prerequisites	41
II. Log Statistics.....	42
III. Alpha miner.....	46
IV. Inductive miner	46
V. Heuristic miner.....	47
VI. Direct-Follows Graph (DFG).....	47
7.2 L1 Loops.....	48
I. Visualizations.....	49
II. Comparisons	50
III. Incident Response	50
IV. False Positive Detection	51
7.3 Infrequent Events	51
I. Visualizations.....	52
II. Comparisons	54
III. Incident Response	55
IV. False Positive Detection	55
7.4 Outside of working hours	55
I. Visualizations.....	57
II. Incident Response	58
III. False Positive Detection	58
7.5 Flow Diagram.....	59



7.6 Evaluation of results	61
8. Conclusions.....	62
Table of Figures.....	63
References	64
Appendix A – Sample Dataset Statistics in JSON Format.....	66
LogonID only as concept:name	66
LogonID User System as concept:name	69



Περίληψη

Η παρούσα διπλωματική προτείνει μια process-aware μεθοδολογία για την ανίχνευση ύποπτης δραστηριότητας σε εταιρικά συστήματα μέσω ανάλυσης αρχείων καταγραφής συμβάντων (event logs) των Windows. Στόχος είναι ο εντοπισμός εσωτερικών απειλών, μοντελοποιώντας και αναλύοντας τη συμπεριφορά των χρηστών χρησιμοποιώντας τεχνικές process mining. Για τον σκοπό αυτό, τα αρχεία καταγραφής των Windows συλλέγονται από όλους τους χρήστες και Windows συστήματα ενός οργανισμού και μετατρέπονται σε μορφή συμβατή για τις τεχνικές process mining που θα πραγματοποιηθούν. Τα μοντέλα διαδικασιών (process models) παράγονται με εργαλεία όπως το ProM ή με την προγραμματιστική γλώσσα Python και τις αντίστοιχες βοηθητικές βιβλιοθήκες που αφορούν το process mining (PM4PY). Τα μοντέλα αυτά βοηθούν στην οπτικοποίηση και ανάλυση της συχνότητας, του χρόνου και της ακολουθίας συμβάντων που σχετίζονται με συγκεκριμένους χρήστες ή συστήματα. Λιγότερο συχνές διαδικασίες επισημαίνονται για περαιτέρω ανάλυση, βάσει της υπόθεσης ότι οι κακόβουλες ή μη εξουσιοδοτημένες ενέργειες εμφανίζονται σπανιότερα από ενέργειες ρουτίνας που εκτελούνται καθημερινά σε έναν οργανισμό. Οι χρονικές στιγμές (timestamps) θεωρούνται χρήσιμες μόνο όταν αποκαλύπτουν ενέργειες εκτός εργασιακού ωραρίου, οι οποίες αντιμετωπίζονται ως δυνητικά ύποπτες. Επαναλαμβανόμενες δραστηριότητες όπως η εκτέλεση μιας ενέργειας πολλαπλές φορές σε ένα συγκεκριμένο χρονικό διάστημα (L1 loops), βοηθούν στον εντοπισμό επιθέσεων brute-force ή γενικότερα αυτοματοποιημένων επιθέσεων. Απομονωμένες ακολουθίες συμβάντων χωρίς λογική σύνδεση με άλλα συμβάντα μπορεί επίσης να υποδεικνύει προσπάθειες μη εξουσιοδοτημένης πρόσβασης ή κακή χρήση του συστήματος. Τα αποτελέσματα δείχνουν ότι η μέθοδος μπορεί να αναδείξει αποτελεσματικά ανωμαλίες που συνδέονται με μια ύποπτη δραστηριότητα. Με την συλλογή των συμβάντων σε ένα κεντρικό σημείο και την αυτοματοποιημένη επεξεργασία τους, η προσέγγιση αυτή προσφέρει μια πρακτική πορεία προς την παρακολούθηση ύποπτων δραστηριοτήτων και απειλών μέσα σε ένα εταιρικό περιβάλλον σε πραγματικό χρόνο. Μετέπειτα έρευνες, πέρα από το score της παρούσας εργασίας, μπορούν στη συνέχεια να ξεκινούν κατά περίπτωση.



Abstract

This paper proposes a process-aware methodology to detect suspicious activity in enterprise systems through the analysis of Windows event logs. The goal is to identify insider threats by modeling and analyzing user behavior using process mining techniques. To achieve this, Windows event logs are collected from all users within an organization and transformed into a process mining-compatible format. The process models are generated using ProM tools or custom Python scripts. These models help visualize and analyze the frequency, timing, and sequence of events tied to specific users or systems. Less frequent processes are flagged for further analysis, under the assumption that malicious or unauthorized actions are less common than routine activity. Timestamps are only considered useful when identifying actions that are for example outside of working hours, which are treated as potentially suspicious. Repetitive patterns, such as L1 loops, help detect brute-force or automated attacks. Isolated event sequences with no logical connection to other events may also indicate unauthorized access attempts or system misuse. The results demonstrate that this method can effectively highlight anomalies that may be linked to suspicious activity. With centralized log collection and automated processing, this approach offers a practical path toward real-time monitoring of enterprise environments. Follow-up investigations, beyond the scope of this work, can then be initiated as needed.

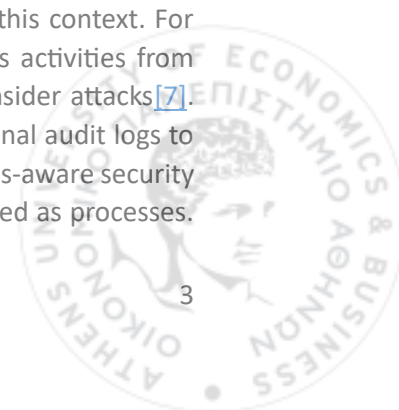


1. Introduction

Insider threats are among the most detrimental cybersecurity challenges facing organizations today. Unlike external attackers, insiders already have trusted access to systems, making their malicious actions much harder to detect[1]. These threats often remain stealthy for long periods, as the perpetrators can leverage legitimate credentials and intimate knowledge of internal processes to evade detection[1]. As a result, insider incidents are on the rise. A recent industry survey reported that 83% of organizations experienced at least one insider attack within the past year[2]. The combination of privileged access and inconspicuous behavior means that insider threats can cause severe damage before being noticed, underscoring the need for more effective monitoring and detection techniques within enterprises.

In modern enterprises, Windows-based systems form a core part of the IT infrastructure, and Windows event logs serve as a primary diagnostic and auditing resource for those systems. Windows event logs record a wealth of information about system and user activities, providing valuable insights when piecing together an incident or suspicious activity[3]. They offer comprehensive logging of application errors, system events, and security-related events. Data which analysts and SOC teams routinely examine for signs of intrusion[3]. Windows organizes its logs into categories such as *Application*, *System*, and *Security* logs, each capturing different event types[3]. For example, the Security log records authentication events and access control changes (e.g. login attempts or file deletion) that can reveal suspicious behavior[3]. Additionally, Windows systems can be augmented with Sysmon (System Monitor) to enhance logging detail. **Sysmon** is a Microsoft Sysinternals tool that extends the standard Windows logs by recording low-level system activities like process creations, network connections, and file modifications[4]. These rich event streams from Windows (including Security, System, Application logs and Sysmon telemetry) are a trove of information for detecting anomalies and have been used in practice for troubleshooting and forensic investigations[3]. The challenge is how to intelligently harness this data to spot the subtle patterns of an insider attack.

Process mining is a relatively young data analysis discipline that offers a promising, process-centric approach to analyzing such event data. It bridges the gap between data mining and process modeling by extracting actual process flows from event logs[5]. In other words, process mining techniques can automatically reconstruct how activities (events) are executed and related, yielding models that describe the “as-is” processes recorded in the logs. This has proven invaluable in business process management for tasks like process discovery, conformance checking, and performance analysis[5]. In the security domain, understanding the sequence and context of events is equally crucial, for instance, to trace the steps an insider took to exfiltrate data or escalate privileges. However, process mining remains an underutilized technique in cybersecurity monitoring, especially for insider threat detection, which is still considered an under-researched area[6]. Only a handful of recent works have begun to explore its potential in this context. For example, Zhu *et al.* developed a system that learns normal profiles of business activities from event logs and flags deviations in the content or order of events as potential insider attacks[7]. Similarly, Macák *et al.* (2020) proposed leveraging process mining on organizational audit logs to help detect insider threats[6]. These early studies highlight the promise of process-aware security monitoring, wherein the behavioral patterns of users or programs can be analyzed as processes.



Nonetheless, such approaches are far from mainstream, and their application to the rich logging ecosystems of Windows environments remains largely untapped.

This thesis aims to fill that gap by introducing a process-aware methodology to detect suspicious activity in enterprise Windows systems through event log analysis. In contrast to prior work that applied process mining to Windows logs for IT troubleshooting (e.g. using event logs to diagnose Windows 10/11 system errors[5]), our focus is explicitly on insider threat behaviors.

To evaluate the proposed methodology in a realistic yet controllable setting, we conducted a focused case study within a small enterprise environment consisting of three Windows-based workstations, each assigned to one employee. All systems run the latest 64-bit Windows 10/11 Enterprise build and are hardened with Microsoft’s recommended “Advanced Audit Policy Configuration.” In addition, Sysmon v15 is deployed with a finely tuned configuration that captures high-value telemetry process creations, network connections, registry modifications, and file hashes, while suppressing obvious noise. Every event source (Security, System, Application, and Sysmon) is forwarded in real time via Windows Event Forwarding (WEF) to a central collector hosted on a hardened Windows Server 2022 machine. Upon arrival, the logs are routed into separate source-specific datasets, where a preprocessing stage applies noise filtering, and enrichment to prepare the data for analysis. The cleaned datasets are then converted to the XES format and ingested by a Python pipeline built on PM4Py that runs discovery, conformance, and anomaly-detection algorithms to uncover behavioral deviations indicative of insider activity.

The remainder of this thesis is organized as follows. **Chapter 2** presents a detailed examination of Windows event logging mechanisms, including the structure of event records, the role of Sysmon, audit policy configuration, and the use of Windows Event Forwarding (WEF) for centralized collection. **Chapter 3** introduces the concept of insider threats, highlighting behavioral patterns and early technical indicators. **Chapter 4** provides a comprehensive overview of process mining, including the theoretical background, tools used (ProM and PM4Py), and the specific algorithms selected for this study. **Chapter 5** outlines the methodology, from log collection and pre-processing to anomaly detection using process models. **Chapter 6** describes the experimental case study setup in detail, covering the configuration of employee workstations and the WEF server, and the construction and transformation of the event-log dataset. **Chapter 7** presents the practical usage of ProM and Python code to conduct the mining tasks, the implementation results, including visualizations of detected L1 loops, infrequent events, and out-of-hours activities, along with incident response strategies and false-positive mitigation. Finally, **Chapter 8** concludes the thesis by summarizing the contributions, reflecting on the findings, and suggesting directions for future work.



2. Log Analysis

In this section we take a deep dive into how Windows logs are generated, where they reside, which add-ons should be installed, and which settings must be configured before exporting them for dataset creation. It's important to understand how Windows stores and manages its event data, and how to boost the logging of insider security-relevant events while filtering out routine noise that merely clutters the log space.

2.1 Windows Event Logs

Windows Event Logs are the built-in records of system and security activity in Windows, accessible through the Event Viewer. Event Viewer organizes these logs into separate channels or “folders” such as Application, Security, and System[8]. Each event entry follows a structured schema that includes fields like an Event ID (a numeric identifier for the type of event), the event Source (the component or application that generated the log), a Task Category, the timestamp of the event, and the user account associated with the event[3]. Different logs capture different aspects of the system's behavior – for example, the Security log records security-related events (such as logon attempts and file access audits) whereas the System log reports OS-level events (such as driver failures)[5]. This separation is particularly useful for insider threat monitoring, as it allows analysts to examine various event streams (authentication events, system errors, etc.) in order to spot anomalous activities that might indicate malicious insider behavior.

The following section offers a concise overview of the Windows Event Viewer interface, explaining how logs are organized into categories and detailing the data fields contained in a typical event record. This foundation is essential for the later chapters, where these logs are transformed into a dataset suitable for process-mining analysis.



I. Navigation Panel

The navigation panel of Windows Event Viewer provides a hierarchical view of every channel in which the operating system records events. Working from this panel, an analyst can move rapidly from broad, system-wide logs to narrowly scoped application streams, select a category, and inspect its contents in the main window.

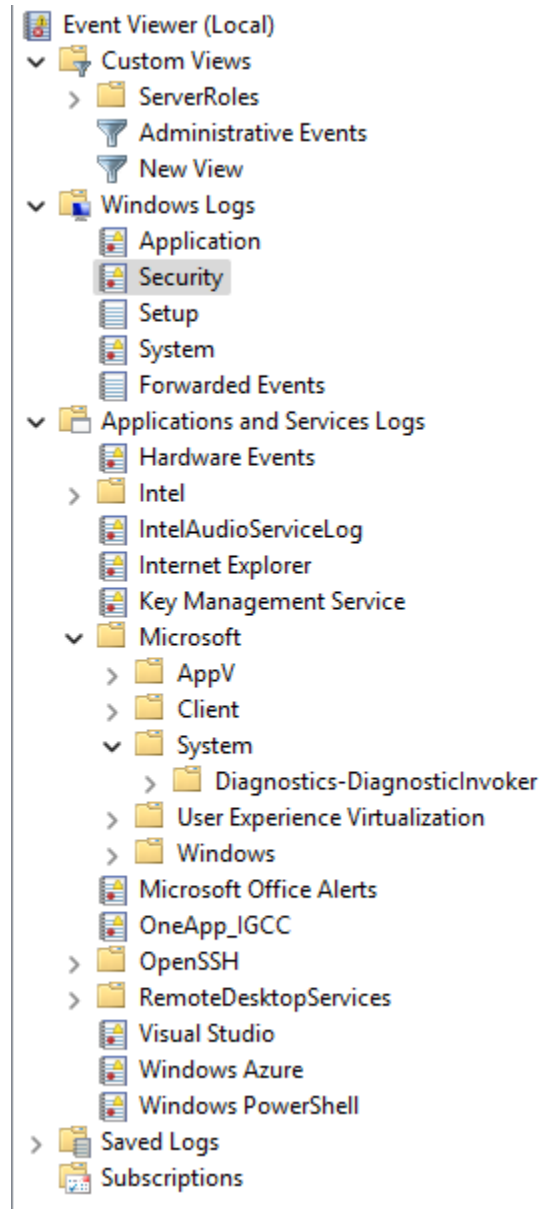
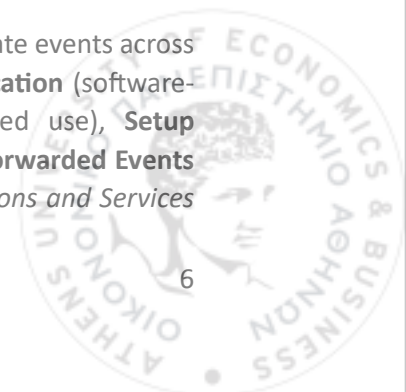


Figure 1: Windows Event Viewer navigation panel.

At the top level, *Custom Views* contains administrator-defined filters that aggregate events across multiple logs. The core logs reside under **Windows Logs**, which includes **Application** (software-generated messages), **Security** (audit events such as logons and privileged use), **Setup** (installation-related activity), **System** (kernel- and driver-level messages), and **Forwarded Events** (logs collected from other hosts via subscriptions). A separate branch, *Applications and Services*



III. Log View

Selecting an individual event opens the detail panel, which exposes the full set of fields recorded for that entry. Although the exact attributes vary by event type, most security-relevant records share a core group of identifiers, including *Security ID* (the user or service account's SID), *Account Name* and *Account Domain*, and a unique *Logon ID* that links related events within the same session. These common fields provide the contextual anchors needed to correlate events across different categories and reconstruct a coherent user activity trace.

The screenshot displays the Windows Security event log interface. At the top, a table lists several security events. The selected event, ID 4688, is shown in detail below. The event description reads: "A new process has been created." The details are as follows:

Field	Value
Creator Subject:	[Redacted]
Security ID:	[Redacted]
Account Name:	aman01stres
Account Domain:	CITE
Logon ID:	0x1DA0BD3A
Target Subject:	[Redacted]
Security ID:	NULL SID
Account Name:	-
Account Domain:	-
Logon ID:	0x0
Process Information:	[Redacted]
New Process ID:	0x244
New Process Name:	C:\Windows\System32\mmc.exe
Token Elevation Type:	TokenElevationTypeLimited (3)
Mandatory Label:	Mandatory Label\Medium Mandatory Level
Creator Process ID:	0x292c
Creator Process Name:	C:\Windows\explorer.exe
Process Command Line:	[Redacted]

Below the details, there is explanatory text about Token Elevation Type and a summary of the event:

Log Name: Security
Source: Microsoft Windows security Logged: 5/22/2025 9:59:26 PM
Event ID: 4688 Task Category: Process Creation
Level: Information Keywords: Audit Success
User: N/A Computer: [Redacted]
OpCode: Info

Figure 3: Security event log description.

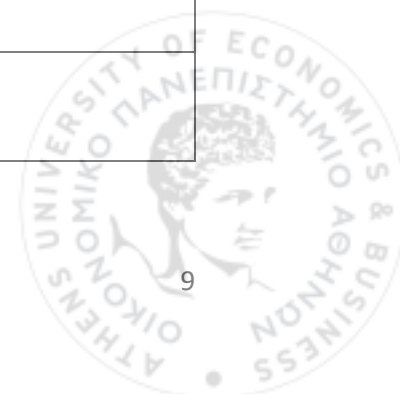
2.2 Sysmon

Sysmon (System Monitor) is an optional Windows system service from Microsoft's Sysinternals suite that extends the default logging capabilities of Windows. Once installed, Sysmon runs continually in the background and records detailed system activity events into a dedicated Windows event log channel. Notably, it logs low-level events that are not captured by standard Windows auditing, including process creation events (with full command-line details), network connection events, modifications to files or registry entries, and other system interactions[9]. This visibility is especially valuable for insider threat detection: Sysmon's logs can reveal subtle malicious behaviors by an insider (for example, an unauthorized program execution or an unusual network data transfer) that would not be apparent from conventional event logs alone. By

supplementing the regular Security and System logs with Sysmon's rich telemetry, an enterprise gains a more comprehensive view of user and process activities, improving the chances of early detection of insider attacks.

Below is a concise map of the extra event types that Sysmon contributes to Event Viewer (with their event IDs)

IDs	Event Types	Description
1	Process Create	Full command line, parent/child linkage, image hash.
2	File Creation Time Changed	Back-dating or time-stomping attempts.
3	Network Connection	Source/destination IPs, ports, protocol, initiating process.
4	Sysmon Service State Changed	Service starts, stops, or crashes.
5	Process Terminate	Orderly or abrupt exits of processes.
6	Driver Loaded	Kernel-mode drivers and their signatures.
7	Image Loaded	DLL or EXE modules mapped into processes.
8	Create remote thread	Cross-process code injection.
9	Raw Disk Read	Direct sector reads that bypass the filesystem.
10	Process Access	OpenProcess, WriteProcessMemory, etc.—useful for privilege-escalation traces.
11	File Create	New files on disk (with hash and path).
12-14	Registry Events	Key/value create, modify, delete.

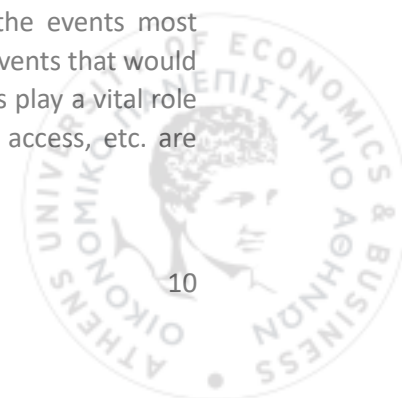


15	File Create Stream Hash	Creation of alternate data streams.
16	Sysmon Configuration Change	On-the-fly rule updates.
17-18	Named Pipe Created/Connected	IPC channels often abused for lateral movement.
19-21	WMI Events	Filters, consumers, and bindings (persistence or reconnaissance).
22	DNS Query	Process-level DNS lookups.
23	File Delete	Tracked deletions (when enabled).
24	Clipboard Change	Copying large volumes of data.
25	Process Tampering	Hollowing, herpaderping, or other image manipulations.
26-27	File Delete Detected / File Executable Detected	Fine-grained file-activity insights.

Together, these additional records give defenders a panoramic, process-centric view of system behavior. Precisely the depth of context needed for process-mining based insider-threat analytics.

2.3 Audit Policies

Audit policies in Windows determine which security-relevant events are logged by the operating system. Administrators configure these policies (via local or group policy settings) to enable logging for critical categories of actions. For instance, recording every user logon/logoff, changes in user privileges or group memberships, and access to sensitive objects, while leaving out less important events. By carefully tuning what is audited, one can capture the important signals (e.g. unauthorized access attempts or privilege escalations) without overwhelming the system with noise [10]. In other words, a well-defined audit policy focuses on recording the events most indicative of misuse or policy violations and filters out the thousands of benign events that would otherwise obscure potential insider threat indicators. These audit configurations play a vital role in insider threat detection (ensuring that suspicious logon patterns, resource access, etc. are tracked) and will be discussed in more detail in later sections.



2.4 Windows Event Forwarding (WEF)

Windows Event Forwarding (WEF) is Microsoft's native, agent-less framework that uses the WS-Management (WinRM) channel to stream any administrative or operational event selected on a Windows host to a designated Windows Event Collector (WEC) server in near real time. In the present implementation WEF constitutes the data-collection backbone. It **de-duplicates** agents and network traffic by applying subscription filters at the source so that only high-value Security, System, Application and Sysmon records traverse the wire. It also **preserves provenance** and cryptographic integrity because the forwarded events remain in their native CSV schema, which is essential for later binding to XES attributes. By centralizing the audit stream on the WEC (Windows Event Collector) server, WEF also establishes a single tamper-resistant evidence store against which the process-mining stage can operate without requiring privileged access to every endpoint, thereby aligning with the least-privilege principle while ensuring comprehensive telemetry coverage.



3. Insider threat

Government guidance treats an insider as anyone who now or previously held authorized access to an organization's people, facilities, information or systems, while an *insider threat* is the possibility that such a person will exploit that access, deliberately or accidentally to cause harm to the organization's mission, assets or reputation. In practice the term covers both malicious actors and well-meaning employees whose mistakes expose the organization to risk.^[24]

Empirical work by the CERT Insider Risk Center shows that malicious insiders usually fall into three broad behavioral patterns: information-technology sabotage, theft of intellectual property, and fraud. Each pattern is driven by a different mix of skill, opportunity and motive, which means the technical footprint of the attack also differs. For example, a saboteur may use privileged scripts to wipe servers, whereas a fraudster manipulates business records. Alongside these explicitly hostile cases, security programs must also contend with negligent users who bypass policy for convenience and with outsiders who take over internal accounts (sometimes called "pseudo-insiders"). Case studies reveal that revenge for perceived workplace injustice and the search for financial gain remain the primary motives in most insider incidents. Competitive advantages, ideological commitment and personal stressors such as debt or pending termination also feature prominently. Security psychologists note that these human factors often surface in HR or performance data weeks before any technical anomaly appears, which is why effective insider-risk programs integrate behavioral as well as system telemetry.

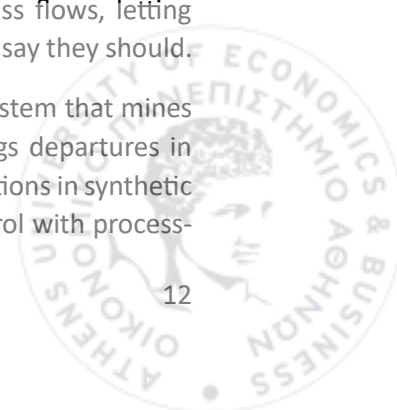
3.1 Early Indicators

Although insider activity is notoriously hard to spot, several recurring warning signs have emerged from incident analyses. Technical indicators include logons from unusual locations or at atypical hours, rapid privilege-escalation attempts, and sudden spikes in data transfers or print jobs. On the human side, unexplained changes in behavior deteriorating performance, escalating conflicts with supervisors, unexpected financial gain or distress, and abrupt plans to resign often precede malicious action. These clues rarely appear in isolation.

3.2 Process mining influence

Insider activity always unfolds as a sequence of low-level actions: a log-on, a registry tweak, a privileged script, an outbound copy. Those actions are faithfully time-stamped in operating-system and application logs, yet traditional analytics inspect counting failed log-ons or watching data-volume thresholds without "asking" whether the order of events tells a different story. Process mining remedies that blind spot by reconstruing raw logs as executable process flows, letting analysts see how insiders actually move through a system instead of how policies say they should.

Early studies illustrate the promise. Zhu et al. built an insider-threat detection system that mines business-process logs, learns normal task sequences for each operator and flags departures in either the content or the order of events, successfully isolating injected insider actions in synthetic cases.^[7] More recently, Japanese researchers combined role-based access control with process-



mining models to uncover real SoD violations in a 1.2-million-event loan-approval dataset, the kind of subtle policy drift that had eluded conventional monitoring.[\[23\]](#)

Taken together, these results suggest that process mining can supply the temporal context and behavioral baselines that insider-threat programmes often lack. The next section therefore delves into the theoretical foundations of process mining, laying the groundwork for the Windows-log methodology introduced later in this thesis.



4. Process Mining

In this section we will talk a little about process mining to start shaping the idea behind the use of this method to detect insider threats. Process mining is a data-driven approach that reconstructs and analyses real-world processes from event logs. Each log entry records at minimum a case identifier, an activity name, and a timestamp. By ordering events that belong to the same case, process-mining algorithms obtain traces that reveal the actual path each process instance followed. Three core capabilities form the discipline: discovery (generating a model purely from the log), conformance (comparing traces with a reference model to quantify deviations), and enhancement (adding performance or resource data to an existing model). These techniques are widely applied in finance, healthcare, customer service, IT service management, and compliance auditing because they convert raw system data into an “as-is” picture of how work truly flows, exposing delays, rework loops, and exceptions.

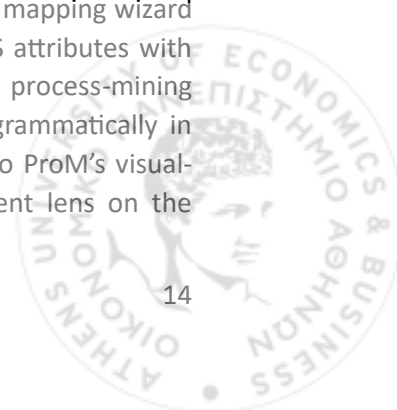
Process mining fits our insider-threat study for two reasons. First, Windows event logs already contain the essential attributes, such as session or logon ID, event type, and timestamp, so the data can be mapped directly into traces without invasive instrumentation besides the required extractions that we made to get the data we want from the description column. Second, anomaly patterns typical of insider misuse naturally manifest in process terms: rapid self-loops signal repeated login failures, isolated fragments expose one-off privilege changes, and timestamp filters isolate activity outside normal working hours. By applying process-discovery algorithms to the transformed logs, we obtain visual models that highlight these deviations in context, making suspicious behavior easy to spot and quantify.

4.1 Tools

1. ProM (Process Mining Workbench)

ProM is an open-source, plugin-oriented platform that has become the de-facto reference environment for research into process-mining algorithms. Its architecture revolves around three artefact types: data objects, plugins, and visualizers, which are all installed and versioned through a built-in package manager. Once an XES log is loaded into the workspace, analysts can invoke discovery plugins, perform conformance checking with alignment diagnostics, or enrich a model with frequency and performance overlays. Because every plugin produces a typed data object, results can be chained: a Petri net discovered by Heuristic Miner can immediately be replayed against the same log, and the replay can then be decorated with bottleneck statistics or converted to BPMN for business stakeholders.

In this thesis ProM serves two complementary purposes that sit upstream and downstream of the automated Python pipeline. First, we rely on ProM’s Import framework to transform the cleansed CSV datasets produced by the ETL stage into fully compliant XES logs. Its built-in mapping wizard lets us bind case identifiers, activity names, and timestamps to the correct XES attributes with minimal manual effort, guaranteeing that the data can be consumed by any process-mining engine. Second, after discovery and conformance checking are executed programmatically in PM4Py, the resulting Petri nets and directly-follows graphs are re-imported into ProM’s visual-analytics plugins. This secondary, interactive inspection acts as an independent lens on the



models, allowing analysts to zoom into low-frequency paths, overlay performance metrics, and cross-check that the structures learned in Python align with domain intuition before finalizing any insider-threat findings.

II. *PM4Py*

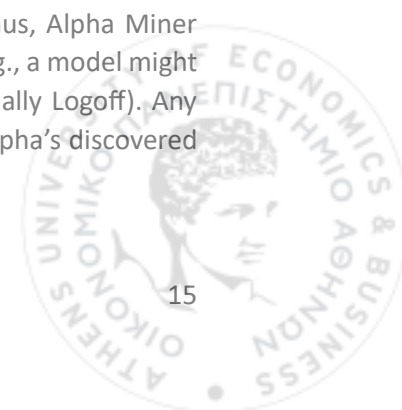
PM4Py brings the core capabilities of ProM into the Python data-science ecosystem. It reads event data directly from XES, CSV, or pandas DataFrames, and exposes one-line wrappers for the major discovery algorithms. Because models and logs are ordinary Python objects, users can pass them to NumPy for numerical analysis or to Matplotlib for bespoke visualizations; the library therefore sits naturally inside modern ETL and machine-learning pipelines.

Within our methodology PM4Py functions as the computational core of the detection job. Once the cleaned datasets are converted to XES by ProM, the log files are loaded into a Python pipeline that couples PM4Py with standard data-science libraries. The script runs several discovery algorithms in sequence such as Alpha++, Inductive Miner, Heuristics Miner, and a Direct-Follows Graph extractor with each generating a process model that captures the day's activity from all users with a different analytical angle. The resulting Petri nets and DFGs are rendered to PNG/SVG for quick visual inspection. Analysts then review the models for tell-tale insider-threat patterns such as short self-loops of failed logons, rare privilege-escalation branches, or off-hours execution paths. Any trace whose conformance score or structural context breaches a configurable threshold is flagged and forwarded to the central incident response team for triage. Because the entire workflow is code-driven rather than GUI-based, it runs unattended, can be version-controlled, and is easily re-parameterized to refine detection sensitivity as the organization's threat landscape evolves. Together, **ProM** and **PM4Py** let us move seamlessly from interactive exploration to industrial-scale automation: ProM handles log conversion and visual scrutiny, while PM4Py drives the nightly discovery and conformance checks over multiple Windows events generated each week. Together they fulfil the study's single overriding aim, which is detecting insider threats accurately while keeping false-positive noise to a minimum.

4.2 Methods

III. *Alpha/Alpha++ Miner*

The Alpha algorithm is one of the earliest process discovery methods, which infers a Petri net by looking at direct succession and concurrency patterns in the log. It works well on noise-free, simple processes but struggles with complex, real-world data[11]. In the context of Windows logs, pure Alpha Miner may be too brittle. Real user behavior often includes concurrent events (overlapping actions), loops, and infrequent deviations that violate Alpha's assumptions. Thus, Alpha Miner might serve as a baseline to get a simple model of the main flows of activities (e.g., a model might show that typically, a Logon is followed by some file access events and eventually Logoff). Any behavior not captured (or not fitting) this simple model could be flagged, but Alpha's discovered model may have low fitness if the log is noisy.



Alpha++ miner on the other hand is an enhanced variant of the classic Alpha family that retains the intuitive succession-based reasoning of the original algorithm while addressing many of its well-known shortcomings. By incorporating additional causal tests, improved noise-tolerance rules, and a systematic treatment of short loops, Alpha++ is able to discover larger classes of Petri nets, including models with complex parallelism, length-one and length-two loops, and certain infrequent but legitimate paths that standard Alpha would either misrepresent or discard.

IV. *Inductive Miner*

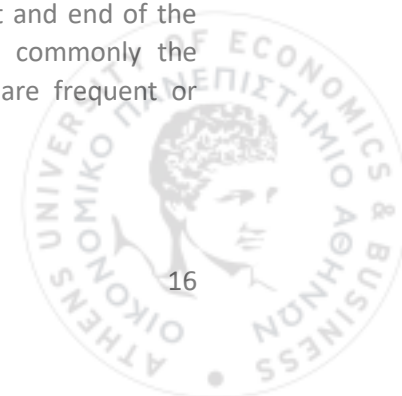
Inductive miner produces sound process models (often Petri nets or process trees) that represent the log behavior and always yield a consistent model. It recursively decomposes the log and can handle complex structures like parallelism and optional paths. For insider threat use, an Inductive Miner can be used to discover a normative process model of user activity while ignoring infrequent traces. The resulting model is usually a good representation of normal behavior flows, and since it's sound, we can perform conformance checking against it.

V. *Heuristic Miner*

The Heuristics Miner is more robust to noise and frequency variations. It looks at the dependency frequencies between events (how often one event follows another) and filters out infrequent paths and activities as noise. This is valuable for insider threat detection: **common behavior** will form the main process model, while **rare behaviors** (potentially malicious) are filtered out of the model as exceptions. For example, if normally users rarely print sensitive documents or change passwords, those events might not appear strongly in the discovered model and thus stand out as isolated edges or be omitted entirely. HM can also detect short loops and concurrency: for instance, it can identify if an event tends to repeat back-to-back (loop) or if order varies [12]. An *L1 loop* (an event repeating immediately) in a trace might indicate retried actions. If the heuristic miner model normally doesn't include such loops, the presence of a loop of login attempts in a trace would be an anomaly.

VI. *Direct-Follows Graph*

The Directly-Follows Graph (DFG) is a fundamental representation in process mining that captures the observed sequential flow of activities in an event log [13]. In a DFG, each node represents an activity (a distinct event type) and a directed edge from one node to another indicates that the first activity is directly followed by the second activity in at least one recorded case. This graph-based model is typically derived by scanning process execution traces and linking activities in the order they occur, often designating special source and sink nodes for the start and end of the process. Each edge can be annotated with quantitative information, most commonly the frequency of the direct follow relation, providing insight into which paths are frequent or infrequent.



5. Methodology

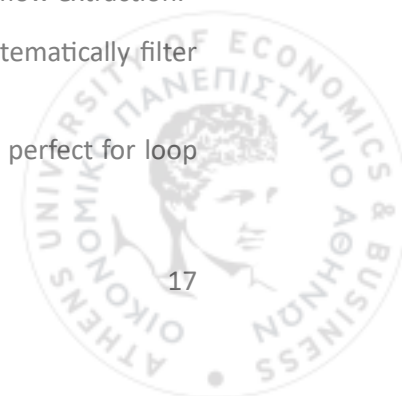
The core objective is to develop a methodology for systematically collecting Windows event logs, preprocessing and transforming these logs into suitable event traces, and then mining them to identify unusual or suspicious patterns of activity. By applying process mining techniques, such as process model discovery to capture normal usage workflows, and conformance check to detect deviations, the methodology will uncover anomalous process flows that may indicate malicious insider actions or other security violations. Through this process-centric analysis of Windows event logs, this thesis seeks to demonstrate a novel approach for insider threat detection that integrates the temporal and contextual information in logs into a coherent behavioral model. Ultimately, the proposed methodology will show how leveraging Windows logs in a process mining framework can enhance the detection of stealthy suspicious activities in enterprise systems, contributing a new dimension to cybersecurity monitoring and insider threat defense. The following sub section describes the case study in which the methodology was applied on.

5.1 Case study

The study follows a single-case embedded design: a hardened but otherwise typical SME network (three Windows 10/11 Enterprise workstations and one Windows Server 2022 collector) is instrumented for two continuous weeks. The constrained scope enables deep inspection of every event while still generating sufficiently heterogeneous traces to exercise the mining pipeline. All hosts are configured with Microsoft's *Advanced Audit Policy* baseline, and Sysmon is installed with a tuned configuration that captures high-value telemetry yet suppresses obvious noise. Event sources (Security, System, Application and Sysmon) are forwarded in real time via Windows Event Forwarding (WEF) to the central collector. Logs are exported daily as CSV to preserve the verbose *Description* field. A Python script loads each file into a *pandas DataFrame* and extracts four key attributes with regular expressions System, User, Logon ID and a one-sentence Event summary, and placing them in dedicated columns. Entries lacking a user or logon identifier, or containing malformed timestamps, are dropped to prevent trace fragmentation. Routine background events (e.g. service health checks) are also filtered out, producing an analyst-ready dataset of actionable, user-related activities. The cleansed CSV is imported into ProM's CSV-to-XES wizard, where *System + User + Logon ID* form the *case:concept:name*, Event maps to *concept:name*, and the original timestamp is bound to *time:timestamp* (format M/d/yyyy H:mm). The resulting XES file is then re-ingested by PM4Py to exploit Python's data-science stack while retaining XES semantics.

Four (selected) complementary discovery algorithms are applied sequentially with default parameters unless stated otherwise:

- **Alpha++ Miner** – baseline, noise-intolerant Petri nets for simple control-flow extraction.
- **Inductive Miner (IM-f)** – sound nets that preserve parallelism and systematically filter spurious behavior.
- **Heuristics Miner** – frequency-weighted causal nets robust to noise and perfect for loop detection.



- **Direct-Follows Graph (DFG)** – frequency and performance views for quick visual triage.

Implementation details and code listings are provided in later chapters/sections.

In this context, the detection focuses on three specific behavioral indicators associated with insider threats: **L1 loops**, such as repeated failed logon attempts, **infrequent events**, which may signify abnormal or one-off actions like privilege escalation or account manipulation, and **out-of-hours** activity, defined as any action occurring outside the organization's standard operating window. These indicators are systematically derived from the mined models and form the basis of the rule-driven anomaly detection logic implemented in later stages.

Because ground-truth labelling of genuine insider incidents is rare, controlled injections were scripted: e.g. 20 failed logons in 60 s, creation of a dormant admin account, and file access outside of working hours. False-positive mitigation relies on a triage playbook that enriches alerts with HR shift calendars, VPN logs and physical-access data.

The experiment complies with GDPR and institutional policies. Only synthetic or anonymized user identifiers are stored. No content data (documents, e-mails) are processed. All monitoring is performed on locally owned equipment.

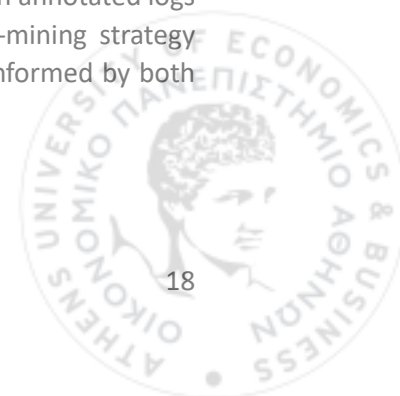
5.2 Testing workflow

Once the event-log dataset has been successfully mined using the selected discovery algorithms, the analysis proceeds through a structured post-processing workflow designed to triage, investigate, and validate potential anomalies. The first step involves visual inspection of the resulting process models, with a focus on identifying structural irregularities such as L1 loops, infrequent event paths, or activity outside defined working hours. These patterns serve as entry points for deeper forensic inspection.

Events or traces flagged as suspicious are then examined in detail using the original event log data, mapped back via timestamps and session identifiers. From here, the incident response workflow is initiated. This includes verifying the legitimacy of the activity (e.g., was the user authorized to perform an out-of-hours access?) and enriching the context with external sources such as VPN logs, HR calendars, and physical access records (e.g., entry badge logs). This correlation step helps distinguish malicious behavior from policy-compliant deviations.

To address false positives, the workflow incorporates a triage mechanism that evaluates anomalies in the context of known benign patterns. For example, automated system updates, IT maintenance activity, or legitimate account provisioning during off-hours may generate unusual traces that are nonetheless authorized. These cases are marked accordingly to prevent alert fatigue in future iterations.

The outcome of this workflow is a prioritized list of validated incidents, along with annotated logs and visual models that support reporting and potential escalation. This post-mining strategy ensures that each anomaly is processed systematically and that decisions are informed by both data-driven insights and organizational context.



5.3 Selected PM techniques

Below there is a table that describes for each incident type, the process mining techniques that were chosen to be applied after testing many process mining methods, the output model and a small description on why we used them.

Incident Type	PM Techniques applied	Output Model	Why this choice
L1 loops	<ul style="list-style-type: none">• Direct-Follows → Workflow Net• Direct-Follows → Frequency view• Heuristics Miner	Sound WF-net, frequency-weighted DFG, causal net	All three keep explicit self-loops while filtering incidental noise.
Infrequent events	<ul style="list-style-type: none">• Direct-Follows → Frequency view• Alpha++ Miner	Frequency-weighted DFG, sound Petri net	Preserves single-occurrence traces yet reduces visual clutter.
Out-of-hours	<ul style="list-style-type: none">• Inductive Miner (IM-f variant)	Sound Petri net	Filters routine daytime traces and isolates after-hours paths for clear contrast.

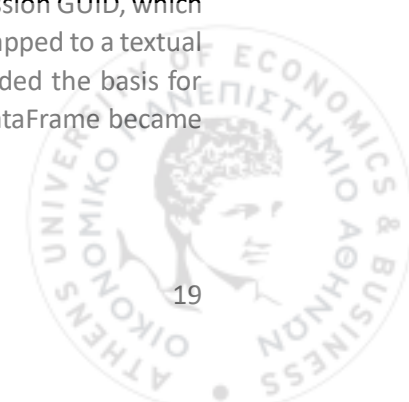
The techniques above strike a balance between model clarity, loop retention, and noise tolerance. They will be applied and analyzed in **Chapter 7**, which is the implementation section.

5.4 Pre-Implementation Strategy

The implementation of the analysis pipeline was guided by a structured and goal-driven design process. From the beginning, the primary objective was to transform Windows event logs into a process mining compatible format, extract behaviorally meaningful patterns, and detect deviations that could indicate suspicious insider activity. The plan was to build this capability in a modular way using Python, combining the power of pandas for data handling with the functionality of PM4Py for process discovery.

The first step in the envisioned workflow was to ingest the raw event log data from CSV format into a pandas DataFrame. This allowed immediate access to tabular operations, filtering, sorting, and regular expression processing. The goal was to make the data analyst-ready while preserving key forensic attributes such as timestamp, event name, user, system, and logon ID.

Next, it was essential to structure the log in a way that aligned with process mining concepts. The idea was to define each trace by combining the System, User, and Logon ID or Session GUID, which together form a unique session-like identifier. Each event within the trace was mapped to a textual summary extracted from the event description. This composite structure provided the basis for generating valid XES (eXtensible Event Stream) cases, where each row in the DataFrame became an event, and traces grouped events by session.



Once structured, the was to convert the pandas DataFrame into XES, the standard input format for ProM and PM4Py miners. This would allow seamless switching between visual inspection in ProM and programmable analysis in Python. Timestamp normalization and field mapping were treated as mandatory steps to ensure that each event had a well-defined concept:name and time:timestamp, both required by process mining engines.

Following data preparation, the next design goal was to apply multiple complementary mining algorithms in sequence to analyze the event traces from different perspectives. These included Alpha++ for basic Petri net extraction, Inductive Miner for noise filtering and parallelism, Heuristics Miner for robustness and loop detection, and Direct-Follows Graph (DFG) for frequency-based exploration. These miners were chosen specifically for their alignment with the case study's goals: detecting L1 loops (e.g., repeated failed logins), infrequent events (potential anomalies), and out-of-hours activity (policy violations).

Each mined model was also intended to be visualized, supporting interpretability and manual triage of suspicious behavior. From the outset, the strategy incorporated calls to the visualizer modules in PM4Py, enabling graphical outputs for Petri nets, process trees, heuristic nets, and directly-follows graphs. This ensured that the final output was not just a numeric or textual anomaly score, but a traceable, auditable representation of user behavior.

Finally, log statistics were included by design, such as the number of traces, distinct variants, and event class frequency, to provide a quantitative foundation for interpreting model complexity and variability. These statistics support the broader analytical framework, allowing anomalies to be judged not only by structure but also by frequency and time distribution.

A rough breakdown of the approach can be seen below in the flow diagram:

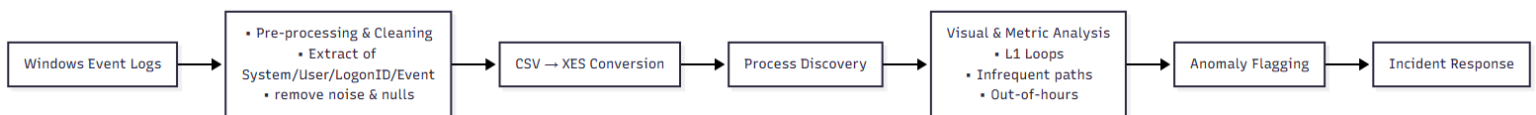


Figure 4: Flow chart of the approach.

6. Case Study Setup

This section presents a comprehensive walkthrough of the experimental environment used to validate the proposed methodology. It begins by detailing how each employee workstation was instrumented through the installation of Sysmon and the enforcement of granular Windows Audit Policies to enable high-fidelity event tracking. Following that, it outlines the deployment of the central collection infrastructure. Specifically, the configuration of a Windows Server instance as a Windows Event Forwarding (WEF) collector. The section then shifts focus to the raw data, describing how Windows event logs were collected, pre-processed, and transformed into an analysis-ready format by extracting key identifiers such as username, system name, and logon session ID, while eliminating background noise. Next, it discusses the installation and use of ProM as a conversion and mining toolkit, with emphasis on the transformation of CSV log exports into XES format. Finally, the section concludes with a listing of the core Python implementation scripts used to perform process mining, including the environment setup, log statistics generation, and the sequence of discovery algorithms applied to the dataset.

6.1 Systems Setup

The first step is to prepare each employee system by installing Sysmon (System Monitor) on each employee workstation to capture detailed event-level telemetry beyond what is available through native Windows auditing alone. Once Sysmon was deployed with a custom XML configuration tailored to reduce noise and retain only security-relevant signals, we proceeded to apply Advanced Audit Policies on each device. These policies were designed to enable critical subcategories such as logon events, privilege use, and object access, ensuring comprehensive coverage of user activity for later analysis.

1. *Sysmon*

Sysmon (System Monitor) is a Windows system service and device driver that logs system activity to the Windows Event Log, providing detailed information on process creation, network connections, driver loading, and more. For this setup, Sysmon was installed on each employee workstation using the official distribution provided by Microsoft Sysinternals. The tool was downloaded from <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>.

After downloading, Sysmon was deployed using the command-line interface with a customized configuration XML file, which defined specific event types to capture while suppressing unnecessary noise. The installation command used was:

- Sysmon64.exe -accepteula -i sysmon-config.xml



```

PS D:\downloads\Sysmon> ./Sysmon64.exe -accepteula -i sysmon-config.xml

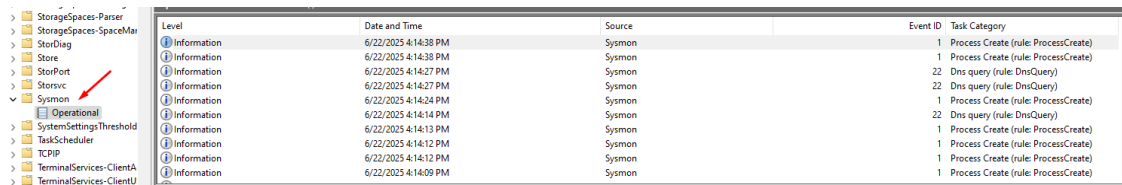
System Monitor v15.15 - System activity monitor
By Mark Russinovich and Thomas Garnier
Copyright (C) 2014-2024 Microsoft Corporation
Using libxml2. libxml2 is Copyright (C) 1998-2012 Daniel Veillard. All Rights Reserved.
Sysinternals - www.sysinternals.com

Loading configuration file with schema version 4.50
Sysmon schema version: 4.90
Configuration file validated.
Sysmon64 installed.
SysmonDrv installed.
Starting SysmonDrv.
SysmonDrv started.
Starting Sysmon64..
Sysmon64 started.

```

Figure 5: Successful installment of Sysmon v15.

Once installed successfully, Sysmon creates a new dedicated log folder in the **Windows Event Viewer**. Below, we can observe that a new folder named "**Microsoft → Windows → Sysmon → Operational**" has appeared, confirming that the service is running and actively recording security-relevant telemetry. This confirms the agent's presence on the system and validates that it is properly integrated with the local event log infrastructure.



Level	Date and Time	Source	Event ID	Task Category
Information	6/22/2025 4:14:38 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:38 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:27 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	6/22/2025 4:14:27 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	6/22/2025 4:14:24 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:14 PM	Sysmon	22	Dns query (rule: DnsQuery)
Information	6/22/2025 4:14:13 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:12 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:12 PM	Sysmon	1	Process Create (rule: ProcessCreate)
Information	6/22/2025 4:14:09 PM	Sysmon	1	Process Create (rule: ProcessCreate)

Figure 6: Sysmon Logs on Event Viewer.

II. Audit Policies

Audit policies in Windows allow system administrators to track and log various security-relevant events, such as logon attempts, privilege use, object access, and system changes. These policies are essential for producing structured, high-quality logs suitable for security monitoring and process mining.

To apply these policies, administrators can access the Local Security Policy interface by running secpol.msc from the Run dialog (Windows + R). Once opened, audit settings can be configured under "**Local Policies → Advanced Audit Policy Configuration → System Audit Policies**"

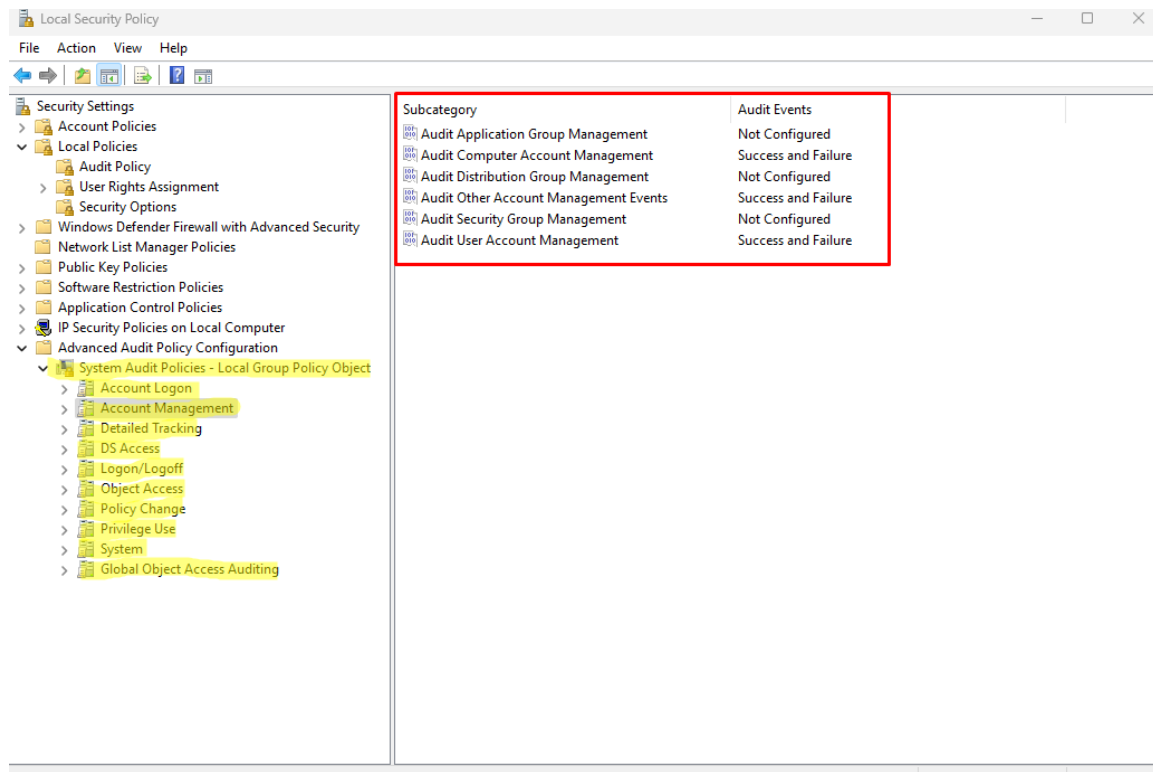


Figure 7: Audit Policy Interface.

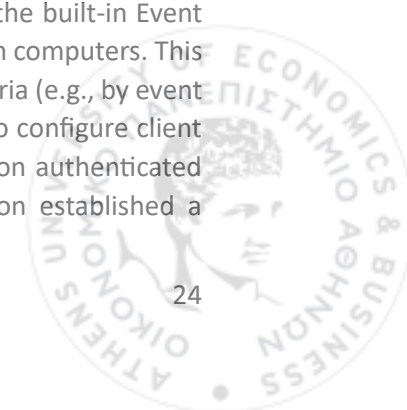
For the purposes of this thesis, the following audit policies were applied:

Account Logon	Audit Credential Validation	Success and Failure
	Audit Computer Account Management	Success
	Audit Other Account Management Events	Success and Failure
	Audit Security Group Management	Success and Failure
	Audit User Account Management	Success and Failure
	Audit Plug and Play Events	Success
	Audit Process Creation	Success
	Audit Directory Service Access	Success and Failure
	Audit Directory Service Changes	Success and Failure
Logon/Logoff	Audit Account Lockout	Success and Failure
	Audit Group Membership	Success
	Audit Logoff	Success
	Audit Logon	Success and Failure
	Audit Special Logon	Success
Object Access	Audit Removable Storage	Success and Failure
Policy Change	Audit Audit Policy Change	Success and Failure
	Audit Authentication Policy Change	Success
	Audit Authorization Policy Change	Success
Privilege Use	Audit Sensitive Privilege Use	Success and Failure
System	Audit IPsec Driver	Success and Failure
	Audit Other System Events	Success and Failure
	Audit Security State Change	Success
	Audit Security System Extension	Success and Failure
	Audit System Integrity	Success and Failure

In summary, these settings capture every event category that can signal privilege misuse or unauthorized access yet avoids the excessive logging that would obscure anomalies in noise. These settings provide the raw, security-rich telemetry required for the insider-threat detection pipeline developed in the following chapters, where the collected events are transformed, mined, and evaluated for suspicious patterns.

6.2 Server Setup

The next phase involved configuring a Windows Event Forwarding (WEF) server on a dedicated Windows Server machine to act as the central log collector for all employee workstations. The WEF setup was based on the native Windows Event Collector (WEC) service, which was first enabled and set to start automatically. A subscription was then created using the built-in Event Viewer interface to define which event types should be collected and from which computers. This included specifying the event channels (e.g., Security, Sysmon), the filtering criteria (e.g., by event ID), and the source endpoints (the employee devices). Group Policy was used to configure client systems to forward their logs to the WEF server, ensuring that each workstation authenticated properly and transmitted events over the WinRM protocol. This configuration established a



reliable, centralized log pipeline, enabling continuous monitoring without the need for agents on each host.

I. WEF server

The Windows Event Forwarding (WEF) server was deployed on a dedicated Windows Server machine, configured to collect event logs from all participating workstations in real time. The foundation of this setup is the **Windows Event Collector (WEC)** service, which must be manually enabled and set to start automatically. This can be done via the Services console (services.msc) or using the following PowerShell

- Set-Service -Name Wecsvc -StartupType Automatic
- Start-Service Wecsvc

With the collector service running, the next step was to configure **Windows Remote Management (WinRM)** to allow secure log transfer. This was accomplished using:

- winrm quickconfig

This command ensures that a listener is active and that the WinRM service is accepting incoming connections from client devices.

The actual log collection is defined through **Subscriptions**, which are created using the **Event Viewer** interface under "**Subscriptions → Create Subscription**". For the purposes of this implementation, a **Collector-Initiated Subscription** was used. This allowed the server to explicitly define which client machines to poll and which events to collect. Primarily security-related logs such as successful and failed logons (4624, 4625), process creation (4688), and Sysmon event IDs.

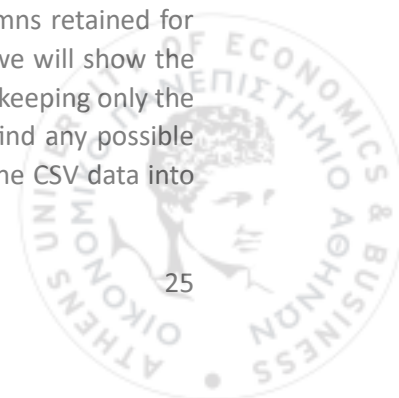
To ensure that client systems forward events correctly, **Group Policy** was applied locally. The relevant policy is located under: "**Computer Configuration → Administrative Templates → Windows Components → Event Forwarding**"

The key setting enabled is "**Configure target Subscription Manager**", where the WEF server's address is specified in the format:

Once applied, clients begin sending events to the WEF server, which stores them in the **Forwarded Events** log visible within Event Viewer. This configuration allows for secure, scalable, and agentless log aggregation, which is ideal for centralized process analysis and anomaly detection.

6.3 Data Description

In this section first we will describe the event-log schema, identifying the columns retained for process mining and clarifying what each attribute serves in the analysis. Next, we will show the process of filtering and cleaning of the data with the goal of removing noise and keeping only the important information needed to execute the process mining techniques and find any possible suspicious or even malicious activity. Finally, we will describe how we convert the CSV data into XES format, rendering the log ready for subsequent process-mining.



Windows Event Viewer supports exporting logs in its native **.evtx** format or as comma-separated values (.csv). The CSV option is selected here because it can be parsed directly with Python and then transformed to an eXtensible Event Stream (XES).

Logs are extracted per category. Security, Application, System, and any additional channels enabled by the audit policy (e.g., Sysmon). This separation allows each dataset to be pre-processed according to its own field conventions. For example, Application logs treat **Source** as the originating program and rarely populate **Task Category**, whereas Security logs use a fixed source name and rely on **Task Category** to label audit sub-types. Sysmon events follow yet another pattern, omitting session identifiers that are present elsewhere. Isolating categories therefore enables category-specific parsing and normalization before the data sets are eventually merged for XES conversion.

I. Dataset Preparation

We will be using as an example the logs from the “Security” category. The first step is to export the logs of a system in CSV format and inspect them. Opening the exported CSV we notice the following columns: “Level”, “Date and Time”, “Source”, “Event ID”, “Task Category”, and an empty column. This column is basically the description of an event log. This column contains a lot of useful information that we will be using for process-mining. The problem here is that we need this information to be in their own separate columns, so it is required to extract them from that column into a new one without destroying anything else. We will call this column the “Description” column. The important information that can be extracted is the system name, the User’s name, the logon ID of the current session when the event was logged, a short sentence that describes what the event log is about and many more.

Every type of Event log varies when it comes to what is included in the description column, so it’s not ideal to say that the same information can be extracted from that column. For example, not every log has a logon ID or user. This usually means the action was executed by the system itself, which in most cases means that it doesn’t hold much value for process mining. The approach here will be to only keep track of the Event IDs that are valuable to us. The list of events that may indicate insider threat or suspicious activity in general is quite big and won’t be listed here fully, but for the sake of understanding we will list some of these event IDs.

Event ID	Category	Description
4624	Successful logon	Spot logons at odd hours, from unusual hosts, or with rare logon types that can signal hijacked or mis-used accounts.
4625	Failed logon	Clusters of failures suggest password guessing or a user probing accounts they shouldn’t access.
4648	Logon with explicit credentials	Indicates “Run As” or scheduled tasks using another account, which is common in credential-theft or lateral moves.
4663	Attempted access to object	Gives file name and access mask. Perfect for catching confidential-file reads by non-owners.

4670	Permissions on object changed	Unauthorized ACL changes pave the way for later data theft without generating access errors.
4672	Special privileges assigned to logon	Flags domain admins or SeDebug privilege; if it's not an admin's machine, investigate.
4674	Operation on privileged object	Detects attempts to tamper with objects protected by privilege-based access control.
4688	New process created	Provides parent/child chain to spot suspicious binaries (e.g., cmd.exe launched from Outlook).
4719	System audit policy changed	Attackers often turn off noisy audits before illicit actions.
4720	User account created	The creation of rogue accounts is one of the clearest insider red flags.
4722	User account enabled	Re-enabling an old account avoids fresh-account scrutiny. 4720 Event ID also triggers this Event automatically.
4723	Attempt to change password (user)	Excessive self-service changes can indicate account takeover recovery efforts.
4724	Attempt to reset password (admin)	Admin resets outside ticketed workflow may be privilege escalation.
4725	User account disabled	Disabling an account after using it can hide audit trails.
4726	User account deleted	Deleting an account erases future evidence; investigate immediately.
4727	Security-enabled global group created	New groups can be used to mass-assign rights without notice.
4740	Account locked out	Repeated lockouts often follow failed brute-force or disgruntled sabotage attempts.
1100	Event logging service shut down	Cutting logging is a classic step to operate undetected.
1101	Audit events dropped	High volume of dropped events can mean intentional log flooding.
1102	Security log cleared	Immediate sign of tampering triggers an alert every time.
1104	Security log full	If not configured to overwrite, logging stops and activity goes dark.

Many more Event IDs exist that are scattered inside different event categories on event viewer. Some may not even be related to insider threat and look innocent at first, but by linking logs based on a single component like "logon ID" for example, process mining can indicate an unusual flow of actions later in the analysis. The complexity of an event can increase significantly based on the context of the case. It's up to the security administrators or engineers of the organization to deem an action suspicious or threatening.



II. Dataset Construction

In this section we will show how the original exported CSV file looks like and the steps that we will follow afterwards to construct the dataset in such a way that it has the necessary information required for process-mining.

To facilitate the preprocessing and construction of the event-log dataset, Python was used as the primary data-handling tool due to its flexibility and powerful data manipulation libraries. The raw Windows event log data, exported in CSV format, was loaded into a Pandas DataFrame, which allowed for structured and efficient processing. Initial steps included loading the file, inspecting its structure, and configuring display settings for clarity within the Jupyter environment. Subsequent steps involved cleaning the data, transforming timestamps into standardized formats, selecting relevant columns (such as Event ID, Timestamp, and User), and renaming them according to process mining schema requirements. A table representing the column names used and their mappings can be seen below:

Column Name	Mapped to column	Purpose
Event	concept:name	Identifies the type of action or event.
Timestamp	time:timestamp	Defines the exact time the event occurred
Logon ID	case:concept:name	Uniquely identifies the session or trace
User	case:concept:name	Used as part of the composite case ID (e.g., System User Logon ID)
System	case:concept:name	Used as part of the composite case ID (e.g., System User Logon ID)

To begin the dataset construction process, Security logs were exported from the Windows Event Viewer in CSV format. These logs primarily consist of routine user and system activity, such as successful logons and logoffs. The exported file was imported into a Jupyter Notebook environment and loaded into a structured DataFrame using Python's pandas library, enabling efficient inspection, filtering, and transformation of the data. In this particular dataset, a number of repeated failed logon attempts were deliberately generated and included, in order to simulate suspicious behavior that can later be analyzed through process mining techniques. A screenshot of the resulting DataFrame is presented below to illustrate the typical schema of a Windows Security event log as represented in tabular format.

Level	Date and Time	Source	Event ID	Task Category	Description
0	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4672	Special Logon	Special privileges assigned to new logon.\r\n\r\n...
1	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4627	Group Membership	Group membership information.\r\n\r\n\r\nSubject:...\r\n\r\n...
2	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4624	Logon	An account was successfully logged on.\r\n\r\n\r\n...
3	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4634	Logoff	An account was logged off.\r\n\r\n\r\nSubject:...\r\n\r\n...
4	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4634	Logoff	An account was logged off.\r\n\r\n\r\nSubject:...\r\n\r\n...
5	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\n\r\nCreator:...\r\n\r\n...
6	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\n\r\nCreator:...\r\n\r\n...
7	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4801	Other Logon/Logoff Events	The workstation was unlocked.\r\n\r\n\r\nSubject:...\r\n\r\n...
8	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\n\r\nCreator:...\r\n\r\n...
9	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4673	Sensitive Privilege Use	A privileged service was called.\r\n\r\n\r\nSubject:...\r\n\r\n...
10	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4672	Special Logon	Special privileges assigned to new logon.\r\n\r\n\r\n...
11	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4627	Group Membership	Group membership information.\r\n\r\n\r\nSubject:...\r\n\r\n...
12	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4624	Logon	An account was successfully logged on.\r\n\r\n\r\n...
13	Information 5/19/2025 14:45:30 PM	Microsoft-Windows-Security-Auditing	4627	Group Membership	Group membership information.\r\n\r\n\r\nSubject:...\r\n\r\n...
14	Information 5/19/2025 12:45	Microsoft-Windows-Security-Auditing	4624	Logon	An account was successfully logged on.\r\n\r\n\r\n...
15	Information 5/19/2025 12:45	Microsoft-Windows-Security-Auditing	4648	Logon	A logon was attempted using explicit credential... A privileged service was called.\r\n\r\n\r\nSubject:...\r\n\r\n...
16	Information 5/19/2025 12:45	Microsoft-Windows-Security-Auditing	4673	Sensitive Privilege Use	A privileged service was called.\r\n\r\n\r\nSubject:...\r\n\r\n...
17	Audit Failure 5/19/2025 12:33	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
18	Audit Failure 5/19/2025 12:33	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
19	Audit Failure 5/19/2025 12:33	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
20	Audit Failure 5/19/2025 12:33	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
21	Audit Failure 5/19/2025 12:33	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
22	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
23	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
24	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
25	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
26	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
27	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
28	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
29	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
30	Audit Failure 5/19/2025 12:34	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
31	Audit Failure 5/19/2025 12:35	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
32	Audit Failure 5/19/2025 12:35	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
33	Audit Failure 5/19/2025 12:35	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
34	Audit Failure 5/19/2025 12:35	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
35	Audit Failure 5/19/2025 12:36	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
36	Audit Failure 5/19/2025 12:36	Microsoft-Windows-Security-Auditing	4625	Logon	An account failed to log on.\r\n\r\n\r\n...
37	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\n\r\n...
38	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
39	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
40	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
41	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	5379	User Account Management	Credential Manager credentials were read.\r\n\r\n\r\n...
42	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
43	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\n\r\n...
44	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
45	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
46	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
47	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...
48	Information 1/11/2025 17:09	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\n\r\n...

Figure 8: Pandas DataFrame of security event logs.

The figure above displays the initial format of the Windows Security Event Log as loaded into a pandas DataFrame. Each row represents a single event record, and the columns include the following:

- **Level:** Indicates the severity or type of the event (e.g., Information, Audit Failure).
- **Date and Time:** The exact timestamp when the event occurred.
- **Source:** The source or component that generated the event, typically Microsoft-Windows-Security-Auditing.
- **Event ID:** A numeric identifier that specifies the type of event (e.g., 4624 for successful logon, 4625 for failed logon).
- **Task Category:** A high-level categorization of the event's purpose (e.g., Logon, Process Creation).
- **Description:** A detailed textual explanation of the event, often including information such as usernames, machine names, and logon IDs.

While the existing columns provide basic structural metadata about each event, several crucial attributes are missing from the dataset in their own dedicated fields. Specifically, the system name, the account name involved in the event, a summary of the event's action in a single sentence, and the logon ID, which is vital for correlating multiple logs into a single logical event. These are not

provided as separate columns. Instead, all this information is embedded within the verbose Description field. To make this dataset suitable for process mining and insider threat detection, this missing information will be extracted from the Description column using regular expressions and string parsing. As a result, four new columns will be created: one each for the system name, user name, event action (i.e., a short label for what occurred), and the logon ID. These additional attributes will enable event grouping, case identification, and more granular behavioral analysis.

User extraction

Assuming the CSV has been imported and transformed into a DataFrame, we then try to extract for every row from the description column the User that was active when an action happened. Then we add the output into a new column inside the existing DataFrame with the name “User”. The output should look like this:

	Level	Date and Time	Source	Event ID	Task Category	Description	User
0	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
1	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
2	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
3	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
4	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
5	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
6	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.\r\n\r\nSubject\r\n\r\nSec...	amandilaras
7	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\nCreator...	LOCAL SERVICE
8	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\nCreator...	amandilaras
9	Information	5/31/2025 18:29	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.\r\n\r\nCreator...	amandilaras

Figure 9: DataFrame with the new User column.

To extract the account name associated with each event, a regular expression was constructed to parse the Description column, which contains structured yet unformatted text. Specifically, the pattern searches for the field labeled "Account Name:" followed by the actual username value. The regular expression used for this task is

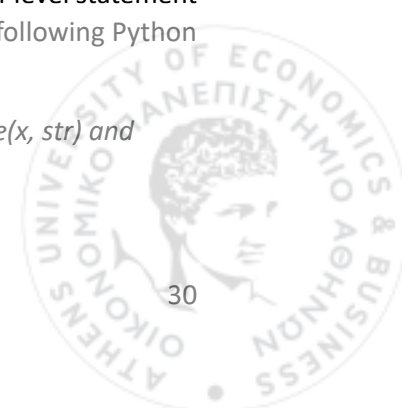
$$Account\s*Name:\s*(\r\n)+$$

This pattern identifies the phrase “Account Name:” (allowing for optional whitespace) and captures all characters following it up to the next line break. The captured value is then assigned to a new column named “User”.

Event extraction

To provide a concise summary of each log entry, the first line of the Description field was extracted and stored in a new column named Event. This line typically contains a brief, high-level statement that describes the nature of the event (e.g., “An account failed to log on.”). The following Python expression was used to perform this transformation:

```
df["Event"] = df["Description"].apply(lambda x: x.strip().splitlines()[0] if isinstance(x, str) and x.strip() else None)
```



This code snippet checks if the value in the Description column is a non-empty string, removes leading and trailing whitespace, splits the text into lines, and extracts the first line. This first line which is always a one-sentence summary of the event gets assigned to the new Event column.

LogonID extraction

A similar approach was applied to extract the **Logon ID**, which is a critical field used to correlate related events across the dataset. The Logon ID is embedded within the *Description* column and follows a consistent pattern labeled as "**Logon ID:**". To retrieve this value, the following regular expression was used:

$$\text{Logon}\backslash s^*ID:\backslash s^*([\r\n]^+)$$

This pattern locates the label "Logon ID:" (again accounting for optional whitespace) and captures the value that follows until the end of the line. The extracted value is then stored in a new column named *LogonID*, which plays a central role in grouping events into sessions for process mining analysis.



System name extraction

The CSV format exported from Windows Event Viewer does not explicitly include the name of the system on which the logs were generated. This information is present only in the XML export format, where it appears within the <Computer> tag of each event entry. However, when logs are collected in an organizational environment, it is typically assumed that each exported log file originates from a known host, especially when centralized log collection mechanisms (e.g., Windows Event Forwarding or SIEM agents) are in place. Therefore, for the purpose of this dataset, it is assumed that the source system of each CSV file is already known. Based on this assumption, the system name can be directly assigned to a new column titled **System** and uniformly applied to all rows within that file to reflect the host from which the logs originated. If we used the XML instead, we would then again use a regex to capture the name from within the <Computer> tag.

	System	User	Level	Date and Time	Source	Event ID	Task Category	Event	Description	Logon ID
0	LAPTOP-21-01	User1	Information	5/1/2025 1:18	Microsoft-Windows-Security-Auditing	4648	Logon	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	0x5a83ac7
1	LAPTOP-21-01	User2	Information	5/1/2025 1:51	Microsoft-Windows-Security-Auditing	4769	Kerberos Service Ticket Operations	A Kerberos service ticket was requested.	A Kerberos service ticket was requested.	0xa9f915d
2	LAPTOP-21-01	User4	Information	5/1/2025 2:29	Microsoft-Windows-Security-Auditing	4648	Logon	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	0x1e49bea
3	LAPTOP-21-01	User3	Information	5/1/2025 2:39	Microsoft-Windows-Security-Auditing	4769	Kerberos Service Ticket Operations	A Kerberos service ticket was requested.	A Kerberos service ticket was requested.	0x28c6c90
4	LAPTOP-21-01	User4	Information	5/1/2025 3:14	Microsoft-Windows-Security-Auditing	4648	Logon	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	0xbf15aff
5	LAPTOP-21-01	User4	Information	5/1/2025 3:40	Microsoft-Windows-Security-Auditing	4768	Kerberos Authentication Service	A Kerberos authentication ticket (TGT) was req...	A Kerberos authentication ticket (TGT) was req...	0x6df8e57
6	LAPTOP-21-01	User1	Information	5/1/2025 5:02	Microsoft-Windows-Security-Auditing	4688	Process Creation	A new process has been created.	A new process has been created.	0x7a261e1
7	LAPTOP-21-01	User1	Information	5/1/2025 5:07	Microsoft-Windows-Security-Auditing	4689	Process Termination	A process has exited.	A process has exited.	0x7a261e1
8	LAPTOP-21-01	User4	Information	5/1/2025 5:53	Microsoft-Windows-Security-Auditing	4769	Kerberos Service Ticket Operations	A Kerberos service ticket was requested.	A Kerberos service ticket was requested.	0x15907be
9	LAPTOP-21-01	User2	Information	5/1/2025 6:05	Microsoft-Windows-Security-Auditing	4624	Logon	An account was successfully logged on.	An account was successfully logged on.	0x3a32b19
10	LAPTOP-21-01	User2	Information	5/1/2025 6:05	Microsoft-Windows-Security-Auditing	4672	Special Logon	Special privileges assigned to new logon.	Special privileges assigned to new logon.	0x3a32b19
11	LAPTOP-21-01	User2	Information	5/1/2025 6:05	Microsoft-Windows-Security-Auditing	4627	Group Membership	Group membership information.	Group membership information.	0x3a32b19
12	LAPTOP-21-01	User1	Information	5/1/2025 6:09	Microsoft-Windows-Security-Auditing	4624	Logon	An account was successfully logged on.	An account was successfully logged on.	0x81dd370
13	LAPTOP-21-01	User1	Information	5/1/2025 6:09	Microsoft-Windows-Security-Auditing	4672	Special Logon	Special privileges assigned to new logon.	Special privileges assigned to new logon.	0x81dd370
14	LAPTOP-21-01	User1	Information	5/1/2025 6:09	Microsoft-Windows-Security-Auditing	4627	Group Membership	Group membership information.	Group membership information.	0x81dd370
15	LAPTOP-21-01	User3	Information	5/1/2025 6:23	Microsoft-Windows-Security-Auditing	4634	Logoff	An account was logged off.	An account was logged off.	0x3a32b19
16	LAPTOP-21-01	User4	Information	5/1/2025 6:23	Microsoft-Windows-Security-Auditing	4634	Logoff	An account was logged off.	An account was logged off.	0x81dd370

Figure 10: The completed dataset.



The final version of the dataset, shown in the figure above, was produced after extracting all relevant attributes from the *Description* field and organizing them into clearly defined columns. These include the **System**, **User**, **Event**, and **Logon ID**, which were parsed and assigned based on the methods previously described. Minor adjustments were made to the column order and formatting to enhance clarity and consistency for the reader. Additionally, any log entries that lacked critical information, specifically the user or logon ID, were excluded from the final dataset, as they do not contribute meaningfully to the detection or reconstruction of potential insider threat scenarios. The resulting structured dataset is now ready to be used in process mining simulations and behavioral analysis. The process of filtering the logs and removing unnecessary noise is discussed in the next section.

III. Noise Cleaning

To improve the quality and relevance of the dataset, a noise cleaning process was applied to remove log entries that do not contribute meaningful behavioral information. First, any logs that lacked a valid Account Name or Logon ID were excluded, as these fields are essential for associating actions with specific users and for grouping related events into cohesive cases. Additionally, events with missing or malformed timestamps were removed to prevent misalignment in temporal sequencing. Logs that represented system-level background operations not tied to a specific user (such as periodic service health checks or audit policy changes without a user context) were also considered noise and filtered out. This cleaning step ensured that the final dataset contains only actionable, user-related events that can be meaningfully analyzed in the context of insider threat detection.

6.4 ProM

In this section, we present the installation and usage of **ProM**, a widely used open-source framework for process mining. We begin by outlining how the ProM tools were installed and configured on the analysis machine, followed by a brief overview of the user interface and its core components relevant to this study. After establishing familiarity with the environment, the section focuses on the transformation process from raw CSV log files to the **XES (eXtensible Event Stream)** format, which is ProM's standard input format for process mining algorithms. This conversion is a critical preprocessing step that allows Windows event data to be analyzed using the various mining plugins ProM provides.



I. Installing ProM

ProM is a Java-based, extensible framework for process mining that supports a wide range of discovery, conformance, and enhancement algorithms. For this study, two components of the ProM ecosystem were installed: the **ProM Framework (ProM Lite)** and the **ProM Package Manager**, which is used to manage and update the tool's modular plugin architecture.

The tools were downloaded from the official ProM website <https://www.promtools.org>

We selected the **ProM Full v6.14** distribution to avoid limitations associated with the Lite version and to ensure compatibility with all necessary mining algorithms, including Alpha++, Inductive Miner, Heuristics Miner, and various utility plugins for data import and export.

In parallel, the **ProM Package Manager (ProMPM.exe)** was used to inspect and verify that all relevant packages were present and up to date. While the full version includes most plugins by default, the package manager allows fine-grained control over plugin versions and optional modules, ensuring a stable and reproducible mining environment. This setup provided the foundation for importing event logs, transforming them into XES format, and performing visual and algorithmic analysis throughout the implementation phase.

Below are a few screenshots to demonstrate the above points concerning the ProM Package Manager:

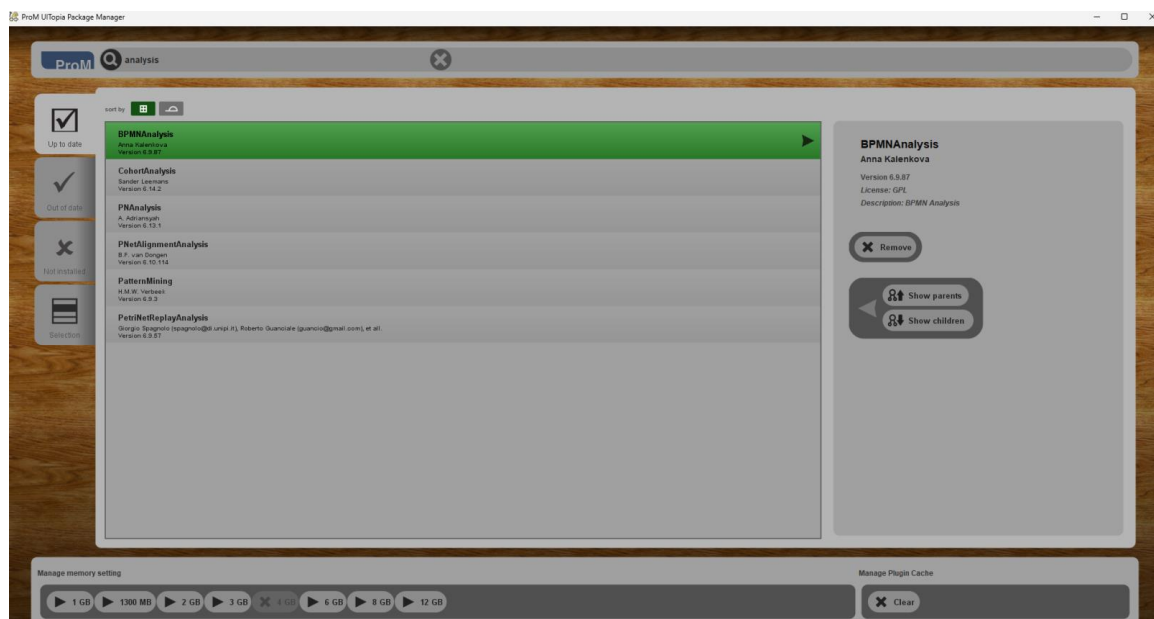


Figure 11: ProM Package Manager browsing installed packages.



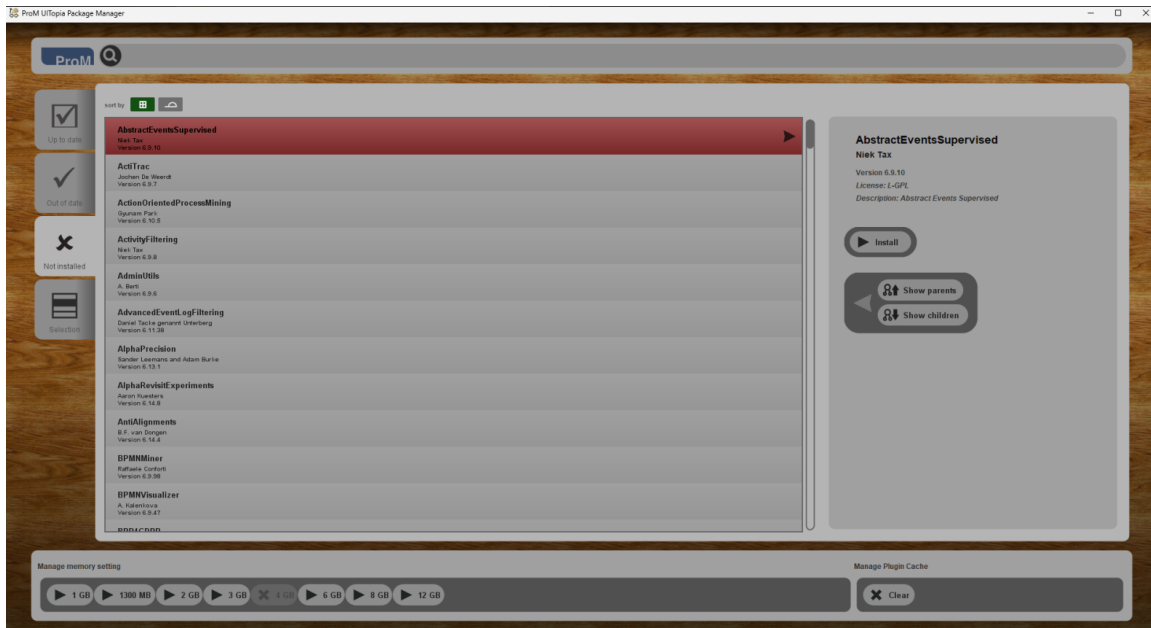


Figure 12: ProM Package Manager Installation of new Packages.

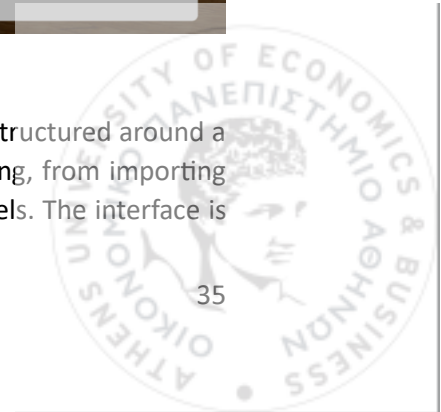
II. Using ProM

After successfully installing the required packages, we can proceed with using ProM Tools. Below we can see a picture of the main interface of ProM:



Figure 13: ProM Tools main interface.

The image above displays the main interface of ProM Full. The application is structured around a modular, three-panel design that supports the full workflow of process mining, from importing logs, to executing mining algorithms, to visually inspecting the resulting models. The interface is



divided into three primary views: Workspace, Action, and View, each accessible through tabs at the top of the application window.

In the Workspace view, users can manage all available resources, including imported logs (e.g., in CSV or XES format), discovered process models (e.g., Petri nets, directly-follows graphs), and analytical outputs. The left-hand sidebar provides navigation filters such as All, Favorites, Imported, and Selection, allowing users to quickly locate resources. Sorting options are also available to organize items by type, name, or date of import. Once a resource is selected, additional options appear for viewing, running mining actions, exporting, or marking the item as a favorite.

The Action view is used to execute process mining plugins. Based on the selected input resource(s), ProM displays a list of compatible mining algorithms and transformation tools. Users can configure plugin parameters and launch them via the "Start" button. This view is critical for applying core techniques such as Alpha++, Inductive Miner, or Heuristics Miner.

The View tab allows users to visually explore the structure and behavior of a selected resource. This includes inspecting Petri nets, directly-follows graphs, and event logs through graphical renderings or statistical summaries. Basic interaction tools are available to zoom, pan, print, or export the views.

At the top-right corner of the interface, the "import..." button allows users to load external files into the environment. To the left of it, the play (▶) and view (👁) icons provide shortcuts to execute a mining action or display the selected resource in detail.

This interface design ensures a smooth and traceable process mining workflow, from raw log ingestion to interactive model analysis within a single, unified environment.



III. XES Conversion

With the cleaned and structured dataset prepared, the next step is to convert the CSV file into XES (eXtensible Event Stream) format, which is the standard input format for most process mining tools. This conversion is necessary to apply process discovery, conformance checking, and enhancement technique. The transformation from CSV to XES will be performed by importing the selected CSV and then running the CSV to XES action as shown below.

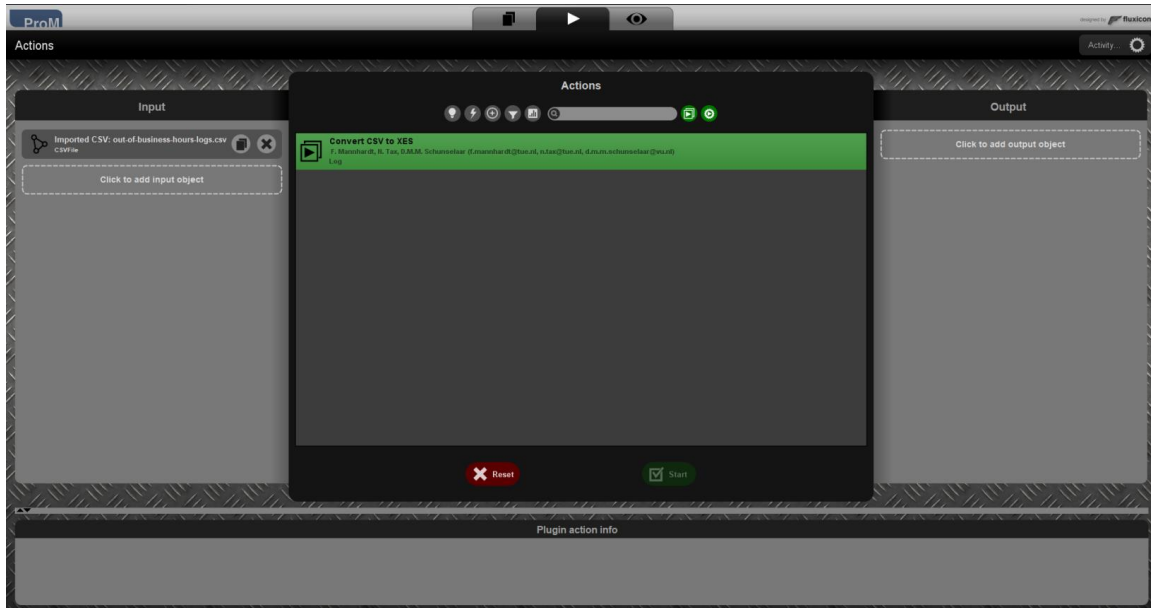


Figure 14: Conversion from CSV to XES action.

During the import process, appropriate columns such as *Case ID* (derived from System, User, and Logon ID), *Activity* (derived from the Event column), and *Timestamp* will be defined to match the event log model. Once converted, the XES log will serve as the input for subsequent process mining analysis aimed at identifying deviations, suspicious sequences, and anomalous behavior patterns within the user activity traces.

Figure 15: CSV to XES conversion configuration process.

In the figure above, the mapping of CSV columns to standard XES attributes is configured. Under the Case Column section, the fields *System*, *User*, and *Logon ID* are selected. These columns together form a unique context for each trace (or "case") in the process mining model. By combining them, we ensure that events are grouped not just by user session (via *Logon ID*), but also with awareness of the system and user account on which the activity occurred, which is crucial for detecting insider activity in multi-user, multi-host environments.

In the Event Column, the *Event* field is selected. This column contains a one-line textual description extracted from each log's *Description* field and serves as the label for each activity in the discovered process model. This simplifies the complexity of raw event descriptions and makes the output models easier to interpret. For the Start Time, the *Date and Time* column is selected, and its format is specified (M/d/yyyy H:mm), allowing ProM to correctly interpret and sort events chronologically.

We finally export the produced XES version of our dataset, and we import it back in our Jupyter directory where we already performed the dataset preparation earlier. The XES is then programmatically transformed back into CSV using **PM4Py** (Process Mining for Python), an open-source Python library that provides process-mining functionality comparable to ProM but with the flexibility of the Python ecosystem. PM4Py is particularly useful when one needs to integrate process-mining results with statistical modelling, machine-learning pipelines, or custom visualizations in Jupyter.

```
log = pm4py.read_xes('windows_security_14d_enhanced.xes')
pd = pm4py.convert_to_dataframe(log)
pd.to_csv('windows_security_14d_enhanced.csv', index=False)
```

Figure 16: XES conversion back to CSV for process mining.

The snippet first **reads** the XES file (`read_xes`), producing an internal PM4Py event-log object. It then **converts** this object into a pandas DataFrame (`convert_to_dataframe`), preserving the trace, event, and timestamp information in tabular form. Finally, the DataFrame is **exported** to a CSV file (`to_csv`). What we get is the following table:

	Description	conceptname	Level	lifecycle/transition	Event ID	timestamp	Source	Task Category	case:conceptname
3949	A Kerberos authentication ticket (TGT) was req...	A Kerberos authentication ticket (TGT) was req...	Information	start	4768	2025-05-01 01:13:00+00:00	Microsoft-Windows-Security-Auditing	Kerberos Authentication Service	LAPTOP-21-01 amandilaras Da420e0d
1985	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	Information	start	4648	2025-05-01 01:18:00+00:00	Microsoft-Windows-Security-Auditing	Logon	LAPTOP-21-01 amandilaras D65a83ac7
2325	A new process has been created.	A new process has been created.	Information	start	4688	2025-05-01 01:26:00+00:00	Microsoft-Windows-Security-Auditing	Process Creation	LAPTOP-21-01 amandilaras D675697b
2326	A process has exited.	A process has exited.	Information	start	4689	2025-05-01 01:28:00+00:00	Microsoft-Windows-Security-Auditing	Process Termination	LAPTOP-21-01 amandilaras D675697b
4062	A Kerberos service ticket was requested.	A Kerberos service ticket was requested.	Information	start	4769	2025-05-01 01:51:00+00:00	Microsoft-Windows-Security-Auditing	Kerberos Service Ticket Operations	LAPTOP-21-01 amandilaras D6a9915d
3505	A new process has been created.	A new process has been created.	Information	start	4688	2025-05-01 02:12:00+00:00	Microsoft-Windows-Security-Auditing	Process Creation	LAPTOP-21-01 amandilaras D69265406
3506	A process has exited.	A process has exited.	Information	start	4689	2025-05-01 02:13:00+00:00	Microsoft-Windows-Security-Auditing	Process Termination	LAPTOP-21-01 amandilaras D69265406
2589	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	Information	start	4648	2025-05-01 02:14:00+00:00	Microsoft-Windows-Security-Auditing	Logon	LAPTOP-21-01 amandilaras D7282750
444	A logon was attempted using explicit credentials.	A logon was attempted using explicit credentials.	Information	start	4648	2025-05-01 02:29:00+00:00	Microsoft-Windows-Security-Auditing	Logon	LAPTOP-21-01 amandilaras D11e490ea

Figure 17: XES converted table.

Following the reconversion of the XES file back into CSV format using PM4Py, several new columns were introduced into the dataset to align with the standard **XES meta-model**. These columns are automatically added by PM4Py to represent key event attributes required for process mining:

- **concept:name:** This column holds the name of the activity or event, as defined in the XES structure. It corresponds to the *Event* column mapped during the XES import in ProM and serves as the primary label used during process discovery.
- **lifecycle:transition:** Indicates the stage of the activity lifecycle. In this dataset, all entries have the value "start", reflecting that each log entry marks the beginning of an activity instance. More complex logs may also include "end" or other lifecycle markers. All events are labeled with a value of "start" because Windows Event Logs do not follow an activity lifecycle model. Each log entry represents a discrete occurrence that happens at a specific point in time, without an associated end or duration. As a result, there is no natural "end" for most events, and only the start timestamp is available for process mining purposes.
- **time:timestamp:** Represents the exact timestamp of the event in ISO 8601 format (with timezone offset), enabling precise temporal sequencing and time-based analysis during process mining.
- **case:concept:name:** This column uniquely identifies the trace or case to which each event belongs. It was constructed by combining the *System*, *User*, and *Logon ID* fields during the CSV-to-XES conversion step in ProM. This allows events to be grouped into logical sessions or user activity flows for analysis.

These XES-specific columns are critical for enabling ProM, PM4Py, and other process mining tools to interpret the event log structure correctly and generate accurate process models, timelines, and conformance checks.

7. Implementation

In this section, a case study is presented to demonstrate how process mining techniques can be applied in a real-world organizational setting to detect potential insider threats. The objective is to showcase, through controlled examples, how specific patterns in user activity logs can reveal suspicious behavior that may otherwise go unnoticed through traditional log inspection.

The scenario assumes that logs have been collected from 3 employees within an organization over a continuous period of two weeks. All preprocessing steps, such as extracting the logs from Windows Event Viewer, structuring them into a unified dataset, removing irrelevant or incomplete entries, and converting the dataset into a process mining compatible format are considered to have already been completed. As a result, a single, clean CSV file is assumed to be available, ready to be used as input in process mining tools such as ProM or PM4Py.

To simplify the proof of concept and isolate different types of suspicious behavior, the CSV is split into 3 different parts. Each one simulates typical routine user activity combined with a specific behavioral pattern indicative of an insider threat. These three behavioral indicators, commonly studied in the context of security monitoring, are:

1. **Infrequent Events:** Isolated or rare activities that deviate from a user's normal behavior or from organizational norms, such as the unexpected creation of a new user account.
2. **L1 Loops (Self-loops):** Repetition of the same activity within a short time frame, which may signal malicious intent (e.g., repeated failed login attempts indicative of password guessing).
3. **Out-of-Hours Activity:** Events that occur outside established business hours, which could signal unauthorized or unsupervised access to company systems. This concept can also apply to companies that have set a policy, disallowing any employee to use company hardware outside of working hours.



7.1 Code Listing

This section contains the complete Python pipeline used in this study. All preprocessing and analysis were carried out in a Jupyter environment because it allows rapid iteration, inline visualization, and direct interaction with large log files. Using pandas for data wrangling and PM4Py for process-mining operations, the script loads the cleaned CSV dataset, converts it to a PM4Py event-log object, and then executes some selected process mining technique for each dataset. For every technique the code generates a corresponding process model and exports the visualization as a PNG that is later embedded in the Implementation chapter. The notebook also annotates each output with basic statistics such as trace count and distinct activity names, so that readers can verify their own results against the figures in this paper.

No other languages or external tools are required. Every step from log import to PNG export runs inside the single notebook, ensuring that the methodology is fully reproducible on any machine with a standard Python stack.

I. Prerequisites

The script starts by loading the main libraries. pandas handles general data manipulation, while the pm4py package provides every process-mining feature used in this work. Log conversion tools bring the cleaned CSV into a PM4Py event-log object, and discovery modules load the Alpha, Inductive, Heuristics, and Direct-Follows algorithms. Matching visualization modules create Petri nets, process trees, heuristics nets, and directly-follows graphs. A helper converter turns process trees into Petri nets when that representation is required. Finally, the script widens pandas display settings so that all columns and full row content are visible during interactive inspection in the notebook.

```
import pandas as pd
import pm4py
from pm4py.objects.conversion.log import converter as log_converter
from pm4py.objects.log.importer.xes import importer as xes_importer

# process mining
from pm4py.algo.discovery.alpha import algorithm as alpha_miner
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristics_miner
from pm4py.algo.discovery.dfg import algorithm as dfg_discovery

# visualization
from pm4py.visualization.petri_net import visualizer as pn_visualizer
from pm4py.visualization.process_tree import visualizer as pt_visualizer
from pm4py.visualization.heuristics_net import visualizer as hn_visualizer
from pm4py.visualization.dfg import visualizer as dfg_visualization

# misc
```



```
from pm4py.objects.conversion.process_tree import converter as pt_converter

pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', 1000)
```

II. *Log Statistics*

The code then imports the XES file and stores it as a PM4Py event-log object named `log`. Calling `pm4py.get_start_activities(log)` lists every activity that can begin a trace, confirming the log's structure.

```
log = xes_importer.apply('windows_security_14d_enhanced.xes')
pm4py.get_start_activities(log)

log[1]

log[0][0]

variants_count = case_statistics.get_variant_statistics(log)
variants_count = sorted(variants_count, key=lambda x: x['count'], reverse=True)
variants_count[:10]
```

To illustrate the event format, `log[1]` prints the attributes of the second trace, and `log[0][0]` drills further to show the first event of the first trace, including the event ID, timestamp, user name and other audit fields that will later become process-model attributes.

The trace shown would look something like this:

```
{
  "attributes": {
    "concept:name": "0x10107a5"
  },
  "events": [
    {
      "User": "User1",
      "Description": "A Kerberos authentication ticket (TGT) was requested.",
      "concept:name": "A Kerberos authentication ticket (TGT) was requested.",
      "Level": "Information",
      "lifecycle:transition": "start",
      "Event ID": 4768,
      "time:timestamp": "2025-05-11T10:37:00Z",
      "System": "LAPTOP-21-01",
      "Source": "Microsoft-Windows-Security-Auditing",
      "Task Category": "Kerberos Authentication Service"
    }
  ]
}
```



```
}
```

The event of the first trace again would have this format:

```
{  
  "User": "User1",  
  "Description": "An account failed to log on.",  
  "concept:name": "An account failed to log on.",  
  "Level": "Information",  
  "lifecycle:transition": "start",  
  "Event ID": 4625,  
  "time:timestamp": "2025-05-07T02:04:00Z",  
  "System": "LAPTOP-21-01",  
  "Source": "Microsoft-Windows-Security-Auditing",  
  "Task Category": "Logon"  
}
```

Next, the script measures behavioral diversity. The `variants_filter_get_variants(log)` function groups traces that share an identical activity sequence and returns a dictionary whose keys are variant labels and values are lists of traces that follow that variant. Printing the length of this dictionary shows that eleven distinct variants exist in the log. The code then calculates how often each variant occurs with `case_statistics.get_variant_statistics(log)`, sorts the resulting list in descending order of frequency and prints the ten most common variants. This quick scan confirms that the dataset contains a manageable variety of execution paths before any discovery algorithms are applied.

```
[  
  {  
    "variant": [  
      "An account was successfully logged on.",  
      "Special privileges assigned to new logon.",  
      "Group membership information.",  
      "An account was logged off."  
    ],  
    "count": 592  
  },  
  {  
    "variant": [  
      "Group membership information.",  
      "Special privileges assigned to new logon.",  
      "An account was successfully logged on.",  
      "An account was logged off."  
    ],  
    "count": 213  
  },  
  {
```



```
"variant": [  
  "A new process has been created.",  
  "A process has exited."  
],  
"count": 210  
},  
{  
  "variant": [  
    "An account was successfully logged on.",  
    "Group membership information.",  
    "Special privileges assigned to new logon.",  
    "An account was logged off."  
  ],  
  "count": 169  
},  
{  
  "variant": [  
    "Special privileges assigned to new logon.",  
    "Group membership information.",  
    "An account was successfully logged on.",  
    "An account was logged off."  
  ],  
  "count": 146  
},  
{  
  "variant": [  
    "Special privileges assigned to new logon.",  
    "An account was successfully logged on.",  
    "Group membership information.",  
    "An account was logged off."  
  ],  
  "count": 146  
},  
{  
  "variant": [  
    "A logon was attempted using explicit credentials."  
  ],  
  "count": 140  
},  
{  
  "variant": [  
    "Group membership information.",  
    "An account was successfully logged on.",  
    "Special privileges assigned to new logon.",  
    "An account was logged off."  
  ],  
  "count": 128  
},
```



```
{
  "variant": [
    "A Kerberos service ticket was requested."
  ],
  "count": 70
},
{
  "variant": [
    "A Kerberos authentication ticket (TGT) was requested."
  ],
  "count": 70
}
]
```

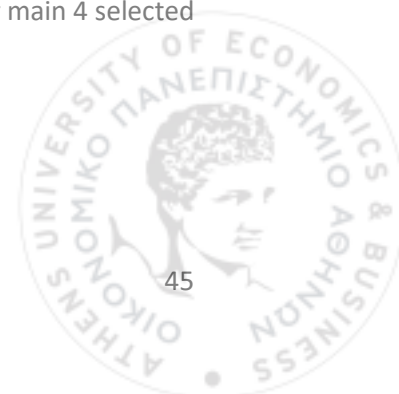
This next part of the code imports the `attributes_filter` module from `PM4Py` and then calls `get_attribute_values`, passing the event log and the attribute name `"concept:name"`. `PM4Py` scans every event in the log, counts how many times each activity label appears, and returns a dictionary in which the keys are activity names and the values are their respective frequencies. Storing this dictionary in `activities` gives a quick overview of how often each type of Windows security event occurs in the dataset.

```
from pm4py.algo.filtering.log.attributes import attributes_filter
activities = attributes_filter.get_attribute_values(log, "concept:name")
activities
```

In our example the dictionary lists every distinct activity label in the log and the number of times each one occurs:

```
{
  "An account failed to log on.": 17,
  "A Kerberos authentication ticket (TGT) was requested.": 70,
  "An account was successfully logged on.": 1394,
  "Group membership information.": 1394,
  "Special privileges assigned to new logon.": 1394,
  "An account was logged off.": 1394,
  "A new process has been created.": 210,
  "A process has exited.": 210,
  "A logon was attempted using explicit credentials.": 140,
  "A Kerberos service ticket was requested.": 70
}
```

Then, we move on to executing the process mining techniques, which will be our main 4 selected miners in this case.



III. *Alpha miner*

This subsection shows the process of running the Alpha Miner on the event log, producing a Petri-net model (net) together with its required initial and final markings. These objects are passed to the Petri-net visualizer, which renders the net (gviz) and then opens the graphic in a viewer window. In one step you obtain both the formal control-flow model discovered from the log and an on-screen diagram of that model.

```
net, initial_marking, final_marking = alpha_miner.apply(log)
gviz = pn_visualizer.apply(net, initial_marking, final_marking)
pn_visualizer.view(gviz)
```

This next block tells PM4Py to redraw the Petri net with frequency information. A parameter dictionary sets the output format to PNG. The variant argument selects the FREQUENCY visualizer, which shades and labels each place and transition according to how often it appears in the log (the log itself is supplied via log=log). The call to pn_visualizer.apply returns a graphics object gviz, and pn_visualizer.view(gviz) opens the resulting frequency-annotated net for inspection.

```
parameters = {pn_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}
gviz = pn_visualizer.apply(net, initial_marking, final_marking,
                          parameters=parameters,
                          variant=pn_visualizer.Variants.FREQUENCY,
                          log=log)
pn_visualizer.view(gviz)
```

IV. *Inductive miner*

Our next goal is to execute the Inductive Miner on the event log, which yields a **process tree** named "tree". The tree is then rendered with the process-tree visualizer and displayed in an interactive window, giving a hierarchical view of the log's control flow.

```
tree = inductive_miner.apply(log)

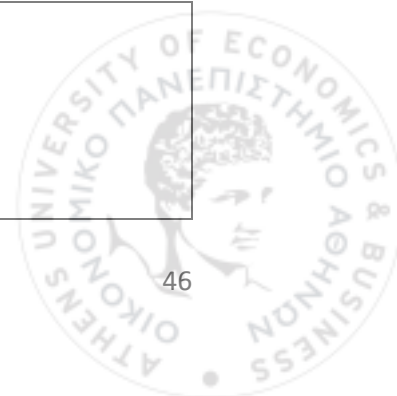
gviz = pt_visualizer.apply(tree)
pt_visualizer.view(gviz)

pn_visualizer.save(gviz, "Inductive_miner.png")
```

The second block converts the same process tree into a Petri net using pt_converter.apply, providing the net plus its initial and final markings. The Petri-net visualiser draws this model, shows it on screen, and saves the diagram to disk as Inductive_miner_petri_net.png.

```
net, initial_marking, final_marking = pt_converter.apply(tree,
                                                         variant=pt_converter.Variants.TO_PETRI_NET)

gviz = pn_visualizer.apply(net, initial_marking, final_marking)
pn_visualizer.view(gviz)
pn_visualizer.save(gviz, "Inductive_miner_petri_net.png")
```



V. *Heuristic miner*

The code for the Heuristic Miner applies itself to the log with a dependency-threshold parameter set to 0.5, generating a heuristics net (heu_net) that keeps only relations whose causal strength meets or exceeds that value. The net is visualized with `hn_visualizer.apply`, shown in a viewer window, and saved to disk as a PNG.

```
heu_net = heuristics_miner.apply_heu(log,  
parameters={heuristics_miner.Variants.CLASSIC.value.Parameters.DEPENDENCY_THRESH: 0.5})  
gviz = hn_visualizer.apply(heu_net)  
hn_visualizer.view(gviz)  
  
hn_visualizer.save(gviz,"Heurctic_miner.png")
```

VI. *Direct-Follows Graph (DFG)*

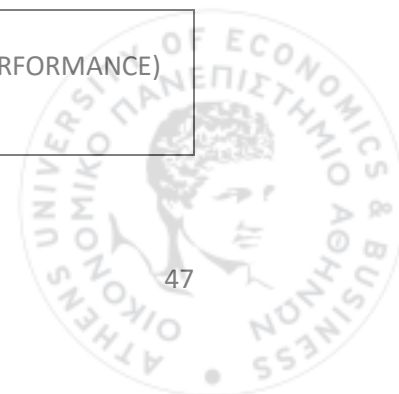
For Directly-Follows Graph (DFG) we execute it to the event log and then render a frequency-annotated version of that graph.

1. `dfg_discovery.apply(log)` scans the log and counts every immediate succession of activities, producing a dictionary whose keys are (source, target) activity pairs and whose values are occurrence counts.
2. `dfg_visualization.apply` is called with the FREQUENCY variant, so each edge in the graph is drawn with a thickness or color that reflects how often the corresponding transition appears in the log.
3. `dfg_visualization.view(gviz)` opens the resulting diagram in an interactive viewer, and `dfg_visualization.save(gviz, "DFG_with_frequency.png")` writes the image to disk for later inclusion.

```
dfg = dfg_discovery.apply(log)  
  
from pm4py.visualization.dfg import visualizer as dfg_visualization  
gviz = dfg_visualization.apply(dfg, log=log, variant=dfg_visualization.Variants.FREQUENCY)  
dfg_visualization.view(gviz)  
dfg_visualization.save(gviz, "DFG_with_frequency.png")
```

This next block rebuilds the directly-follows graph so that each edge stores the average time taken to move from one activity to the next rather than a count. The performance visualizer then draws the graph, coloring or labelling each edge with that throughput time, shows it on screen, and saves the image for later use.

```
dfg = dfg_discovery.apply(log, variant=dfg_discovery.Variants.PERFORMANCE)  
gviz = dfg_visualization.apply(dfg, log=log, variant=dfg_visualization.Variants.PERFORMANCE)  
dfg_visualization.view(gviz)  
dfg_visualization.save(gviz, "DFG_with_time.png")
```



The call to `dfg_mining.apply(dfg)` takes the directly-follows graph you built earlier and converts it into a *workflow net* (a sound Petri net) together with its initial (im) and final (fm) markings. The Petri-net visualizer then draws this model and displays it.

```
from pm4py.objects.conversion.dfg import converter as dfg_mining
net, im, fm = dfg_mining.apply(dfg)
gviz = pn_visualizer.apply(net, im, fm)
pn_visualizer.view(gviz)
dfg_visualization.save(gviz, "DFG_with_Workflow_Net.png")
```

7.2 L1 Loops

An **L1 loop** occurs when the same activity is logged twice in immediate succession within a single process trace. In a directly-follows graph this appears as a self-edge, and in a Petri-net model it is represented by a transition that can fire consecutively without an intervening activity. Within Windows security auditing, such loops arise when the operating system records the identical event code back-to-back multiple times (just like our example with the multiple failed logons). Ordinary workflows generate long stretches of identical events; therefore, dense self-loops typically signal abnormal behavior. A burst of “4625 - Failed Logon” records, a rapid series of “4688 - Process Creation” events for the same executable, or repeated “4656 / 4663 - Object Access” denials can each reflect password-brute force attempts, malicious script loops, or exploratory privilege escalation.

Detecting these patterns requires discovery techniques that preserve or highlight self-loops rather than abstracting them away. The Heuristics Miner accommodates them explicitly by considering the weight of directly-follows relations where an activity precedes itself; its noise-tolerance settings can be tuned so that infrequent insider traces are kept instead of discarded. Inductive Miner (particularly the IM-f variant) also models self-repeating behavior through loop cuts while guaranteeing soundness, making it suitable for larger, noisy audit logs in which brute-force attacks may dominate. On smaller or more controlled datasets, Alpha++ and other short-loop extensions to the original α -algorithm provide formal loop detection, though they are less forgiving of noisy traces. Even before full discovery, a directly-follows graph with a frequency filter offers a quick visual cue: a thick self-edge immediately reveals aggressive repetition, which can be corroborated with local process-model mining to isolate frequent A A fragments embedded in longer traces.

Significance hinges on context. A pair of identical events can be benign, for instance, two legitimate logons after a screen-lock, but suspicion rises when the repeated event represents a failure code, which occurs unusually often within the same session, day & time, or emanates from a workstation atypical for the user. Incorporating such contextual checks alongside loop-aware mining techniques not only brings genuinely malicious patterns to the surface but also curbs false positives, aligning with the research overarching goal of precise insider-threat detection.



I. Visualizations

The following figure shows the model obtained when the log is projected to a Directly-Follows Graph (DFG) and automatically completed to a sound workflow net. This view emphasizes causal order and makes every possible start-to-end path explicit: repeated execution of the same activity appears as a circular arc that returns to the originating node. In our data only the transition “An account failed to log on” forms such a self-edge, revealing a length-one loop that corresponds to consecutive authentication failures within the same logon session. No other event repeats itself.

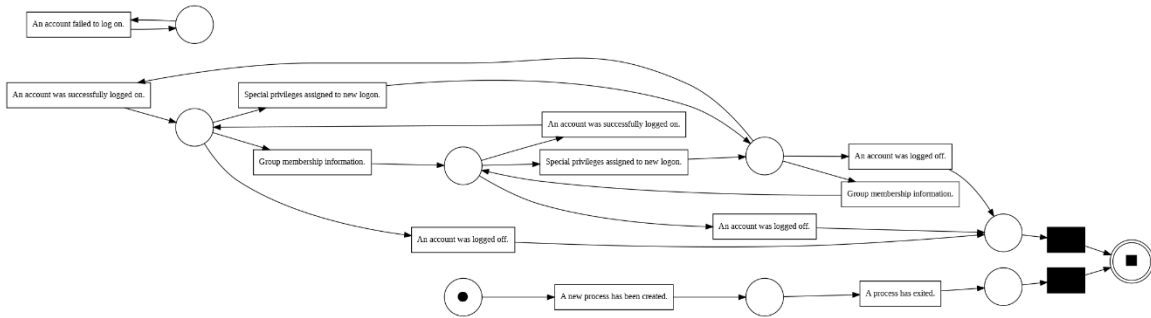


Figure 18: Directly follows Graph with Workflow Net.

The next example presents the same DFG but now enriched with raw frequency counts. By coloring or thickening arcs according to their weight the model highlights dominant paths while still preserving low-support behavior. The self-edge on the failed-logon activity carries a count of sixteen, confirming that the loop is neither a modelling artefact nor an isolated coincidence: the system really did record sixteen back-to-back failures for one user-session trace.

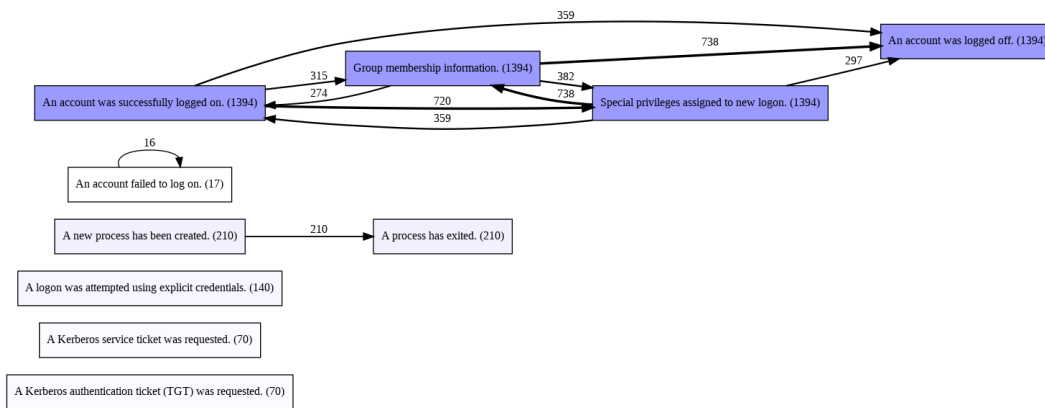
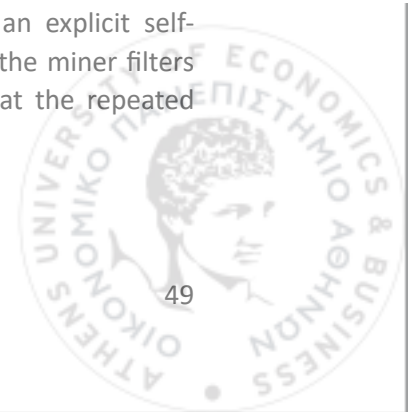


Figure 19: Direct follows Graph with frequency.

The last figure represented below is produced with the Heuristics Miner, whose dependency measures tolerate noise while retaining loops of length one. The result resembles a causal net annotated with absolute counts. Here again the failed-logon node displays an explicit self-transition, while all other activities show only forward-moving edges. Because the miner filters relations below its threshold, the presence of a surviving self-loop signals that the repeated failures stand out statistically against normal traffic.



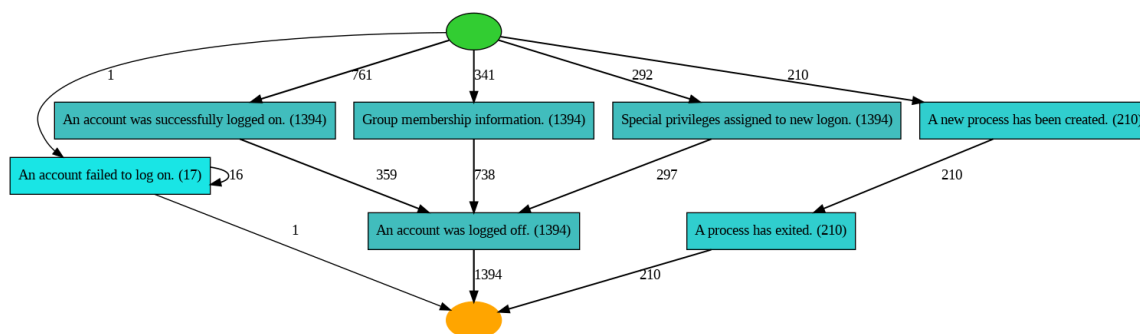


Figure 20: Heuristic miner output for L1 loops example.

II. Comparisons

Taken together, the three visualizations confirm both the existence and the significance of an L1 loop in the dataset, which provide evidence that the account in question attempted multiple logons in rapid succession, a pattern frequently associated with brute-force guessing or mistyped credentials rather than routine user behavior.

Of the three visualizations, the Heuristics-Miner model offers the most reliable way to expose length-one loops in practice. The frequency-annotated DFG gives a quick, intuitive snapshot, but its clarity disappears as soon as the log grows beyond a modest size; in a real corporate environment with tens of thousands of traces the self-edge would be lost in a web of arcs. The workflow-net variant introduces soundness but does so by completing the model with artificial places, which can mask a self-loop if the miner resolves it through hidden transitions. By contrast, the Heuristics Miner balances both needs: it retains explicit self-transitions yet applies dependency and frequency thresholds that can be tuned to filter noise without discarding low-support and meaningful loops. This adaptability makes it more robust across heterogeneous datasets, whether the log contains a handful of brute-force attempts or hundreds of repeated failures, and therefore better suited to consistent insider-threat detection in live, evolving environments.

III. Incident Response

When an L1 loop is flagged in the produced graph, the first step is to treat the associated session as a potential incident case. An analyst should immediately correlate the logon ID with supporting context: the user's identity, the endpoint involved, the corresponding network traffic, and any complementary security controls such as endpoint detection alerts or firewall logs. By assembling this wider picture, the team can confirm whether the loop reflects malicious intent, a mistyped password, or an automated system task.

This process-mining graph offers a strategic value for an organization. Recurrent loops can be monitored over time to measure the effectiveness of policy changes and user training. The visual model also provides clear evidence for audit reports, showing that the security team not only captured the anomaly but followed a documented, risk-based procedure to resolve it.

IV. *False Positive Detection*

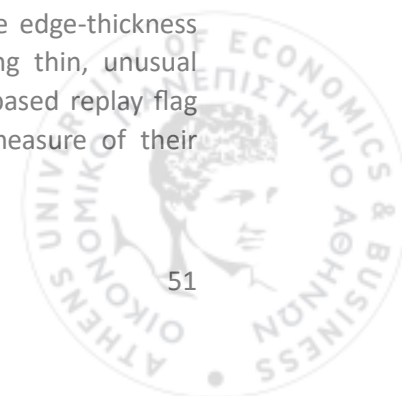
To keep false positives under control, every flagged loop should pass through a checklist. The analyst verifies the timestamp against approved maintenance windows, checks whether the originating user normally works during that period, and examines whether the source IP or workstation deviates from the user's usual pattern. If any of these factors align with an established baseline, the event can be downgraded or closed with a note. If the loop involves sensitive resources or appears outside the baseline, it should move to an investigative queue for deeper forensics and, if necessary, user verification. If deemed necessary, the forensics team can contact the employee to ask questions about the incident. This can help a significant amount with the final judgement of the team.

The organization should formalize this flow within an incident response playbook. The playbook assigns roles for triage, investigation, and remediation, defines service-level targets for each stage, and specifies when to escalate to legal or human resources. Automating the initial enrichment phase, for example by using a security orchestration platform, shortens response times and reduces analyst fatigue.

7.3 Infrequent Events

Events that appear only once or a handful of times across an otherwise uniform set of traces often carry greater diagnostic value than the high-volume transactions that dominate a log. In an enterprise Windows environment, a rare event might record the creation of a new local user, a single invocation of an administrative tool, or an isolated instance of a sensitive object-access failure. Because insider misuse typically diverges from everyday routines, such events stand out as outliers when the log is projected into a directly-follows graph: they either contribute a thin edge between two activities or remain disconnected from the main structure altogether. Because these events occur so rarely, most mining tools discard them as low-frequency noise. To spot them, we must lower those frequency filters or use algorithms that keep even single-occurrence events in the model.

Heuristics Miner accommodates infrequent traces by lowering its dependency and frequency thresholds, allowing even a single occurrence to produce a visible arc in the discovered model if its causal strength is sufficient. Inductive Miner's "IMf" variant takes the complementary approach of first filtering noise and then re-inserting low-frequency behavior where it does not break model soundness, thus keeping legitimate outliers while discarding random noise. Local Process Model mining is particularly useful here because it focuses on small, statistically significant fragments; a fragment that appears once yet links activities rarely seen together may point to an insider action executed in isolation. When visualizing directly-follows graphs, configuring the edge-thickness scale to present raw counts rather than normalized percentages helps bring thin, unusual pathways into view. Finally, conformance-checking techniques such as token-based replay flag low-frequency traces with high deviation scores, providing a quantitative measure of their anomalous nature.



Contextual interpretation remains essential: a single failed service start on a workstation reboot is benign, whereas a lone “4720 - User Account Created” on a domain controller warrants immediate investigation.

1. Visualizations

The graph presented below is a directly-follows graph rendered with absolute frequency counts. Edges grow thicker as the number of observed transitions increases, so the main authentication cycle successful logon, privilege assignment, group-membership change, and logoff, dominates the picture with dark, heavy arcs. Events that occur rarely, such as “A user account was enabled” or “A user account was created,” remain connected to the graph but are linked by almost hair-line edges and appear at the periphery. Their thin connections and single-digit counts make them immediately recognizable as outliers, even though the visual clutter produced by the many routine paths makes the overall structure hard to read.



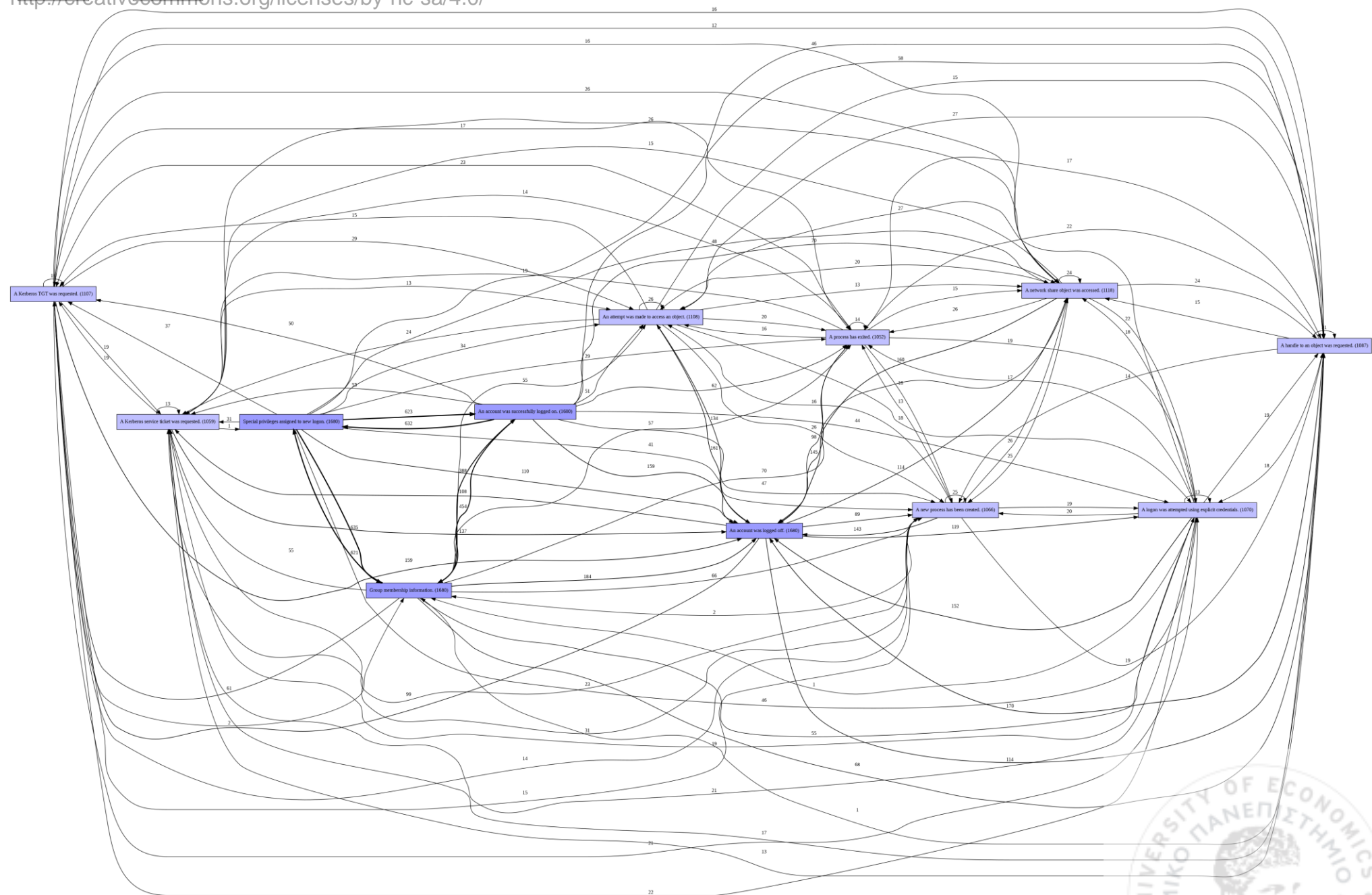


Figure 21: DFG with frequency for Infrequent events.



The next graph shown applies the Alpha++ miner to the same log and converts the result to a Petri-net model. Because this algorithm preserves every observable transition, the infrequent activities survive as discrete places and transitions, yet the model is far less tangled than the raw DFG with frequency. The core authentication loop still carries counts in the thousands, but the two one-off management events stand out at the top of the net, connected to the start and end places by arcs labelled with a frequency of one. In effect, the miner distils the log into a decent process model while retaining the statistical information needed to spot anomalies: anything that appears only once or twice is immediately visible because its arcs and markings bear the lowest numbers in the diagram. By comparing the cluttered but exhaustive DFG to the more structured Alpha++ net, the reader can see how discovery techniques with different abstraction levels expose the same infrequent events while simultaneously improving interpretability, which is a combination that is essential when the goal is to flag rare insider actions without losing the broader process context.

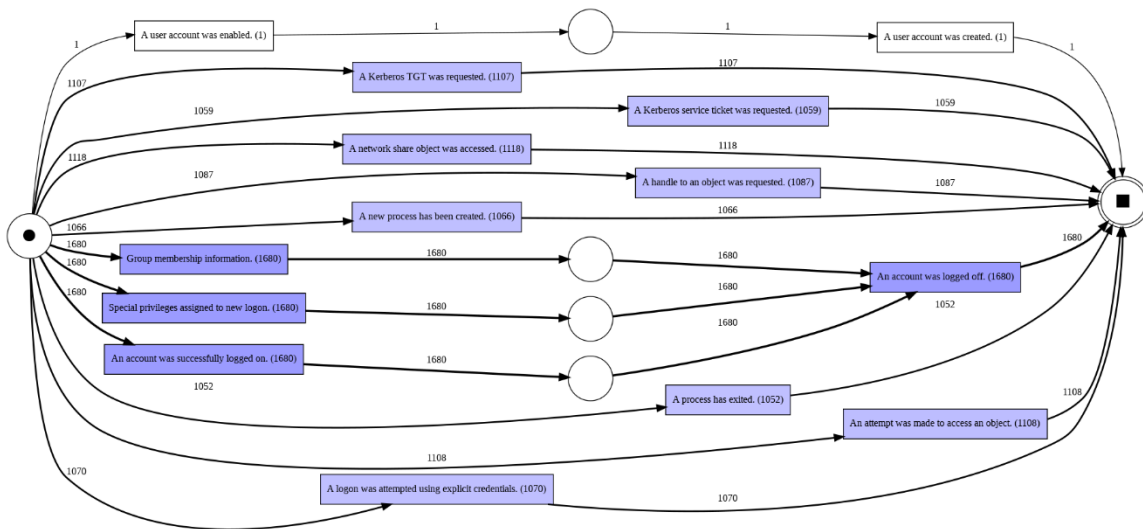
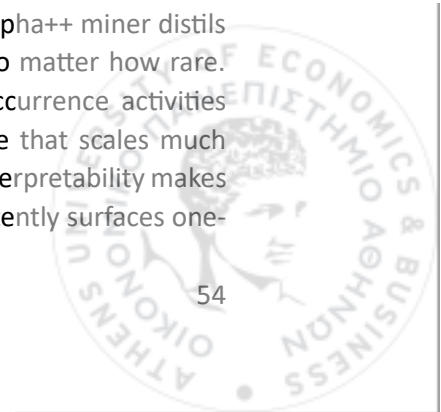


Figure 22: Alpha++ miner with frequency for infrequent events.

II. Comparisons

Among the two representations used for infrequent-event analysis, the Alpha++ Petri-net is the more dependable choice for operational monitoring. The raw frequency DFG clearly flags outliers by leaving them on thin, peripheral arcs, but that readability degrades rapidly as the number of routine transitions, and their corresponding edges explode. In contrast, the Alpha++ miner distils the log into a sound process model while still preserving every transition, no matter how rare. Because each arc in the Petri-net carries its observed count, the single-occurrence activities remain visible, yet they are embedded in a cleaner, noise-reduced structure that scales much better to large or volatile datasets. This balance between completeness and interpretability makes the Alpha++ net more robust for real-world insider-threat detection: it consistently surfaces one-



off or low-frequency actions without overwhelming analysts with the visual clutter inherent in a full, unfiltered DFG.

III. Incident Response

When an infrequent event surfaces in the process-mining model, the security team should open an incident case tied to that exact record and its trace. The first task is to enrich the alert with surrounding context: who executed the action, on which host, under what ticket or maintenance window change, and whether any parallel controls (for example, configuration-management or identity-governance systems) recorded a matching, legitimate action. If the organization operates a change-management database, the analyst checks whether the rare event aligns with an approved request.

IV. False Positive Detection

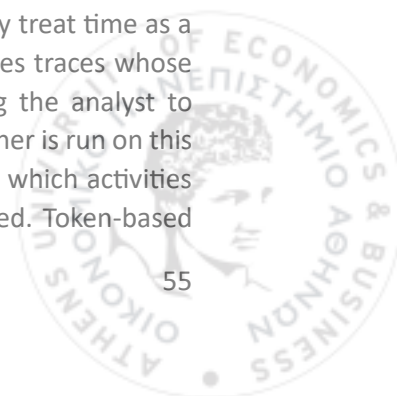
Controlling false positives hinges on verifying intent and authorization. An isolated “user account enabled” entry during a routine hiring cycle may be harmless, whereas the same entry on a dormant server at midnight is likely suspicious. The triage checklist therefore compares the event against business-hour policies, validates that the account owner or system administrator had a scheduled task at the given time, and inspects network or application logs for corroborating activity. Events that fail any of these checks move to a deeper investigation that may include memory capture, file-system imaging, or direct user interviews.

The organization formalizes this workflow in its incident-response playbook. Triage rules specify which roles validate change tickets, how long an analyst may spend before escalating, and what evidence must accompany a hand-off to legal or human resources. Automation platforms can pre-populate tickets with context from configuration management and privilege-management systems, shortening cycle time and reducing analyst workload.

7.4 Outside of working hours

Events that occur outside the organization’s defined working hours are particularly revealing in insider-threat scenarios. A timestamp that falls well beyond the normal shift window, whether in the early hours of the morning, during weekends, or on officially recorded holidays, suggests a deviation from routine business practice. In Windows security auditing, such timestamps often accompany privileged actions that administrators ordinarily perform during maintenance windows. When they appear on ordinary user accounts, they merit closer scrutiny.

Performance-oriented miners are well suited to exploiting this label because they treat time as a first-class dimension. In ProM, applying the Performance/Timeframe filter isolates traces whose events fall wholly or partially outside the accepted working interval, allowing the analyst to generate a model composed solely of after-hours activity. When the Inductive Miner is run on this filtered log, it often reveals simplified process structures that highlight precisely which activities occur out of hours and in what order, since day-to-day noise has been removed. Token-based



conformance replay offers an alternative approach: computing replay costs with respect to a “normal-hours” reference model assigns high deviation scores to traces that include off-hours events, ranking them automatically for further investigation. Even a directly-follows graph can be enriched by coloring edges according to the proportion of off-hours events they carry, visually separating nocturnal or weekend activity from the daytime backbone.

Interpretation again depends on context. A scheduled backup service undoubtedly produces events at night, but an isolated “4722 - User Account Enabled” or a cluster of process-creation events from a helpdesk workstation at 03:00 raises an immediate red flag.



I. Visualizations

In this case a Petri-net model was used to discover the Inductive Miner. Because every trace (daytime and overnight) is included, the net shows the complete authentication cycle, object-access attempts, process creation and termination, and a scattering of network events. The start place fans out to many activities, reflecting the diversity of actions that occur during business hours, while the end place converges those paths back through routine logoff or ticket-granting steps. Although comprehensive, the model offers little visual contrast between legitimate daytime traffic and the few traces that fall outside the shift window.

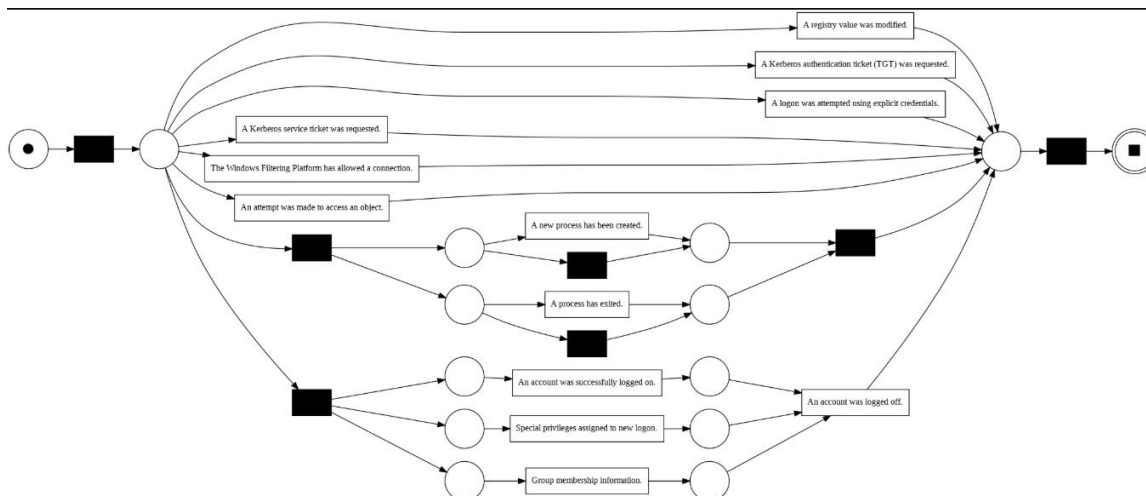


Figure 23: Out-of-hours Inductive miner graph (w/o time limit)

Unlike the previous graph though, this one presents the **after-hours subset** produced by filtering the same log on timestamps later than 18:00. With the daytime activity removed, the net collapses into a much sparser structure: only Kerberos ticket requests, explicit-credential logons, and the final logoff path remain. The absence of group-membership changes, privilege assignments, and most process-creation events make the residual model easy to inspect; any overnight deviation would therefore stand out immediately.

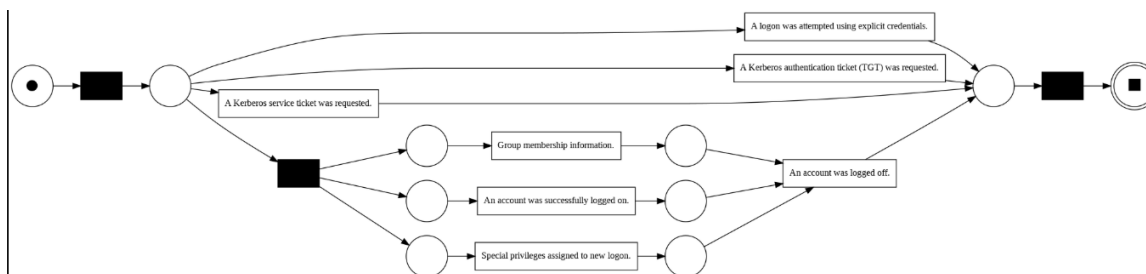


Figure 24: Out-of-hours Inductive miner graph (with time limit)

For isolating activity outside working hours, the attribute-filtered Inductive Miner model of the after-hours subset provides the most dependable representation. The global model that combines day and night traffic preserves completeness, but the sheer volume of routine traces conceals the few overnight paths that matter for insider-threat detection. Once the log is filtered on the “after_hours” attribute and rediscovered, the resulting Petri net contains only the activities that

only occurred during the restricted time frame. This produces a compact, noise-free structure where any unexpected deviation stands out immediately, regardless of how large or diverse the original dataset becomes. Because the approach relies on a simple timestamp filter followed by a miner that guarantees soundness, it remains consistent even when business hours vary across departments or evolve over time, making it the most practical option for real-world monitoring.

II. Incident Response

When the process-mining model highlights activity that occurred outside the organization's defined working hours, the incident-response workflow begins with immediate context gathering. The analyst links the timestamped event to the user account, host, and physical or virtual location, then checks change-management records or on-call schedules to confirm whether a late-night maintenance window was planned. If the user has legitimate after-hours duties, such as database backups or emergency patching, the alert can be downgraded once supporting evidence is verified.

Beyond the immediate response, the after-hours model supports long-term policy tuning. Tracking the volume and type of legitimate overnight activity allows the team to tighten maintenance windows without disrupting critical operations. Trend analysis reveals whether users consistently work late without formal approval, prompting either process adjustments or targeted training. The visual trace of authorized versus unauthorized after-hours events also provides clear evidence for audits, demonstrating that the organization not only detects policy deviations but follows a documented procedure to resolve them.

III. False Positive Detection

Reducing false positives in this scenario depends on a robust reference calendar. The security team should maintain a baseline that includes normal shift times, weekend rotations, public holidays, and authorized maintenance periods. The incident playbook instructs analysts to compare every after-hours event against this baseline. If the event falls outside any approved window, or if the user involved is not listed on an on-call roster, the case escalates. Deeper investigation may include reviewing VPN logs, badge-in records, and endpoint telemetry to determine whether the access originated from an expected location and whether additional suspicious actions followed. The organization's playbook assigns response times and responsibilities. Tier-1 analysts perform the initial validation, while tier-2 investigators collect forensic images, review related logs, and, if needed, engage human resources or legal counsel. Automated enrichment through security-orchestration tools speeds up this cycle by pulling calendar data, user role information, and network context into the incident ticket as soon as the alert is generated.



7.5 Flow Diagram

A flow chart has been made to illustrate the complete insider-threat workflow used in this study. Logs from multiple systems feed a four-step dataset-preparation block, after which process-mining detects anomalies and flags an incident. The alert enters an “Incident Response playbook” that houses parallel triage checklists for out-of-hours activity, infrequent events, and L1 loops. Results from these checks converge on a decision gateway: benign cases close with a note, whereas suspicious cases trigger deeper investigation and may be escalated to HR or Legal before final closure.



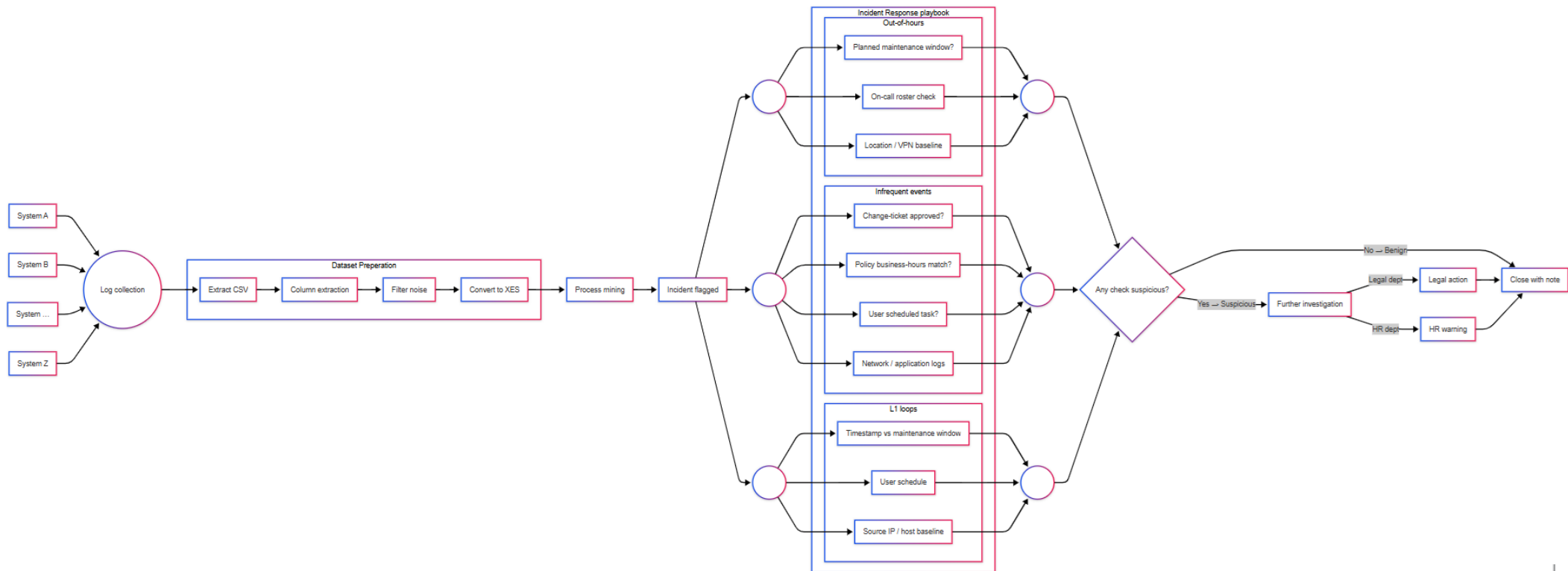


Figure 25: End-to-End Insider-Threat Detection and Response Workflow



7.6 Evaluation of results

Compared with the techniques that dominate Windows-log monitoring today, static SIEM rule sets, keyword dashboards in Event Viewer, and generic anomaly-scoring models, the process-mining approach delivers two decisive advantages. First, it adds explicit control-flow context. Conventional rules fire when a single event matches a pattern (for example, five consecutive 4625 failures). Generic anomaly detectors improve on this by counting unusual frequencies, yet they still treat each record as an isolated point in time. Process discovery, by contrast, rebuilds the whole sequence of actions a user or machine actually executed, revealing how one step causally leads to the next. Self-loops that encode password-guessing bursts, one-off branches that hide privilege escalation, or paths whose timestamps sit wholly outside business hours emerge naturally in the mined model and can be inspected in seconds, whereas the same behavior is buried in thousands of raw lines when viewed through a rule or n-gram lens.

Second, the method is inherently explainable. A Petri net or directly-follows graph shows the exact place where a suspect trace diverges from the dominant flow, so an analyst can justify an escalation without reverse-engineering why a black-box score spiked.

Scalability also compares favorably. The computational cost of discovery grows with log volume, but it does so linearly and can be mitigated with daily batching or incremental miners, whereas rule sets grow combinatorially as administrators attempt to cover every corner case. The main trade-off is resource usage: process discovery over tens of millions of events demands more memory and, occasionally, cluster time. Yet this cost is offset by the reduction in analyst hours spent triaging noisy alerts and by the richer forensic artefacts generated for compliance audits.

In summary, applying process mining to Windows event logs replaces a fragmentary, event-centric view with a coherent behavioral narrative. It tightens detection accuracy, shortens investigation cycles, and scales predictably, offering a demonstrable improvement over both traditional rule-matching and unsupervised point-anomaly models while preserving the full evidentiary value of the original logs.



8. Conclusions

This study has shown that process mining can turn routine Windows event logs into an effective source of behavioral intelligence for insider-threat detection. By converting raw audit data into an XES event stream and applying discovery algorithms, we exposed three categories of anomalies. Length-one loops revealed repeated logon failures linked to password-guessing activity, infrequent events uncovered isolated administrative actions such as unexpected account creation, and out-of-hours traces highlighted activity occurring well beyond normal working schedules. In each case the process-aware perspective added contextual detail that simple rule matching cannot provide, allowing analysts to visualize entire behavioral sequences rather than single log lines.

The practical impact of this approach is two-fold. First, an automated pipeline spanning CSV extraction, attribute filtering, XES conversion and miner execution, can update behavioral baselines almost in real time, giving security teams a fresh view of emerging patterns without manual intervention. Second, the Petri nets and directly-follows graphs produced by the miners present a concise, audit-ready record that demonstrates systematic monitoring and review.

Three limitations temper these benefits. The methodology assumes complete and correctly configured audit logging, because in the case of missing channels or weak audit policies blind spots would start to appear. It also relies on the premise that process deviations correlate with malicious intent, which means an insider who faithfully mimics routine workflows may slip through undetected. Finally, very large logs can introduce performance bottlenecks during conversion or discovery unless preprocessing thresholds are tuned with care.

Future work should explore the integration of conformance-checking metrics to rank deviations automatically, the enrichment of traces with complementary data sources such as network flows or endpoint telemetry, and large-scale evaluations across multi-domain environments. Investigating supervised or semi-supervised classifiers that ingest process-mining features may further improve alert prioritization.

Despite these challenges, the results demonstrate that a toolset originally developed for business-process analysis can serve as a powerful complement to traditional security monitoring. When applied to well-curated Windows event logs, process mining reshapes disjointed audit entries into coherent behavioral models, providing organizations with a structured and visually intuitive means of spotting, investigating and documenting insider threats.

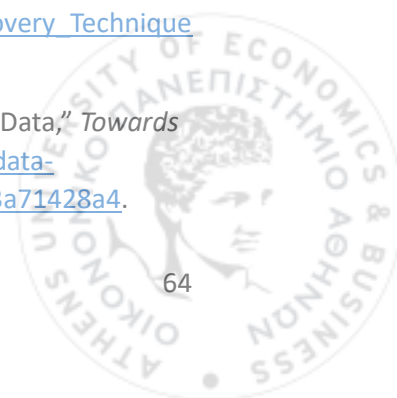


Table of Figures

Figure 1: Windows Event Viewer navigation panel.	6
Figure 2: Windows Event Viewer security log panel.	7
Figure 3: Security event log description.	8
Figure 4: Flow chart of the approach.	20
Figure 5: Successful installment of Sysmon v15.	22
Figure 6: Sysmon Logs on Event Viewer.....	22
Figure 7: Audit Policy Interface.....	23
Figure 8: Pandas DataFrame of security event logs.....	29
Figure 9: DataFrame with the new User column.....	30
Figure 10: The completed dataset.	32
Figure 11: ProM Package Manager browsing installed packages.....	34
Figure 12: ProM Package Manager Installation of new Packages.	35
Figure 13: ProM Tools main interface.....	35
Figure 14: Conversion from CSV to XES action.	37
Figure 15: CSV to XES conversion configuration process.....	38
Figure 16: XES conversion back to CSV for process mining.	39
Figure 17: XES converted table.....	39
Figure 18: Directly follows Graph with Workflow Net.....	49
Figure 19: Direct follows Graph with frequency.	49
Figure 20: Heuristic miner output for L1 loops example.....	50
Figure 21: DFG with frequency for Infrequent events.....	53
Figure 22: Alpha++ miner with frequency for Infrequent events.....	54
Figure 23: Out-of-hours Inductive miner graph (w/o time limit).....	57
Figure 24: Out-of-hours Inductive miner graph (with time limit).....	57
Figure 25: End-to-End Insider-Threat Detection and Response Workflow.....	60

References

- [1] University of Hawai'i – Cyber, "Distinguishing and Understanding Insider Threats," *Forensics Weekly Executive Summary*, 2024. [Online]. Available: <https://westoahu.hawaii.edu/cyber/forensics-weekly-executive-summmaries/distinguishing-and-understanding-insider-threats/>.
- [2] IBM Security Think Blog, "83 Percent of Organizations Reported Insider Threats in 2024," 2024. [Online]. Available: <https://www.ibm.com/think/insights/83-percent-organizations-reported-insider-threats-2024/>.
- [3] HackTheBox Blog, "Decoding Windows Event Logs: A Definitive Guide for Incident Responders," 2023. [Online]. Available: <https://www.hackthebox.com/blog/decoding-windows-event-logs-a-definitive-guide-for-incident-responders>.
- [4] Blumira Labs, "The Benefits of Sysmon for Security Monitoring," 2022. [Online]. Available: <https://www.blumira.com/blog/sysmon-benefits>.
- [5] CEUR-WS, "Process Mining for Cyber-Security Log Analysis," *Proc. SIMPDA 2018* (Short Paper), vol. 2270, pp. 28–33, 2018. [Online]. Available: <https://ceur-ws.org/Vol-2270/short5.pdf>.
- [6] Annals of Computer Science & Information Systems, "Risk-Aware Business-Process Discovery," vol. 25, pp. 1–6, 2019. [Online]. Available: https://annals-csis.org/Volume_25/drp/85.html.
- [7] Springer, "Integrating Risk Information into Business-Process Modelling," in *Business Process Management Workshops*, 2016, pp. 546–557. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-49109-7_44.
- [8] ManageEngine, "Types of Windows Event Logs," ManageEngine Knowledge Base. [Online]. Available: <https://www.manageengine.com/products/eventlog/kb/types-of-windows-event-logs.html>
- [9] Microsoft, "Sysinternals Sysmon – System Monitor," 2025. [Online]. Available: <https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>.
- [10] Microsoft, "Planning and Deploying Advanced Security Audit Policies (Windows 10)," 2023. [Online]. Available: <https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/planning-and-deploying-advanced-security-audit-policies>.
- [11] ResearchGate, "Advanced Process Discovery Techniques – State of the Art," pre-print, 2024. [Online]. Available: https://www.researchgate.net/publication/378827766_Advanced_Process_Discovery_Techniques.
- [12] M. Gupta, "Process Mining to Assess App-User Behaviour from Click-Stream Data," *Towards Data Science*, Medium, 2022. [Online]. Available: <https://medium.com/towards-data-science/process-mining-to-assess-app-user-behavior-from-clickstream-data-8e53a71428a4>.



- [13] ProcessMining.org, “Process Discovery – Directly-Follows Graphs,” 2025. [Online]. Available: <https://www.processmining.org/process-discovery.html>.
- [14] IEEE, “AD2: Anomaly-Detection on Active-Directory Log Data for Insider-Threat Monitoring,” in *Proc. IEEE Int. Conf. Big Data*, 2015. [Online]. Available: <https://ieeexplore.ieee.org/document/7389698>.
- [15] Springer, “Advances on P2P, Parallel, Grid, Cloud and Internet Computing,” *Lecture Notes on Data Engineering and Communications Technologies*, vol. 1, 2017. [Online]. Available: <http://link.springer.com/10.1007/978-3-319-49109-7>.
- [16] M. Raptaki, *Utilizing Automated Process Mining of Information Systems for Risk-Graph Construction* (Master’s thesis, in Greek). Athens, Greece: University of Piraeus, 2021. [Online]. Available: ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ
- [17] Elsevier, “Toward Automated Process Discovery in Service-Oriented Architectures,” *Computer Standards & Interfaces*, vol. 29, no. 5, 2007. [Online]. Available: <https://linkinghub.elsevier.com/retrieve/pii/S174228760700045X>.
- [18] IEEE, “SMARTSCOPY: Process-Mining-Assisted Vulnerability Discovery,” in *Proc. IEEE/ACM Int. Symp. Microarchitecture*, 2018. [Online]. Available: <https://ieeexplore.ieee.org/document/8530773>.
- [19] World Scientific, “Modelling and Analysis of Security Workflows,” *Int. J. Co-operative Inf. Syst.*, vol. 22, no. 3, 2013. [Online]. Available: <https://www.worldscientific.com/doi/abs/10.1142/S0218213013600130>.
- [20] EUDL, “Business-Process Mining for Insider Threat Detection,” in *Proc. CollaborateCom 2013*, 2013. [Online]. Available: <http://eudl.eu/doi/10.4108/icst.collaboratecom.2013.254136>.
- [21] A. Srivastava and J. Medina, “A Survey of Process Mining,” *arXiv preprint*, arXiv:1506.04200, 2015. [Online]. Available: <http://arxiv.org/abs/1506.04200>.
- [22] IEEE, “Federated Process Mining for Large-Scale Log Data,” in *Proc. IEEE BigData*, 2021. [Online]. Available: <https://ieeexplore.ieee.org/document/9336573>.
- [23] P. Tucker, “Insider-threat detectors fail too often. A new tool could help plug leaks,” *Defense One*, Jan. 13, 2025. [Online]. Available: <https://www.defenseone.com/technology/2025/01/insider-threat-detectors-fail-too-often-new-tool-could-help-plug-leaks/402156/>.
- [24] Cybersecurity and Infrastructure Security Agency (CISA), “Insider Threat Mitigation,” CISA, n.d. [Online]. Available: <https://www.cisa.gov/topics/physical-security/insider-threat-mitigation>.



Appendix A – Sample Dataset Statistics in JSON Format

LogonID only as concept:name

```
{  
  "attributes": {  
    "concept:name": "0x100403e"  
  },  
  "events": [  
    {  
      "User": "User2",  
      "concept:name": "A network share object was accessed.",  
      "Level": "Information",  
      "lifecycle:transition": "start",  
      "Event ID": 5140,  
      "time:timestamp": "2025-05-11T16:42:00Z",  
      "System": "LAPTOP-21-02",  
      "Source": "Microsoft-Wind"  
    }  
  ]  
}  
  
{  
  "User": "User2",  
  "concept:name": "Group membership information.",  
  "Level": "Information",  
  "lifecycle:transition": "start",  
  "Event ID": 4627,  
  "time:timestamp": "2025-05-02T14:08:00Z",
```



```
"System": "LAPTOP-21-02",  
"Source": "Microsoft-Windows-Security-Auditing",  
"Task Category": "Group Membership"  
}  
[  
  {  
    "variant": ["A Kerberos TGT was requested."],  
    "count": 709  
  },  
  {  
    "variant": ["A process has exited."],  
    "count": 663  
  },  
  {  
    "variant": ["A network share object was accessed."],  
    "count": 663  
  },  
  {  
    "variant": ["An attempt was made to access an object."],  
    "count": 660  
  },  
  {  
    "variant": ["A new process has been created."],  
    "count": 658  
  },  
  {  
    "variant": ["A logon was attempted using explicit credentials."],  
    "count": 657  
  },  
]
```



```
{
  "variant": ["A Kerberos service ticket was requested."],
  "count": 657
},
{
  "variant": ["A handle to an object was requested."],
  "count": 653
},
{
  "variant": [
    "An account was successfully logged on.",
    "Special privileges assigned to new logon.",
    "Group membership information.",
    "A Kerberos TGT was requested.",
    "An account was logged off."
  ],
  "count": 18
},
{
  "variant": [
    "Group membership information.",
    "Special privileges assigned to new logon.",
    "An account was successfully logged on.",
    "A network share object was accessed.",
    "An account was logged off."
  ],
  "count": 16
}
]
```



LogonID | User | System as concept:name

```
{  
  "attributes": {  
    "concept:name": "User3 |LAPTOP-21-03|0x100521c"  
  },  
  "events": [  
    {  
      "concept:name": "A logon was attempted using explicit credentials.",  
      "Level": "Information",  
      "lifecycle:transition": "start",  
      "Event ID": 4648,  
      "time:timestamp": "2025-05-12T14:45:00Z",  
      "Source": "Microsoft-Windows-Security-Auditing",  
      "Task Category": "Logon"  
    }  
  ]  
}
```

```
{  
  "concept:name": "An account was successfully logged on.",  
  "Level": "Information",  
  "lifecycle:transition": "start",  
  "Event ID": 4624,  
  "time:timestamp": "2025-05-07T16:47:00Z",  
  "Source": "Microsoft-Windows-Security-Auditing",  
  "Task Category": "Logon"  
}
```

```
[
```



```
{
  "variant": ["The Windows Filtering Platform has allowed a connection."],
  "count": 350
},
{
  "variant": ["An attempt was made to access an object."],
  "count": 350
},
{
  "variant": ["A registry value was modified."],
  "count": 350
},
{
  "variant": ["A new process has been created.", "A process has exited."],
  "count": 332
},
{
  "variant": ["A logon was attempted using explicit credentials."],
  "count": 328
},
{
  "variant": [
    "An account was successfully logged on.",
    "Special privileges assigned to new logon.",
    "Group membership information.",
    "An account was logged off."
  ],
  "count": 269
},
```



```
{
  "variant": [
    "Special privileges assigned to new logon.",
    "An account was successfully logged on.",
    "Group membership information.",
    "An account was logged off."
  ],
  "count": 232
},
{
  "variant": ["A Kerberos authentication ticket (TGT) was requested."],
  "count": 228
},
{
  "variant": ["A Kerberos service ticket was requested."],
  "count": 209
},
{
  "variant": [
    "Group membership information.",
    "Special privileges assigned to new logon.",
    "An account was successfully logged on.",
    "An account was logged off."
  ],
  "count": 208
}
]
```

