

MASARYK UNIVERSITY
FACULTY OF INFORMATICS



Comparison of data-oriented and process-oriented approaches for insider attack detection from audit logs

BACHELOR'S THESIS

Marek Laššák

June 2022

DECLARATION

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Marek Laššák

Advisor: RNDr. Martin Macák

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor RNDr. Martin Macák for his guidance, availability and positive attitude. I would also like to thank Safetica Technologies s.r.o. and Ing. Miloš Minařík, Ph.D., for providing the audit log data and consultation. Last but not least, I am deeply grateful to my family and friends for supporting me throughout my studies.

ABSTRACT

This thesis studies and compares the automatized process-oriented and data-oriented methods, namely process mining and data mining, and discusses their applicability for insider attack detection from the audit log data provided by Safetica Technologies s.r.o. The theoretical part presents an overview of the insider threat domain. Subsequently, it introduces the fields of process mining and data mining, where it explains the essential terminology, data format requirements and selected implementations of these methods. In the practical section, the paper describes the methodology behind the choice of techniques, obtaining data from the provided audit logs, the assembled data profile, and the implemented workflows for pre-processing and the chosen methods. The workflows in question were implemented using relevant open-source Python frameworks for process mining and data mining. The thesis then compares process mining to data mining and process mining to decision tree learning, which is chosen as the primary representative of data mining methods. The comparison is approached theoretically and practically. The practical one involves illustrative implementations of process mining and data mining techniques for insider attack detection in created insider attack scenarios. Finally, based on the theoretical and practical observations, the thesis is concluded with a discussion about the usefulness of process mining and data mining for insider attack detection.

KEYWORDS

insider threat, insider attack, audit logs, Safetica, process mining, data mining, decision tree learning, Python

CONTENTS

ACRONYMS	IX
INTRODUCTION	X
1 STATE OF THE ART	1
1.1 Challenges	1
1.2 Insider attack types	1
1.3 Insider profile	1
1.4 Existing solutions	2
2 PROCESS MINING	3
2.1 Definitions	3
2.2 Data representation	4
2.3 Process models	5
2.3.1 Overview	5
2.3.2 Petri nets	6
2.3.3 Other notations	9
2.4 Types of process mining	11
2.4.1 Process discovery	11
2.4.2 Conformance checking	12
2.4.3 Process enhancement	14
3 DATA MINING	15
3.1 Definition	15
3.2 Data representation	15
3.3 Supervised learning techniques	16
3.3.1 Decision tree learning	16
3.3.2 Linear regression	17
3.4 Unsupervised learning techniques	18
3.4.1 Clustering	18
3.4.2 Association rule learning	18
4 METHODOLOGY	20
4.1 Choice of approach	20
4.2 Used technologies	20
4.2.1 Pandas	20
4.2.2 PM4Py	21

Contents

4.2.3	Scikit-learn	21
4.3	Dataset	21
4.4	Implementation overview	23
4.4.1	Preprocessing	23
4.4.2	Process mining workflow	25
4.4.3	DTL workflow	26
5	COMPARISON	28
5.1	Theoretical comparisons	28
5.1.1	Process-oriented and data-oriented approaches	28
5.1.2	Process mining and decision tree learning	29
5.2	Insider attack scenarios	30
5.2.1	Manipulation of configuration files	30
5.2.2	Exfiltration of data to external devices	35
5.3	Practical comparison	41
5.4	Discussion	42
	CONCLUSION	44
A	APPENDIX	46
	BIBLIOGRAPHY	47

LIST OF FIGURES

2.1	A Petri net (WF-net) describing the behavior in Table 2.1.	8
2.2	A BPMN diagram fitting the behavior in Table 2.1.	9
2.3	A DFG fitting the behavior in Table 2.1.	10
2.4	A process tree fitting the behavior in Table 2.1.	10
4.1	The visualization of the general pre-processing workflow.	24
4.2	The visualization of the PM workflow.	25
4.3	The visualization of the DTL workflow.	27
5.1	Sabotage scenario – the obtained heuristic net of the non-malicious behavior.	31
5.2	Sabotage scenario – test cases.	32
5.3	Sabotage scenario – elbow plot established before k-modes clustering.	34
5.4	Data exfiltration – test cases.	37
5.5	Data exfiltration scenario – process discovery results.	38
5.6	Data exfiltration – the decision tree obtained from the training data.	40

ACRONYMS

ARL	Association Rule Learning
BP	Business Process
BPMag	Business Process Managment
BPMN	Business Process Modeling Notation
BPMod	Business Process Modeling
CC	Conformance Checking
DFG	Directly-Follows Graph
DM	Data mining
DTL	Decision Tree Learning
FO	File Operation
IA	Insider Attack
IT	Insider Threat
LR	Linear Regression
PD	Process Discovery
PE	Process Enhancement
PM	Process Mining
PM4Py	Process Mining for Python
SL	Supervised Learning
UL	Unsupervised Learning
WF-net	Workflow Net

INTRODUCTION

Information technology has become an essential part of everyday life and industry. Inside the business sector, these technologies have provided space for efficient cost-management, flexibility, improved productivity and quality of service, and innovation [1]. Thus, business organizations harness this capacity, e.g., through the employment of custom enterprise systems [2]. In addition, the use of such systems allows for automatized data logging and subsequent analyses [3].

From the cost-management perspective, failing to ensure the security of the company's assets can result in significant expenses [4]. Threats that materialize into attacks may lead to scenarios like disclosure of confidential data or denial of service [5]. Consequently, that may cause losses in company's revenues, intellectual property or reputation [6]. Hence, the question of security threats has become an important talking point within organizations.

Insider threats (IT) form a significant subset of such threats. They carry malicious intent directed toward the organization by people internal to the organization [7]. The aspiring attacker already owns access to the internal structure and can hide their malicious actions. Combined with the potential damage, containment becomes a lengthy and costly process. As stated in [8], 27% of data breaches had involved internal actors. Financially, over 15 million USD is spent on average per incident, with the number of incidents rising by 44% from 2020 to 2022 [9]. As a result, there is a motivation to mitigate such threats and efficiently detect incidents.

Despite the numerous challenges, multiple solution approaches have been introduced. Among them, automatized **detection** from different data sources presents a way to identify malicious behavior. Security companies, like Safetica¹, provide the infrastructure to handle insider threats and data leaks. From there, recordings of user actions in the form of **audit logs** can be obtained. The logs can then be used as a data source for different analyses of insider behavior.

Traditionally, behavior patterns are inferred from large data sources using machine learning algorithms. They are used for knowledge-gathering from past data and prediction of new inputs. Such automatized data-oriented methods are referred to as **data mining** [10].

In recent decades, **process mining** emerged as a newer data science discipline [3]. This field aims to automatize the tasks of discovery, analysis and improvement of real business processes. Unlike data mining, it attempts to identify ordered sequences of activities that model process flow. Such models can accurately describe behavior and

¹<https://www.safetica.com/>

be used as a reference point to evaluate new inputs. In turn, process mining became a promising option for insider attack detection.

This thesis aims to utilize and compare both approaches to attempt the identification of IAs from audit logs provided by Safetica. The threat scenarios focus on internal data leaving the company and sabotage. For the practical application demonstration, **decision tree learning** is selected as the representative of traditional data-oriented approaches since it shares several comparable features with process mining. Additionally, these two approaches show the potential to complement each other [3]. The implementation part uses state-of-the-art open-source data science frameworks. Hence, the programming language chosen for implementation is Python [11] since it is a base language for proven and well-documented modules [12, 13].

The thesis is composed of five main chapters:

- In the first chapter – **State of the art** – a general outline of ITs is presented, followed by an overview of relevant solution attempts.
- The second chapter – **Process mining** – provides the essential knowledge for understanding the process-oriented view. The field of process mining is introduced, starting from the organizational business process perspective. Subsequently, the chapter describes data requirements, process modelling and process mining methodologies.
- As the third chapter, **Data mining** explores the traditional data-oriented approaches. It provides the fundamental terminology and classification of methods with examples.
- The fourth chapter – **Methodology** – describes the data profile, utilized technologies, implemented functionality and workflows.
- In the fifth chapter – **Comparison** – features, capabilities and results of the used methods are analyzed, compared and discussed both generally and within the IT context.

The outcome of this thesis is a comparison of decision tree learning and process mining and a discussion about their application to audit log data within the insider attack detection context. The features and capabilities of both approaches are analyzed and compared theoretically, whereas the subsequent practical analysis involves illustrative insider attack detection scenarios.

1 STATE OF THE ART

Not exclusive to the insider threat (IT) field, the security of an organization's digital and physical assets is an ever-evolving and complex problem [5]. It is challenged by the constant influx of new exploit types. New prevention, detection and mitigation methods are being deployed to address this issue. Despite these efforts, the threat is always ahead of the solution – an unprecedented exploit can be found. In addition, the organizations consider the financial costs of security solutions [14].

1.1 CHALLENGES

Insider threats have additional features that make them challenging to manage [6]. In this context, the **insider** is a person already present within the organization – they know the structure of the internal systems and often possess legitimate access to them. Additionally, the number of insiders is generally lower than regular employees. Hence, once the IT materializes into an **insider attack** (IA), their actions are easier to masquerade among the regular traffic. Furthermore, that may lead to the lack of reference material for certain solution approaches [15]. Finally, the diversity in insider profiles and IA types makes establishing an efficient security workflow challenging.

1.2 INSIDER ATTACK TYPES

Different sources [16, 17] show that there is a multitude of IA classifications available. A general outline is provided by Capelli et al. [18]. It distinguishes between three types of IAs with malicious intent. Firstly, **sabotage** refers to inflicting damage to an organization or an individual. Secondly, **theft** addresses espionage and use of the intellectual property for own or third-party benefits. Lastly, **fraud** refers to the unauthorized addition, deletion or modification of an organization's data. Examples of such behavior include insider social engineering, disclosure of critical data or its destruction. It is worth noting that some of these actions, e.g., loss of classified data, can happen unintentionally. In turn, this further complicates the process of identifying real IAs.

1.3 INSIDER PROFILE

As described in [19], there are different motivations for the orchestration of IAs. A self-motivated attacker could be driven into performing malicious acts for financial,

personal or political reasons. Alternatively, an insider candidate can be persuaded by a third-party element. Another class of ITs refers to trained personnel who are purposely installed into an organization.

At the same time, the perspective of unintentional attackers should be considered. For instance, Wall's article [20] classifies them based on employee profiles that can cause data leaks. These include victims of social engineering, accidental data leakers and employees who disregard security policies. The latter category refers to those who intend to make their lives easier (underminers) or be more effective (overambitious employees).

1.4 EXISTING SOLUTIONS

Analogically to the variety in IA types, the pool of solution approaches is large [16]. Ranging from prevention policies to placement of decoys, cybersecurity, data science, social studies and business management are deployed together to handle ITs. The following overview is focused on relevant data and process-oriented solutions, while the related terminology is further established in [Chapter 2](#) and [Chapter 3](#).

There are numerous **data-oriented** techniques for understanding and identification of insider behavior available. However, the most widely adopted approaches are anomaly-based [16]. Such works [7, 15, 21, 22] utilize **data mining** methods to learn behavior patterns from collected user activity data. Precisely, a model of allowed behavior is established from the learning results. Subsequently, behavior that significantly diverges from the modelled norm is analyzed.

Process-oriented methods introduce potential alternatives for IA detection. For example, the work of Bishop et al. [23] utilizes process modelling to study how an election process can be compromised from the inside. Furthermore, **process mining** (PM) allows for the automatization of such solutions. Rodríguez et al. [24] attempt to employ it for insider attacker profiling. Macák's proposal [25] discusses PM as a promising option for anomaly detection and IT mitigation. Concrete examples of PM deployment for the purpose of IA detection are featured in [26, 27, 28].

For further reference, the reader is pointed to IT-related surveys [16, 17]. The access to the aforementioned technologies is facilitated either through commercial and academic tools, such as RapidMiner¹ and ProM², or open-source modules, such as the ones featured in [Chapter 4](#).

¹<https://rapidminer.com/>

²<https://www.promtools.org/doku.php>

2

PROCESS MINING

This chapter introduces the essential terms of the process mining field and describes the general data profile. Furthermore, an overview of process modeling is included, with Petri nets as the focal notation. The last section is dedicated to types of process mining, along with technique examples.

2.1 DEFINITIONS

In a general sense, **process** is a collection of interrelated activities that are initiated in response to an action which aims to achieve a specific goal for the consumer of the process [29]. Within this frame of reference, the focus is on a specific subset of processes – **business processes** (BP). Their activities are called business operations and consist of employees, machines, systems, materials and methods. These are organized together to design, create, and deliver the product or service to the customer.

Naturally, such processes are expected to be initiated repeatedly and in multiple ways [3]. Activity sequences may form non-trivial flows, such as cycles or concurrency. Therefore, it is essential to differentiate between process instances. A **process instance** is defined as an ordered sequence of **events**, where each event represents a particular activity, i.e., a well-defined step, of such instance [3].

Business process management (BPMag) is a field dedicated to tasks such as discovery, analysis, improvement and design of BPs to maximize their efficiency [30]. A vital element of this management approach is **business process modeling** (BPMoD). It provides notations for a comprehensive understanding of a BP in the form of process models [31]. These models can be further used in analyses, simulation and presentation. In the context of BPMag, it is worth noting that modeling of BPs can be based on assumptions, i.e. it does not have to stem from real-world data. That can potentially cause problems, as an incorrectly assumed model may not reflect reality [3]. As one of the focal points within this thesis, process models with examples are explored further in [Section 2.3](#).

At the same time, systems utilized by the business operations have the potential to generate, collect and store large amounts of data [3]. Business-wise, analyses of this data may yield valuable results, e.g., discovering differences between stakeholder expectations and reality. As a sub-discipline of data science, **data mining** (DM) focuses on digitally stored data and automatizes the tasks of knowledge gathering [10]. Since DM is further utilized within the scope of this thesis, [Chapter 3](#) revisits this field in greater detail.

Process mining (PM) can be defined as a field that aims to automatize the tasks of discovery, analysis and improvement of real processes by extracting knowledge from data readily available in today's systems [3]. With these goals in mind, PM represents a connection between the fields of BPMoD and DM through overcoming their limitations. Analogically to DM and contrarily to BPMoD, PM stems from recorded, not assumed data. Vice-versa, PM can comprehend data in a process-oriented way, i.e., create and utilize process models.

Generally, the pool of PM methods is divided into three categories [3]. **Process discovery** algorithms aim to build a process model from input event data. Contrarily, **conformance checking** methods utilize already discovered models. These methods measure the differences between particular process instances and such models. Finally, **process enhancement** techniques use the knowledge gained from event data to modify an existing process model in order to improve its quality. All three types of PM, together with algorithm examples, are revisited in [Section 2.4](#).

A complementary classification, not exclusive to PM, is defined with regards to model-data relationship [3]. In parallel to process discovery, **play-in** methods take data as input and use it to infer models directly. On the other hand, **play-out** approaches take an existing model and generate behavior data from it. Lastly, **replay** methods inputs consist of both model and data. This data is then simulated on the given model. Conformance checking and process enhancement can both be included in the latter category.

2.2 DATA REPRESENTATION

In PM, the universe of objects is composed of individual event records and is called an **event log** [3]. It is represented as a two-dimensional table, where each row corresponds to an event, while each column represents one of its attributes. However, further specification of attributes is required to utilize event logs as input for PM algorithms.

Namely, the identifications of **case** and **activity** attributes are mandatory. These attributes are essential to utilize the event data in a process-oriented way. Unique values of the chosen **case** attribute represent distinct process instances. Therefore, events that share the same **case** value are discrete steps of the same instance. At the same time, **activity** attribute values represent labels for events observed in the log. In other words, such values correspond to all distinct types of actions performed by the process instances.

The remaining requirement is to define the ordering of events within each process instance. By default, the flow of events in time is determined naturally by their positions within the event log [3]. However, provided that event records contain **timestamps** representing the time of execution, the PM algorithms can use this additional information for ordering. In addition, having timestamps allows for measuring process performance.

By identifying mandatory attributes and order, process instances can be recovered from the event data. An established alternate term for a process instance is a **trace** [3]. Formally, it is represented as an ordered multiset of events. By extension, an event log can be represented as a multiset of traces.

The remaining attributes serve purely as additional data. Among them, **resources** provide specifics on what is responsible for the execution of the event. This added knowledge may provide a deeper insight into the examined process during process enhancement.

As an example, [Table 2.1](#) represents actions within e-shop order processing. Aiming to observe the steps of this process, attribute `action` is chosen as the activity and `order_id` as the case. In addition, `date_time` is chosen as timestamp, other attributes become resources. The event log is established, and three different traces representing the distinct orders can be identified. Visually, different process models inferred from this setup can be observed within [Section 2.3](#). Different choices of the case and activity attributes change the point of view, e.g., changing `activity` attribute from `action` to `assignee` showcases which employees were involved during the processing of the order.

order_id	action	assignee	date_time	order_size
1	registered	John	01-01-2022 18:42:00	2
2	registered	John	04-01-2022 13:40:00	1
3	registered	John	04-01-2022 13:45:00	4
2	cancelled	John	04-01-2022 13:52:00	1
2	closed	Matt	04-01-2022 14:21:00	1
1	sent	Rebecca	06-01-2022 06:17:00	2
3	halted	Rebecca	07-01-2022 11:12:00	4
1	closed	Matt	08-01-2022 16:48:00	2
3	cancelled	Rebecca	10-01-2022 18:12:00	4
3	closed	Matt	15-01-2022 17:34:00	4

Table 2.1: Event log example.

2.3 PROCESS MODELS

2.3.1 OVERVIEW

The goal of process modeling is to capture dynamic aspects of processes [32]. A process model describes the process and defines its means of operation.

In practice, the importance and use of process modeling are relevant for various objectives [33]. Modeling provides insight into the design of a process, allowing the stakeholders to identify faults before the process idea gets implemented. As models provide tools for discussion, it opens space for subsequent adjustment and validation

of design decisions. In terms of analysis, models can be used to measure process performance and identify undesired behavior, such as deadlocks.

As for the modeling languages themselves, a plethora of notations is available [3]. The differences are found in terms of modeling capabilities and degree of formality [33]. Notations such as **Petri nets** provide a formal apparatus that can be used for scientific research. On the other side of the spectrum, models such as Data Flow Diagrams serve as an informal language that is easy to understand but unusable for rigorous purposes. In the middle, notations like BPMN or UML activity diagrams can model non-trivial constructions while retaining a simpler notation than some rigorous alternatives. With consideration of their respective limitations, conversions between notations are possible [3]. Ultimately, the choice of modeling language depends on the circumstances, e.g., the nature of the task or the knowledge level of the target audience.

While modeling processes, undesired behavior must be considered in advance [3]. Examples of such behavior are deadlocks, livelocks or unreachable parts of a model. For that reason, most notations utilize or assume various correctness criteria, such as **soundness** described in the following section.

2.3.2 PETRI NETS

Petri nets are among the most researched notations that allow for modeling of concurrency, choices and iteration [3]. Due to their mathematical origin, various formal analysis techniques can be applied to them [34]. The following paragraphs demonstrate that they are executable and provide an intuitive graphical notation.

Formally, Petri net is defined as a triplet (P, T, F) , where [34]

1. P is a finite set of **places**.
2. T is a finite set of **transitions** and $P \cap T = \emptyset$.
3. $F \subseteq (P \times T) \cup (T \times P)$ is a **flow relation**.

Graphically, places are represented as circles and transitions as squares. Flow relation defines directed arcs, visualized by arrows that connect places to transitions and transitions to places. As such, Petri net is a bipartite graph $(P \cup T, F)$ where P and T represent its disjoint sets of vertices [3].

INFORMAL DESCRIPTION OF BEHAVIOR

Informally, this fixed structure can be viewed as a simulation ground for a game of **tokens** [34]. Tokens are graphically represented as dots and can be assigned to places. Each unique token distribution defines a **marking**, i.e., net state. Transitions cannot contain tokens and serve as gateways between places. As such, each transition has its input places and output places. Once each of its input places contains a token, the transition becomes **enabled** for firing. Upon being **fired**, tokens from the input places are consumed, while each output place is granted a new token, potentially changing the marking.

FORMAL DESCRIPTION OF BEHAVIOR

Formalizing this behavior on a Petri net $N = (P, T, F)$, **marking** is defined as a function $m : P \mapsto \mathbb{N}_0$ [34]. This function maps a non-negative integer amount of tokens to each place $p \in P$ and this value is further referred to as $m(p)$. All possible markings of net N form a set M .

Input and output places are defined in terms of flow relation F [34]. A place p is an **input place** of a transition t if $(p, t) \in F$. Input places of a transition t form a set $\bullet t = \{p \mid (p, t) \in F\}$. Inverting the types of start and destination nodes, a place p is an **output place** of a transition t if $(t, p) \in F$. The set $t\bullet = \{p \mid (t, p) \in F\}$ all output places of a transition t . $\bullet t$ and $t\bullet$ are referred to as **preset** and **postset** of t , respectively. Consequently, the term of **transition enabling** can be defined. Precisely, transition t is enabled at marking m if and only if $\forall p \in \bullet t : m(p) > 0$ [34].

Finally, to formalize the **firing** of the transition, weight function w is defined as $w : (P \times T) \cup (T \times P) \mapsto \{0, 1\}$ where [34]

$$w((x, y)) = \begin{cases} 1 & (x, y) \in F \\ 0 & (x, y) \notin F \end{cases}$$

Utilizing weight function w and given a marking m , transition t can fire if and only if it is enabled at m . Upon being fired, a new marking m' is yielded where for all places $p \in P : m'(p) = m(p) - w((p, t)) + w((t, p))$.

LABELED PETRI NET

Aiming to introduce Petri nets to model business processes, correlations between process data and model can be made. Each transition can be seen as a representation of an observable activity within the process [3]. Directed arcs reflect how these activities may follow up on each other. Therefore, it is reasonable to label transitions to achieve better comprehension. Such **labeled Petri net** is formally defined as (P, T, F, A, l) , where A is a set of labels and l is a labeling function $l : T \mapsto A$ that assigns a label to each transition. As a part of their representational bias, Petri nets require each label to be unique to a single transition. The labeling function can be a means to bypass this requirement.

WORKFLOW NETS

As previously mentioned in [Section 2.1](#), BPs are expected to be executed repeatedly, creating distinct instances. The lifetime of an instance is precisely delimited by the first and last event within its ordered event sequence. Consequently, a general model for any instance contains distinct source and sink places, representing the creation and termination of the instance. The event sequence then forms a path between these two points.

Workflow nets (WF-nets) are a subclass of Petri nets that reflects this behavior [3]. Generalizing the preset and postset definition to include both types of nodes, a workflow net is defined as a labeled Petri net (P, T, F, A, l) , where [3, 35]

1. i is a single source place, i.e., $\{p \in P \mid \bullet p = \emptyset\} = \{i\}$.
2. o is a single sink place, i.e., $\{p \in P \mid p\bullet = \emptyset\} = \{o\}$.
3. Every node is on path from i to o , i.e.,

$$\forall n \in P \cup T : (i, n) \in F^* \wedge (n, o) \in F^*$$

where F^* is the reflexive transitive closure of relation F .

Additionally, the criterion of **soundness** is introduced to constrain undesired behavior within a WF-net. A WF-net is **sound** if and only if all the following conditions are met [3]:

1. A place cannot contain more than one token (safeness).
2. If the sink place is marked, all other places should be empty (proper completion).
3. It is always possible to mark the sink place from any marking (option to complete).
4. There are no dead transitions, i.e. for each transition $t \in T$, there is at least one firing sequence that enables it (absence of dead parts).

Due to their nature, WF-nets are a reasonable representation model for PM purposes since traces refer to process instances and can be seamlessly mapped to firing sequences [3]. Within the scope of this thesis, the definitions above provide the necessary tools to construct process models using labeled Petri nets. If deemed necessary, the reader is pointed to extended definitions found in [35]. Assuming that case and activity attributes are `order_id` and `action`, respectively, Figure 2.1 is an example of a WF-net that fits the behavior observed in Table 2.1.

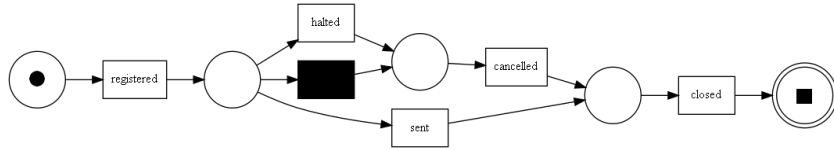


Figure 2.1: A Petri net (WF-net) describing the behavior in Table 2.1.

2.3.3 OTHER NOTATIONS

In the context of PM, **directly-follows graph** (DFG) is a graph in which nodes correspond to activities and which directed edges correspond to directly-follows relationships between the activities [36]. As such, DFGs offer a basic outline of the internal structure of the process. Assuming that the necessary data is provided, the model can be extended with frequency and performance information. As demonstrated later in Section 2.4.1, DFGs are utilized by various process discovery algorithms. By extension, commercial PM software often produces process discovery results in the form of a DFG. However, the notation is limited by its directly-follows nature. Specifically, the absence of non-local scope may lead to misleading models. In addition, DFGs may struggle to capture concurrency concisely. Depending on the input data, that may result in a complicated model that does not show a clear process structure. A sample DFG is provided in Figure 2.3.

Business Process Modeling Notation (BPMN) is a process notation standardized by a Object Management Group consortium [37]. This notation is developed for use in BPMag settings. Unlike DFGs, it captures complex behavior, such as XOR-splits or parallel tasks. Similarly to Petri nets, BPMN models support replay and verification [38, 39]. On the other hand, BPMN models do not utilize places. Instead, a trivial process flow is depicted using only arrows and activities while specific node types are introduced to model different types of non-trivial behavior. In addition, BPMN allows for further model extensions, such as guard conditions in XOR-splits or identification of sub-processes. As a result, these detailed models can be easier to understand across different roles in BPMag. Figure 2.2 is an example of this notation.

Process trees are a representative of block-structured process models [40]. The special feature of this model class is that they are sound by construction. A process tree is an n-ary tree, where leaf nodes represent activities and internal nodes represent operators. These operators model essential structures, such as sequence (seq), exclusive choice (xor), parallel composition (and) and loop (loop). Due to the inherent soundness property, process trees can be utilized as a base for soundness-maintaining conversions into other notations. A process tree object is shown in Figure 2.4.

All three exemplary models were generated from data in Table 2.1 with `order_id` and `action` in the roles of `case` and `activity` attributes, respectively. Other notations and model classes can be explored in [3, 41].

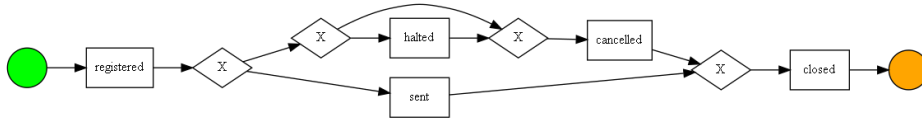


Figure 2.2: A BPMN diagram fitting the behavior in Table 2.1.

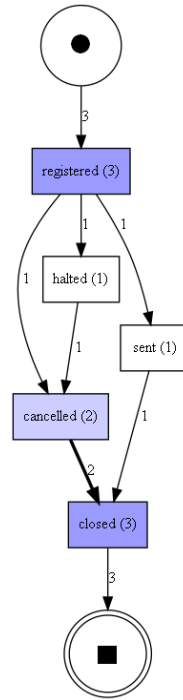


Figure 2.3: A DFG fitting the behavior in Table 2.1.

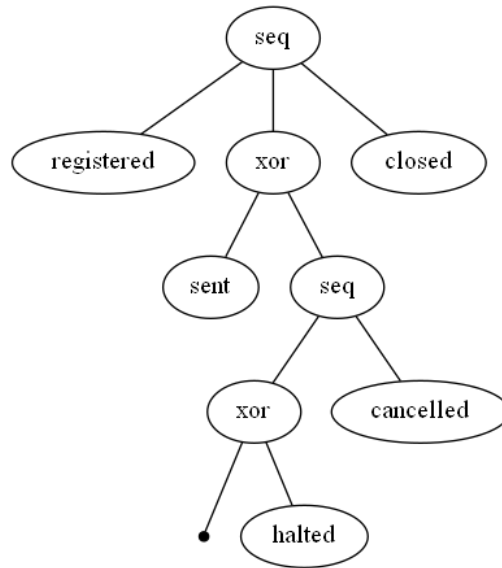


Figure 2.4: A process tree fitting the behavior in Table 2.1.

2.4 TYPES OF PROCESS MINING

As mentioned at the beginning of this chapter, PM methodology can be divided into the tasks of **process discovery**, **conformance checking** and **process enhancement**. Despite being mutually exclusive in terms of their goal, their results may follow up on each other [3]. Therefore all three categories serve a crucial role in managing business processes respectively.

2.4.1 PROCESS DISCOVERY

Process discovery (PD) represents the task of process model reconstruction from the given event log [3, 42]. Discovery itself is often preceded by assessment and pre-processing of the event data. Subsequently, a suitable PD algorithm is applied, resulting in a model that captures the recorded reality.

Generally, the stakeholders are interested in representative, i.e., the most frequent behavior. Identifying this behavior provides a reference point for further assessment. Consequently, that can lead to an explanation of anomaly traces.

MODEL QUALITY

The quality of the discovered model can be evaluated using four criteria [3]. Firstly, **fitness** states that the model should allow for behavior recorded in the data. Secondly, **precision** declares that the model should not allow for behavior that significantly differs from the recorded one. Thirdly, **generalization** states that the model should generalize the example behavior. Finally, **simplicity** criterion requires the model to be as simple as possible. Naturally, these criteria may interfere, e.g., a model with high fitness and precision can lack simplicity. For that reason, PD algorithms offer ways to adjust the abstraction level manually.

CHALLENGES

PD is often challenged by various factors [3]. **Representational bias** describes the implicit limitations of output model notation. For example, Petri nets do not allow two transitions to share the same label. Consequently, a PD algorithm could produce unnecessarily complicated structures. In addition, the **correctness** of the inferred model should be considered.

From the data perspective, event logs may contain **noise**, i.e., infrequent anomaly traces. Alternatively, event data may not provide enough representative behavior. Instead, a large amount of alternatives is presented, causing **incompleteness**. Lastly, **invisible tasks** represent behavior that is not explicitly recorded in the log, yet it is compatible and convenient to put in the model.

The following section demonstrates that some of these challenges can be overcome by choosing a suitable PD algorithm.

PROCESS DISCOVERY APPROACHES

The PM toolkit offers a variety of PD approaches [3]. The algorithms differ mainly in their capabilities, implementation, performance and result notation.

α -algorithm and its extensions utilize the direct predecessor-successor relation between activities [42]. From there, they infer more complicated constructions and produce a Petri net. α -algorithm depends on the presence and order of activities within the log, disregarding their frequency. In turn, that makes it prone to noise and incompleteness. The representational bias of Petri nets disallows it to model duplicate transitions. It is unable to utilize invisible tasks and handle loops of lengths one and two [43]. Finally, the resulting model is not guaranteed to be sound.

In contrast, **heuristic mining** takes frequency into consideration [44]. Constructs are added to the resulting model only if the frequency of particular behavior meets a predefined threshold. Consequently, this reduces problems with noise and incompleteness. However, the potentially valuable anomaly behavior is discarded. Various implementations, such as [44] and [45], produce a heuristic dependency model (heuristic net), which can be converted to a Petri net. Similarly to the α -algorithm, this model has no soundness guarantees.

Inductive mining algorithms aim to logically divide the input log into smaller sublogs [3, 43]. Depending on the implementation, it can be applied recursively. This procedure defines a process tree that can be translated to a Petri net. Process trees are sound by definition, and that property is retained by the Petri net. Unlike previous algorithms, invisible tasks can be utilized. That is particularly useful for mapping process trees to WF-nets. In addition, the inductive approach is highly modifiable and extensible. These qualities make it one of the most popular PD options [43].

Summary of features is provided in Table 2.2. Detailed formal descriptions, along with other approaches, such as genetic, fuzzy or region-based mining, can be explored further in [3].

Feature	α -algorithm	Heuristic mining	Inductive mining
considers frequency	no	yes	yes
utilizes invisible tasks	no	no	yes
handles short loops	no	yes	yes
soundness of result	no	no	yes

Table 2.2: PD algorithms features summarized.

2.4.2 CONFORMANCE CHECKING

The term **conformance checking** (CC) refers to the task of confronting event data with an already existing process model [3]. This model describes the learned and therefore expected behavior. A CC algorithm provides the method to evaluate new input traces based on their similarity to such behavior. Unlike PD, where the result is created from past data, CC can handle new inputs in real-time.

CC can be utilized to measure the quality of models inferred by PD. Different models based on the same log can be ranked according to their fitness. In this context, fitness represents the percentage of traces that fit the modeled behavior according to the used CC approach. While fitness could be considered the paramount quality, it is often beneficial to complement it with other metrics such as appropriateness [46].

In a BPMag setting, CC can become an auditing tool. Here, the model represents the template and constraints of the behavior of resources. Low fitness indicates the mismatch between stakeholder expectations and reality. That potentially signals the need for change in process structure or actions of its resources.

Naturally, CC can be used to pinpoint anomaly traces. Upon detection, it can be determined whether the irregular behavior is desired or not. An example of desired behavior could be an optimization achieved through performing an unexpected action. On the other hand, a suspicious sequence of operations might imply an insider security threat.

CONFORMANCE CHECKING APPROACHES

Analogically to PD, there are multiple ways to approach CC [3]. The provided examples differ mainly in terms of result accuracy and complexity.

Causal footprints present a CC method based on directly-follows relationships between activities [47]. These relationships are learned and recorded inside a footprint matrix. Such matrices are calculated for the log and model, respectively. Subsequently, matrices are compared, and result fitness is produced as a ratio of matching elements and matrix size. This approach is independent of model notation and can be utilized for model-to-model and log-to-log comparison. While intuitive, footprint comparison lacks in several departments [3]. Notably, footprints are limited by their local directly-follows scope, disregarding the global outline. For example, two different processes may share the same footprint. Additionally, unlike other methods, frequencies are not being considered.

Using Petri nets as a base notation, **token-based replay** can be utilized [46]. The amounts of produced and consumed tokens are recorded during replay for each trace. If the replay is halted due to the trace not being compatible, missing tokens are provided, counted, and the replay is resumed. Upon reaching the final marking, the model is checked for the remaining tokens. Finally, trace fitness is calculated using these four values. Additionally, recording these values for each place in the model highlights which parts cause deviations. Such trace replay can be extended to evaluate overall log fitness. However, in addition to notation restriction, token-based replay struggles with large models. In exceptional cases, the replay of traces on such models may yield large amounts of tokens that devalue overall fitness [3].

Alignments method aims to find the most fitting path in the model in regards to the input trace from the log [43]. A predefined cost function assigns zero to matches, while different positive values are mapped to mismatches depending on their severity. Therefore, the optimal alignment is the one assigned with minimal value. An alignment evaluated with zero signifies that the trace fits into the model. This approach

is independent of notation and allows to pinpoint problematic parts conveniently. The knowledge of optimal alignment can be helpful in further analyses, e.g., during process enhancement [3]. Nevertheless, the implementation is significantly more complicated than previous methods, resulting in higher computational and memory demands.

Other applicable CC methods can be observed in [43]. Table 2.3 is a recapitulation of previously mentioned algorithms and their key features.

Feature	Causal footprints	Token-based replay	Alignments
notation dependent	no	yes	no
considers frequency	unable	able	able
performance	fastest	slower	slowest
memory requirements	lowest	lower	highest
large model problems	no	yes	no

Table 2.3: CC algorithm features.

2.4.3 PROCESS ENHANCEMENT

The third and last category of PM, **process enhancement** (PE) is concerned with altering of existing models [3]. As a crucial part of any business operation life cycle, it provides valuable viewpoints on the process and is a means of process evolution.

Model can be **extended** with nonessential event data. Particularly, timestamps can be used to measure process performance, such as throughput time of activities. Henceforth, this allows for the identification of bottlenecks. At the same time, resource attributes can be used to highlight the roles of people and devices. Other noteworthy insights include frequencies and guard conditions [3, 48]. Finally, PE can be deployed to **repair** the model to address the differences within recorded behavior.

3 DATA MINING

This chapter introduces the basic terms of data mining and its use cases. Furthermore, it presents a high-level view of data mining techniques and describes selected methods in greater detail.

3.1 DEFINITION

Data mining (DM) is defined as an automatized process of discovering patterns in data with the intention of gaining advantageous information [10]. Within this frame of reference, the term **data** characterizes large quantities of digital information stored in databases. Alternatively, DM is an application of machine learning methods to such data [49].

3.2 DATA REPRESENTATION

In this domain, the universe of objects is called a **dataset** [50]. It can be viewed as a generalization of the previously established process mining event logs described in Section 2.2. Similarly, it is visualized as a two-dimensional table, where each row corresponds to an individual record, and each column represents an attribute of such record. Unlike in PM, there is no general mandatory pre-identification of attribute roles like case or activity [3]. However, as shown in the following sections, different dataset format constraints may emerge depending on the DM method.

When choosing an appropriate DM technique to solve a problem at hand, it is essential to identify the type of attributes within the dataset. An example of this can be seen in the Chapter 4, where it refers to a crucial implementation detail during DM-related pre-processing. The definitions and descriptions of DM methods provided in sections Section 3.3 and Section 3.4 involve attribute types, respectively. Hence, a simple type classification is provided below.

Generally, any attribute can be classified either as **categorical** or as **continuous** [50].

Categorical attributes are represented by discrete values extracted from the corresponding pre-defined finite domain sets. This subcategory can be further split into nominal and ordinal attributes, the difference being that the values of an ordinal attribute can be arranged in a meaningful order. An example of an ordinal domain is {LOW, MEDIUM, HIGH}, while {TRUE, FALSE} would be a typical case of a nominal domain. The values of a categorical attribute serve as labels to classify records.

Continuous attributes are represented by numerical values. Unlike categorical attributes, they can be used in mathematical calculations. A simple example would

be a column containing the final test scores on a scale from 0 to 100 points in a student dataset.

In some cases, conversion from continuous to categorical attribute or vice versa is required. To illustrate, the continuous final test score attribute from the last paragraph could be categorized into {PASS, FAIL}. Label PASS represents the number of points from 50 to 100, while label FAIL corresponds to values from 0 to 49. Contrarily, categorical values {PASS, FAIL} could be converted to 1 and 0, respectively.

In general, the pool of DM techniques is divided into two main categories – **supervised** and **unsupervised** learning [50]. With the advancements in the DM field in recent decades, other categories, e.g., semi-supervised learning and reinforced learning, were introduced [49]. Some concepts, such as anomaly-based detection, can overlap into multiple categories and offer different points of view. The focus of the following sections is to explain the terms of supervised and unsupervised learning and provide examples of such methods.

3.3 SUPERVISED LEARNING TECHNIQUES

Supervised learning (SL) techniques distinguish themselves from the unsupervised methods by the additional requirement of labeled data records. Specifically, a subset of attributes is chosen as an **output (response) variable** ahead of the execution of a particular SL algorithm. The remaining attributes are marked as **input (predictor) variables**. Finally, the aim is to describe the mapping from input to output variables [49].

While many SL algorithms are available [51], the final choice ultimately depends on the given dataset. The split in attribute typing described in previous [Section 3.1](#) defines two subsets of SL techniques — **classification** and **regression**.

Classification methods, such as decision trees, assume a categorical response variable [50]. They aim to classify it according to the predictor variables accurately.

Contrastingly, **regression methods** presume a continuous response variable. The aim is to define a function representing the predictor-response relationship with the least amount of error [49]. An example of a regression technique is linear regression.

3.3.1 DECISION TREE LEARNING

Decision tree learning (DTL) is an example of classification method. It assumes a single categorical response variable that labels the data records [50]. The final product of the DTL algorithm is a tree with maximal arity of n , where n is the size of the largest domain set among all the predictor attribute domains. In order to avoid an over-complicated tree structure, continuous attributes are typically pre-processed to categorical ones.

Each internal node of the decision tree represents a decision point and corresponds to a subset of all records. Children of such nodes represent disjoint subsets of their parent’s record set. The distribution of the parent’s set is defined by the unique values of a particular predictor attribute. The tree-building algorithm chooses this splitting

attribute according to its specifications. The implemented splitting criteria involve entropy or Gini index [51]. Leaf nodes represent the final classification of records according to the response variable.

Finally, each path from the root to the leaf node corresponds to a unique classification rule. This rule can be described as **IF** x **THEN** y , where x is typically the conjunction of preconditions and y is the conclusion [10]. Preconditions are mapped to values of splitting predictor attributes, respectively, while the conclusion is mapped to a value of the response variable.

Decision trees provide **data compression** and **prediction** [50]. A concise model representing thousands of records in a human-readable way is inferred by correctly setting the tree-building algorithm parameters. For prediction purposes, a decision tree can be learned from a training dataset. Subsequently, this model can be utilized to classify new inputs. A typical application of decision trees is the detection of unauthorized financial transactions. In this case, a tree is learned based on the parameters of past transactions. The bank system can then deem the transaction suspicious if it does not conform to the customer's usual behaviour.

3.3.2 LINEAR REGRESSION

Linear regression (LR) is a representative of regression analysis. Attributes are processed as continuous. Generally, any type of regression can be defined as

$$y = f(x_1, \dots, x_n) + \epsilon$$

where y is the response variable, f is a statistical model and ϵ is the random error [52]. x_1 to x_n represent predictor variables that are the input for the model. In LR, this model is required to be linear. Therefore, it is defined as

$$f(x_1, \dots, x_n) = \beta_0 + \beta_1 \cdot x_1 + \dots + \beta_n \cdot x_n$$

The values of coefficients β_1 to β_n are initially unknown and are learned based on the dataset. Each record i is evaluated and produces error value ϵ_i which is the distance between result value and evaluation of the model, i.e.,

$$\epsilon_i = |y_i - f(x_{i_1}, \dots, x_{i_n})|$$

The ultimate goal is to define a model f that fits the overwhelming majority of records with the least amount of error.

LR provides three primary use cases [52]. Firstly, the resulting model describes the casual relationship between responses and predictors. For illustration, a dataset of employee data containing age, years of experience, productivity rating, and average sleep per night is provided. After applying LR, it can be seen how productivity rating depends on the other factors. Secondly, using this gained knowledge, it can be distinguished which predictors are significant, e.g., age could be identified to have little impact. At the same time, the amount of sleep could be deemed the most influential

factor in employee productivity. Last but not least, a trained model can be used to predict new inputs. For instance, the productivity of a 50-year-old employee with ten years of experience and an average of 4.5 hours of sleep can be estimated.

3.4 UNSUPERVISED LEARNING TECHNIQUES

As implied at the beginning of [Section 3.3](#), **unsupervised learning** (UL) techniques do not use labeled data. In other words, there is no input-output distinction, and every attribute is viewed as input. UL algorithms aim to identify regularities in such data [49].

3.4.1 CLUSTERING

Clustering methods seek to aggregate similar data records into groups called clusters [49]. There are various approaches to achieving this goal [53]. To name a few, distance-based methods [51], such as k-means clustering, project the dataset into an n-dimensional space and iteratively split it into subspaces that correspond to the final clusters. Some algorithms, such as k-means, are intended for continuous data, whereas other solutions, e.g., k-modes clustering, were created for categorical inputs [54, 55]. Alternatively, hierarchical clustering iteratively merges smaller, similar clusters into larger ones, thus creating a new hierarchy level with each iteration. Each level defines a clustering with a different degree of abstraction. Other types of clustering may utilize the records' statistical distributions and probability properties. Some methods require each record to belong to a single cluster, while others allow cluster intersections [10].

A simple example of a clustering use case is voter analysis. Having a dataset of voter data, such as age, chosen political party, level of education, relationship status, and the number of children, the demography of the voter base could be identified. For example, clustering may be employed to identify the most prominent groups sharing mutual political views. By extension, a political party could implement targeted strategies to improve communication with different voter groups by learning this knowledge. In a more general sense, clustering can be used to divide a sizeable problem into smaller ones to improve the performance of computations and the readability of DM results. As a concept, clustering has many use cases in other fields, such as image compression, information retrieval, or bioinformatics [49].

3.4.2 ASSOCIATION RULE LEARNING

Association rule learning (ARL) shares several similarities with DTL described in [Section 3.3.1](#). Firstly, it typically handles attributes as categorical [50]. Secondly, its goal is to infer logical rules from the dataset, and such rule can also be described as **IF** x **THEN** y . However, as there is no response variable to limit the scope of rule discovery, any pattern can be discovered. In addition to that, while basic DTL addresses

3 Data mining

only a single response variable as the conclusion, ARL is capable of considering combinations of multiple attributes in that role [10]. Thus typically, the x represents a conjunction of preconditions, and y represents a conjunction of conclusions.

A naive, unbound ARL algorithm discovers any pattern within the data as long as it is present, regardless of its relevancy [3]. This fact implies that even a tiny dataset could theoretically produce an overwhelming number of rules. Therefore, it is crucial to filter discovered rules according to their informative value. There are multiple metrics for measuring the relevancy of a rule, such as support and confidence [10]. Limiting the scope of rule discovery by pre-defining threshold values for these metrics can lead to a comprehensive set of rules with high information value.

ARL can be utilized to conclude from a dataset. For example, applying ARL on a log of suspicious operations to discover a threat can lead to identifying common root causes [56]. In the context of election data example from Section 3.4.1, ARL could discover patterns such as voters under 30 years of age with higher education generally vote for ecology-oriented parties and are not married.

4 METHODOLOGY

This chapter describes the implementation of previously-established methods to the audit log data provided by Safetica Technologies s.r.o. The input dataset is explored, along with the descriptions of involved technologies, functionality, and workflows.

4.1 CHOICE OF APPROACH

As declared in the thesis goals, the implementation part utilizes relevant Python frameworks for process and data-oriented solution attempts. On the process-oriented side, PM is chosen for implementation since it is a popular automatized solution in both academia and industry [12]. In contrast, as shown in Chapter 3, the DM toolkit presents a plethora of distinct approaches. Thus, a particular method is chosen as the representative. In this role, the choice of DTL is based on existing parallels with PM. More specifically, both process discovery and DTL data compression infer a model that represents the behavior recorded in input data. In addition, conformance checking and DTL prediction share the idea of confronting an existing model with new inputs. While the similarities end at this conceptual level, both methods show the potential to offer valuable and unique points of view on the problem [3]. Their respective results could complement each other, particularly in the IA detection context. This potential is further explored in Chapter 5.

4.2 USED TECHNOLOGIES

4.2.1 PANDAS

Pandas [57] is an open-source Python framework for data analysis and manipulation. Its main features include data import and export functionality and efficient in-memory management of large datasets. The focus of the implementation is on being easy to use and high performance. The following sections demonstrate that the framework's functionality is present within multiple mainstream data-science-related libraries for Python.

Contribution details, along with the documentation, are available on the official website¹. Within this thesis, version 1.4.1. (February 2022) is utilized as the primary data pre-processing tool.

¹<https://pandas.pydata.org/>

4.2.2 PM4Py

Process Mining for Python (PM4Py) [12] is an open-source PM framework developed by the Fraunhofer Institute for Applied Information Technology. PM4Py contains state-of-the-art algorithms and tools for all three types of PM and complementary data analysis. To achieve this, PM4Py was developed following existing popular data science Python frameworks. The project’s idea is to provide easier access to extensions and customized usage of PM methods. That exceeds commercial PM tools, such as Disco², which tend to offer a user-friendly, but ultimately black-box toolkit.

The project’s documentation, source code and developer contact information are reachable from the official website³. This thesis utilizes the currently newest version 2.2.20.1. (April 2022) for PM-related purposes.

4.2.3 SCIKIT-LEARN

Scikit-learn (sklearn) [13] is an open-source Python module for medium-scale data mining solutions. The framework offers a well-documented and accessible toolkit of SL and UL methods, including classification, regression and clustering algorithms. Through inclusion of other mainstream data-handling and computation modules, such as previously mentioned pandas, scipy [58] or numpy [59], the module is accessible and computationally efficient. Furthermore, it provides means for visualization, model selection and pre-processing.

The project’s official website⁴ contains links to the documentation, source code and developer contact. This thesis utilizes version 1.0.2. (December 2021) for DM-related purposes.

4.3 DATASET

The input dataset, further referenced as `file_log`, contains records of user file handling within a single organization provided by Safetica Technologies s.r.o. It incorporates anonymized data from various audit logs, most notably the file operation and application usage logs. These types of logs were chosen because the combination of their contents allowed for interesting process-oriented analyses. The establishment of the dataset involved the joining of multiple relational database tables, feature selection and filtering. In addition, as demonstrated in the following sections, the obtained data required further pre-processing. Each row of the dataset refers to a distinct file operation (FO) recorded by the system. These records include the following attributes:

- **id** – a unique integer ID of the record.
- **id_user** – an integer ID referencing the user who performed the FO.

²<https://fluxicon.com/disco/>

³<https://pm4py.fit.fraunhofer.de/>

⁴<https://scikit-learn.org/stable/>

- **id_pc** – an integer ID referencing the device where the FO was executed.
- **operation_type** – an integer representing the type of FO (e.g., Move, Delete).
- **action** – an integer value that describes the type of action taken by the system to the FO (e.g., Allow, Deny).
- **date_time** – the timestamp specifying when the FO happened.
- **file_name** – the file name string.
- **file_size** – an integer value representing the file size in bytes.
- **src_type** – an integer representing the type of location where the FO originally took place (e.g., Local filesystem path, External device).
- **dest_type** – an integer representing the type of location where the file appeared after the FO (e.g., after Move or Copy), utilizes same values as `src_type`.
- **src_path** – the filesystem path string referencing the location of the file before the FO.
- **dest_path** – the filesystem path string referencing the location of the file after the FO.
- **is_tagged** – a boolean flag (represented as an integer) signifying whether the file is tagged explicitly with a degree of importance.
- **app_name** – a name string of the application within which context the FO was executed.
- **to_safe_zone** – a boolean value (represented as an integer) signifying whether the destination is verified as a safe location.

The provided dataset is considered to contain only regular, non-malicious behavior. For the selected IA scenarios, the malicious behavior is created manually and can be implanted into the `file_log`. The attribute that labels such behavior is present within the dataset as:

- **is_malicious** – a boolean value (represented as an integer) signifying whether the FO is labeled as malicious.

Finally, the following attribute is not present in input data and is established during pre-processing:

- **file_id** – an integer ID of the file involved in the FO.

4.4 IMPLEMENTATION OVERVIEW

A small Python project for the handling of `file_log` data was implemented as a part of this thesis. It utilizes the aforementioned Python libraries for data loading, processing, customizable data filtering and application of PM and DTL methods. The project code and installation manual are available in [Appendix A](#).

4.4.1 PREPROCESSING

INPUT DATA FORMAT AND LOADING

The input data assembled from the provided database tables is contained in the form of a csv file with `;` as the value separator. The ordering, type and meaning of the values conform to the dataset description in [Section 4.3](#). Analogically, optional training data from a separate file can be loaded and included. The loaded data is then represented and further processed in memory in the form of `pandas.DataFrame` object.

HANDLING OF INVALID AND EMPTY VALUES

The first part of pre-processing handles missing values of the string attributes. The respective default placeholder value replaces such values. Following that, `file_size` column is checked for negative values. Negative amounts are replaced with 0, effectively marking their size as unknown.

The final part of this phase involves records of certain types of FOs, e.g., `open` or `copy`. Within these records, one of the `src_path` and `dest_path` is an empty path string. That is expected since these FOs happen in-place, and thus no path change is recorded. However, for the following processing, the `src_path` and `dest_path` values are synchronized with the non-empty string of the pair.

HANDLING OF STRING ATTRIBUTES

In the second part, each unique `src_path` and `dest_path` is assigned to a unique integer identifier. The same process is then applied to `app_name` strings. That is done to reduce the complexity of further operations and allow the DTL implementation to utilize these attributes. By default, both of these string-to-integer mappings are stored to separate `json` files for reference. If these files already exist and the user trusts their content, there is an option to load their content instead of re-creating the mapping. That allows for optimization in a scenario where non-string attributes were changed in the input csv file, and data reload is required.

ESTABLISHMENT OF FILE IDENTIFIERS

Particularly for the PM view on the FOs, knowing which events refer to each file is essential. However, the source audit logs do not contain an explicit file identifier attribute. Using `file_name` or `src_path` in this role is insufficient. This is due to certain FOs, such as the instances of `Move`, where `src_path` and `dest_path` values expectedly

differ. Hence, the transitivity between paths would not be captured. For that reason, reachability trees are established using path identifiers to capture such relations. Once established, a new column for `file_id` is created within the `DataFrame`. Finally, each record receives its `file_id` value based on the root of the dependency tree that contains its `src_path` ID.

RESULT DATASETS

At this point, the general pre-processing is technically complete. With the exception of the `file_name` and `date_time` attributes, the `DataFrame` object contains purely numeric values in each column. A copy of this `DataFrame` is made and is later stored into the first output csv file. This numeric-content version of the pre-processed data is intended for DTL workflow.

Returning to the original `DataFrame`, the numeric values of the selected columns in the original are translated to their respective string labels. Thus, a human-readable version of the dataset is created. The complete mappings used for these translations are present within the project code referenced in [Appendix A](#). Finally, it is stored in the second output csv file. This second dataset is intended for PM workflow.

The whole pre-processing mechanism is summarised in [Figure 4.1](#).

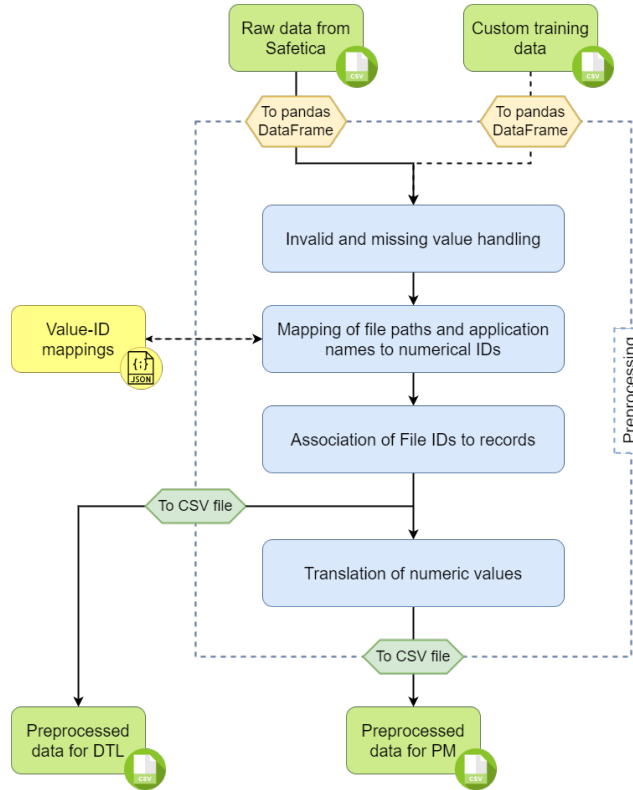


Figure 4.1: The visualization of the general pre-processing workflow.

4.4.2 PROCESS MINING WORKFLOW

The PM workflow used within this thesis is built mainly on the functionality from PM4Py. It starts with the loading of translated csv file obtained from pre-processing to DataFrame. Subsequently, the roles of the attributes can be specified. Namely, the presence of `case`, `activity` and `timestamp` attributes is mandatory. Optional inclusion of `src_type` and `dest_type` through concatenation with respective activity labels is supported. Once the roles are specified, the DataFrame is ordered by `timestamp` and converted to an `PM4Py.EventLog` object using the built-in converter.

At this point, the log is practically ready for PM application. However, to manage the scope of the data for scenario-based examination, several custom filters tailored to the `file_log` profile have been implemented.

The first round of filtering is executed at the event level. In other words, the events are treated as individual units, and the sequence relation is disregarded. In this category, the selection of events within the given time interval has been utilized.

The second round operates on the trace-level, i.e., considers the event sequences. PM4Py provides functionality for basic filter types, trace trimming and the creation of custom filters. For instance, filtering based on trace length, activity presence and trace trimming was implemented in the thesis project.

Finally, the filtered log can be used as input for adjustable PD and subsequently CC implementations provided by PM4Py.

The complete PM workflow is captured in [Figure 4.2](#).

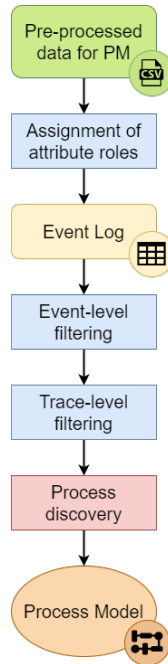


Figure 4.2: The visualization of the PM workflow.

4.4.3 DTL WORKFLOW

The DTL workflow is facilitated through `sklearn` and `pandas` functionality. Before executing the algorithm, the desired predictor and response attributes are identified, along with the parameters for the DTL classifier. Notably, the proportional split of the input dataset into training and test parts can be specified. The requirement of numeric attribute values is enforced by the `sklearn` implementation of DTL. Once the parameters are provided, the loading of pre-processed numeric-content csv file commences. The dataset is contained in the form `pandas.DataFrame`.

Once the data is loaded, the dataset is filtered based on the desired attribute values. This includes filtering based on `date_time` interval, `operation_type`, `src_type`, and `dest_type`.

After that, the attributes selected neither as predictor nor response variables are removed from the dataset. For the `file_log` examination purposes, the attributes are expected to be categorical. To allow this, several remaining attributes, such as `src_type` or `file_size`, are further categorized to reduce the diversity in their contents. Further specification of these categories is provided in the code referenced in [Appendix A](#).

Once the data is processed, the `DataFrame` object is split into the predictor and response `DataFrame`, respectively. However, the handling of values has to be addressed before applying DTL. In particular, the `sklearn` implementation treats attribute values as ordinal. That is not desired since the dataset contains nominal values. Hence, both datasets are extended with dummy data to simulate the nominal handling of attribute values.

Finally, a `sklearn.tree.DecisionTreeClassifier` object is obtained using the predictor and response datasets. The pre-specified parameters adjust the process of model inference. The described workflow is summarized in [Figure 4.3](#).

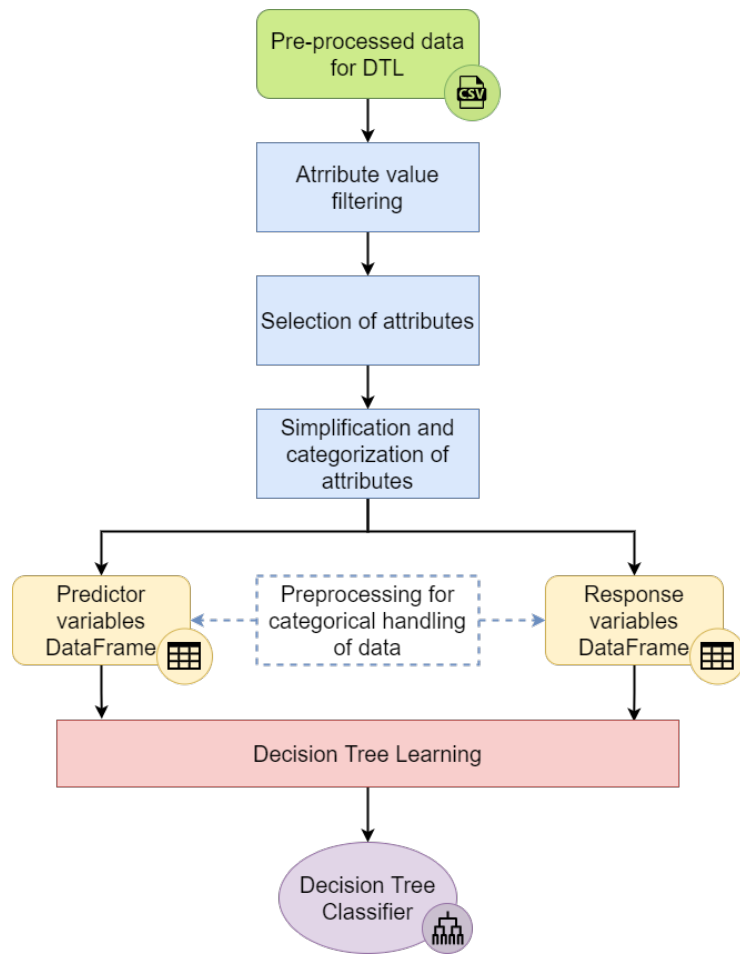


Figure 4.3: The visualization of the DTL workflow.

5 COMPARISON

This chapter features a theoretical comparison of automatized data and process-oriented approaches. Furthermore, the illustrative applications of PM and DTL are provided and compared in the context of IA scenarios to showcase their unique solution capabilities. Finally, a discussion about their respective usefulness is presented.

5.1 THEORETICAL COMPARISONS

The first section examines the general PM and DM approach comparison, while the second focuses on PM and DTL. Both comparisons originate from the theoretical background featured in [Chapter 2](#) and [Chapter 3](#).

5.1.1 PROCESS-ORIENTED AND DATA-ORIENTED APPROACHES

As suggested in [Section 2.1](#), DM can be viewed as a parent discipline of PM. Both methodologies stem from real data and provide automatized tools to conclude patterns and information from such data. In the context of the data-model relationship, more parallels can be identified. For instance, that can be seen in the selected SL methods in [Section 3.3](#), where the models are inferred and can be then used to predict new inputs. These two actions can be related to PD and CC, respectively.

The main difference between data and process mining is that DM methods do not consider end-to-end processes. In other words, DM algorithms are unable to discover nor analyze process models expressed in terms of Petri nets or BPMN diagrams [3].

As observed in [Chapter 3](#), most DM approaches consider each record of being an independent unit of information. On the contrary, the PM event records relate to other events by sharing a mutual case identifier. Additionally, the ordering of such related events matters, as it determines their flow in time. Those significant restrictions are automatically disregarded within the individualistic nature of most DM approaches. Hence, it inherently leads to fewer pre-existing dataset content and format restrictions. Most notably, there is no universal prerequisite to assign roles to attributes, unlike mandatory case and activity role identifications in PM event logs.

Despite their differences, PM and DM methods can be combined to achieve better results [48, 60]. Assuming that the data profile allows it, multiple viewpoints on the same problem can be obtained. In addition, use cases provided by this crossover are problem decomposition, a deeper examination of previously discovered models, and enhancement of model quality [3].

To conclude this general comparison of automatized data and process-oriented approaches, it can be claimed that the focus of traditional DM methods is on the **horizontal** plane of the dataset since it focuses on the individual records and the values of their attributes. On the other hand, the functionality of PM is based on the **vertical** view of the data, focusing mainly on the sequential relations between events. As observed in the following sections, both perspectives provide a unique way of data handling, and their combination can enhance their results.

5.1.2 PROCESS MINING AND DECISION TREE LEARNING

The parallels between PM and DTL have already been mentioned in the previous chapter. The following comparison focuses on both methods' play-in, replay, and change of perspective features since these three areas show the most significant distinctions. Additionally, these particular areas are relevant for IA detection implementation and understanding of their respective capabilities.

PLAY-IN FEATURES

The previously established data view direction is the primary source of differences. As implied in [Section 3.3.1](#), DTL handles the data horizontally, identifying which predictor attribute brings the most information value at the time towards further classification. Being an SL method, DTL requires the identification of the response attribute for pattern mining.

On the other hand, PM does not require a response variable. Once the `case` and `activity` roles are identified, PD can infer models directly without adhering to a pre-specified target. The view of the data is vertical and non-essential attributes are disregarded during model discovery.

REPLAY FEATURES

The individualistic approach of DM is also apparent in DTL prediction. Here, each record is classified separately according to the provided decision tree. This classification is based purely on the content of the record's attributes.

On the contrary, PM is not a classification method and operates the replay at the trace level. CC methods, such as trace alignments and token-based replay, measure how closely the trace conforms to the modeled behavior and highlight mismatch points. By default, the focus is only on the `activity` values and their sequential relations, whereas the other attributes are not considered.

CHANGE OF PERSPECTIVE

Without changing the dataset content and size, the DTL scope can be changed by selecting a different response variable. Alternatively, it can be modified by including or reducing predictor attributes. However, both of these adjustments presume that

the content of the selected features is compatible with the particular DTL implementation and suitable for the given data profile.

In PM, the change of perspective can be achieved through a sensible choice of different case or activity attributes. Combining multiple attributes in either of these roles is also possible by concatenating their values at the event level. The adjustments to the selection of non-mandatory variables are not reflected in PD results. Nonetheless, such adjustments may be relevant for PE.

SUMMARY

It can be concluded that the implementation of the common concepts significantly differs between the two techniques and that these differences stem mainly from the observations made in the [Section 5.1.1](#).

5.2 INSIDER ATTACK SCENARIOS

To illustrate the IA detection capabilities of both methods, I created two simplified IA scenarios based on the audit log content. The first scenario features an unlabeled dataset, whereas the second one features pre-identified, labeled malicious behavior.

5.2.1 MANIPULATION OF CONFIGURATION FILES

In the first scenario, I aimed to detect sabotage of the company's systems. In this case, the sabotage was carried out by illegally manipulating the configuration files. This detection illustration involved an unlabeled dataset with limited attributes.

DATASET

Due to the lack of reference data for instances of write FOs in `file_log`, I had to create the data manually. For demonstration purposes, I established a simplified dataset based on the `file_log` content. The following attributes were included to the dataset:

- `file_id`,
- `id_user`,
- `operation_type`,
- `date_time`.

This dataset contains thirty-five FO records associated with eight different configuration files and four users. A data sample is provided in [Table 5.1](#). These records are further regarded as non-malicious, authorized behavior.

5 Comparison

file_id	id_user	operation_type	date_time
1	1	Open	2021-04-22 12:29:00
2	1	Open	2021-04-22 13:23:00
2	1	Open	2021-04-22 13:24:06
...	...	...	...

Table 5.1: Sabotage scenario – dataset sample.

PROCESS DISCOVERY

I proceeded to use PD to obtain the process model from this data. To learn what FO sequences were performed on the files, `file_id` and `operation_type` were chosen as case and activity attributes, respectively. Subsequently, a heuristic net was discovered using heuristic mining. In addition, a Petri net and process tree were obtained by inductive mining. Due to the small sample size, both PD applications were adjusted for maximal model fitness. The result of heuristic mining can be seen in [Figure 5.1](#). There, it can be observed what was happening to the given files. Disregarding the obvious uses of `create` and `open` FOs, cases where files were edited through the use of `write` operations can be seen. In addition, some cases involved file movements or uploads to version control repositories.

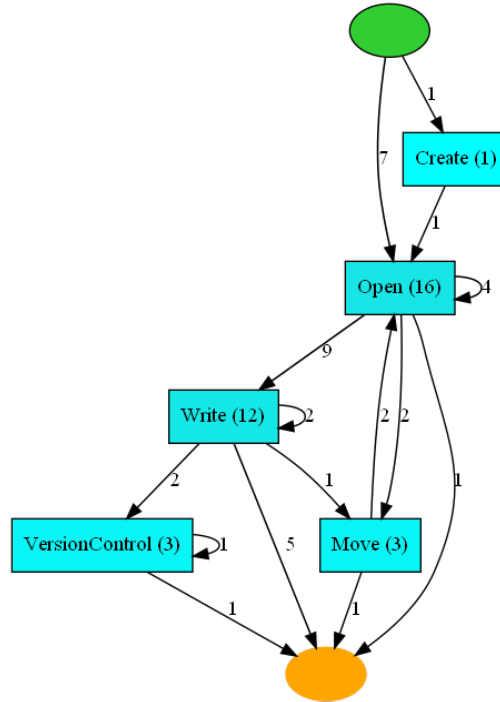


Figure 5.1: Sabotage scenario – the obtained heuristic net of the non-malicious behavior.

TEST CASES

Having established the non-malicious behavior model, I created three new file handling test cases to compare with the model. The traces representing them can be observed in [Figure 5.2](#). Case 1 represents a malicious deletion of the configuration file, preceded by a copy operation and uploading of the file to the web. Case 2 involves a download of a corrupted configuration file and its instalment to the system through Rename and Move operations. Lastly, test case 3 presents a non-malicious editing of the file content and subsequent uploading to the version control.

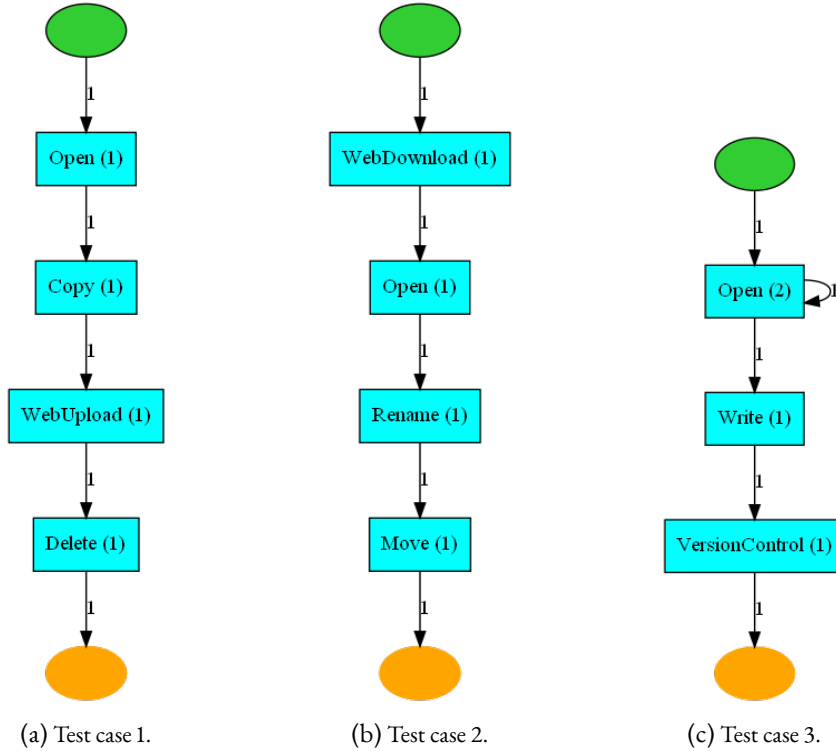


Figure 5.2: Sabotage scenario – test cases.

CONFORMANCE CHECKING

In the last phase, I applied CC between each test trace and the previously established models of non-malicious behavior. The process tree and the Petri net obtained by inductive mining were initially considered for the role of reference models during CC. As for the specific CC methods, I attempted to use these models for trace alignments and token-based replay. However, as seen in [Appendix A](#), the token-based replay failed to produce worthwhile results. That was due to the obtained Petri net, which contained too many invisible tasks and, in turn, allowed for more behavior than intended and failed to detect expected deviances before the replay reached the sink place. That was also the case with the Petri net obtained by heuristic mining. The

5 Comparison

final PD attempt with the α -algorithm was expectedly unable to capture the behavior by being unable to model short loops. Hence, I chose trace alignments with the process tree as the underlying model for CC.

I aimed to obtain two main metrics using this method - alignment cost and trace fitness. In this case, the alignment cost refers to the number of mismatches between the trace and the most similar behavior allowed by the model. For instance, the alignment for test case 2 is shown in [Table 5.2](#). There, two mismatches can be identified since the `WebDownload` (`WebDL`) and `Rename` operations are not present in the model. The mismatches caused by the use of the invisible transitions in the model (marked as τ) are ignored during cost function calculation [43]. Based on the alignment, trace fitness was calculated. CC results for all three test traces are summarized in [Table 5.3](#). In these results, it can be seen that the test cases representing the sabotage significantly diverge from the modeled norm, while test case 3 fits the expected behavior.

WebDL	»	Open	Rename	»	Move
»	τ	Open	»	τ	Move

Table 5.2: Sabotage scenario – alignment of test case 2 (top row) against the model.

Test case	Alignment cost	Trace fitness
1	3	40%
2	2	60%
3	0	100%

Table 5.3: Sabotage scenario – trace alignment results.

This use case shows that PM can infer a process model using just two mandatory attributes despite the limited dataset scope. The model was then examined visually to assess what FOs were performed on top of the configuration files. Next, trace alignment facilitated automatized comparison of new inputs to the established model. The obtained metrics could be then forwarded to the IA detection system. Based on the pre-defined threshold values, such a system would be able to flag the unfitting behavior based on its severity. The obtained trace alignments could be then used for a deeper analysis of where the deviances happened. Furthermore, this potential could be enhanced since `PM4Py` allows for the implementation of custom cost functions [43]. Lastly, the detected insider activity can be linked to particular users based on the presented dataset.

The detailed alignment results, along with the models and the mentioned token-based replay results, are available in [Appendix A](#).

DM APPROACH

In this sabotage scenario, I could not properly demonstrate the use of DM-based IA detection due to the lack of suitable reference data. As for DTL and other SL meth-

5 Comparison

ods, the data content did not present a candidate for response behavior, thus leaving only UL methods for consideration. Considering that the input attributes are categorical, I chose k-modes clustering for demonstration purposes. Before running the algorithm, I included the test cases in the dataset to see whether the differences show in the obtained patterns. Here, I used all four attributes, with `date_time` being reduced to the respective hour during the calculations for simplicity. The used k-modes implementation selects starting representatives at random, which may yield slightly different results between multiple runs of the algorithm. In the following illustrative instance, I chose eight as the target number of clusters (κ) following the results of the elbow method. The visualization of the elbow method is featured in [Figure 5.3](#). Finally, a sample of clustering results is provided in [Table 5.4](#). The entire process of obtaining the described instance and its result clusters are provided in [Appendix A](#).

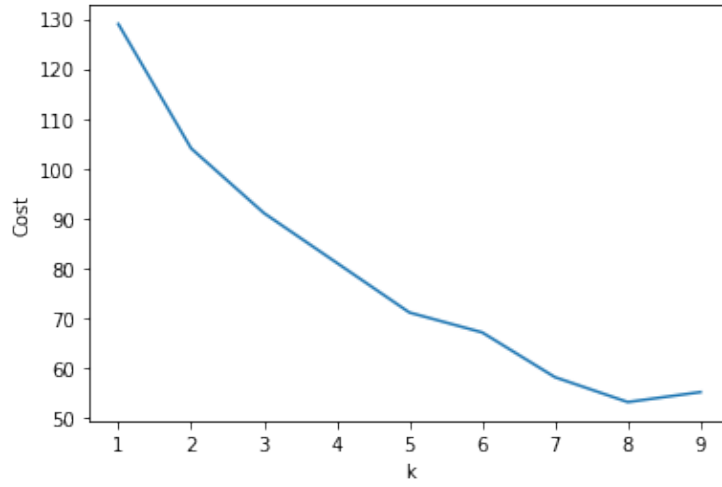


Figure 5.3: Sabotage scenario – elbow plot established before k-modes clustering.

file_id	id_user	operation_type	date_time	Cluster
1	1	Open	2021-04-22 12:29:00	6
2	1	Open	2021-04-22 13:23:00	1
2	1	Open	2021-04-22 13:24:06	1
2	1	Write	2021-04-22 13:24:47	0
...	...	...	...	...

Table 5.4: Sabotage scenario – clustering sample results.

In this small dataset, k-modes clustering partially managed to group file records based on the common attributes. For example, [Table 5.5](#) shows that a cluster was assembled from FOs related to the same file and performed by the same user within a common timeframe. From there, record order could be established manually through the timestamp attributes, practically simulating the results of PD. However, as seen

5 Comparison

in Table 5.4, the clustering failed to include all records in case of `file_id` 2, despite sharing three out of four values. In addition, it is worth noting that more realistic and complex datasets may not provide such convenient data properties and choice of k . Hence, clustering is not reliably capable of identifying trace-like structures. In addition, there is no real alternative to CC. Individual FO records could be assigned to clusters based on similarity measurements. Nonetheless, such an isolated classification does not bring information value towards identifying the FO as malicious in the scope of this dataset.

file_id	id_user	operation_type	date_time	Cluster
4	3	Open	2021-04-27 12:11:00	7
4	3	Move	2021-04-27 12:12:12	7
4	3	Open	2021-04-27 13:23:06	7
4	3	Write	2021-04-27 13:23:49	7

Table 5.5: Sabotage scenario – cluster number 7.

While applying UL methods to this limited dataset was technically possible, its data profile, combined with the individual approach to records, ultimately did not produce valuable outputs for IA detection. Comparing the capabilities of clustering to PM, the records were still treated individually, even at the cluster level, since the algorithm is based on their respective attributes, not sequential relations. Thus, since neither the isolated FO records nor the clusters carried enough features for malicious behavior identification, I deemed the application of clustering not useful for this dataset.

It is worth noting that in a more suitable dataset, the results of clustering and other UL methods may identify valuable common patterns, which may lead to a deeper understanding of the dataset or help divide the main problem into smaller ones [60]. Alternatively, clustering can be employed to help identify anomalous behavior in specific circumstances [22].

5.2.2 EXFILTRATION OF DATA TO EXTERNAL DEVICES

In the second scenario, I attempted to illustrate the detection of unauthorized data exfiltration to external devices. The detection is implemented using PM and DTL, showcasing their respective features and capabilities for IA detection.

DATASET

In this case, I was interested in the specific instances of `Move` FOs. The requirement was that the `dest_type` attribute must be one of the following external device types:

- external USB (`ExternUsb`),
- external CD/DVD (`ExternCdDvd`),

5 Comparison

- unspecified external device (`ExternOther`),
- mobile phone (`Mobile`).

Unlike in the previous scenario, the `file_log` contained twenty different cases for reference. Therefore, I was able to utilize it directly. This dataset was previously presented in [Section 4.3](#). Such records were labeled as non-malicious, authorized actions.

As a training base for detection, I manually created five IA sequences containing an illegal `Move` operation to an external device. These instances were pre-labeled as malicious through `is_malicious` attribute. That was done to allow for DTL demonstration. Finally, for detection illustration, I created three more unclassified test cases.

PM APPROACH

To see what happened to the files, I chose `file_id` and `operation_type` as case and activity attributes, respectively. However, the event log had to be filtered in advance to focus on the file movements to external devices. For that reason, I concatenated the `operation_type` values with the respective `dest_type` value at the event level. Then, the traces that did not contain a `Move` instance to external device were filtered out.

Analogically to the previous scenario, I proceeded to apply heuristic and inductive mining to obtain the heuristic net, Petri net and process tree with maximal fitness. That was done separately for all three input categories (non-malicious, malicious, test). The three test cases can be observed in [Figure 5.4](#), while the heuristic mining results performed on non-malicious and malicious behavior are presented in [Figure 5.5](#). These models and the inductive mining results are provided in [Appendix A](#).

In the case of non-malicious and malicious input groups, I aimed to use the inductively discovered models for CC against test inputs. The Petri nets were used for token-based replay, whereas the process trees were used for trace alignments. The results are provided in tables [Table 5.6](#), [Table 5.7](#), [Table 5.8](#) and [Table 5.9](#), respectively.

Test file_id	Alignment cost	Trace fitness
1	2	60%
3	2	50%
4	2	50%

Table 5.6: Data exfiltration – trace alignments to non-malicious behavior.

Test file_id	Fitness	Obtained token metrics			
		Missing t.	Consumed	Remaining	Produced
1	100%	0	9	0	9
3	83,33%	1	6	1	6
4	83,33%	1	6	1	6

Table 5.7: Data exfiltration – the results of token-based replay on the non-malicious behavior.

5 Comparison

Test file_id	Alignment cost	Trace fitness
1	1	83%
3	1	80%
4	1	80%

Table 5.8: Data exfiltration – trace alignments to malicious behavior.

Test file_id	Fitness	Obtained token metrics			
		Missing t.	Consumed	Remaining	Produced
1	100%	0	5	0	5
3	83%	1	6	1	6
4	100%	0	5	0	5

Table 5.9: Data exfiltration – the results of token-based replay on the malicious behavior.

In the results, it can be seen that the token-based replay showcases higher fitness values compared to the trace alignment. That was again due to the obtained Petri net not being restrictive enough. The alignments output shows that the test cases share more similarities to malicious behavior. A detection mechanism could be implemented analogically to the previous scenario, i.e., through automatized evaluation of the obtained metrics. Significant differences from the allowed behavior, or high similarity to the malicious one, would be highlighted by such a mechanism. In addition, both trace alignments and token-based replay specify where the mismatches happened. The detected behavior could be analyzed and even linked to the particular users or devices using the other event attributes. The full extent of the mentioned CC results is provided in [Appendix A](#).

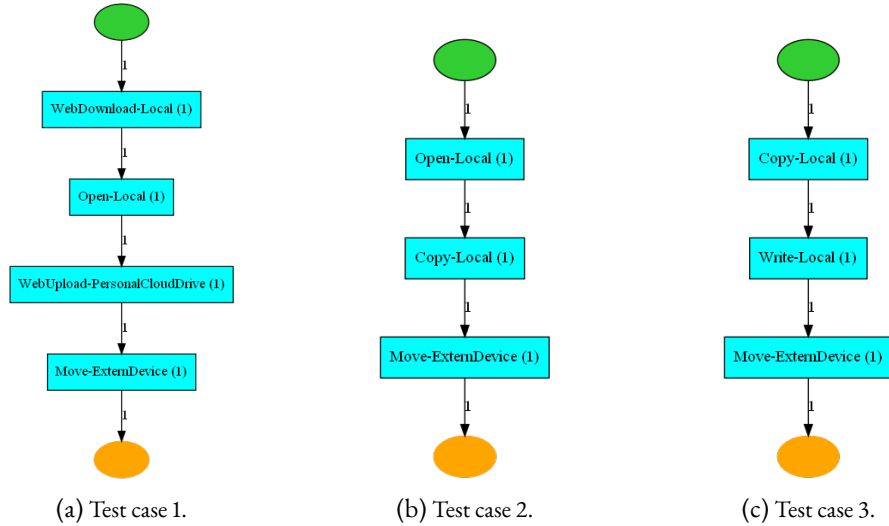
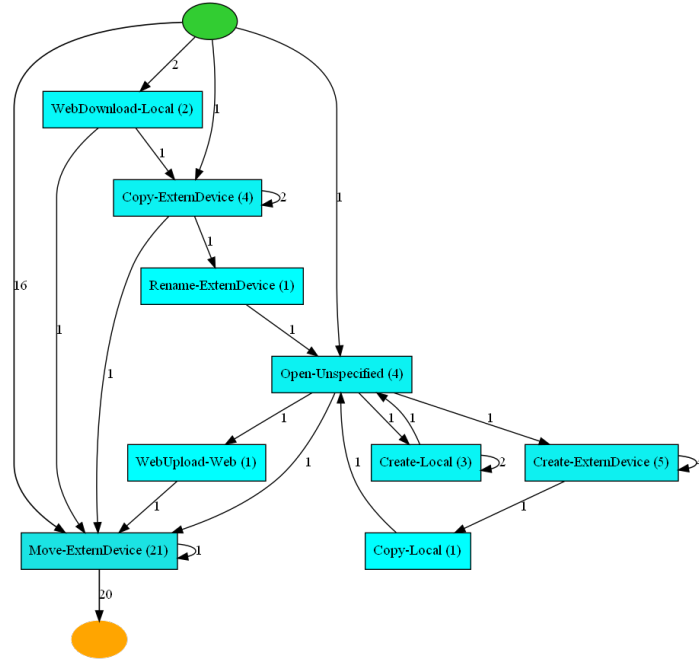
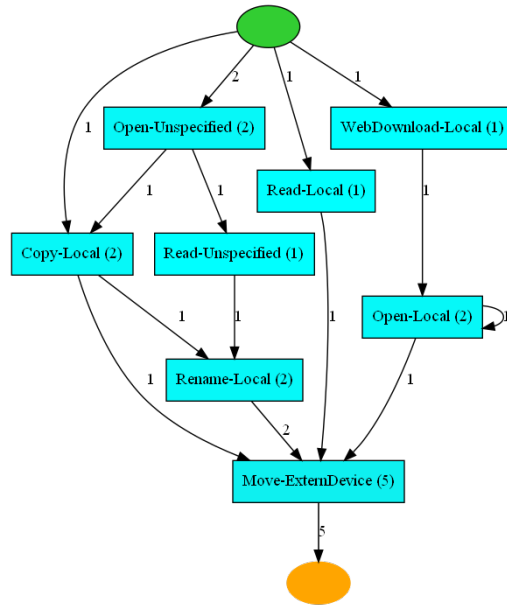


Figure 5.4: Data exfiltration – test cases.

5 Comparison



(a) Heuristic net of the non-malicious behavior from `file_log`.



(b) Heuristic net of the manually created malicious behavior.

Figure 5.5: Data exfiltration scenario – process discovery results.

DTL APPROACH

For the DTL application, I also utilized the three previously-mentioned input data groups. In the beginning, I included the pre-labeled malicious `Move` instances to the original `file_log`, merging the non-malicious and malicious behavior as training input for DTL. Subsequently, this data was filtered to contain only the `Move` FO records that lead to external devices. That resulted in a labeled training dataset of twenty-six records. The labeling was done through the `is_malicious` attribute.

The same attribute was the obvious choice for a response variable. For the role of predictors, I chose the following attributes:

- `file_size`,
- `src_type`,
- `dest_type`,
- `is_tagged`,
- `to_safe_zone`.

I chose these attributes to learn what file parameters lead to the detection of data exfiltration, regardless of the user or PC. The first three predictor attribute values were categorized for simplicity corresponding to the workflow in [Section 4.4.3](#).

Regarding the small data sample size, I performed DTL aiming for maximal model fitness. Thus, no input dataset splitting was performed, nor did I change the default parameters. As an exception, I tried both Gini index and entropy in the role of the tree node splitting criterion. I chose the Gini index-based DTL result for the following demonstrative purposes since it produced a more compact model for illustration. This model is presented in [Figure 5.6](#). Both obtained models can be accessed in the [Appendix A](#).

The decision tree in [Figure 5.6](#) is a demonstration of DTL data compression. To understand the visualized behavior patterns, a translation of node splitting conditions (top row in each non-leaf node) is provided below:

- `['file_size']_3 ≤ 0.5` – the file size is not in range (10MB, 50MB].
- `['file_size']_4 ≤ 0.5` – the file size is not in range (50MB, 1GB].
- `['dest_type']_9 ≤ 0.5` – the destination type is not a verified mobile phone.
- `['is_tagged']_1 ≤ 0.5` – the file is not tagged with a degree of importance.

From there, the acquired decision rules can be examined. For example, it can be seen that `Move` instances that involved files with size not in range (10MB, 50MB] and moved to an unverified mobile phone were labeled as malicious. While this example

5 Comparison

is just illustrative, it is worth noting that the real potential of DTL data compression lies in the handling of large labeled data sources [50].

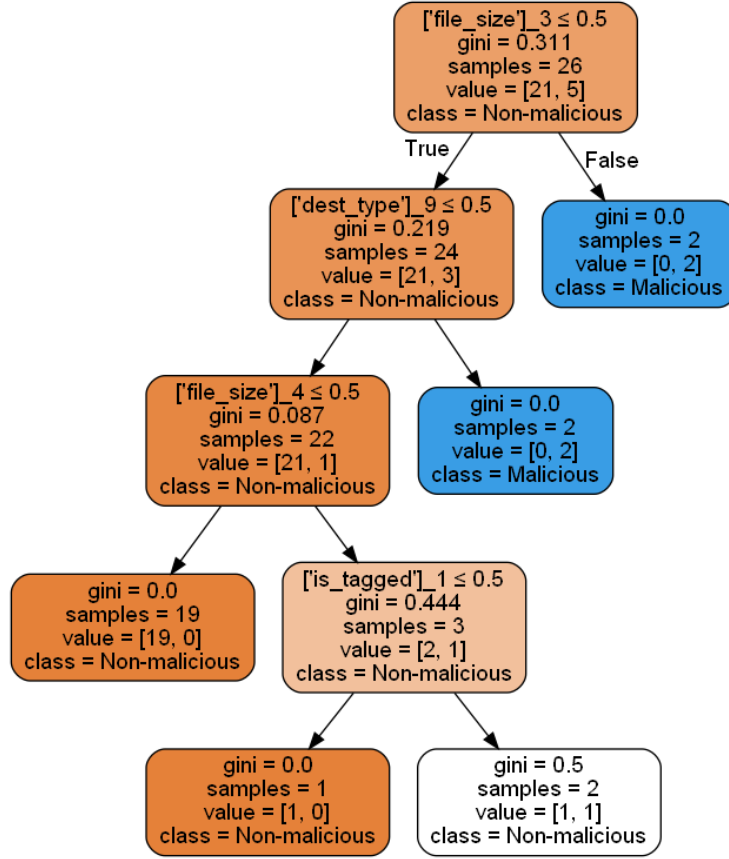


Figure 5.6: Data exfiltration – the decision tree obtained from the training data.

Having obtained the model, I could use it as a base for the DTL prediction of new inputs. To showcase that, I used the properties of the `Move` instances from the three previously-mentioned test cases. The DTL prediction of such instances according to the model in Figure 5.6 is shown in Table 5.10. The mappings between the numeric values and their respective labels are provided in the project code in Appendix A. This prediction process can be further automatized using the `sklearn` functionality [51].

file_size	src_type	dest_type	is_tagged	to_safe_zone	is_malicious
5	1	2	1	0	0
2	1	2	1	1	0
2	1	9	0	0	1

Table 5.10: Data exfiltration – the results of DTL prediction.

Assuming that enough quality training data is provided for the model establishment, DTL prediction could be used to check new `Move` instances. If the attribute values of the input instance correspond to previously confirmed malicious records, the detection system could issue a warning. The dataset profile shows that such records could also be associated with files, users and workstations. In addition, DTL data compression allows us to see which attributes and which of their values were significant for classification. However, being an SL method, the potential of DTL depends on the quality and amount of suitable labeled training data.

5.3 PRACTICAL COMPARISON

The following section analyses the observations made during the previously described IA scenarios. Furthermore, it addresses the observed differences within proposed IA detection mechanisms.

In the first scenario, a small data sample limited to four attributes was presented. Utilizing the suitable profile of this dataset for process-oriented analysis, PM techniques, particularly PD and CC, were deployed to assess the situation with file handling and subsequently attempt detection of configuration file sabotage. These PM implementations of play-in and replay concepts showed that once a suitable choice of case and activity attribute roles were provided, PM could effectively discover and further utilize a process model even on such a small scale without any further knowledge. In this context, a proper application of the DM counterparts in this scenario was not possible due to the lack of a suitable attribute profile. Hence, this case shows that the differences in the dataset profile requirements and play-in implementations may prevent PM and DM from being sensibly applicable to the same dataset. In this regard, it should be noted that the argument about the technique applicability can go both ways since different datasets may not possess the attribute profile for a sensible choice of case and activity roles while being fitting for DM techniques.

In the second scenario, the input dataset allowed for a proper application of both PM and DTL since it featured pre-identified malicious behavior. Thus, it was possible to disregard the data profile requirements factor from the previous scenario and focus on the contrast in the play-in and replay aspects. Such differences, originating from the vertical and horizontal handling of the dataset, were the most apparent during the IA detection. PM and DTL, respectively, provided two different takes on this matter.

Firstly, in the case of **behavior assessment**, both methods allowed to pinpoint behavior related to the particular type of `Move` instances. However, PD operated at the trace level, thus providing the surrounding context to each `Move` instance. From there, it was analyzed which FOs preceded the file movements. The non-mandatory attributes of the events did not contribute to this phase. On the other hand, DTL accessed the `Move` events individually, dismissing the potentially useful process-oriented view of the situation. Instead, DTL data compression identified the common patterns within these isolated `Move` instances and presented which of their attribute values contributed the most to them being labeled as malicious or non-malicious.

Secondly, in the case of **IA detection** itself, the respective trace-based and event-based disparity remained present. In the PM solution, CC compared whole traces of events to the previously discovered process model. The resulting measurements were based purely on the activity sequences, not the other attributes of the individual events. On the contrary, DTL prediction classified isolated `move` records based on their attribute values and the known model.

To summarize, the observations made during the evaluation of scenarios confirmed the differences mentioned in the theoretical comparison, particularly in the attribute roles, play-in and replay departments. In the IA detection context, the horizontal and vertical data handling provided two distinct approaches for IA detection and related behavior assessment. Their potential is discussed in the following section.

5.4 DISCUSSION

This section discusses the potential and applicability of the previously examined automatized process and data-oriented approaches.

Firstly, PM was studied in the role of a process-oriented IA detection tool. Assuming a sensible choice of mandatory attributes, PD allows for obtaining a process model of the recorded behavior in an automatized manner. Such models can be then used for visual assessment of the situation. Hence, the stakeholders could receive an outline of how the processes function in reality. Following that, the differences between their expectations and reality can be identified. This gained knowledge can be used to model new or adjust existing security policies. Furthermore, CC presents an automatized way to confront such models with new inputs. The differences can be detected, measured and evaluated practically in real-time, thus making PM an appealing option for IA detection and related behavior assessment.

Secondly, DM capabilities were examined as a data-oriented solution, focusing primarily on DTL. In the case of DTL and other SL methods, their definition determines that their applicability depends on the presence of labeled data. To satisfy that requirement, the labeling would have to be done in advance, e.g., manually or based on the results of other techniques, such as PM or UL. As mentioned in [15], that may be a substantial initial hurdle to overcome. However, if the required data is provided, DTL can identify the most significant patterns that contributed to the classification, which can be helpful for a deeper understanding of the proceedings. In addition to those analytical capabilities, the obtained models could be used as an event-level alternative to real-time detection of potentially malicious actions. As for the UL methods, while they do not require labeled data, their usefulness depends on the given dataset profile and content. In the presented scenario, the received results did not show reliable outputs and capabilities for IA detection.

Thirdly, in section [Section 5.1.1](#), the existing potential in the combination of PM and DM methods was acknowledged. In this regard, provided with a suitable dataset, DTL can support PE by analyzing decision points in Petri nets [43, 48]. Another example featured clustering to sensibly dissect the problem into smaller, easier-to-handle

event logs [60]. Those particular and other combinations of PM and DM could be harnessed for IA detection-related purposes.

Lastly, it is worth noting that the question of source data and model quality should be addressed. That is relevant for both PM and DM approaches. The provided examples featured simplified, illustrative applications performed on a small scale. A real-world usage should involve larger source datasets. The practical application shows that the audit logs present a promising source of such data and, under the right circumstances, could be used for automatized process and data-oriented analyses. Concerning the obtained models, the common DM challenges should be considered, such as overfitting and underfitting [49, 51]. Analogically, the PM model quality criteria were addressed in [Section 2.4.1](#).

To summarize, there is a considerable IA detection potential in both automatized process-oriented and data-oriented approaches and their combinations. The degree of their applicability is determined by the data profile and content. I am of the opinion that PM methods can be a useful primary tool for behavior assessment and IA detection. I base this claim on the observations made in the previous analyses. Mainly, PM provides automatized ways to obtain process models without prior identification of malicious behavior. It provides the necessary insight into what is happening within the organization. The results can help adjust the organization's security policies and be used for trace-level IA detection. At the same time, I also consider DTL a potentially useful secondary tool for further analyses and IA detection. Once enough labeled data is provided, e.g., manually or based on the results of other techniques, DTL can be used to examine patterns at the individual event level. That can, in turn, enhance understanding of the behavior and be used as an event-level alternative to trace-based IA detection. In addition, other DM methods may improve the results of the two techniques and provide unique, valuable points of view. Henceforth, I conclude that both of the examined approaches have the potential to be a valuable part of automatized IA detection workflows, with PM in the role of a primary IA detection and behavior assessment tool.

CONCLUSION

This thesis outlines insider threats and insider attacks and describes the challenges related to insider attack detection. In this regard, it provides an overview of the existing solutions both from the process-oriented and data-oriented methodology. The outline is focused on the automatized approaches, particularly process mining and data mining. The paper then describes the essential terminology, data profile requirements and algorithms of process mining and data mining fields, respectively. Following the theoretical background and the methodology explanation, the thesis compares process mining to data mining and process mining to decision tree learning from a theoretical standpoint. Subsequently, it presents two implemented demonstrative insider attack detection scenarios using process and data mining methods. The chosen approaches are evaluated and compared in terms of their capabilities and methodology. Finally, based on the observations, the thesis discusses the insider attack detection potential of process mining and data mining. There, it mentions the challenges related to their applicability and concludes that both approaches have the potential to provide unique and valuable outputs and that such results could be combined with process mining in the role of a primary insider attack detection and behavior assessment tool.

FUTURE WORK

Firstly, the illustrative application of process mining and data mining for insider attack detection was performed on a limited and partially artificial scope. A real-world application would be expected to involve acquiring high-quality models adjusted to the organization's requirements. The potential of combining process mining and decision tree learning for obtaining, verifying and enhancement of such models could be examined further in a real-world deployment.

Secondly, since decision tree learning is only one of the many data-oriented methods available, further comparisons between process mining and other suitable data mining methods could be explored in the future. For example, the potential of unsupervised learning techniques, such as association rule learning, could be examined as a complementary data-oriented alternative to process discovery for behavior assessments. Additionally, provided with suitable source data with continuous response variables, insider attack detection through regression-based supervised learning and process mining could be studied. Ultimately, such research could prove beneficial for designing new insider attack detection methodologies or enhancing the existing ones.

5 Comparison

Lastly, the provided observations and workflows could be used as an inspiration for creating a more general-purpose automatized insider attack detection module.

Based on the observations presented within this thesis, I consider process mining to be a valuable option for insider attack detection and behavior assessment. Thus, I encourage further developments in deploying process mining-based insider attack detection workflows alongside the traditional data-oriented approaches.

A APPENDIX

- `bc-thesis-impl` - the directory containing the Python project used in the implementation part of this thesis.
- `results` - the directory containing the results presented in [Chapter 5](#).
- `README.md` - the manual for the project's installation and re-obtaining of results.

BIBLIOGRAPHY

1. ALBERTIN, A. and MOURA, R. The Benefits of Information Technology in Business Performance. In: [online]. 2004 [visited on 2022-04-12]. Available from: <https://aisel.aisnet.org/amcis2004/121>.
2. DUMAS, M.; VAN DER AALST, W. M., and TER HOFSTEDE, A. *Process-aware information systems: bridging people and software through process technology*. Hoboken, N.J: Wiley-Interscience, 2005. ISBN 9780471663065.
3. VAN DER AALST, W. M. P. *Process Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016. Available from DOI: [10.1007/978-3-662-49851-4](https://doi.org/10.1007/978-3-662-49851-4).
4. HARVEY, M. and LUSCH, R. Protecting the core competencies of a company: Intangible asset security. *European Management Journal*. 1997, vol. 15, no. 4, pp. 370–380. Available from DOI: [https://doi.org/10.1016/S0263-2373\(97\)00017-0](https://doi.org/10.1016/S0263-2373(97)00017-0).
5. LAVINIA MIHAELA, C. Current security threats in the national and international context. *Journal of accounting and management information systems*. 2020, vol. 19, pp. 351–378. Available from DOI: [10.24818/jamis.2020.02007](https://doi.org/10.24818/jamis.2020.02007).
6. MYERS, J.; GRIMAILA, M. R., and MILLS, R. F. Towards insider threat detection using web server logs. In: *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research Cyber Security and Information Intelligence Challenges and Strategies – CSIIRW '09*. Oak Ridge, Tennessee: ACM Press, 2009. Available from DOI: [10.1145/1558607.1558670](https://doi.org/10.1145/1558607.1558670).
7. GAVAI, G.; SRICHARAN, K.; GUNNING, D.; ROLLESTON, R.; HANLEY, J., and SINGHAL, M. Detecting Insider Threat from Enterprise Social and Online Activity Data. In: *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats*. Denver Colorado USA: ACM, 2015, pp. 13–20. Available from DOI: [10.1145/2808783.2808784](https://doi.org/10.1145/2808783.2808784).
8. *SEI Cyber Minute: Insider Threats* [online]. 2017 [visited on 2022-03-30]. Available from: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=496626>.
9. *2022 Ponemon Institute Cost of Insider Threats: Global Report* [online]. 2022 [visited on 2022-03-31]. Available from: <https://www.proofpoint.com/us/resources/threat-reports/cost-of-insider-threats>.
10. WITTEN, I. H. and FRANK, E. *Data mining: practical machine learning tools and techniques*. 2nd ed. Amsterdam ; Boston, MA: Morgan Kaufman, 2005. Morgan Kaufmann series in data management systems. ISBN 9780120884070.

Bibliography

11. VAN ROSSUM, G. and DRAKE, F. L. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.
12. BERTI, A.; VAN ZELST, S., and VANDER AALST, W. M. P. *Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science*. 2019. Available from DOI: [10.48550/ARXIV.1905.06169](https://doi.org/10.48550/ARXIV.1905.06169).
13. PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B., and GRISEL O., e. a. Scikit-learn: Machine Learning in Python. 2011, vol. 12, pp. 2825–2830. Available from DOI: [10.48550/ARXIV.1201.0490](https://doi.org/10.48550/ARXIV.1201.0490).
14. THOMAS, R. Total cost of security: a method for managing risks and incentives across the extended enterprise. 2009. Available from DOI: [10.1145/1558607.1558677](https://doi.org/10.1145/1558607.1558677).
15. YUAN, S.; ZHENG, P.; WU, X., and TONG, H. Few-shot Insider Threat Detection. In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. Virtual Event Ireland: ACM, 2020, pp. 2289–2292. Available from DOI: [10.1145/3340531.3412161](https://doi.org/10.1145/3340531.3412161).
16. HOMOLIAK, I.; TOFFALINI, F.; GUARNIZO, J.; ELOVICI, Y., and OCHOA, M. Insight Into Insiders and IT: A Survey of Insider Threat Taxonomies, Analysis, Modeling, and Countermeasures. *ACM Computing Surveys*. 2019, vol. 52, no. 2, 30:1–30:40. Available from DOI: [10.1145/3303771](https://doi.org/10.1145/3303771).
17. SANZGIRI, A. and DASGUPTA, D. Classification of Insider Threat Detection Techniques. In: *Proceedings of the 11th Annual Cyber and Information Security Research Conference*. Oak Ridge TN USA: ACM, 2016, pp. 1–4. Available from DOI: [10.1145/2897795.2897799](https://doi.org/10.1145/2897795.2897799).
18. CAPPELLI, D.; MOORE, A., and TRZECIAK, R. *The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes (Theft, Sabotage, Fraud)*. 2012. ISBN 9780321812575.
19. COLE, E. and RING, S. *Insider Threat*. San Diego; Los Angeles: Newnes [Imprint] Elsevier Science & Technology Books Sony Electronics [distributor], 2009. ISBN 9780080489056.
20. WALL, D. S. Enemies within: Redefining the insider threat in organizational security policy. *Security Journal*. 2013, vol. 26, no. 2, pp. 107–124. Available from DOI: [10.1057/sj.2012.1](https://doi.org/10.1057/sj.2012.1).
21. MORIANO, P.; PENDLETON, J.; RICH, S., and CAMP, L. J. Insider Threat Event Detection in User-System Interactions. In: *Proceedings of the 2017 International Workshop on Managing Insider Security Threats*. Dallas Texas USA: ACM, 2017, pp. 1–12. Available from DOI: [10.1145/3139923.3139928](https://doi.org/10.1145/3139923.3139928).
22. LENKOVÁ, L. *Detekcia neobvyklých prenosov dát z hľadiska pracovnej doby*. 2021. Available from: <https://is.muni.cz/th/nibwj/>. Bachelor's thesis. Masaryk University, Faculty of Informatics, Brno.

Bibliography

23. BISHOP, M.; CONBOY, H. M.; HUONG PHAN; SIMIDCHIEVA, B. I.; AVRUNIN, G. S.; CLARKE, L. A.; OSTERWEIL, L. J., and PEISERT, S. Insider Threat Identification by Process Analysis. In: *2014 IEEE Security and Privacy Workshops*. San Jose, CA: IEEE, 2014, pp. 251–264. Available from DOI: [10.1109/SPW.2014.40](https://doi.org/10.1109/SPW.2014.40).
24. RODRIGUEZ, M.; BETARTE, G., and CALEGARI, D. A Process Mining-based approach for Attacker Profiling. In: *2021 IEEE URUCON*. Montevideo, Uruguay: IEEE, 2021, pp. 425–429. Available from DOI: [10.1109/URUCON53396.2021.9647342](https://doi.org/10.1109/URUCON53396.2021.9647342).
25. MACÁK, M. *Process-aware Insider Threat Detection and Mitigation in Organizations*. 2021. Available from: <https://is.muni.cz/th/wv698/>. Rigorous thesis. Masarykova univerzita, Fakulta informatiky, Brno.
26. MACÁK, M.; VANÁT, I.; MERJAVÝ, M.; JEVOČIN, T., and BÜHNOVÁ, B. Towards Process Mining Utilization in Insider Threat Detection from Audit Logs. In: *2020 Seventh International Conference on Social Networks Analysis, Management and Security (SNAMS)*. Paris, France: IEEE, 2020, pp. 1–6. Available from DOI: [10.1109/SNAMS52053.2020.9336573](https://doi.org/10.1109/SNAMS52053.2020.9336573).
27. ZHU, T.; GUO, Y.; JU, A.; MA, J., and WANG, X. An Insider Threat Detection Method Based on Business Process Mining: *International Journal of Business Data Communications and Networking*. 2017, vol. 13, no. 2, pp. 83–98. Available from DOI: [10.4018/ijbdcn.2017070107](https://doi.org/10.4018/ijbdcn.2017070107).
28. ŠROL, E. *Process Mining usage for potential insider threat identification utilizing PM4Py*. 2021. Available from: <https://is.muni.cz/th/iv3ye/>. Bachelor's thesis. Masaryk University, Faculty of Informatics, Brno.
29. ROSING, M. V. *The complete business process handbook: body of knowledge from process modeling to BPM*. 1st edition. Waltham, MA: Elsevier, 2014. ISBN 9780127999593.
30. PANAGACOS, T. *The Ultimate Guide to Business Process Management: Everything You Need to Know and how to Apply it to Your Organization*. Amazon Digital Services LLC - KDP Print US, 2012. ISBN 9781477486139.
31. AGUILAR-SAVÉN, R. Business process modelling: Review and framework. *International Journal of Production Economics*. 2004, vol. 90, pp. 129–149. Available from DOI: [10.1016/S0925-5273\(03\)00102-6](https://doi.org/10.1016/S0925-5273(03)00102-6).
32. RUSSELL, N.; VAN DER AALST, W. M., and TER HOFSTEDE, A. H. M. Business Process Modeling. In: *Workflow Patterns: The Definitive Guide*. 2016, pp. 27–59. ISBN 9780262029827.
33. VAN DER AALST, W. M. P. Process-Aware Information Systems: Lessons to Be Learned from Process Mining. In: *Transactions on Petri Nets and Other Models of Concurrency II: Special Issue on Concurrency in Process-Aware Information Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–26. Available from DOI: [10.1007/978-3-642-00899-3_1](https://doi.org/10.1007/978-3-642-00899-3_1).

34. VAN DER AALST, W. and STAHL, C. *Modeling business processes : a Petri net-oriented approach*. MIT Press, 2011. Information Systems. ISBN 9780262015387.
35. VANDER AALST, W. M. P.; VAN HEE, K. M.; TER HOFSTEDE, A. H. M.; SIDOROVA, N.; VERBEEK, H. M. W.; VOORHOEVE, M., and WYNN, M. T. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*. 2011, vol. 23, no. 3, pp. 333–363. Available from DOI: [10.1007/s00165-010-0161-4](https://doi.org/10.1007/s00165-010-0161-4).
36. VAN DER AALST, W. M. A practitioner’s guide to process mining: Limitations of the directly-follows graph. *Procedia Computer Science*. 2019, vol. 164, pp. 321–328. Available from DOI: [10.1016/j.procs.2019.12.189](https://doi.org/10.1016/j.procs.2019.12.189).
37. *Business Process Model and Notation (BPMN), Version 2.0.2* [online]. 2014 [visited on 2022-02-04]. Available from: <https://www.omg.org/spec/BPMN/>.
38. KALENKOVA, A. A.; VAN DER AALST, W. M. P.; LOMAZOVA, I. A., and RUBIN, V. A. Process mining using BPMN: relating event logs and process models. In: *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*. Saint-malo France: ACM, 2016, pp. 123–123. Available from DOI: [10.1145/2976767.2987688](https://doi.org/10.1145/2976767.2987688).
39. YAMASATHIEN, S. and VATANAWOOD, W. An approach to construct formal model of business process model from BPMN workflow patterns. In: *2014 Fourth International Conference on Digital Information and Communication Technology and its Applications (DICTAP)*. 2014, pp. 211–215. Available from DOI: [10.1109/DICTAP.2014.6821684](https://doi.org/10.1109/DICTAP.2014.6821684).
40. BUIJS, J.; DONGEN, B., and VAN DER AALST, W. M. P. A Genetic Algorithm for Discovering Process Trees. In: 2012, pp. 1–8. Available from DOI: [10.1109/CEC.2012.6256458](https://doi.org/10.1109/CEC.2012.6256458).
41. WESKE, M. *Business Process Management: Concepts, Languages, Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019. ISBN 9783662594315 9783662594322. Available from DOI: [10.1007/978-3-662-59432-2](https://doi.org/10.1007/978-3-662-59432-2).
42. ACCORSI, R.; STOCKER, T., and MÜLLER, G. On the exploitation of process mining for security audits: the process discovery case. In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC '13*. Coimbra, Portugal: ACM Press, 2013. Available from DOI: [10.1145/2480362.2480634](https://doi.org/10.1145/2480362.2480634).
43. PROCESS MINING GROUP FRAUNHOFER FIT. *PM4Py, official documentation* [online]. 2021 [visited on 2022-04-14]. Available from: <https://pm4py.fit.fraunhofer.de/docs>.
44. WEIJTERS, A. and VAN DER AALST, W. M. P. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*. 2003, vol. 10, pp. 151–162. Available from DOI: [10.3233/ICA-2003-10205](https://doi.org/10.3233/ICA-2003-10205).

Bibliography

45. WEIJTERS, A.; VAN DER AALST, W. M. P., and ALVES DE MEDEIROS, A. Process mining with the HeuristicsMiner algorithm. In: Technische Universiteit Eindhoven, 2006. ISBN 978-90-386-0813-6.
46. ROZINAT, A. and VAN DER AALST, W. Conformance checking of processes based on monitoring real behavior. *Information Systems*. 2008, vol. 33, no. 1, pp. 64–95. Available from DOI: [10.1016/j.is.2007.07.001](https://doi.org/10.1016/j.is.2007.07.001).
47. MOLKA, T.; REDLICH, D.; DROBEK, M.; CAETANO, A.; ZENG, X.-J., and GILANI, W. Conformance checking for BPMN-based process models. In: *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. Gyeongju Republic of Korea: ACM, 2014, pp. 1406–1413. Available from DOI: [10.1145/2554850.2555061](https://doi.org/10.1145/2554850.2555061).
48. ROZINAT, A. and VAN DER AALST, W. M. P. Decision Mining in ProM. In: 2006, pp. 420–425. Available from DOI: [10.1007/11841760_33](https://doi.org/10.1007/11841760_33).
49. ALPAYDIN, E. *Introduction to machine learning*. 2nd ed. Cambridge, Mass: MIT Press, 2010. Adaptive computation and machine learning. ISBN 9780262012430.
50. BRAMER, M. *Principles of Data Mining*. 3rd ed. 2016. London: Springer London : Imprint: Springer, 2016. Undergraduate Topics in Computer Science. ISBN 9781447173076.
51. SCIKIT-LEARN DEVELOPERS. *Scikit-learn, official user guide* [online]. 2007-2022 [visited on 2022-04-03]. Available from: https://scikit-learn.org/stable/user_guide.html#user-guide.
52. YAN, X. and SU, X. *Linear regression analysis: theory and computing*. Singapore; Hackensack, NJ: World Scientific, 2009. ISBN 9789812834102.
53. ROKACH, L. and MAIMON, O. Clustering Methods. In: *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2005, pp. 321–352. Available from DOI: [10.1007/0-387-25465-X_15](https://doi.org/10.1007/0-387-25465-X_15).
54. NATALIANI, Y. and YANG, M.-S. Feature-Weighted Fuzzy K-Modes Clustering. In: *Proceedings of the 2019 3rd International Conference on Intelligent Systems, Metaheuristics & Swarm Intelligence*. Male Maldives: ACM, 2019, pp. 63–68. Available from DOI: [10.1145/3325773.3325780](https://doi.org/10.1145/3325773.3325780).
55. VOS, N. J. de. *k-modes categorical clustering library* [online]. 2015–2021 [visited on 2022-05-02]. Available from: <https://github.com/nicodv/kmodes>.
56. JOSHI, G. Anomaly Extraction Using Association Rule Mining. In: [online]. 2014 [visited on 2022-03-02]. Available from: http://www.ijera.com/papers/Vol4_issue1/Version%202/M41028892.pdf.
57. REBACK, J.; JBROCKMENDEL; MCKINNEY, W.; VANDEN BOSSCHE, J., and AUGSPURGER Tom, e. a. *pandas-dev/pandas: Pandas 1.4.1*. Zenodo, 2022. Available from DOI: [10.5281/ZENODO.6053272](https://doi.org/10.5281/ZENODO.6053272).

Bibliography

58. VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M., and REDDY Tyler, e. a. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, vol. 17, pp. 261–272. Available from DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
59. HARRIS, C. R.; MILLMAN, K. J.; VAN DER WALT, S. J.; GOMMERS, R., and PAULI VIRTANEN, e. a. Array programming with NumPy. *Nature*. 2020, vol. 585, no. 7825, pp. 357–362. Available from DOI: [10.1038/s41586-020-2649-2](https://doi.org/10.1038/s41586-020-2649-2).
60. SONG, M.; GÜNTHER, C. W., and VAN DER AALST, W. M. P. Trace Clustering in Process Mining. In: *Business Process Management Workshops*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 109–120. ISBN 978-3-642-00328-8.