Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance

W.M.P. van der Aalst and A.K.A. de Medeiros

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{w.m.p.v.d.aalst, a.k.medeiros}@tm.tue.nl

Abstract. One approach to secure systems is through the analysis of audit trails. An audit trail is a record of all events that take place in a system and across a network, i.e., it provides a trace of user/system actions so that security events can be related to the actions of a specific individual or system component. Audit trails can be inspected for the presence or absence of certain patterns. This paper advocates the use of *process mining techniques* to analyze audit trails for security violations. It is shown how a specific algorithm, called the α-algorithm, can be used to support security efforts at various levels ranging from low-level intrusion detection to high-level fraud prevention.

Keywords: Process mining, security, audit trails, pattern discovery, data mining, Petri nets.

1 Introduction

Fueled by the omnipresence of event logs in transactional information systems (cf. WFM, ERP, CRM, SCM, and B2B systems), process mining has become a vivid research area [5, 6]. Until recently, the information in these event logs was rarely used to analyze the underlying processes. Process mining aims at improving this by providing techniques and tools for discovering process, control, data, organizational, and social structures from event logs, i.e., the basic idea of process mining is to diagnose processes by mining event logs for knowledge. So far, process mining research has focussed on process discovery and process improvement. In this paper, we focus on the application of process mining to security issues.

When considering an enterprise information system, security plays a role at different levels, i.e., from the level of UNIX processes to the level of interorganizational business processes. Security policies may refer to things ranging from cryptography and role-based access control to auditing and the four-eyes principle. Security violations may be conducted by hackers but also by white-collar criminals, cf. the discussions on "corporate governance" following the Enron and Parmalat scandals). Literature on security can be split into computer security [11] and auditing [31]. Although computer security and auditing are at very different levels, the absence or presence of certain behavioral patterns may indicate

security violations. Therefore, *audit trails* can be useful. Fortunately, many enterprise information systems store relevant events in some structured form. For example, workflow management systems typically register the start and completion of activities [3]. ERP systems like SAP log all transactions, e.g., users filling out forms, changing documents, etc. Business-to-business (B2B) systems log the exchange of messages with other parties. Call center packages but also general-purpose CRM systems log interactions with customers. These examples show that many systems have some kind of *event log* often referred to as "audit trail", "history", "transaction log", etc. [5, 8, 21, 32]. The event log typically contains information about events referring to an *activity* and a *case*. The case (also named process instance) is the "thing" which is being handled, e.g., a customer order, a job application, an insurance claim, a building permit, etc. The activity (also named task, operation, action, or work-item) is some operation on the case. Typically, events have a *timestamp* indicating the time of occurrence. Moreover, event logs typically also contain information on the actor, i.e., person or system component, executing or initiating the event. We will refer to such an actor as the *originator* or performer. Based on this information several tools and techniques for process mining have been developed [2, 4, 5, 7, 8, 13, 22, 23, 28, 32, 35].

Process mining is useful for at least two reasons. First of all, it could be used as a tool to find out how people and/or procedures really work. Consider for example processes supported by an ERP system like SAP (e.g., a procurement process). Such a system logs all transactions but in many cases does not enforce a specific way of working. In such an environment, process mining could be used to gain insight in the actual process. Another example would be the flow of patients in a hospital. Note that in such an environment all activities are logged but information about the underlying process is typically missing. In this context it is important to stress that management information systems provide information about key performance indicators like resource utilization, flow times, and service levels but *not* about the underlying business processes (e.g., causal relations, ordering of activities, etc.). Second, process mining could be used for *Delta analysis*, i.e., comparing the actual process with some predefined process. Note that in many situations there is a descriptive or prescriptive process model. Such a model specifies how people and organizations are assumed/expected to work. By comparing the descriptive or prescriptive process model with the discovered model, discrepancies between both can be detected and used to improve the process. Consider for example the so-called reference models in the context of SAP. These models describe how the system should be used. Using process mining it is possible to verify whether this is the case. In fact, process mining could also be used to compare different departments/organizations using the same ERP system.

Clearly, both aspects (discovery and delta analysis) are relevant for computer security and auditing. For example, in [18] an approach for intrusion detection is presented. This method inspects audit trails and uses fixed-length patterns to distinguish *self* (i.e., normal process execution) from *other* (i.e., a potential

security violation). In [36] this is extended to variable length patterns. Unfortunately, approaches such as [18, 36] do not consider the process structure and are unable to detect parallelism and causality. Therefore, we explore the concept of process mining and one algorithm in particular (the α-algorithm, [7]) in the context of security.

The remainder of this paper is organized as follows. Section 2 introduces the concept of process mining. Section 3 introduces the basic notation and presents the basic α-algorithm. Then the paper focuses on two problems: *Detecting Anomalous Process Executions* (Section 4) and *Checking Process Conformance* (Section 5). Section 6 provides some related work. Finally, Section 7 concludes the paper.

2 Process Mining: An overview

The goal of process mining is to extract information about processes from transaction logs [5]. We assume that it is possible to record events such that (i) each event refers to an *activity* (i.e., a well-defined step in the process), (ii) each event refers to a *case* (i.e., a process instance), (iii) each event can have a *performer* also referred to as *originator* (the actor executing or initiating the activity), and (iv) events have a *timestamp* and are totally ordered. Table 1 shows an example of a log involving 19 events, 5 activities, and 6 originators. In addition to the information shown in this table, some event logs contain more information on the case itself, i.e., data elements referring to properties of the case. For example, the case handling system FLOWer logs every modification of some data element.

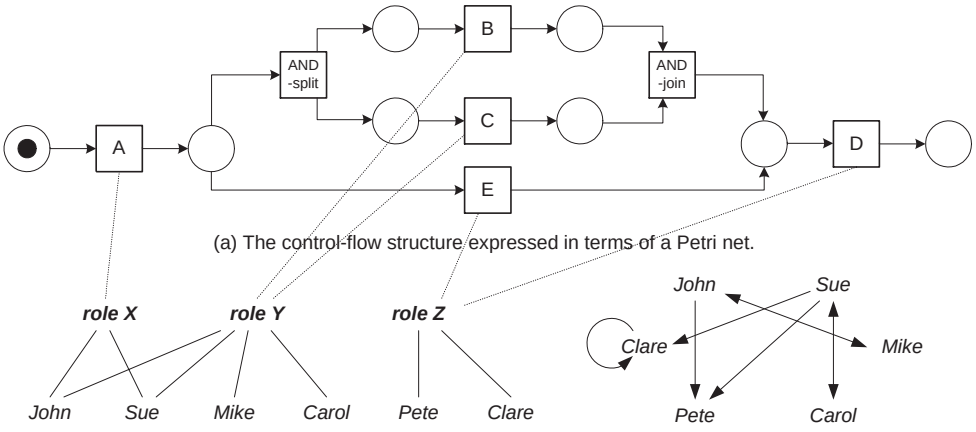
case id	activity id	originator	timestamp	case id	activity id	originator	timestamp
case 1	activity A	John	9-3-2004:15.01	case 5	activity A	Sue	10-3-2004:13.05
case 2	activity A	John	9-3-2004:15.12	case 4	activity C	Carol	11-3-2004:10.12
case 3	activity A	Sue	9-3-2004:16.03	case 1	activity D	Pete	11-3-2004:10.14
case 3	activity B	Carol	9-3-2004:16.07	case 3	activity C	Sue	11-3-2004:10.44
case 1	activity B	Mike	9-3-2004:18.25	case 3	activity D	Pete	11-3-2004:11.03
case 1	activity C	John	10-3-2004:9.23	case 4	activity B	Sue	11-3-2004:11.18
case 2	activity C	Mike	10-3-2004:10.34	case 5	activity E	Clare	11-3-2004:12.22
case 4	activity A	Sue	10-3-2004:10.35	case 5	activity D	Clare	11-3-2004:14.34
case 2	activity B	John	10-3-2004:12.34	case 4	activity D	Pete	11-3-2004:15.56
case 2	activity D	Pete	10-3-2004:12.50				

Table 1. An event log (audit trail).

Event logs[1] such as the one shown in Table 1 are used as the starting point for mining. We distinguish three different perspectives: (1) the process perspective, (2) the organizational perspective and (3) the case perspective. The *process perspective* focuses on the control-flow, i.e., the ordering of activities. The goal of mining this perspective is to find a good characterization of all possible paths, e.g., expressed in terms of a Petri net [30] or Event-driven Process Chain (EPC) [23, 24]. The *organizational perspective* focuses on the originator field, i.e., which

[1] In the context of security, event logs should be interpreted as audit trails.

performers are involved and how are they related. The goal is to either structure the organization by classifying people in terms of roles and organizational units or to show relation between individual performers (i.e., build a social network [9, 10, 12, 15, 19, 20, 27, 29, 33, 34]). The *case perspective* focuses on properties of cases. Cases can be characterized by their path in the process or by the originators working on a case. However, cases can also be characterized by the values of the corresponding data elements. For example, if a case represent a replenishment order it is interesting to know the supplier or the number of products ordered.



(a) The control-flow structure expressed in terms of a Petri net.

(b) The organizational structure expressed in terms of a activity-role-performer diagram.

(c) A sociogram based on transfer of work.

Fig. 1. Some mining results for the process perspective (a) and organizational (b and c) perspective based on the event log shown in Table 1.

The process perspective is concerned with the "How?" question, the organizational perspective is concerned with the "Who?" question, and the case perspective is concerned with the "What?" question. To illustrate the first two consider Figure 1. The log shown in Table 1 contains information about five cases (i.e., process instances). The log shows that for four cases (1, 2, 3, and 4) the activities A, B, C, and D have been executed. For the fifth case only three activities are executed: activities A, E, and D. Each case starts with the execution of A and ends with the execution of D. If activity B is executed, then also activity C is executed. However, for some cases activity C is executed before activity B. Based on the information shown in Table 1 and by making some assumptions about the completeness of the log (i.e., assuming that the cases are representative and a sufficient large subset of possible behaviors is observed), we can deduce the process model shown in Figure 1(a). The model is represented in terms of a Petri net [30]. The Petri net starts with activity A and finishes with activity D. These activities are represented by transitions. After executing A there is a choice between either executing B and C in parallel or just executing activity E. To execute B and C in parallel two non-observable activities (AND-split and AND-join) have been added. These activities have been added for routing purposes only and are not present in the event log. Note that for

this example we assume that two activities are in parallel if they appear in any order. By distinguishing between start events and complete events for activities it is possible to explicitly detect parallelism.

Figure 1(a) does not show any information about the organization, i.e., it does not use any information on the people executing activities. However, Table 1 shows information about the performers. For example, we can deduce that activity A is executed by either John or Sue, activity B is executed by John, Sue, Mike or Carol, C is executed by John, Sue, Mike or Carol, D is executed by Pete or Clare, and E is executed by Clare. We could indicate this information in Figure 1(a). The information could also be used to "guess" or "discover" organizational structures. For example, a guess could be that there are three roles: X, Y, and Z. For the execution of A role X is required and John and Sue have this role. For the execution of B and C role Y is required and John, Sue, Mike and Carol have this role. For the execution of D and E role Z is required and Pete and Clare have this role. For five cases these choices may seem arbitrary but for larger data sets such inferences capture the dominant roles in an organization. The resulting "activity-role-performer diagram" is shown in Figure 1(b). The three "discovered" roles link activities to performers. Figure 1(c) shows another view on the organization based on the transfer of work from one individual to another, i.e., not focus on the relation between the process and individuals but on relations among individuals (or groups of individuals). Consider for example Table 1. Although Carol and Mike can execute the same activities (B and C), Mike is always working with John (cases 1 and 2) and Carol is always working with Sue (cases 3 and 4). Probably Carol and Mike have the same role but based on the small sample shown in Table 1 it seems that John is not working with Carol and Sue is not working with Mike.[2] These examples show that the event log can be used to derive relations between performers of activities, thus resulting in a sociogram. For example, it is possible to generate a sociogram based on the transfers of work from one individual to another as is shown in Figure 1(c). Each node represents one of the six performers and each arc represents that there has been a transfer of work from one individual to another. The definition of "transfer of work from A to B" is based on whether for the same case an activity executed by A is directly followed by an activity executed by B. For example, both in case 1 and 2 there is a transfer from John to Mike. Figure 1(c) does not show frequencies. However, for analysis purposes these frequencies can be added. The arc from John to Mike would then have weight 2. Typically, we do not use absolute frequencies but weighted frequencies to get relative values between 0 and 1. Figure 1(c) shows that work is transferred to Pete but not vice versa. Mike only interacts with John and Carol only interacts with Sue. Clare is the only person transferring work to herself.

Besides the "How?" and "Who?" question (i.e., the process and organization perspectives), there is the case perspective that is concerned with the "What?"

[2] Clearly the number of events in Table 1 is too small to establish these assumptions accurately. However, for the sake of argument we assume that the things that did not happen will never happen.

question. Figure 1 does not address this. In fact, focusing on the case perspective is most interesting when also data elements are logged but these are not listed in Table 1. The case perspective looks at the case as a whole and tries to establish relations between the various properties of a case. Note that some of the properties may refer to the activities being executed, the performers working on the case, and the values of various data elements linked to the case. Using clustering algorithms it would for example be possible to show a positive correlation between the size of an order or its handling time and the involvement of specific people.

Orthogonal to the three perspectives (process, organization, and case), the result of a mining effort may refer to *logical* issues and/or *performance* issues. For example, process mining can focus on the logical structure of the process model (e.g., the Petri net shown in Figure 1(a)) or on performance issues such as flow time. For mining the organizational perspectives, the emphasis can be on the roles or the social network (cf. Figure 1(b) and (c)) or on the utilization of performers or execution frequencies.

To address the three perspectives and the logical and performance issues we have developed a set of tools including EMiT [2], Thumb [35], and MinSoN [4]. These tools share a common XML format. For more details we refer to http://www.processmining.org.

3 WF-nets and the α-Algorithm

This section contains the main definitions used in the α-algorithm. For more information on the α-algorithm and its supporting definitions the reader is referred to [7]. We assume some basic knowledge of Petri nets. Readers not familiar with basic concepts such as (P, T, F) as a representation for a Petri net, the firing rule, firing sequences, preset $\bullet x$, postset $x\bullet$, boundedness, liveness, reachability, etc. are referred to [1, 14, 30].

3.1 Workflow Nets

Before introducing the α-algorithm we briefly discuss a subclass of Petri nets called a *WorkFlow nets* (WF-nets). This subclass is tailored towards modeling the control-flow dimension of a workflow[3] or any other case driven process, e.g., logging onto a system. It should be noted that a WF-net specifies the dynamic behavior of a single case in isolation [1].

Definition 3.1. (Workflow nets) Let $N = (P, T, F)$ be a Petri net and $\bar{t}$ a fresh identifier not in $P \cup T$. N is a *workflow net* (WF-net) iff:

1. *object creation*: P contains an input place i such that $\bullet i = \emptyset$,
2. *object completion*: P contains an output place o such that $o\bullet = \emptyset$,
3. *connectedness*: $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected,

[3] Note that we use the words *workflow* and *process* interchangeably.

The Petri net shown in Figure 1 is a WF-net. Note that although the net is not strongly connected, the *short-circuited* net with transition $\bar{t}$ is strongly connected. Even if a net meets all the syntactical requirements stated in Definition 3.1, the corresponding process may exhibit errors such as deadlocks, tasks which can never become active, livelocks, garbage being left in the process after termination, etc. Therefore, we define the following correctness criterion.

Definition 3.2. (Sound) Let $N = (P, T, F)$ be a WF-net with input place i and output place o. N is *sound* iff:

1. *safeness*: $(N, [i])$ is safe,
2. *proper completion*: for any marking $s \in [N, [i]\rangle$, $o \in s$ implies $s = [o]$,
3. *option to complete*: for any marking $s \in [N, [i]\rangle$, $[o] \in [N, s\rangle$, and
4. *absence of dead tasks*: $(N, [i])$ contains no dead transitions.

The set of all sound WF-nets is denoted $\mathcal{W}$.

The WF-net shown in Figure 1 is sound. Soundness can be verified using standard Petri-net-based analysis techniques [1, 3].

Most process modeling languages offer standard building blocks such as the AND-split, AND-join, XOR-split, and XOR-join [3]. These are used to model sequential, conditional, parallel and iterative routing. Clearly, a WF-net can be used to specify the routing of cases, i.e., process instances. *Tasks*, also referred to as *activities*, are modeled by transitions and causal dependencies are modeled by places and arcs. In fact, a place corresponds to a *condition* which can be used as pre- and/or post-condition for tasks. An AND-split corresponds to a transition with two or more output places, and an AND-join corresponds to a transition with two or more input places. XOR-splits/XOR-joins correspond to places with multiple outgoing/ingoing arcs. Given the close relation between tasks and transitions we use the terms interchangeably.

Our process mining research aims at rediscovering WF-nets from event logs. However, not all places in sound WF-nets can be detected. For example places may be implicit which means that they do not affect the behavior of the process. These places remain undetected. Therefore, we limit our investigation to WF-nets without implicit places.

Definition 3.3. (Implicit place) Let $N = (P, T, F)$ be a Petri net with initial marking s. A place $p \in P$ is called implicit in (N, s) if and only if, for all reachable markings $s' \in [N, s\rangle$ and transitions $t \in p\bullet$, $s' \geq \bullet t \setminus \{p\} \Rightarrow s' \geq \bullet t$.[4]

Figure 1 contains no implicit places. However, adding a place p connecting transition A and D yields an implicit place. No mining algorithm is able to detect p since the addition of the place does not change the behavior of the net and therefore is not visible in the log.

For process mining it is very important that the structure of the WF-net clearly reflects its behavior. Therefore, we also rule out the constructs shown in

[4] $[N, s\rangle$ is the set of reachable markings of net N when starting in marking s, $p\bullet$ is the set of output transitions of p, $\bullet t$ is the set of input places of t, and $\geq$ is the standard ordering relation on multisets.

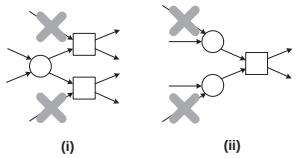


Fig. 2. Constructs not allowed in SWF-nets.

Figure 2. The left construct illustrates the constraint that choice and synchronization should never meet. If two transitions share an input place, and therefore "fight" for the same token, they should not require synchronization. This means that choices (places with multiple output transitions) should not be mixed with synchronizations. The right-hand construct in Figure 2 illustrates the constraint that if there is a synchronization all preceding transitions should have fired, i.e., it is not allowed to have synchronizations directly preceded by an XOR-join. WF-nets which satisfy these requirements are named *structured workflow nets* and are defined as:

Definition 3.4. (SWF-net) A WF-net $N = (P, T, F)$ is an *SWF-net* (Structured workflow net) if and only if:

1. For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|p \bullet| > 1$ implies $|\bullet t| = 1$.
2. For all $p \in P$ and $t \in T$ with $(p, t) \in F$: $|\bullet t| > 1$ implies $|\bullet p| = 1$.
3. There are no implicit places.

3.2 The α-Algorithm

The starting point for process mining is the event log. A log is a set of traces. Event traces and logs are defined as:

Definition 3.5. (Event trace, event log) Let T be a set of tasks. $\sigma \in T^*$ is an *event trace* and $W \in \mathcal{P}(T^*)$ is an *event log*.[5]

From an event log, ordering relations between tasks can be inferred. In the case of the α-algorithm, every two tasks in the event log must have one of the following four ordering relations: $>_W$ (follows), $\rightarrow_W$ (causal), $\|_W$ (parallel) and $\#_W$ (unrelated). These ordering relations are extracted based on local information in the event traces. The ordering relations are defined as:

Definition 3.6. (Log-based ordering relations) Let W be an event log over T, i.e., $W \in \mathcal{P}(T^*)$. Let $a, b \in T$:

- $a >_W b$ if and only if there is a trace $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ and $i \in \{1, \ldots, n-2\}$ such that $\sigma \in W$ and $t_i = a$ and $t_{i+1} = b$,
- $a \rightarrow_W b$ if and only if $a >_W b$ and $b \not>_W a$,
- $a \#_W b$ if and only if $a \not>_W b$ and $b \not>_W a$, and
- $a \|_W b$ if and only if $a >_W b$ and $b >_W a$.

To ensure the event log contains the minimal amount of information necessary to mine the process, the notion of log completeness is defined as:

[5] T^* is the set of all sequences that are composed of zero of more tasks from T. $\mathcal{P}(T^*)$ is the powerset of T^*, i.e., $W \subseteq T^*$.

Definition 3.7. (Complete event log) Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$. W is an *event log of* N if and only if $W \in \mathcal{P}(T^*)$ and every trace $\sigma \in W$ is a firing sequence of N starting in state $[i]$ and ending in state $[o]$, i.e., $(N, [i])[\sigma\rangle(N, [o])$. W is a *complete* event log of N if and only if (1) for any event log W' of N: $>_{W'} \subseteq >_W$, and (2) for any $t \in T$ there is a $\sigma \in W$ such that $t \in \sigma$.

For Figure 1, a possible complete event log W is $\{ABCD, ACBD, AED\}$. From this complete log, the following ordering relations are inferred:

- (follows) $A >_W B$, $A >_W C$, $A >_W E$, $B >_W C$, $B >_W D$, $C >_W B$, $C >_W D$ and $E >_W D$.
- (causal) $A \rightarrow_W B$, $A \rightarrow_W C$, $A \rightarrow_W E$, $B \rightarrow_W D$, $C \rightarrow_W D$ and $E \rightarrow_W D$.
- (parallel) $B \|_W C$ and $C \|_W B$.

Now we can give the formal definition of the α-algorithm followed by a more intuitive explanation.

Definition 3.8. (Mining algorithm α) Let W be an event log over T. The $\alpha(W)$ is defined as follows.

1. $T_W = \{t \in T \mid \exists_{\sigma \in W} t \in \sigma\}$,
2. $T_I = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{first}(\sigma)\}$,
3. $T_O = \{t \in T \mid \exists_{\sigma \in W} t = \mathit{last}(\sigma)\}$,
4. $X_W = \{(A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$,
5. $Y_W = \{(A, B) \in X_W \mid \forall_{(A', B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Longrightarrow (A, B) = (A', B')\}$,
6. $P_W = \{p_{(A,B)} \mid (A, B) \in Y_W\} \cup \{i_W, o_W\}$,
7. $F_W = \{(a, p_{(A,B)}) \mid (A, B) \in Y_W \wedge a \in A\} \cup \{(p_{(A,B)}, b) \mid (A, B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}$, and
8. $\alpha(W) = (P_W, T_W, F_W)$.

The α-algorithm works as follows. First, it examines the event traces and (Step 1) creates the set of transitions (T_W) in the process, (Step 2) the set of output transitions (T_I) of the source place , and (Step 3) the set of the input transitions (T_O) of the sink place[6]. In steps 4 and 5, the α-algorithm creates sets (X_W and Y_W, respectively) used to define the places of the discovered WF-net. In Step 4, the α-algorithm discovers which transitions are causally related. Thus, for each tuple (A, B) in X_W, each transition in set A causally relates to *all* transitions in set B, and no transitions within A (or B) follow each other in some firing sequence. These constraints to the elements in sets A and B allow the correct mining of AND-split/join and XOR-split/join constructs. Note that the XOR-split/join requires the fusion of places. In Step 5, the α-algorithm refines set X_W by taking only the largest elements with respect to set inclusion. In fact, Step 5 establishes the exact amount of places the discovered net has (excluding the source place i_W and the sink place o_W). The places are created in Step 6 and connected to their respective input/output transitions in Step 7. The discovered

[6] In a WF-net, the source place i has no input transitions and the sink place o has no output transitions.

WF-net is returned in Step 8. Figure 3 shows the result of applying the α-algorithm to the log shown in Table 1, i.e. $W = \{ABCD, ACBD, AED\}$. Note that the α-algorithm is not able to discover the AND-split and AND-join shown in Figure 1 (these are not in the log), but is still able to construct an equivalent WF-net.

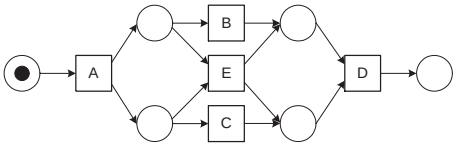


Fig. 3. The WF-net discovered by the α-algorithm (based on Table 1).

Finally, we define what it means for a WF-net to be *rediscovered* and roughly characterize the class of processes for which the α-algorithm works correctly.

Definition 3.9. (Ability to rediscover) Let $N = (P, T, F)$ be a sound WF-net, i.e., $N \in \mathcal{W}$, and let α be a mining algorithm which maps event logs of N onto sound WF-nets, i.e., $\alpha : \mathcal{P}(T^*) \rightarrow \mathcal{W}$. If for any complete event log W of N the mining algorithm returns N (modulo renaming of places), then α is able to *rediscover* N.

Theorem 3.10. Let $N = (P, T, F)$ be a sound SWF-net and let W be a complete event log of N. If for all $a, b \in T$ $a\bullet \cap \bullet b = \emptyset$ or $b\bullet \cap \bullet a = \emptyset$, then $\alpha(W) = N$ modulo renaming of places.

Note that no mining algorithm is able to find names of places. Therefore, we ignore place names, i.e., α is able to rediscover N if and only if $\alpha(W) = N$ modulo renaming of places. Also note the requirement not allowing "short loops". Using the refinement described in [26] this additional requirement can be avoided.

4 Detecting Anomalous Process Executions

In Subsection 3.2, we presented how the α-algorithm can mine the cases in Table 1, and discover a process (see Figure 3) that describes *all* possible behaviors. A similar reasoning holds for security issues if we consider the *event traces* to be *audit trails*, and the *cases* as e.g. *session ids*. In this section we show (i) how to use the α-algorithm to discover the acceptable or normal behavior in systems and (ii) how to use the discovered net to detect undesired behavior.

The α-algorithm discovers a net that models all acceptable behavior whenever the *complete log* given as input has *only acceptable* audit trails and the discovered net is a sound WF-net. For example, imagine a website that is used to sell products. Assume every user in this website has a shopping basket that can be edited at any time. If the shopping basket contains products when the user leaves the website, the user basket's status is saved and is retrieved when the user enters the website again. Possible user actions are described by the WF-net in Figure 4. Now, assume we do not know the net in Figure 4, but we do have a complete log of acceptable audit trails. For instance, let this audit log be $W_{OK} = \{$"*Enter, Select*

Product, Add to Basket, Cancel Order", "*Enter, Select Product, Remove from Basket, Cancel*", "*Enter, Select Product, Add to Basket, Continue Shopping, Select Product, Remove from Basket, Continue Shopping, Select Product, Add to Basket, Proceed to Checkout, Fill in Delivery Info, Fill in Payment Info, Provide Password, Process Order, Finish Checkout*", "*Enter, Select Product, Remove from Basket, Proceed to Checkout, Fill in Payment Info, Fill in Delivery Info, Provide Password, Process Order, Finish Checkout*"}. Given W_{OK} as input, the α-algorithm discovers the net shown in Figure 4.

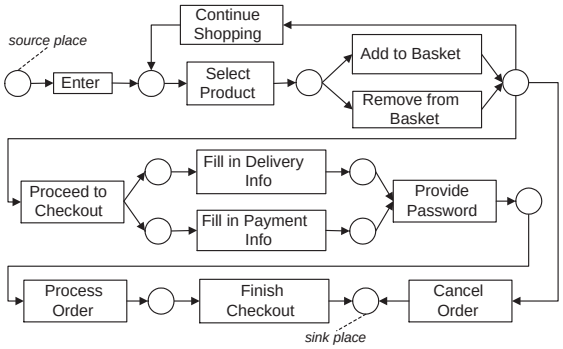


Fig. 4. Example of a process description to buy products at a website.

Once the net is discovered, the conformance of *every* new audit trail can be verified by playing the "token game". Note that anomalous audit trails do not correspond to possible firing sequences in the "token game" for the discovered net. Furthermore, the "token game" detects the point in which the audit trail diverges from the normal behavior and allows also for the *real time* verification of trails. For example, let us verify the new audit log $W_{NOK} = \{$"*Enter, Select Product, Remove from Basket, Proceed to Checkout, Fill in Delivery Info, Fill in Payment Info, Provide Password, Process Order, Finish Checkout*", "*Enter, Select Product, Remove from Basket, Proceed to Checkout, Fill in Payment Info, Fill in Delivery Info, Process Order, Finish Checkout*"$\}$ by playing every trace in W_{OK} in the net in Figure 4. The first audit trail in W_{NOK} is an acceptable one. Note that this trail *is not* in W_{OK}, but it can be generated by the discovered net. The second trail is an anomalous one because it does not contain the task *Provide Password*. By playing the "token game", we see that two tokens get stuck in the input places of *Provide Password*. In other words, the "token game" *explicitly* shows the point where the anomalous behavior happened. The EMiT tool supports the "token game" and indicates deadlocks and remaining tokens.

Note that the α-algorithm correctly discovered the net in Figure 4 without requiring the "training" complete log W_{OK} to show *all possible behavior* (the first trace in W_{NOK} is not in W_{OK}). However, because the α-algorithm aims at discovering the *process perspective*, it does not capture constraints that relate to data in the system, like the maximum number of times a loop may iterate. For the example in Figure 4, the loop can be executed an unlimited number of times

without violating security issues. Nonetheless, if the loop would correspond to user attempts to log into the system, a maximum number of loop iterations must be set. If this is the case, the discovered WF-net must be explicitly modified to incorporate the required data-related constraints. As a final remark, we would like to point out that the simple idea of playing the "token game" can also be used without applying the α-algorithm, i.e., by explicitly modeling the process. However, given the evolving nature of systems and processes, the α-algorithm is a useful tool to keep the "security process" up-to-date. For example, if an audit trail "does not fit" but does not correspond to a violation, then it can be added to the event log used by the α-algorithm. Audit trails that seemed OK, but turned out to be potential security breaches can be removed from the log. By applying the α-algorithm to the modified event log, a new and updated "security process" can be obtained without any modeling efforts.

5 Checking Process Conformance

The ordering relations can be used to check system properties. In Section 4, a process model is derived from acceptable audit trails. The discovered net is then used to check new audit trails. In this case, every audit trail must comply with the process. However, sometimes security applies *only to a part* of the process. For example, for the process in Figure 4, the critical security issue is to execute the task *Provide Password* before *Process Order*. In other words, task *Provide Password* should *cause* task *Process Order*. The process fragment for this situation is construct (a) in Figure 5. This construct is mapped to the ordering relation *Provide Password*$\rightarrow$*Process Order*. Thus, given an audit log, we can check if this pattern holds *for the system*. I.e., considering *all* audit trails, we check if the ordering relations that are equivalent to the desired pattern hold. Back to our example in Section 4 and considering an audit log $W = W_{OK} \cup W_{NOK}$, we do infer the relation *Provide Password*$\rightarrow_W$*Process Order*. Thus, we can conclude that the process described by W contains the pattern shown in Figure 5(a).

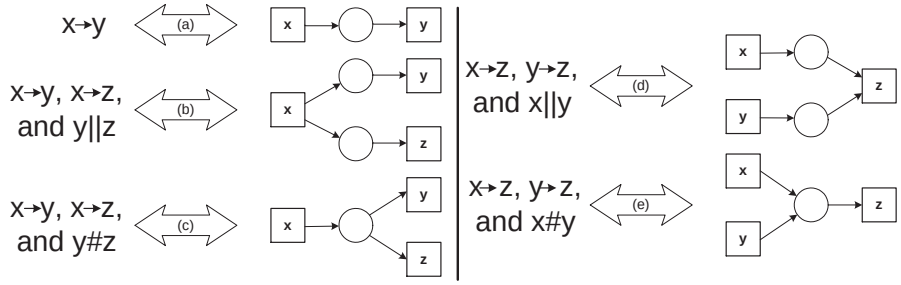


Fig. 5. Relating the log-based relations $>_W$, $\rightarrow_W$, $\|_W$, and $\#_W$ to basic Petri-net constructs.

The approach to check process conformance verifies if a pattern holds, but does not assure this is *always* the case. Full conformance can be verified by

combining this approach with the one in Section 4. The difference is that now we play the "token game" *with the subnet*. By playing every event trace in the desired pattern, we check if there is always a causal relation between *Provide Password* and *Process Order*. Note this will not be the case for all trails in W and the anomalous one will be detected. The main advantage of the approach for checking process conformance is that it does not require a complete audit log for the *whole* process, but only for the tasks involved in the pattern. Figure 5 illustrates the basic patterns that can be used to build process fragments.

6 Related Work

The idea of process mining is not new [2, 5, 7, 8, 13, 22, 23, 25, 28, 32, 35] and most techniques aim at the control-flow perspective. However, process mining is not limited to the control-flow perspective. For example, in [4] we use process mining techniques to construct a social network. For more information on process mining we refer to a special issue of Computers in Industry on process mining [6] and a survey paper [5]. In this paper, unfortunately, it is impossible to do justice to the work done in this area.

The focus of this paper is on the α-algorithm. For more information on the algorithm, we refer to [2, 7, 25, 35]. In [26] one of the problems raised in [25] is tackled ("short loops") and should be considered as an extension of [7].

In the security domain there are related papers dealing with intrusion detection based on audit trails [18, 36]. These paper break "normal behavior" into smaller patterns and then compare actual audit trails using these patterns. Note that, unlike the α-algorithm, these approaches do not consider explicit process models.

There have been many formal approaches towards security, e.g., using Petri nets or process algebras [16, 17]. Unlike our approach they typically focus on verification of a design rather than analyzing the actual behavior.

To support our mining efforts we have developed a set of tools including EMiT [2], Thumb [35], and MinSoN [4]. These tools share a common XML format. For more details we refer to www.processmining.org.

7 Conclusion

In this paper, we explored the application of process mining techniques in security. First, we introduced process mining and then we focused on one algorithm to mine the process perspective. Then we showed the application of this algorithm to security issues. First we discussed the detection of anomalous process executions in the mined WF-net by playing the "token game" for concrete cases. Then, we showed that process conformance can be checked by comparing process fragments with the discovered WF-net.

We would like to emphasize that we consider the application of the α-algorithm at any level of security, i.e., from low-level intrusion detection to

high-level fraud presenting. The focus on *Corporate Governance* and governmental regulations such as *Sarbanes-Oxley Act* trigger the development of tools to enforce and check security at the level of business processes. We believe that organizations will increasingly need to store and monitor audit trails. Process mining techniques such as the α-algorithm can assist in these efforts.

Acknowledgements
The author would like to thank Ton Weijters, Boudewijn van Dongen, Minseok Song, Laura Maruster, Eric Verbeek, Monique Jansen-Vullers, Hajo Reijers, Michael Rosemann, and Peter van den Brand for their on-going work on process mining techniques and tools at Eindhoven University of Technology.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst and B.F. van Dongen. Discovering Workflow Performance Models from Timed Logs. In Y. Han, S. Tai, and D. Wikarski, editors, *International Conference on Engineering and Deployment of Cooperative Information Systems (EDCIS 2002)*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer-Verlag, Berlin, 2002.
3. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
4. W.M.P. van der Aalst and M. Song. Mining Social Networks: Uncovering interaction patterns in business processes. In M. Weske, B. Pernici, and J. Desel, editors, *International Conference on Business Process Management (BPM 2004)*, Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2004.
5. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
6. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
7. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. QUT Technical report, FIT-TR-2003-03, Queensland University of Technology, Brisbane, 2003. (Accepted for publication in IEEE Transactions on Knowledge and Data Engineering.).
8. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
9. A.A. Bavelas. A Mathematical Model for Group Structures. *Human Organization*, 7:16–30, 1948.
10. H.R. Bernard, P.D. Killworth, C. McCarty, G.A. Shelley, and S. Robinson. Comparing Four Different Methods for Measuring Personal Social Networks. *Social Networks*, 12:179–216, 1990.
11. M. Bishop. *Computer Security: Art and Science*. Addison-Wesley, Boston, USA, 2003.
12. R.S. Burt and M. Minor. *Applied Network Analysis: A Methodological Introduction*. Sage, Newbury Park CA, 1983.

13. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
14. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
15. M. Feldman. Electronic Mail and Weak Ties in Organizations. *Office: Technology and People*, 3:83–101, 1987.
16. R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.
17. R. Focardi, R. Gorrieri, and F. Martinelli. A Comparison of Three Authentication Properties. *Theoretical Computer Science*, 291(3):285–327, 2003.
18. S. Forrest, A.S. Perelson, L. Allen, and R. Cherukuri. Self-Nonself Discrimination in a Computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212. IEEE Computer Society Press, Los Alamitos, California, 1994.
19. L.C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40:35–41, 1977.
20. L.C. Freeman. Centrality in Social Networks: Conceptual Clarification. *Social Networks*, 1:215–239, 1979.
21. D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. In P. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. Snodgrass, editors, *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 159–168. Morgan Kaufmann, 2001.
22. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
23. IDS Scheer. ARIS Process Performance Manager (ARIS PPM). http://www.ids-scheer.com, 2002.
24. G. Keller and T. Teufel. *SAP R/3 Process Oriented Implementation*. Addison-Wesley, Reading MA, 1998.
25. A.K.A. de Medeiros, W.M.P. van der Aalst, and A.J.M.M. Weijters. Workflow Mining: Current Status and Future Directions. In R. Meersman, Z. Tari, and D.C. Schmidt, editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer-Verlag, Berlin, 2003.
26. A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters. Process Mining: Extending the α-algorithm to Mine Short Loops. BETA Working Paper Series, WP 113, Eindhoven University of Technology, Eindhoven, 2004.
27. J.L. Moreno. *Who Shall Survive?* Nervous and Mental Disease Publishing Company, Washington, DC, 1934.
28. M. zur Mühlen and M. Rosemann. Workflow-based Process Monitoring and Controlling - Technical and Organizational Issues. In R. Sprague, editor, *Proceedings of the 33rd Hawaii International Conference on System Science (HICSS-33)*, pages 1–10. IEEE Computer Society Press, Los Alamitos, California, 2000.
29. H. Nemati and C.D. Barko. *Organizational Data Mining: Leveraging Enterprise Data Resources for Optimal Performance*. Idea Group Publishing, Hershey, PA, USA, 2003.

30. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1998.
31. J.C. Robertson. *Auditing*. Irwin, Burr Ridge, USA, 1993.
32. M. Sayal, F. Casati, and M.C. Shan U. Dayal. Business Process Cockpit. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. Morgan Kaufmann, 2002.
33. J. Scott. *Social Network Analysis*. Sage, Newbury Park CA, 1992.
34. S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, Cambridge, 1994.
35. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.
36. A. Wespi, M. Dacier, and H. Debar. Intrusion Detection Using Variable-Length Audit Trail Patterns. In H. Debar, L. Me, and S.F. Wu, editors, *Recent Advances in Intrusion Detection*, volume 1907 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, Berlin, 2000.