

Modeling and Verifying Security Policies in Business Processes

Mattia Salnitri¹, Fabiano Dalpiaz², and Paolo Giorgini¹

¹ University of Trento, Italy

{mattia.salnitri, paolo.giorgini}@unitn.it

² Utrecht University, the Netherlands

f.dalpiaz@uu.nl

Abstract. Modern information systems are large-sized and comprise multiple heterogeneous and autonomous components. Autonomy enables decentralization, but it also implies that components providers are free to change, retire, or introduce new components. This is a threat to security, and calls for a continuous verification process to ensure compliance with security policies. Existing verification frameworks either have limited expressiveness—thereby inhibiting the specification of real-world requirements—, or rely on formal languages that are hardly employable for modeling and verifying large systems. In this paper, we overcome the limitations of existing approaches by proposing a framework that enables: (1) specifying information systems in SecBPMN, a security-oriented extension of BPMN; (2) expressing security policies through SecBPMN-Q, a query language for representing security policies; and (3) verifying SecBPMN-Q against SecBPMN specifications via an implemented query engine. We report on the applicability of our approach via a case study about air traffic management.

Keywords: Information systems, Security policies, BPMN, Compliance

1 Introduction

Information systems are becoming increasingly large, complex, and decentralized. Air Traffic Management (ATM) systems, smart grids, and smart cities are not simple monolithic systems, but rather they consist of a high number of autonomous, heterogeneous, and mutually interdependent components. These systems require new design techniques, in order to prevent crashes with effects on both organizations and society [33].

These systems manage a large amount of private and confidential information; as such, their design shall ensure information assurance and security both in technical terms and from an organizational perspective [26]. Business process models are an adequate abstraction to do so, for they express an information system in terms of the interactions between humans, organizations, and technical systems.

Several modeling languages have been proposed that extend BPMN (Business Process Modelling and Notation) [23]—the de-facto standard notation for representing business processes—with security annotations that individual BPMN elements shall comply with [26, 34]. For example, Rodriguez et al. [26] extended BPMN with a predefined set of security annotations (e.g., attack/harm detection and privacy) that constrain the execution of the annotated tasks of the business process.

However, the security annotations in [26] constrain individual elements in a business process, and do not allow expressing security policies that specify the admissible behavior of the whole business process. Some extensions of BPMN employ a predefined set of policies—BPMN patterns which specify the behavior of the business process—but they do not allow the definition of custom security policies [6, 20]. This is the case, for instance, of SecureBPMN [6], which introduces security elements for expressing, e.g., separation and binding of duties.

Furthermore, business process modelers need to verify whether a process model complies with the specified policies. This verification is required not only when designing the process, but also whenever the participating components do change after its deployment. For example, in an ATM system, a failure in the pilot to control tower communication component requires a quick reconfiguration of the system, which shall be checked for compliance with the security policies.

Existing approaches for compliance checking are inadequate: some focus on general-purpose policies and do not provide support to security policies [3, 11, 17, 28], while others use a too limited set of policies, mainly concerning access control [19, 20, 26].

In this paper, to overcome the limitations of existing approaches, we propose a framework for modeling and verifying the compliance of a business process model with a set of security policies. To do so, we take BPMN-Query (BPMN-Q) [3] as our baseline; BPMN-Q is a query language that enables expressing and verifying generic queries over a BPMN model. We extend BPMN-Q with a number of annotations for expressing security policies. We make the following contributions:

1. The SecBPMN language, which extends BPMN with security annotations.
2. SecBPMN-Q, an extension of BPMN-Q for specifying security policies as queries.
3. An implemented framework for modeling in SecBPMN, specifying security policies in SecBPMN-Q, and running SecBPMN-Q queries against SecBPMN models.
4. We evaluate the applicability of our approach on a case study.

The rest of the paper is structured as follows. Section 2 describes our baseline. Sections 3 and 4 introduce SecBPMN and SecBPMN-Q, respectively. Section 5 discusses related work and shows how we applied our approach on a case study. Finally, Section 6 presents our conclusions and outlines future directions.

2 Baseline

In this section we briefly introduce the baseline of our research: the BPMN-Q language for querying business process models, and the RMIAS security reference model.

While BPMN is an effective means for expressing the interactions among the components in a complex system, it does not offer the possibility to verify whether certain critical properties of the model do hold. For example, when modeling the landing procedure in air traffic management, one cannot automatically verify with BPMN that pilots do confirm the landing trajectory of the plane.

Visual analysis of BPMN models works for small scenarios, but it is ineffective when many models exist, or when they are as large as hundreds of nodes. Moreover, when safety and security properties are concerned, relying on an informal analysis is not

an option, due to the harmful effects of adopting a model that violates them. BPMN-Q is a diagrammatic query language which partially overcomes this limitation, by expressing properties concerning business process models through graphical queries that can be checked against a model [4]. These queries can be seen as patterns that a given BPMN model should comply with. BPMN-Q introduces a set of relations that are functional to define the queries, i.e. the concepts of path, negative path, negative flow as well as an extension of the “data object” element that enable characterizing the state of the object.

Figure 1 shows an example of a BPMN-Q query (taken from our SWIM ATM case study³). The query enables checking whether the flight plan (Reference Business Trajectory or simply RBT) is approved and if the landing documents are checked at least once. The query will match against all business processes where the first activity “Plane RBT generation service” generates the data object “RBT [Proposed]” (between brackets is indicated the state of the data object) and other two activities are executed: (i) “Control Tower communication service” generates the data object “RBT [Accepted]”; (ii) any activity (“@Y”) reads the data object “Landing documents [Approved]”.

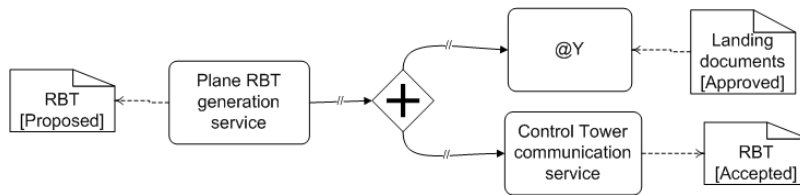


Fig. 1. Exmple of a BPMN-Q query

BPMN-Q enables expressing generic properties over BPMN elements, but does not support the specification of security properties. To overcome this limitation, the languages proposed in this paper extend BPMN and BPMN-Q with primitives for specifying and querying security properties, which we define based on a state-of-the-art reference model for information security. A prominent family of reference models extends the Confidentiality Integrity Availability (CIA) triad [24]. However, their adequacy has been questioned for they characterize a too limited set of properties of a system [25]. Later, more complete reference models were proposed, for example McCumber’s cube [18], which analyzes system security from three different perspectives: information states, critical information characteristics and security measures. The Business Model for Information Security (BMIS) [1] focuses on business environments, and consists of four interconnected elements: organization design and strategy element, people element, process element and technology element. In our work, we choose the Reference Model on Information Assurance and Security (RMIAS) [7], as it is the result of an analysis and classification of security aspects proposed by the most known

³ The System Wide Information Management (SWIM) [2] is a next-generation communication system which enables the secure interchange of information among ATM decision makers. We use it to evaluate the languages in Section 5

reference models on information assurance and security. As far as our knowledge goes, it proposes the most comprehensive set of security aspects, that are listed in Table 1.

Name	Definition
Accountability	An ability of a system to hold users responsible for their actions (e.g. misuse of information).
Auditability	An ability of a system to conduct persistent, non-by passable monitoring of all actions performed by humans or machines within the system.
Authenticity	An ability of a system to verify identity and establish trust in a third party and in information it provides.
Availability	A system should ensure that all system's components are available and operational when they are required by authorised users.
Confidentiality	A system should ensure that only authorised users access information.
Integrity	A system should ensure completeness, accuracy and absence of unauthorised modifications in all its components.
Non-Repudiation	The ability of a system to prove (with legal validity) occurrence/non-occurrence of an event or participation/non-participation of a party in an event.
Privacy	A system should obey privacy legislation and it should enable individuals to control, where feasible, their personal information (user-involvement).

Table 1. Security aspects in RMIAS [7]

3 SecBPMN: a modeling language for secure business processes

We extend BPMN with security annotations covering each of the security aspects in the RMIAS reference model (see Table 1). Every annotation has a graphical syntax and is linked with an existing element of a BPMN model: an activity, a data object, or message flow. Moreover, annotations have attributes that security designers can use to specify detailed information on the security mechanisms⁴ that enforce the policy. All attributes are optional but one: the BPMN element linked with the annotation.

Specifically, our language extends the subset of BPMN—that is supported by BPMN-Q—for specifying orchestrations, which enables expressing interactions among information system components: activities, gateways and data objects. Each security annotation is formalized in terms of one or more predicates, one for every type of BPMN element the annotation can be linked with.

Our graphical syntax was carefully designed according with Moody's guidelines for increasing the usability and comprehensibility of modeling languages [21]. The annotations share three common visual variables: they all have an orange fill color, a solid texture, and a circular shape; they differ in the icon in the middle of the circle. Every security annotation has a visual distance of three from non-security annotations, and a

⁴ They define the low level (software and hardware) functions that implement the controls imposed by the policy [31]

visual distance of one from other security annotations. We decide to use icons instead of abstract symbols because icons are easy to remember and faster to recognize [21]. Leitner et al. [14–16] conducted empirical studies to propose guidelines for representing a set of security aspects. We did not apply such suggestions because they conflict with the recommendation by the security experts that helped us define the security annotations and, moreover, the set of security aspects Leitner et al. took into account covers only partially the security aspects proposed in RMIAS.

Figure 2 shows an example of BPMN extended with security annotations, which shows part of a service composition, offered by different service providers, intended to enable flight tickets booking. There, the security annotations specify the security aspects that the implemented services will comply with. The annotations are defined in Table 2 and explained below.

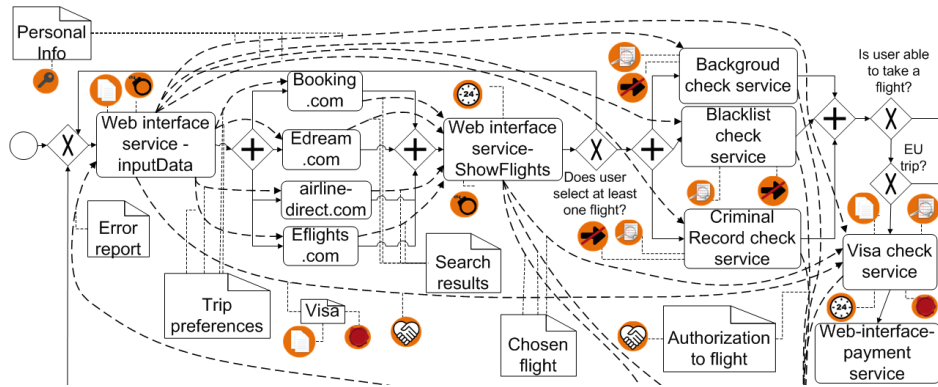


Fig. 2. Example of a SecBPMN business process model

Accountability. It applies only to activities, and expresses the need of monitoring a set of users when executing the activity. Thus, there is only one corresponding predicate named `AccountabilityAct`. It has three parameters: `a` is the activity whose execution has to satisfy the security aspect corresponding to this type of annotation, `enfBy` is a set of security mechanisms used to enforce accountability for the activity, `monitored` is the set of users which are monitored.

If the activity is executed by a user that is not in monitored, the security aspect is satisfied without using the enforcement mechanism. This situation would typically occur with trusted users that do not need be monitored. Security designers can specify they keyword `ALL` in `monitored`, to indicate that all users are held for their actions.

Consider, for example, the predicate `AccountabilityAct("Web interface service - inputData", {RBAC}, {customer})`, which details one of the accountability security annotations in Figure 2. The first attribute contains the activity linked with the security annotation, the second one indicates that RBAC (Role-Based Access Control) [9] will be used to enforce accountability, while the third attribute specifies that only customers have to be monitored while executing that activity.









AccountabilityAct (a: Activity, enfBy: {SecMechanisms}, monitored: {Users})	
AuditabilityAct (a: Activity, enfBy: {SecMechanisms}, frequency: Time) AuditabilityDO (do: DataObject, enfBy: {SecMechanisms}, frequency: Time) AuditabilityMF (mf: MessageFlow, enfBy: {SecMechanisms}, frequency: Time)	
AuthenticityAct (a: Activity, enfBy: {SecMechanisms}, ident: Bool, auth: Bool, trustValue: Float) AuthenticityDO (do: DataObject, enfBy: {SecMechanisms})	
AvailabilityAct (a: Activity, enfBy: {SecMechanisms}, level: Float) AvailabilityDO (do: DataObject, enfBy: {SecMechanisms}, authUsers: {Users}, level: Float) AvailabilityMF (mf: MessageFlow, enfBy: {SecMechanisms}, level: Float)	
ConfidentialityDO (do: DataObject, enfBy: {SecMechanisms}, readers: {Users}, writers: {Users}) ConfidentialityMF (mf: MessageFlow, enfBy: {SecMechanisms}, readers: {Users}, writers: {Users})	
IntegrityAct (a: Activity, enfBy: {SecMechanisms}, personnel: Bool, hardware: Bool, software: Bool) IntegrityDO (do: DataObject, enfBy: {SecMechanisms}) IntegrityMF (mf: MessageFlow, enfBy: {SecMechanisms})	
NonRepudAct (a: Activity, enfBy: {SecMechanisms}, execution: Bool) NonRepudMF (mf: MessageFlow, enfBy: {SecMechanisms}, execution: Bool)	
PrivacyAct (a: Activity, enfBy: {SecMechanisms}, sensitiveInfo: {Info}) PrivacyDO (do: DataObject, enfBy: {SecMechanisms}, sensitiveInfo: {Info})	

Table 2. Security annotations of SecBPMN: predicates and graphical syntax

Auditability. It introduces three variants of security annotation, which are used to express that it should be possible to verify different aspects of the business process: (i) *AuditabilityAct* indicates that it should be possible to keep track of all the actions performed by the executor of the activity *a* when trying to execute that activity; (ii) *AuditabilityDO* indicates that it should be possible to keep track of all the actions that manage (e.g. write, read, store) the data object *do*; (iii) *AuditabilityMF* indicates that it should be possible to keep track of all the actions executed to handle the communication (send and receive actions) within the message flow *mf*.

The three predicates share two parameters: *enfBy* to express a specific set of security mechanisms to be used, and *frequency* to specify how frequently the security checks are performed. If *frequency* is set to zero, the continuous verification is required.

For instance, consider the predicate *AuditabilityAct*("Background check service", {}, 10d), which formalizes one of the auditability annotations in Figure 2. It applies to activity *Background check service*, it does not require a specific technology for checking auditability, and it requires audits to be performed every 10 days.

Authenticity. It comes in two versions, depending on which BPMN elements the annotation applies to. *AuthenticityAct* imposes that identity and/or authenticity of users of

activity *a* are verified. The attribute *enfBy* is the set of security mechanisms to be used while *trustValue* is the minimum level of trust [12] the executor of activity *a* must have. If attribute *ident* is true, anonymous users should not take part in the execution of the activity, while if *auth* is set to true, the identity of users should be verified. *AuthenticityDo* indicates that it should be possible to prove the data object *do* is genuine, i.e. it should be possible to prove that it was not modified by unauthorized parties, and to prove the identity of the entity who generated and/or modified it.

For example, consider the predicate *AuthenticityDO* (“*Visa*”, {*TLS*, *X.509*}), which formalizes an authenticity security annotation in Figure 2. The predicate specifies that the integrity of *Visa* data object should be guaranteed using *TLS* (Transport Security Layer) and *X.509* security mechanisms.

Availability. It applies to three BPMN elements, hence we defined three different versions: (i) *AvailabilityAct* specifies that the activity *a* should be executed every time it’s specified in the business process; (ii) *AvailabilityDO* specifies that the data object *do* should be available when required by the authorized users specified in *authUsers* attribute; (iii) *AvailabilityMF* specifies that it is always possible to communicate through the message flow *mf*.

The predicates share two parameters: *enfBy*, described above, and *level*, i.e., the minimum time percentage that the resource (i.e., activity, data object or message flow, depending on the variant of availability annotation) should be available. In *AvailabilityDO*, security designers can specify that all users are authorized to request the data object, simply specifying the keyword *ALL* in the attribute *authUsers*.

For instance, the predicate *AvailabilityAct*(“*Web interface service - ShowFlights*”, {*SAVE* }, 99.5) specifies that *Web interface service - ShowFlights* has to process at least 99.5% of the total requests, using the *SAVE* (Source Address Validity Enforcement) protocol to prevent denial of service attacks.

Confidentiality. It has two variants: *ConfidentialityDO* which specifies the data object *do* can be accessed only by authorized users, and *ConfidentialityMF* which specifies that only authorized users can use (i.e. send or receive) the message flow *mf*. Both predicates share three parameters: *enfBy*, already described; *readers* i.e. the set of users that are authorized to read the data object (or receive from the message flow); *writers* i.e. the set of users that are authorized to write the data object *do* (or send through the message flow). The attributes *readers* and *writers* allow the usage of the keyword *ALL* to specify that all the users are authorized.

For instance, consider the predicate *ConfidentialityMF* (*mf*(“*Web interface service - inputData*”, “*Visa check service*”), {*TLS*, *RBAC*}, {*controlAuthority*, *VisaOwner*}, {*VisaOwner*}), which details one of the confidentiality annotations in Figure 2. It specifies that only the users *controlAuthority* and *VisaOwner* can receive from the message flow between *Web interface service - inputData* and *Visa check service*, and only *VisaOwner* can send data objects through that channel. This security annotation must be enforced using both *TLS* and *RBAC* security mechanisms.

Integrity. It comes in three variants: (i) *IntegrityAct* specifies that the functionalities of activity *a* should be protected from intentional corruption. Attributes *personnel*, *hardware* and *software* determine if respectively the personnel, hardware or software, in-

volved in the execution of the a , are protected from intentional corruption [10]; (ii) IntegrityDO specifies that the data object do should be protected from intentional corruption [10]; (iii) IntegrityMF specifies that every message exchanged through mf should be protected from intentional corruption. All the predicates share the attribute $enfBy$.

For instance, the predicate IntegrityAct("Visa check service", $\{\}$, false, true, true) specifies one of the integrity annotations in Figure 2. It indicates that software and hardware used to execute Visa check service will be protected from intentional corruption, e.g. unauthorized modifications of the software or hardware robbery.

Non-Repudiation. It is defined as: NonRepudiationAct and NonRepudiationMF. The former indicates that the execution (or non-execution) of activity a should be provable, while the latter specifies that the usage (or non-usage) of the message flow mf should be verifiable. Both the predicates have in common two attributes: $enfBy$, already described before, and $execution$, which specifies if it's required a proof of execution (when it is set to true) or non-execution (when it is set to false) of activity a or message flow mf , in the latter case is required a proof of usage of the communication channel.

For example, the predicate NonRepudiationAct("Blacklist check service", $\{\}$, false) defines one of the non-repudiation annotations in Figure 2. It specifies that it should be possible to prove that Blacklist check service has never been executed. There are no constraints on the security mechanisms that have to be implemented because the parameter is an empty set.

Privacy. It has two variants: (i) privacyACT specifies that activity a should be compliant with privacy legislation, and it should let users to control their own data; (ii) privacyDO is similar to the former one, but is targeted to a specific data object, specified in do . Both predicates share two parameters: $enfBy$, already described before, and $sensitiveInfo$, i.e. the set of sensitive information that must be protected.

For example, consider the predicate PrivacyDO("Personal Info", $\{\}$, {name, surname, dateOfBirth, passportID}), which refines one of the privacy annotations in Figure 2. It specifies that the owner of name, surname, date of birth and passport id information contained in the data object Personal Info should be able to delete the data and, if the information are published, they should be anonymized as required by law, e.g. publish only partial information.

4 Modeling and verifying security policies

We propose the SecBPMN-Q language, an extension of BPMN-Q query language, to model security policies using the security annotations in Table 2. Our query language permits to graphically model security policies, which is a useful feature to support the communication of the specified policies with other stakeholders.

Consider, for example, a textual policy such as "*The visa document must be authenticated and it must be sent through a secure channel which assures the information will not be sniffed or modified by third parties, implementing TLS and X.509 security mechanisms*". Figure 3 models this policy in SecBPMN-Q. Beside the two generic tasks and the path, that are elements of BPMN-Q, the BPMN-Q query is enriched with a message flow (represented as a dashed arrow) which exchange a data object called "Visa".

When executed, this query will match any message flow between two activities which exchange the “Visa” data object. The confidentiality annotation linked to the message flow requires the communication channel to assure the data object will be received only by “Visa owners” and “Control authority”. Moreover, the “Visa” data object has to be protected by unauthorized modifications, implementing the “MD5” security mechanism specified by integrity annotation, and its originality has to be provable using “TLS” and “X.509” security mechanisms, specified in the authenticity annotation. Some optional attributes are not specified, meaning that the security designer is imposing fewer constraints on the specific security mechanism. For example, in Figure 3, *enfBy* and *writers* parameters of ConfidentialityMF are not defined (see the underscore placeholder), hence the predicate will be satisfied, regardless the security mechanisms implemented or the set of users authorized to send data objects through the channel.

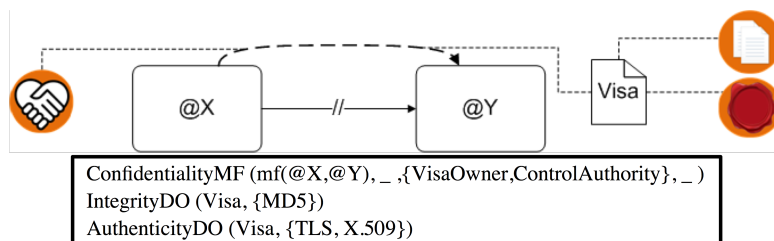


Fig. 3. Example of a security policy and predicates expressed with SecBPMN-Q

In order to verify if the security policies modeled with SecBPMN-Q are satisfied by a SecBPMN-Q business process, we extended the BPMN-Q engine with the implementation of Algorithm 1. The algorithm takes in input a SecBPMN business process and a SecBPMN-Q security policy, and it verifies if there exists a path in the business process that satisfies the security policy. For each path, the algorithm verifies if the security annotations of the business process are of the same type of those in security policy and if they are linked to the same SecBPMN element. If so, the security annotations of the security policy are verified against the security annotations in the business process.

A security annotation of a business process satisfies a security annotation of a security policy if all the attributes of the former are more restrictive of the attributes of the latter. The function *satisfies*, Algorithm 2, checks this property. As first step, Algorithm 2 checks if the security mechanisms specified in the security annotation of the policy are all specified in the security annotation of the business process; if not, it returns false, meaning that the security policy specifies at least a security mechanism that is not implemented in the business process. After that, depending on the type of annotation, the algorithm checks:

- *accountability*, if the monitored users specified in the policy are all monitored by the business process;
- *auditability*, if the frequency of the checks specified in the policy is less or equal than the one specified in the business process;

Algorithm 1 Compliance check of a security policy

```
COMPLIANCE(SecBPMN bp, SecBPMN-Q secPolicy)
1  paths ← FINDPATH(bp, secPolicy)
2  if paths = ∅ then
3    return false
4  for each path ∈ paths do
5    satisfied ← true
6    for each secAnnPolicy ∈ GETSECURITYANNOTATIONS(secPolicy) do
7      for each secAnnPath ∈ GETSECURITYANNOTATIONS(path) do
8        if secAnnPolicy.type = secAnnPath.type then
9          if CHECKTARGET(secAnnPath, secAnnPolicy) then
10             satisfied ← SATISFIES(secAnnPath, secAnnPolicy) ∧ satisfied
11  if satisfied then
12    return true
13 return false
```

Algorithm 2 Pseudo-code of function “satisfies”

```
SATISFIES(SecurityAnnotation SecAnnPath, SecurityAnnotation SecAnnPolicy)
1  if (secAnnPolicy.enfBy ⊆ secAnnPath.enfBy) then
2    return false
3  switch (SecAnnPolicy.type)
4    case AccountabilityAct :
5      return (SecAnnPolicy.monitored ⊆ SecAnnPath.monitored)
6    case AuditabilityAct ∨ AuditabilityDO ∨ AuditabilityMF :
7      return (SecAnnPolicy.frequency ≤ SecAnnPath.frequency)
8    case AuthenticityAct :
9      return ((SecAnnPolicy.ident → SecAnnPath.ident) ∧
10             (SecAnnPolicy.auth → SecAnnPath.auth) ∧
11             (SecAnnPolicy.trustValue ≤ SecAnnPath.trustValue))
12   case AvailabilityAct ∨ AvailabilityDO ∨ AvailabilityMF :
13     return (SecAnnPolicy.value ≤ SecAnnPath.value)
14   case ConfidentialityDO ∨ ConfidentialityMF :
15     return ((SecAnnPolicy.readers ⊆ SecAnnPath.readers) ∧
16             (SecAnnPolicy.writers ⊆ SecAnnPath.writers))
17   case IntegrityAct :
18     return ((SecAnnPolicy.personnel → SecAnnPath.personnel) ∧
19             (SecAnnPolicy.hardware → SecAnnPath.hardware) ∧
20             (SecAnnPolicy.software → SecAnnPath.software))
21   case NonRepudiationAct ∨ NonRepudiationMF :
22     return (SecAnnPolicy.exeution ↔ SecAnnPath.exeution)
23   case privacyAct ∨ privacyMF :
24     return (SecAnnPolicy.sensitiveInfo ⊆ SecAnnPath.sensitiveInfo)
```

- *authenticity*, if *ident* attribute is true in the security annotation specified in the security policy (every user has to be identified) then the same attribute specified in the business process is true. The same criteria is used also for *auth*. The *trustValue* defined in the

security annotation of the security policy has to be less or equal that the value defined in the one specified in the business process, since the security aspects correspondent to the security annotation is satisfied when the trust required is less than the trust offered by the executor of the activity;

- *availability*, if the value specified in the security annotation is less than the value specified in the business process;

- *confidentiality*, if the set of authorized users specified in the security annotation of the security policy is a subset of the authorized users specified in the business process;

- *integrity*, if the personnel attribute (for what concerns IntegrityAct) is true in the security policy then is true in the business process; the same criteria applies for hardware and software. The other two variants of integrity do not need special criteria because they are characterized only by the attribute *enfBy*, that is already checked in the first two lines of the algorithm;

- *non-repudiation*, if the attribute *execution* is the same in both the security annotations, since it specifies two different constraints;

- *privacy*, if the set of sensitive information specified in the security policy is included in the set specified in the business process.

The SecBPMN engine fixes a number of bugged functionalities and comes with a manual which explains the installation of all the required software packages⁵.

When a SecBPMN-Q security policy is checked, the interface of SecBPMN-Q engine presents to the users all the business processes in the repository that have at least one path (graphically highlighted in the business process) that satisfies the security policy specified. Figure 4 shows the result of the SecBPMN-Q query shown in Figure 3 with the SecBPMN-Q process shown in Figure 2. The path highlighted in Figure 4 satisfies the security policy in Figure 3: (i) the first activity of the path, i.e. “Web interface service - inputData”, is linked with a message flow to the last activity of the path, i.e., “Visa check service”; (ii) the message flow is used to exchange the data object “Visa” and it assures confidentiality of the transferred data object; (iii) integrity and authenticity of the “Visa” data object are preserved. Assuming the predicates that details the security annotations of the security policy are less restrictive of the predicates of the business process, the path, and consequently the business process, satisfies the security policy.

5 Discussion

The literature offers a number of graphical modeling languages for expressing security aspects in business process models. These languages support a predefined set of security policies that a designer can use; examples are SecureBPMN [6], other extensions of BPMN e.g. [19, 26, 29, 34], or UML profiles, such as UMLsec [13]. The advantage of these languages is that they are easy to learn and to use [21], thereby requiring a moderately low effort for security designers to specify a secure business process. The price to pay for using these modeling languages is in their limited expressiveness: these graphical modeling languages do not permit to define custom security policies, thereby

⁵ The extended version of the engine and the manual can be found at <http://www.secbpmn.disi.unitn.it>

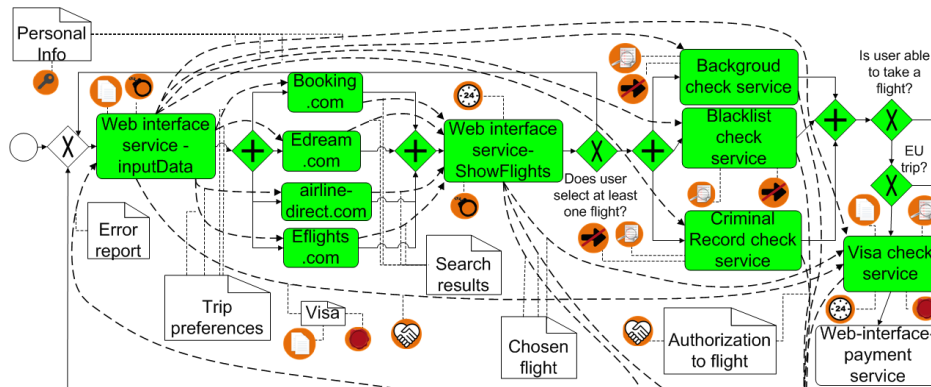


Fig. 4. Result of the query based on SecBPMN-Q policy in Figure 3 against the SecBPMN model in Figure 2

preventing the creation of domain-specific variants. As such, existing verification engines (e.g., [28, 30, 32, 34, 35]) that enable the automated verification of these models do also support a fixed set of hard-coded security policies.

Other graphical languages have been proposed to check the compliance of a process with a query. For example, the Business Process – Query Language (BP-QL) [5] permits to create graphical queries that are checked against processes modeled using the Web Services Business Process Execution Language (WSBPEL) [22]. BP-QL permits to search paths that are compliant with patterns that are defined through the query language proposed; however, BPMN is not used as a basis for the query language. Similarly, the Business Process Query Language (BPQL) [8] permits to graphically define both queries and business process models using the same language. Unfortunately, BPQL is not based on BPMN, hence the learning process is likely to be less quick than that with by BPMN-Q, for this latter language is based on the well-known standard.

Other approaches build on formal languages (e.g., first-order logic, temporal logic, etc.). This trend of work is characterized by high expressiveness. For example, Liu et al. [17] propose a language and a framework which statically verifies a business process against a formally expressed regulatory requirements, while Rushby [27] proposes a language and a framework which checks if the code of a software system diverges from specified behaviors (i.e., policies). The main drawback of these approaches is their low usability, for they require a substantial effort for formalizing business processes and security policies. Moreover, they can hardly be used at runtime, for their verification requires more time, due to the use of a more expressive logics.

We applied SecBPMN and SecBPMN-Q on a case study about a SWIM [2] ATM system, that is part of the Aniketos⁶ European project. The ATM system consists of a large number of autonomous and heterogeneous components, which interact with each other to enable air traffic management operations: pilots, airports personnel, national airspace managers, meteo services, radars, etc. In such a complex information system,

⁶ www.aniketos.eu

ensuring security is critical, for security leaks may result in severe consequences on safety and confidentiality. Experts from the Aniketos project analyzed the security requirement document provided with the case study, and identified 27 active entities and more than 60 security policies. We studied these security policies and modeled them using SecBPMN-Q. After that, we examined the documentation about the case study and we defined four business processes, each containing a number of nodes (activities and gateways) between 28 and 58.

Being based on BPMN, we did not experience particular issues in modeling the processes described in the documentation using SecBPMN. SecBPMN-Q enabled us to model all the security policies elicited by the experts but two cases:

- security policies concerning redundancy, which we could represent only at a high-level of abstraction, without managing to express if the fallback activities have to be performed by the same or a different executor. This limitation was inherited by BPMN-Q, which does not support BPMN swim-lanes and pools. To overcome this limitation, we plan to introduce swim-lanes and pool elements in an extension of SecBPMN/SecBPMN-Q;
- security policies about the non-delegation of an activity, i.e., preventing that third parties execute one activity or parts of it. Even in this case, our future work includes introducing additional elements to the meta-model to support this type of policy.

This preliminary evaluation shows the applicability of the proposed languages for modeling security policies and security-annotated business processes in a non-trivial scenario. However, more extensive evaluation is required for our approach, including experimentation on other domains, assessing the scalability of our algorithms, and checking how well novices and business process experts learn our languages.

6 Conclusions and future work

This paper has introduced SecBPMN and SecBPMN-Q, a modeling language for modeling security-annotated business processes, and a query language for expressing security policies, respectively. Our languages are supported by a toolset that supports both modeling and the execution of queries. Moreover, we have applied our approach on a complex information system for air traffic management.

Our approach overcomes the deficiencies of existing approaches, which either suffer from a limited expressiveness—being graphical languages that support only a predefined set of security annotations—or from limited scalability—being reliant on expressive temporal logics, thereby inhibiting efficient runtime verification.

Our approach is not yet complete, and opens the doors to several future directions: (1) apply the languages to different domains; (2) assess the learnability and usability of our languages; (3) create a catalogue of patterns representing common security policies; (4) determine the scalability of our automated reasoning mechanisms; (5) include our engine in a workflow system to support security policy-compliant runtime reconfiguration.

Acknowledgement

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos) .

References

1. An introduction to the business model for information security. Technical report, ISACA, 2009. <http://www.isaca.org/Knowledge-Center/Research/ResearchDeliverables/Pages/An-Introduction-to-the-Business-Model-for-Information-Security.aspx>.
2. Federal Aviation Administration. SWIM ATM case study, last visited March 2014. http://www.faa.gov/about/office_ org/headquarters_ offices/ato/service_ units/techops/atc_ comms_ services/swim/.
3. A. Awad. BPMN-Q: A language to query business processes. In *EMISA*, volume P-119 of *LNI*, pages 115–128, St. Goar, Germany, 2007. GI.
4. A. Awad. *A compliance management framework for business process models*. PhD thesis, 2010.
5. C. Beerli, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes with BP-QL. *Information Systems*, 33(6):477–507, September 2008.
6. A. D. Brucker, I. Hang, G. Lückemeyer, and R. Ruparel. SecureBPMN: Modeling and Enforcing Access Control Requirements in Business Processes. In *Proc. of SACMAT'12*, pages 123–126.
7. Y. Cherdantseva and J. Hilton. A reference model of information assurance and security. In *Eighth International Conference on ARES*, pages 546–555, Sept 2013.
8. D. Deutch and T. Milo. Querying structural and behavioral properties of business processes. In *Database Programming Languages*, volume 4797 of *LNCS*, pages 169–185. 2007.
9. D.F. Ferraiolo, J.A. Cugini, and D.R. Richard Kuhn. Role-based access control (rbac): Features and motivations. 1995.
10. D. Firesmith. Specifying reusable security requirements. *JOT*, 3(1):61–75, 2004.
11. A. Ghose and G. Koliadis. Auditing business process compliance. In *Proc. of the 5th ISOC, IC SOC '07*, pages 169–180. Springer-Verlag, 2007.
12. A. Josang, R. Ismail, and C. Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618 – 644, 2007.
13. J. Jurjens. Umlsec: Extending uml for secure systems development. In *In Proc. of 5th UML, UML '02*, pages 412–425, London, UK, 2002. Springer-Verlag.
14. M. Leitner, M. Miller, and S. Rinderle-Ma. An analysis and evaluation of security aspects in the business process model and notation. In *Proc. of ARES*, pages 262–267, 2013.
15. M. Leitner and S. Rinderle-Ma. A systematic review on security in process-aware information systems - constitution, challenges, and future directions. *Inf. Softw. Technol.*, 56(3):273–293, 2014.
16. M. Leitner, S. Schefer-Wenzl, S. Rinderle-Ma, and M. Strembeck. An experimental study on the design and modeling of security concepts in business processes. In *Proc. of PoEM*, pages 236–250, 2013.
17. Y. Liu, S. Müller, and K. Xu. A static compliance-checking framework for business process models. *IBM Syst. J.*, 46(2):335–361, April 2007.
18. J. McCumber. Information systems security: A comprehensive model. *Proceeding of the 14th National Computer Security Conference, NIST Baltimore, MD*, 1991.
19. M. Menzel, I. Thomas, and C. Meinel. Security requirements specification in serviceoriented business process management. In *Proc of ARES '09*, pages 41–48.

20. G. Monakova, A. D. Brucker, and A. Schaad. Security and safety of assets in business processes. In *Applied Computing* 27, pages 1667–1673, USA, 2012. ACM.
21. D. Moody. The physics of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Softw. Eng.*, 35:756–779, 2009.
22. OASIS. Web Services Business Process Execution Language. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>, Apr 2007.
23. OMG. BPMN 2.0. <http://www.omg.org/spec/BPMN/2.0>, Jan 2011.
24. D. Parker. Our excessively simplistic information security model and how to fix it. *ISSA Journal*, pages 12–21, 2010.
25. D. B. Parker. *Fighting computer crime - a new framework for protecting information*. Wiley, 1998.
26. A. Rodríguez, E. Fernández-Medina, and M. Piattini. A BPMN extension for the modeling of security requirements in business processes. *IEICE Trans. on Information and Systems*, 90(4):745–752, 2007.
27. J. Rushby. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, 75:167–177, 2002.
28. S. Sadiq, G. Governatori, and K. Namiri. Modeling control objectives for business process compliance. In *Proc. of the 5th BPM, BPM'07*, pages 149–164, 2007.
29. M. Saleem, J. Jaafar, and M. Hassan. A domain-specific language for modelling security objectives in a business process models of soa applications. *AISS*, 4(1):353–362, 2012.
30. M. Salnitri, F. Dalpiaz, and P. Giorgini. Aligning service-oriented architectures with security requirements. In *proc. of OTM'12*, pages 232–249, 2012.
31. P. Samarati and S. Vimercati. Access control: Policies, models, and mechanisms. In *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 137–196. 2001.
32. R. Schmidt, C. Bartsch, and R. Oberhauser. Ontology-based representation of compliance requirements for service processes. In *Proc. of CEUR '07*.
33. I. Sommerville, D. Cliff, R. Calinescu, J. Keen, T. Kelly, M. Kwiatkowska, J. Mcdermid, and R. Paige. Large-scale complex it systems. *Commun. ACM*, 55(7):71–77, 2012.
34. C. Wolter, M. Menzel, A. Schaad, P. Miseldine, and C. Meinel. Model-driven business process security requirement specification. *JSA*, 55(4):211 – 223, 2009.
35. F. Yip, A.K.Y. Wong, N. Parameswaran, and P. Ray. Rules and ontology in compliance management. In *In Proc. of EDOC*, pages 435–435, 2007.