STS-Tool 3.0: Maintaining Security in Socio-Technical Systems

Mattia Salnitri, Elda Paja, Mauro Poggianella, and Paolo Giorgini

University of Trento, Trento, Italy
{mattia.salnitri, elda.paja, poggianella, paolo.giorgini}@unitn.it

Abstract. In this paper, we present STS-Tool 3.0: a software tool that helps security requirement engineers in maintaining high level of security in socio-technical systems. STS-Tool 3.0 allows to specify social/organizational security requirements and to enforce them in part of the implementation of socio-technical systems.

1 Introduction

Socio-technical systems are an interplay of social (human and organizations) and technical subsystems, which interact with one another to reach their objectives. Examples of socio-technical systems are smart cities, air traffic management systems and payment systems, i.e. systems where banks, payment services and e-shops interact to transfer monetary values.

Subsystems in socio-technical systems are autonomous, heterogeneous and weakly controllable, but they highly depend on each other. For example, in the payment system the banks and payment services are autonomous but without interacting with each other, both of the systems will not be able to transfer monetary values.

In such interconnected environment is central maintaining a high level of security: a security issue of one subsystem may cause a chain reaction and impact many other subsystems. For example, in a payment socio-technical system, a security issue on the information integrity of the payment orders in one of the banks might compromise the entire system: all the banks and payment services that use the compromised bank may have received payment orders containing unauthorized modifications.

Therefore, it is essential maintaining the level of security specified by the stakeholders, i.e. being sure that security requirements are enforced in socio-technical systems' implementation.

In previous work [10], we have proposed a framework for specifying security requirements, checking their enforcement in business processes executed in such systems and generating a part of the implementation. Specifically, the framework permits to: (i) specify social-organizational security requirements; (ii) generate, from social-organizational models, business processes, i.e., specifications of how subsystems operate and interact; (iii) generate procedural security policies from social-organizational security requirements; (iv) verify security policies against business processes; (v) generate part of the implementation. We provide methodological guidance for each and every activity. The detailed process and phases are presented in [11].

In this paper we illustrate STS-Tool 3.0, the tool that supports the framework proposed in [10, 11]. Specifically, the paper describes how STS-Tool 3.0 supports each phase and the salient new features of STS-Tool 3.0.

The paper is structured as follows. Section 2 introduces the architecture and the implementation of STS-Tool 3.0, while Section 3 describes the content of the demonstration. Section 4 contains the related work and Section 5 concludes.

2 STS-Tool 3.0: architecture and implementation

STS-Tool 3.0 has the ambitious objective of supporting the generation of the implementation of socio-technical systems that enforces of social/organizational security requirements.

Such security requirements are based on high-level of abstraction concepts, such as goals/objectives of actors and delegation of goals/objectives, but, due to the conceptual gap between these concepts and the implementation, it is not possible to directly generate the code that enforces the security requirements.

STS-Tool 3.0 uses business process models to bridge the conceptual gap. It generates business processes, from social/organizational models, that are refined by security requirement engineers and then it transforms such business processes in the implementation. Specifically it uses using STS-ml [4, 7], a goal-based modeling language, for social/organizational models, and SecBPMN2-ml [9], a modeling language for business processes and security, for business process models.

The generation of a secure implementation of socio-technical systems needs business processes that enforce the social/organizational security requirements. STS-Tool 3.0 verifies the enforcement of security requirements by transforming them in security policies, i.e., security requirements in terms of business process elements, and it verifies such policies against the business processes.

STS-Tool 3.0 extends version 2.0, with a number of features, the most relevant are specified below:

- generation of business processes from the social/organizational perspective;
- generation of procedural security policies from social/organizational security requirements;
- verification of business processes against procedural security policies;
- generation of part of implementation of socio-technical system.

Figure 1 shows the architecture of STS-Tool 3.0. The components are divided in three parts: *Model editors* allow the visualization and the modification of models; *Model transformers* are used to transform models; *Automated reasoners* are used to check the consistency of the models and check the enforcement of business process against security policies.

STS-ml editor and *STS-ml compliance verifier* components are part of STS-Tool 2.0. They facilitate the creation of STS-ml models and the automated verification of their coherency. The components added in STS-Tool 3.0, highlighted with a thick border in Fig. 1, are described below.

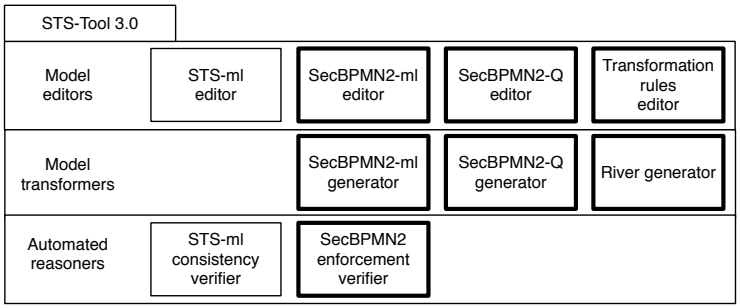

Fig. 1. Components of STS-Tool 3.0.

SecBPMN2-ml editor It permits to specify and to modify business processes with security aspects, using SecBPMN2-ml. This components extends BPMN2 Modeler[1]: an Eclipse[2] plug in for drawing BPMN 2.0 diagrams

SecBPMN2-Q editor It permits to model security policies using SecBPMN2-Q. This component, like *SecBPMN2-ml editor*, extends BPMN2 Modeler.

SecBPMN2-ml generator It automatically generates SecBPMN2-ml diagrams from STS-ml diagrams. SecBPMN2-ml diagrams are generated already filled with SecBPMN2-ml elements that can be derived from STS-ml diagram.

SecBPMN2-Q generator It generates SecBPMN2-Q security policies from security requirements specified in STS-ml diagram.

Transformation rules editor This component permits to visualize and modify transformation rules, i.e. set of rules used to generate the SecBPMN2-ml and SecBPMN2-Q models from, respectively, STS-ml models and the list of social/organizational security requirements.

SecBPMN2 enforcement verifier It is a verification engine that verifies if SecBPMN2-Q security policies are enforced in SecBPMN2-ml business processes. It generates a list of SecBPMN2-ml elements that do not satisfy the security policy and a list of STS-ml elements that are connected to the business process that does not satisfy the security policy. These elements are highlighted in the editors to facilitate their identification.

River generator It generates part of socio-technical system implementations. The component uses River [14], a scripting language for specifying business artifacts and the business logic associated to such artifacts. The generated code implements the management of the data used in the business processes. STS-Tool 3.0 does not generate the functional part of the implementation, i.e. the part of the implementation that executes the actions modelled by the activities of the business processes. The information contained in the strings which identify each activity, is not enough to generate the implementation that execute the activity. For further details, see [8].

[1] https://www.eclipse.org/bpmn2-modeler/
[2] https://www.eclipse.org

2.1 SecBPMN2 enforcement verifier component

The *SecBPMN2 enforcement verifier*, described in this subsection, is the core part of STS-Tool 3.0. We used $DLV^{\mathcal{K}}$ [5] for verifying SecBPMN2-ml business processes against SecBPMN2-Q security policies. $DLV^{\mathcal{K}}$ is a planner: a software engine, based on DLV [6], that generates plans, i.e., sequences of actions, that achieve a set of goals and do not violate given constraints. We transformed business processes in constraints and security policies in goals. Therefore, if $DLV^{\mathcal{K}}$ generates a plan, the security policies are satisfied against the business processes. For further details see [12].

Figure 2 shows the architecture of the component: grey boxes are software components, white boxes are input/output artifacts, while dashed arrows connect the inputs/outputs of the component to internal components.

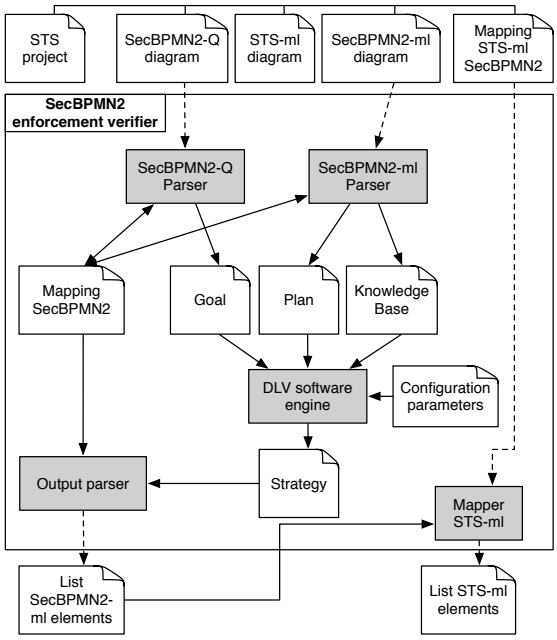

Fig. 2. Architecture of SecBPMN2 enforcement verifier.

SecBPMN2 enforcement verifier receives in input *STS project* that is composed of: *STS-ml diagram*, *SecBPMN2-ml diagram*, *SecBPMN2-Q diagram* and *Mapping STS-ml-SecBPMN2*. The latter part of the input contains the links between STS-ml elements and SecBPMN2-ml elements. It is created during the generation of SecBPMN2-ml diagrams from STS-ml diagrams. *SecBPMN2-ml parser* receives in input a *SecBPMN2-ml diagram* and it creates a *Plan* and the *Knowledge base*, i.e., the list of elements of the business process. Similarly *SecBPMN2-Q parser* component receives in input a *SecBPMN2-Q diagrams* and it creates a set of *Goals*. The *DLV software engine* has

strict limitations on the names of the variables and constants, therefore, the *SecBPMN2-ml parser* and *SecBPMN2-Q parser* generate names that are compliant with the DLV limitations and store in a file called *Map IDs SecBPMN2* the relations between the name of activities, in SecBPMN2-ml diagrams, and the generated names. The *DLV software engine* checks the *Goal* against the *Plan* using the *Knowledge based* and it generates, if possible, a plan, i.e. a list of actions. *Output parser* uses the *MapIDs SecBPMN2* to generate, from the plan, a sequence of SecBPMN2-ml elements that, if executed, will satisfy the security policies. Finally, *Mapper STS-ml* uses the list and *Mapping STS-ml-SecBPMN2* to generate the list of *STS-ml elements*.

3 Demonstration content

We illustrate STS-Tool 3.0 with the payment engine example (Example 1) following the phases of the process shown in Fig. 3.

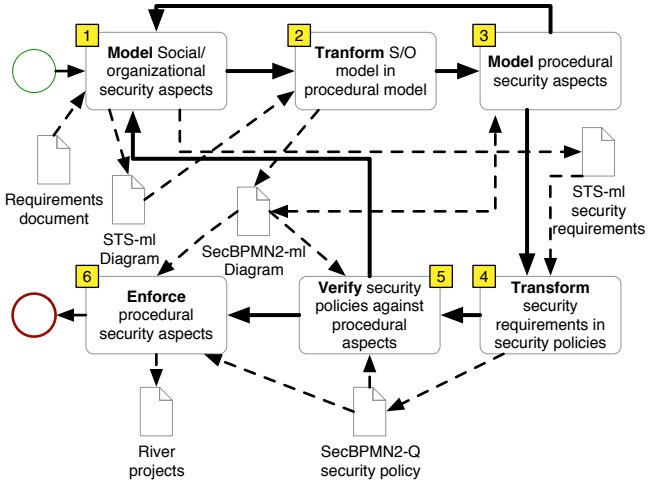

Fig. 3. Methodological process from [11].

Example 1 (Payment Engine). SAP Payment Engine (PE) [13] is a software system created to perform payments for e-commerce shops or, more in general, for services that allow users to pay with electronic methods. Usually, to support the electronic payments, the e-shops implement interfaces to communicate with all the banks they intend to use as source/destination of the payments. But each bank requires a different set of protocols and security measures. Therefore the e-shops are forced to put a noticeable amount of effort to implement different interfaces, and for medium/small companies it is not acceptable investing large quantity of time and money just to allow people to pay the goods/services they offer. The PE minimizes such effort: it contains a set of interfaces with the most known banks in the word that can be used out of the shelf. E-shop

programmers only have to create one interface to transmit the required data to the PE system. The payment system is a socio-technical system since it involves customers, banks, the PE, etc., which interact with each other to perform payments. □

Phase 1 It supports the modeling of socio/organizational aspects of the socio-technical system under consideration. Figure 4 shows a screen shot of STS-Tool 3.0 with an STS-ml diagram in the upper part. Such modelling language focuses on representing the main stakeholders (e.g. *PE* and *Bank* in Fig. 4), their objectives, as well as their interactions (e.g. Fig. 4 the objective *Transfer performed* is delegated from the *PE* to *Bank*). Most importantly, it captures security requirements. For example, in Fig. 4 *Auth* (authorization) and *Ava* (availability) security requirements are specified.
Phase 2 It deals with the transformation of social/organizational models to business processes, receiving in input a social/organizational diagram and generating a procedural diagram of the system-to-be. STS-Tool 3.0 uses SecBPMN2-ml as modelling language for business process with security aspects. It permits to model participants (e.g., *PE* and *Bank* in SecBPMN2-ml diagram shown in Fig. 4) that execute activities (e.g., *Perform transfer* in Fig. 4) and exchange messages (e.g. *Transfer order* in Fig. 4). Such modeling language permits to specify security concepts using security annotations, such as the orange solid circle in Fig. 4 that represents confidentiality of the message transmitted in the communication between the two participants.
Phase 3 Social/organizational models do not contain the information required to generate complete SecBPMN2-ml models. Therefore, this phase is executed to enrich the SecBPMN2-ml models generated by phase 2 with details such as the temporal aspects and security choices on the executed processes.
Phases 1-3 are repeated until security requirement engineers decide they have captured all important (security) details and the procedural model is complete and accurate.
Phase 4 It generates procedural security policies from social/organizational security requirements. This phase permits to translate security requirements, defined in terms of social/organizational concepts, in more operational constraints, as the ones specified in security policies. STS-Tool 3.0 uses SecBPMN2-Q as modeling language for security policies. Figure 4 shows an example of SecBPMN2-Q diagram. Such modelling language extends SecBPMN2-ml, since it permits to use relations such as path, represented with a dashed arrow in Fig. 4.
Phase 5 It deals with the verification of procedural security policies, generated in phase 4, against the procedural model, enriched in phase 3. This phase validates SecBPMN2-ml models by verifying if they enforce the security requirements specified in the social/organizational model. If all security policies are enforced, then the security requirement engineers have the proof that the procedural model meets the security requirements. If the procedural model does not satisfy all procedural security policies, the process starts from the beginning.
Phase 6 It consists in generating part of the implementation from the enriched and verified procedural model. The resulting application will enforce the social/organizational security requirements, because social/organizational security requirements define the security policies that are verified against the procedural model. Therefore, because the transformation enforces all the security aspects defined in the procedural model, the implementation can be considered secure. For further details, see [8].

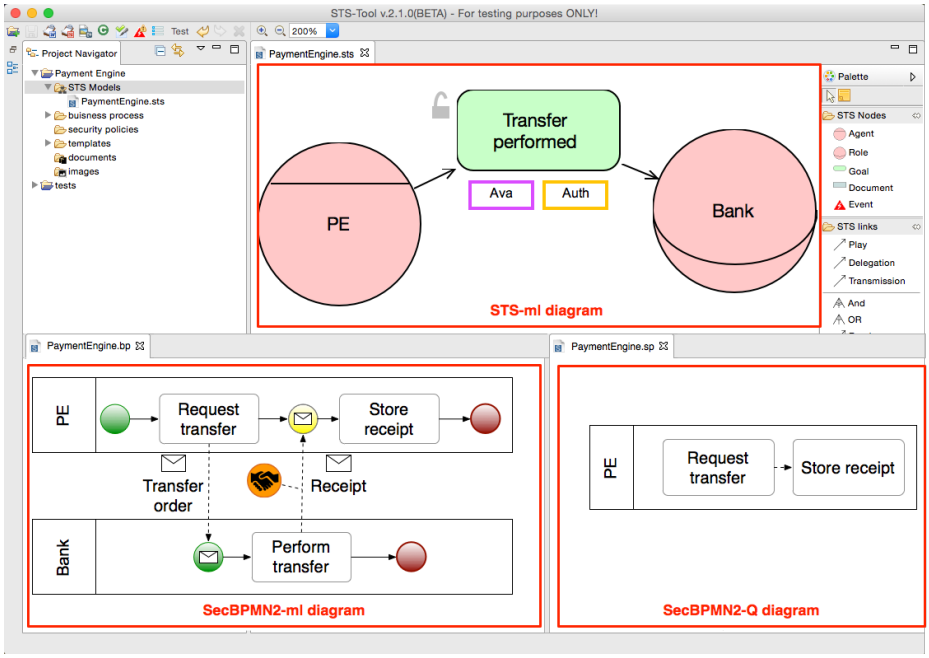

Fig. 4. Screenshot of STS-Tool 3.0.

4 Related work

A number of software tools, e.g., [17, 1, 2], can be used by security requirement engineers for specifying or for designing part of secure socio-technical systems. For example Secure Tropos [17] and STS-Tool 2.0 [16] are focused on specifying security requirements at social/organizational level. While other software tools, such as Adonis [1] or SNP-BPA [15] are focused on the analysis of business processes. For what concerns the implementation of business processes, Altova [3] and Alfresco [2] permit to define and execute business processes executed both by technical components and humans. But they do not generate part of the implementation: they, instead, call a service for each activity that is not executed by humans.

As far as our knowledge goes, no software tool assists security requirement engineers in enforcing security requirements from their early specification until the implementation in socio-technical systems.

5 Conclusion and future work

STS-Tool 3.0 is ready for public use. This version of the tool is the result of an iterative development process, having been tested on multiple case studies and evaluated with practitioners [11]. It has proven suitable to model and reason over models of a large size from different domains [11, 12], such as Air Traffic Management Control

and Telecommunications. Future work about STS-Tool 3.0 includes: (i) integration of other languages for the generation of the implementation of socio-technical systems; (ii) implementing a plug in management system that allows for adding functionalities to STS-Tool 3.0.

Acknowledgement

This research was partially supported by the ERC advanced grant 267856, 'Lucretius: Foundations for Software Evolution', `www.lucretius.eu`.

References

1. Adonis Website. Last visited May '15. http://www.boc-group.com/it/products/adonis/.
2. Alfresco Website. Last visited May '15. http://www.alfresco.com.
3. Altova Website. Last visited May '15. http://www.altova.com.
4. F. Dalpiaz, E. Paja, and P. Giorgini. Security Requirements Engineering via Commitments. In *STAST'11*, pages 1–8.
5. T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres. Planning under incomplete knowledge. In *CL '00*, pages 807–821.
6. N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, July 2006.
7. E. Paja, F. Dalpiaz, and P. Giorgini. Managing Security Requirements Conflicts in Socio-Technical Systems. In *ER '13*, pages 270–283.
8. M. Salnitri, A. Brucker, and P. Giorgini. From Secure Business Process Models to Secure Artifact-Centric Specifications.
9. M. Salnitri, F. Dalpiaz, and P. Giorgini. Modeling and Verifying Security Policies in Business Processes. *BPMDS '14*, pages 200–214.
10. M. Salnitri, E. Paja, and P. Giorgini. Preserving compliance with security requirements in socio-technical systems. In *Proc. of CSP*, pages 49–61, 2014.
11. M. Salnitri, E. Paja, and P. Giorgini. From socio-technical requirements to technical security design: an sts-based framework. Technical report, DISI - University of Trento, 2015.
12. M. Salnitri, E. Paja, and P. Giorgini. Maintaining secure business processes in light of socio-technical systems' evolution. Technical report, DISI - University of Trento, 2015.
13. SAP Payment Engine Website. Last visited March '15. www.sap.com/ services-support/svc/custom-app-development/cnsltg/prebuilt/payment-engine/index.html.
14. SAP SE. *SAP River Developer Guide*, 2014. Document Version 1.0 – 2014-08-21, SAP HANA SPS 08, revision 82.
15. SNP-BPA Website. Last visited May '15. http://www.snp-bpa.com.
16. STS-Tool Website. Last visited May '15. http://www.sts-tool.eu.
17. Tropos Website. Last visited May '15. http://www.troposproject.org.