Towards an Integrated Framework for Model-driven Security Engineering

Jordi Cabot[1,2] and Nicola Zannone[1]

[1] University of Toronto (Canada)
[2] Open University of Catalonia (Spain)
{jcabot,zannone}@cs.toronto.edu

Abstract. Security is a major issue in developing software systems. It is widely recognized that security aspects must be considered in all the phases of the development process from the analysis of the organizational context to the final implementation of the software system. However, current approaches for designing secure systems only target particular security aspects at specific stages of the development process. A unified process combining these different approaches is still missing. This paper surveys several existing techniques and discuss the need of a general framework for integrating them into a single development process.

1 Introduction

Security incidents are a main concern for every organization. They compromise business continuity as well as the trust that customers feel towards the organization. Organizations should adopt the necessary measures to protect their businesses and software systems. However, the design of secure software systems is challenging because of the complexity of modern applications and the number of security facets to be considered.

In recent years, Model Driven Development (MDD) is gaining the attention of both industry and research communities due to its promise to increase productivity in developing, documenting, and maintaining IT systems. MDD emphasizes the use of models during the whole development process and provides automation through model execution, model transformation, and code generation techniques. Therefore, MDD seems an appropriate parading on top of which base a software development process focused on the design of software systems.

Typical MDD approaches (e.g., [18]) and, in general, all software processes [23], start the development process with the modeling and analysis of system requirements and lack of specific techniques and methods for the specification of the organizational context where the system-to-be will operate. Unfortunately, the analysis of security incidents has demonstrated that security is often compromised by exploiting vulnerabilities in the organization structure and security policies adopted by the organization itself, and thus, the analysis of this organizational context must be part of the development process.

The organizational setting has been addressed by early requirements engineering approaches [3, 6, 13, 29]. These approaches provide facilities to represent and understand the relationships between the software system and its organization. However, they do not support the rest of the development process and usually employ modeling languages that are not largely adopted in industry (e.g. specific languages for drawing goal models instead of using UML or related languages as done by all current MDD methods).

Existing research efforts have addressed the problem of designing secure systems by either extending traditional MDD approaches [2, 7, 14, 26] or early requirements engineering approaches [9, 15, 17, 27]. However, existing proposals target only specific security aspects at particular phases of the software development process.

What is still missing is a general framework that considers security throughout the whole software development process. We believe that such integrated framework would facilitate the development of secure systems by providing designers with guidelines to integrate security aspects in all the phases of the development process. This is even more important in the context of MDD where the final software system implementation is (semi)automatically generated from high-level models. In this paper, we provide an overview of existing proposals and discuss the challenges of creating a MDD framework for addressing the development of secure systems.

The paper is organized as follows. We first describe the main features and phases we envision for this framework (Section 2). Then we survey the available proposals for each phase (Sections 3 to 6). Research challenges to achieve the integrated framework are described in Section 7. We end up describing some conclusions and further work.

2 An Integrated Framework for Developing Secure Software Systems

We envision an integrated framework able to capture, model, and analyze all security aspects of the system, where each aspect is addressed in the most appropriate development phase of the process. We believe that, in order to be successful, such framework should present at least the following characteristics:

– It shall consider security aspects in all the phases of the development process.
– It shall emphasize the modeling of the organizational context of the system-to-be as a preliminary step before eliciting the system requirements.
– It shall follow an MDD-based approach.

To fulfill these features, we propose to follow a development process consisting of four main phases (see Fig. 1):

1. *Organizational Context* concerns the understanding of the application domain, including security aspects, by analyzing the organizational setting where the system operates.

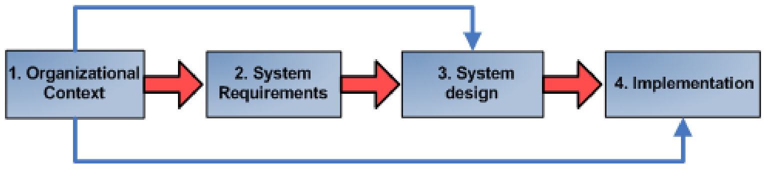

Fig. 1. A unified software process for security development

2. *System Requirements* concerns the analysis of the system-to-be within its operational environment. During this phase, the focus is on the interface between organization procedures and the system, besides the analysis of the system itself.
3. *System Design* completes the previous system specification (e.g., adding component and deployment details) and adapts it to the characteristics of the technology platform where the system will be implemented.
4. *Implementation* (automatically) generates the software system from the design models.

The last three phases of our process can be assimilated to the common software development practices proposed in the literature (see [23] for a survey) to drive designers in the development of software systems. Our framework complements these phases with an initial phase devoted to the analysis of the organizational context in which the system will eventually operate.

Additionally, in this framework, the transition between the different phases is expected to be as automatically as possible, that is, models of a phase p should be (partially) generated from models in phase $p - 1$. Moreover, the analysis of the organizational models influences all subsequent steps of the development process. Not only they are used to elicit system requirements but also to drive how the system is designed and implemented. Since the organizational models capture the goals and intentions of the stakeholders interested in the system, it is important to also consider those goals when making the design and implementation decisions to ensure the system is aligned with them and, this way, improve the stakeholder's satisfaction with the system.

Next sections present the above phases, indicating and discussing the existing security proposals that fit in each phase.

3 Organization Context

MDD approaches focus on the system to be developed, ignoring the analysis of the organizational context in which the system will eventually operate. Instead, in the Requirements Engineering community is generally accepted that understanding the purpose, goals, and intentions of a system within its organizational operational environment is a necessary condition for its successful design and implementation [20]. This is even more important when the system has to meet security requirements because security is often compromised at organizational level, rather than at technical level [1].

Therefore, it is clear that as a first step of our integrated process we need to start by analyzing the organizational environment. A number of requirements engineering frameworks have been proposed to elicit organizational requirements and derive system requirements from them [3, 6, 29]. These frameworks have been adapted to model security aspects of organizations and their IT systems [9, 15, 17, 27]. In what follows we describe the security aspects that these methods permit to model and analyze.

Van Lamswerde extends KAOS [6], a goal-oriented requirements engineering methodology, to address security issues by introducing the notions of obstacle to capture exceptional behaviors [28] and anti-goals to model intentional obstacles set up by attackers to affect security goals [27]. Anti-goals are defined as the negation of security goals such as confidentiality, availability, and privacy and represent the goals of attackers. Anti-goals are refined to form an attack tree on the basis of attackers capabilities as well as software vulnerabilities. Security requirements are defined as the countermeasures to software vulnerabilities or anti-requirements, that is, anti-goals that are realizable by some attacker.

Along the same line, Liu et al. [15] refine the i* modeling framework [29] by analyzing attackers, dependency vulnerabilities among actors and possible countermeasures. All actors are assumed to be potential attackers who inherit capabilities, intentions, and social relationships from the corresponding legitimate actor. Dependencies between these actors can bring vulnerabilities to the system. In particular, dependency analysis is used to identify the vulnerable points in the dependency network. During countermeasure analysis, designers investigate how to protect the system from attackers and vulnerabilities. Elahi et al. [9] extend this work by focusing on how attackers can compromise the system by exploiting vulnerabilities that software components and organizational procedures bring to the system, and on which countermeasures can be adopted to protect the system. The concept of dependency analysis is also proposed in [19] where secure dependencies are defined as a specialization of the security constraint mechanism, which is a restriction related to security issues that can influence the design of the system by restricting some alternative solutions.

A complementary approach is Secure Tropos [10], a requirements engineering methodology supporting system designers in the modeling and analysis of security and privacy facets of systems and their organizational settings. Secure Tropos adopts SI* [17] as modeling framework, which enhances i* with concepts specific to security such as ownership, permission, and trust. The methodology provides a requirements analysis process driving system designers from the elicitation of authorization, availability, and privacy requirements up to their verification. The methodology supports designers in understanding the causes of system vulnerabilities and provides facilities to deal with them by driving designers in revisiting requirements models by either reconstructing the organizational setting of the system or adopting protection mechanisms through the use of security patterns.

These frameworks, when combined, are rich enough to express most of the security aspects that may be relevant at the organizational level. Nevertheless, there does not exist yet a general methodology that proposes how to integrate them in a single organizational specification. This is not a trivial challenge since they use different notations and are not completely orthogonal (i.e., some present overlapping constructs that must be merged and checked for consistency).

4 System Requirements

Once the analysis of the organizational context has been completed, we can start focusing on the specific system requirements. Among the security approaches targeting the system requirements phase, Haley et al. [11] propose abuse frames to determine the impact of security requirements on functional requirements. Abuse frames extend problem frames [13] by considering threats as crosscutting concerns to determine adequate security requirements for the system. Functional requirements describe how assets (i.e., objects to be protected) are used within the system. Threats describe how attackers can exploit vulnerabilities to compromise the security of assets. Security requirements are thus defined as constraints on functional requirements or trust assumptions and are intended to reduce the scope of vulnerabilities. Once security requirements are elicited, the framework uses satisfaction arguments to validate security requirements.

Sindre and Opdahl [26] extend use cases to model security requirements and call this extension *misuse cases*. Misuse cases describe functions that the system should not allow. They are depicted as black ovals to distinguish them from traditional use cases. Misuse cases can be linked to use cases to indicate that the use case is exploited by the misuse case, and use cases to misuse cases to indicate that the use case is a countermeasure against the misuse case. The visualization of these links helps in organizing the requirements specification and in tracing the security requirements to threats that motivated them. Together with misuse cases, the authors introduce the concept of *misuser*, which represent the actor that initiates misuse cases.

The CORAS project [7] proposes a framework for model-based risk assessment. In particular, CORAS provides a UML profile for risk assessment and an integrated risk management and system development framework based on the Unified Process. The profile defines UML stereotypes and rules for specialized UML diagrams. Specifically, the framework makes use of five diagrams: SWOT diagrams, asset diagrams, threat diagrams, state analysis diagrams, and treatment diagrams. SWOT diagrams represent high-level strengths, weaknesses, opportunities and treats and relate them to stakeholders and assets. Asset diagrams are specialized class diagrams where assets are grouped in themes (i.e, human, physical, information, organizational, law and regulation, and software assets) related using standard UML associations. Assets are also related to stakeholders to indicate that the stakeholder owns the asset. Threat and treatment diagrams are based on misuses cases and are used to represent the threats that reduce the value of assets and the treatments to be adopted to prevent such threats. State analysis diagrams are specialized statechart diagrams that specify the (un)desired behavior of the system.

Ideally, a preliminary version of all system requirements models used by these methods should be automatically derived from the analysis of the organizational context. This makes possible to understand *why* the system functionality and protection mechanisms are necessary, besides of how and what. The main problem for providing such transformation is the shift in the modeling language employed. For specifying the organizational context, goal models (written using the i* notation or similar) are typically used, while the system requirements are usually depicted as a set of UML-based diagrams. Although preliminary approaches targeting this transformation have been proposed (e.g. [16, 25]), they do not deal with security aspects (i.e., security extensions to the i* or Tropos models are not transformed into security extensions to UML models). The only work addressing the transformation of security organization and system models in UML is the one in [19], but it presents only some basic high level guidelines rather than a complete mapping.

5 System Design

In this phase, the previous models have to be refined by adding the more low-level details (as deployment and componentization details) needed for the posterior system implementation.

In this sense, Jürjens [14] proposes UMLsec, an extension of UML, for representing security aspects of IT systems such as fair exchange, confidentiality, secure information flaw and secure communications link. UMLsec is a UML profile in which security requirements are represented in form of stereotypes, tagged values, and constraints that can be associated with model elements of activity diagrams, statecharts, sequence diagrams, static structure diagrams, deployment diagrams, and subsystems. For instance, stereotype $\langle\langle secure\ link\rangle\rangle$ is used to ensure that security requirements on the communication are met by the physical layer, and stereotype $\langle\langle no\ flow - down\rangle\rangle$ denotes that data object cannot leak out any information about secret data via non-secret data. Stereotypes are also used to indicate implementation decisions. For instance, stereotypes $\langle\langle Internet\rangle\rangle$, $\langle\langle encrypted\rangle\rangle$, and $\langle\langle LAN\rangle\rangle$ are used on links in deployment diagrams to specify the type of communication link.

Basin et al. [2] propose SecureUML, an UML-based modeling language focusing on the modeling Role-Based Access Control (RBAC) policies and integrating them into a model-driven software development process. In particular, SecureUML is a UML profile that defines RBAC concepts using stereotypes and tagged values that are associated with model elements of class diagrams. Similar approaches have been proposed by Doan et al. [8], who incorporate Mandatory Access Control (MAC) into UML, and by Ray et al. [24], who model RBAC as a pattern using UML diagram template.

Breu et al. [4] propose an approach for the specification of user rights in the context of an object oriented use case driven development process. An informal description of actor permissions is specified in textual way and has the form of an access control list. This informal description of the user rights model is adapted to the actors of use case diagrams. The model is further refined into a complete formal model by considering the system behavior.

All these design security models should be consistent with the previous requirements models (e.g. the RBAC policies for the set of classes implementing a given functionality should take into account the actors with permission to execute that functionality as defined in the system requirements models) and, partially, be automatically generated from them. This problem has not been addressed so far. Another research challenge at this stage of the process is how to change/adapt these models depending on the technical features offered by the technological platform where the system is going to be implemented (e.g., benefiting from advanced security features offered by the platform to simplify the mapping between the models and the final implementation).

6 System Implementation

Secure design models are a necessary but not a sufficient condition to achieve a secure system implementation. Security risks can also be inadvertently introduced during the programming phase. Therefore, secure implementations must consider two different aspects:

1. Secure programming techniques to avoid security vulnerabilities caused by programmer error, as buffer overflows and poor memory management.
2. Guidelines for (automatically) generating secure code according to the security design decisions expressed in the system design models.

For the first aspect, there are several libraries, code samples and programming recommendations that facilitate programmers to avoid these risks (e.g., [12]) depending on the specific technology/language used to implement the system.

On the contrary, the second issue is still an open research problem. It requires to define, for each security modeling primitive, a model-to-code transformation in charge of generating the code excerpt enforcing that particular security aspect. As an example, *secrecy* and *integrity* links in UMLSec deployment diagrams [14] could be guaranteed by means of implementing a public key infrastructure scheme to encrypt the communication between the parties involved in the communication. So far, only few methods provide some kind of code-generation facilities. One of the few exceptions is Breu et al. [4] that propose a model transformation procedure for generating XACML policies [21] from user right models. Similarly, [2, 5] generate the system's access control infrastructure from SecureUML models. However, code-generating techniques addressing other security aspects (and combinations of them) still need to be developed.

7 Research Challenges

Despite the potential benefits of the integrated framework proposed in this paper, there are a number of research challenges that must be solved before making it possible. Some of them have already been introduced in the previous sections.

Clearly, the success of the framework largely depends on our ability to smoothly integrate the different security proposals. Ideally, designers following our integrated process should be unaware that they are using a combination of originally different proposals. To reach this goal we need to address both horizontal and vertical integration of the security techniques our framework consists of:

– *Horizontal integration.* At each phase of the process we have several security proposals available. Though complementary, they are not fully orthogonal so before designers can use them, we need to merge their common aspects and precise which alternative technique will be used to model each specific security aspect (and in which order they should be applied). To maximize the benefits of the framework we also need to study the additional advantages resulting from combining the techniques.
– *Vertical integration.* Security information defined in earlier phases in the process must be carried on to the later phases. This must be done automatically, that is, to avoid redundancies, designers should only define each security aspect once. Therefore, we need to develop a set of model-to-model transformations that transition security concerns between models at different stages of the process. This automatic transition has the additional benefits of minimizing the possibility of errors and ensuring the consistency.

Regarding the vertical integration, the two most challenging transitions are the elicitation of system requirements (functional and non-functional) from the organizational context and the adaptation of design models to the technical features offered by the implementation platform.

For the first one, we have that popular requirements engineering methods as i*/Tropos and their security extensions use the notion of agent and all related mentalistic notions to capture and model system requirements. In particular, goal models defining the organizational context are extended with new actor(s) representing the system and its relations with the stakeholders. Some organizational goals are then delegated to the system and become the system requirements. However, these requirements are still expressed in terms of goals, beliefs and tasks within the goal model. They must be translated into a set of UML model elements (as (mis)use cases) that can be understood and reused by the security techniques employed in the later phases of the process. A complete translation (including security aspects) between both kinds of languages does not exist.

With respect to the design models adaptation, we must take into account that some platforms offer or predefine some security capabilities that may help/impair the implementation of the security aspects defined in the models. Adapting the expressivity of the models to these features would facilitate bridging the gap between the system design and its implementation. Several profiles representing the characteristics of each common platform (as the already existing J2EE profile [22], though it does not include security aspects) must be defined to annotate the models with the appropriate mapping information in order to ease this transition.

All these integration challenges require solving a number of more specific open problems that we can only partially mention due to lack of space:

1. Definition of a common modeling framework (e.g., basing all notations on MOF-compliant metamodels) to facilitate the mapping between different languages;
2. Traceability techniques to link the model elements at different abstraction levels so that it becomes possible to justify which part of a lower-level model covers the security aspect modeled in a higher-level one;
3. Incremental model synchronization techniques that incrementally propagate changes in one model to related models in later (or previous) phases of the development process without a complete recomputation;
4. Consistency analysis techniques that ensure that new security aspects do not contradict existing ones.

8 Conclusions and Future Work

We have argued the necessity of an integrated security engineering MDD process to help designers in the development of secure systems. Such a process should contextualize several security techniques (from goal-modeling techniques for the analysis of the organizational context to detailed design techniques for detecting security vulnerabilities in the components calls in the deployed system) in a unified framework.

As a first step towards this integrated process, we have sketched how the existing security proposals (aimed at specific security aspects and/or particular phases of the development process) can be placed within such framework and the research challenges that must be faced when trying to combine them.

As future work we would like to refine, complete and validate the process and advance in the integration of the different security proposals currently available by facing the research challenges explained before.

Acknowledgments: Work supported by the Spanish Ministry of Education and Science (project TIN2005-06053), the grant 2007 BP-A 00128 (Catalan Government) and by the Hyperion project founded by Canada's NSERC.

References

1. R. Anderson. Why cryptosystems fail. *CACM*, 37(11):32–40, 1994.
2. D. Basin, J. Doser, and T. Lodderstedt. Model Driven Security: from UML Models to Access Control Infrastructures. *TOSEM*, 15(1):39–91, 2006.
3. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. TROPOS: An Agent-Oriented Software Development Methodology. *JAAMAS*, 8(3):203–236, 2004.
4. R. Breu, G. Popp, and M. Alam. Model based development of access policies. *STTT*, 9:457–470, 2007.
5. M. Clavel, V. da Silva, C. Braga, and M. Egea. Model-Driven Security in Practice: An Industrial Experience. In *Proc. of ECMDA-FA'08*, LNCS 5095, pages 326–337. Springer-Verlag, 2008.

6. A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Sci. of Comp. Prog.*, 20:3–50, 1993.
7. F. den Braber, T. Dimitrakos, B. A. Gran, M. S. Lund, K. Stølen, and J. Ø. Aagedal. The CORAS methodology: model-based risk assessment using UML and UP. In *UML and the unified process*, pages 332–357. Idea Group Publishing, 2003.
8. T. Doan, S. Demurjian, T. C. Ting, and A. Ketterl. MAC and UML for secure software design. In *Proc. of FMSE'04*, pages 75–85. ACM Press, 2004.
9. G. Elahi and E. Yu. A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs. In *Proc. of ER'07*, LNCS 4801, pages 375–390. Springer-Verlag, 2007.
10. P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling Security Requirements Through Ownership, Permission and Delegation. In *Proc. of RE'05*, pages 167–176. IEEE Press, 2005.
11. C. Haley, R. Laney, J. Moffett, and B. Nuseibeh. Security requirements engineering: A framework for representation and analysis. *TSE*, 34(1):133–153, 2008.
12. M. Hoiward and D. LeBlanc. *Writing Secure Code*. Microsoft Press, 2003.
13. M. Jackson. *Problem Frames: Analysing and structuring software development problems*. Addison Wesley, 2001.
14. J. Jürjens. *Secure Systems Development with UML*. Springer-Verlag, 2004.
15. L. Liu, E. S. K. Yu, and J. Mylopoulos. Security and Privacy Requirements Analysis within a Social Setting. In *Proc. of RE'03*, pages 151–161. IEEE Press, 2003.
16. A. Martnez, O. Pastor, and H. Estrada. Closing the Gap between Organizational Modeling and Information System Modeling. In *Proc. of WER'03*, pages 93–108, 2003.
17. F. Massacci, J. Mylopoulos, and N. Zannone. An Ontology for Secure Socio-Technical Systems. In *Handbook of Ontologies for Business Interaction*. The IDEA Group, 2007.
18. S. J. Mellor and M. J. Balcer. *Executable UML: A Foundation for Model-driven Architecture*. Addison Wesley, 2002.
19. H. Mouratidis, J. Jürjens, and J. Fox. Towards a Comprehensive Framework for Secure Systems Development. In *Proc. of CAiSE'06*, LNCS 4001, pages 48–62. Springer-Verlag, 2006.
20. B. Nuseibeh and S. Easterbrook. Requirements engineering: a roadmap. In *Proc. of ICSE'00 - Future of Software Eng. Track*, pages 35–46. ACM Press, 2000.
21. OASIS. eXtensible Access Control Markup Language (XACML) Version 2.0. OASIS Standard, 2005.
22. Object Management Group. *UML Superstructure Specification*, 2004.
23. R. Ramsin and R. F. Paige. Process-centered review of object oriented software development methodologies. *ACM Comput. Surv.*, 40(1):1–89, 2008.
24. I. Ray, N. Li, R. France, and D.-K. Kim. Using UML to visualize role-based access control constraints. In *Proc. of SACMAT'04*, pages 115–124. ACM Press, 2004.
25. V. Santander and J. Castro. Deriving Use Cases from Organizational Modeling. In *Proc. of RE'02*, pages 32–42. IEEE Computer Society, 2002.
26. G. Sindre and A. L. Opdahl. Eliciting security requirements with misuse cases. *REJ*, 10(1):34–44, 2005.
27. A. van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proc. of ICSE'04*, pages 148–157. IEEE Press, 2004.
28. A. van Lamsweerde and E. Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *TSE*, 26(10):978–1005, 2000.
29. E. S. K. Yu. *Modelling strategic relationships for process reengineering*. PhD thesis, University of Toronto, 1995.