Model-driven Secure Development Lifecycle

Zhendong Ma, Christian Wagner, Arndt Bonitz, and Thomas Bleier

Safety & Security Department, Austrian Institute of Technology
Siebersdorf 2444, Austria
firstname.lastname@ait.ac.at

Robert Woitsch and Markus Nichterl

BOC Asset Management GmbH
Bäckerstraße 5, 1110 Vienna, Austria
firstname.lastname@boc-eu.com

Abstract

Building security into software development lifecycles and doing it right is hard. To address the challenge, several prominent organizations have published process-oriented security guidelines to bring security activities into a structured way. Although these efforts contribute to measurable improvements in software and system security, they are often too verbose and fuzzy to be implementable in a development lifecycle involving people (e.g., security experts, developers, and managers) with different skillsets. In this paper, we propose the model-driven secure development lifecycle (MD-SDL), an approach that leverages on modeling methods and the advances in model-driven security to simplify the process of efficiently integrating security into development lifecycles for the development of security-critical software and systems.

1 Introduction

Tremedous efforts to secure software and systems have not stopped incidents and security breaches. A main reason is that building security into software systems requires specialized skills and collaborations among the security experts and developers and other stakeholders. A large number of activities related to security needs to be performed through out a software's development lifecycle. In the past, two promising approaches appeared to address security challenges in security engineering in software development lifecycles. The first is model-driven security, which applies models in security engineering to gain more focused views of complex systems and uses levels of abstraction to assist non-security experts (e.g., developers) to implement security in a correct and efficient way. The second approach is the definition of guidelines for security activities involved in a software's development lifecycle. Several organizations comprise lists of "to-dos", i.e., activities regarded as best practices on the technical as well as the organizational level.

Here we propose the model-driven secure development lifecycle (MD-SDL). Our approach combines rigid modeling methods and efficiency gained from model-driven security with process-oriented security development lifecycle guidelines to produce structured, practical, and efficient security engineering practices in software development lifecycles. Section 2 reviews model-driven security and security development lifecycle; Section 3 introduces our MD-SDL approach and our design considerations followed by conclusion in Section 4.

2 Approaches to security in development lifecycle

Model-driven security (MDS), proposed in recent years [3], applies the model-driven architecture (MDA) [11] approach to security engineering. In MDS, models are used to capture and represent complex systems and security requirements. The modeled systems are used as a basis for security analysis or generating software artifacts such as access rules [4], security policies [2], or security configuration files [10]. Since models can simplify and abstract away many details of actual systems, MDS contributes to a more efficient design and development of security-critical software systems [8].

The Microsoft Security Development Lifecycle (SDL) [6] is a software development security assurance process to integrate security and privacy practices into all phases of software development lifecycles. It is divided into five core phases: requirements, design, implementation, verification, and release. In the requirements phase, security and privacy requirements analysis produces a preliminary requirement specification. Quality gates and bug bars define the criteria for acceptable security and privacy level, which must be met in the development lifecycle. Risk assessment identifies areas of focus within the project that need special attention and security and privacy measures. The purpose is to have a cost-effective approach to plan and distribute project efforts. In the design phase, design requirements generate design specifications for security and privacy and specifications on security features. Attack surface reduction minimizes risks related to potential vulnerabilities and exploits. Threat modeling analyzes and identifies security risks of a software architecture in a structured way. In the implementation phase, developers define development toolset with approved security properties, e.g., toolset without known security flaws. Deprecating old and potentially unsafe APIs intends to minimize the risk of security holes in a software. Static analysis of source code is used to perform a code review for the implementation. In the verification phase, dynamic program analysis and fuzz testing verifies the software in a run-time environment. The verification is completed with a re-review of the threat models and the attack surface to ensure that implementations follow the design specifications. In the the release phase, an incident response plan ensures that security will be continuously maintained after the release. A final security review examines all security related activities in the project. Release and archive are control steps in which a software is certified to fulfill security and privacy requirements.

Security Considerations in the System Development Life Cycle guideline [7], proposed by the National Institute of Standards and Technology (NIST), aims at integrating information security in IT system development lifecylces (SDLC). The NIST SDLC consists of five phases: initiation, development/acquisition, implementation/assessment, operations and maintenance, and disposal. In the initiation phase threats, security and privacy requirements and potential constrains are identified with respect to their impacts on the business. In development/acquisition phase, a risk assessment is performed to select and decide security controls. Security architecture integrates different security controls and services into the system. In the implementation/assessment phase, the hardware and software of an information system is installed and integrated into an organization's operational environment. Activities to conduct security control testing, system certification and accreditation are performed. In operations and maintenance phase, the running system is monitored and assessed to determine the fulfillment of security requirements. The system will be tuned according to security and business requirements. The activities in the last disposal phase are defined to ensure that critical information is preserved and sensitive information is safely erased before any software and hardware is decommissioned.

The Systems Security Engineering Capability Maturity Model (SSE-CMM) [12] is a reference framework to assess security engineering capability, i.e., how good is an organization in developing secure products, systems, and services. SSE-CMM emphasizes activities and processes, in which the security engineering capability is measured by how extensive a set of security activities are performed in the security development lifecycle. As a result, SSE-CMM is also a structured collection of generally accepted security engineering practices. In fact, SSE-CMM does not align security activities with different lifecycle phases. All security activities, called base practices (BP), are grouped in process areas (PA) applied across the lifecycle. There are 61 base practices grouped in 11 security-related process areas (e.g., from administer security controls to verify and validate security). A base practice is denoted by the number of process area (PA_no) followed by the base practice number within the process area (BP_no).

3 Model-driven secure development lifecycle

"Model-driven" in this paper means: (1) to use security engineering techniques that follow principles defined in model-driven architecture, (2) to use modeling method engineering to structure and support security activities in a development lifecycle, (3) to use models as a part of the artifacts in the development lifecycle. By adopting a model-driven approach to SDL, we aim at achieving the following goals: (1) To tackle complexity in software system. By applying MDA principles, we can represent and analyze a complex system with different levels of abstraction and arbitrary viewpoints. (2) To bring software into tangibility. Software and information kept in the system are usually invisible and do not have geometric representation [5]. The graphic features and the rigidness of formal modeling language underlying the models can assist the understanding of the intrinsic nature of the software and the state of a system. (3) To conduct security engineering in a structured as well as flexible way. Models (especially graphic models) are more intuitive to understand and easier to follow than texts cluttered in a set of documents. Thus the model-driven approach can ensure that defined security activities are performed within the development lifecycle. Furthermore, models can be resulted from different instantiations from a pre-defined meta model. This brings a "defined" flexibility to different SDL processes for different software and systems. (4) To facilitate clear and effective communications among stakeholders. Models can contribute to clear and understandable communications on specific aspects of a system among the people involved in the project, e.g., communicating functional requirements between users and developers, or communicating security requirements between developers and security experts. (5) To provide accurate and extensive documentation support. Documentations are not only a requirement on software development but also a requirement on assurance and accountability. Model-driven approach can embed documentation efforts into the development lifecycle.

3.1 Model architecture

The model architecture is designed to be a framework for the MD-SDL process. Our design reflects the opinions from both the security experts and the modeling experts. It also reflects our accumulated experiences in security research projects, modeling projects, and software development projects. Figure 1 shows the overall model architecture. Notice that as a first attempt to apply model-driven approach to secure development lifecycle, we focus only on the requirement, design, and implementation phase since our work centers around security design and proof-of-concept implementation. The model architecture is designed to fulfill the aforementioned goals. A part of the architecture consists of loosely connected existing security engineering techniques and tools. Each development phase includes several model components. A model component might further include several sub-components. The model architecture connects these components into a holistic framework and ensure that the artifacts and documents from one component is related to the other corresponding components. In the following, we give a brief description of these components.

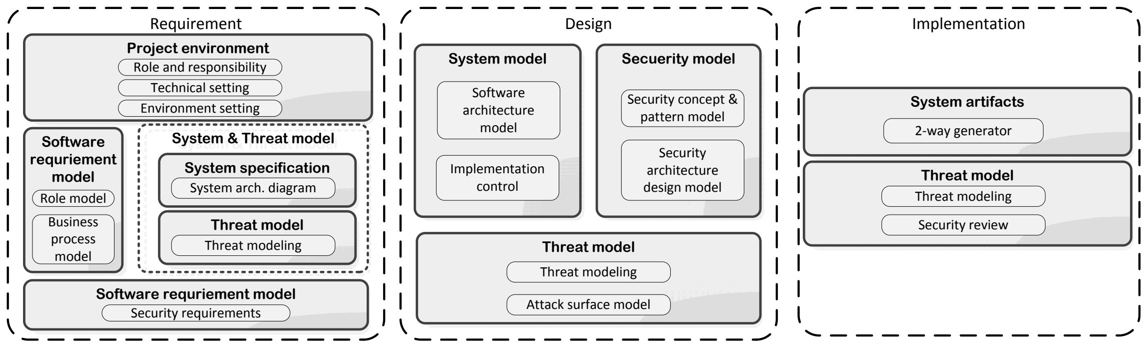

Figure 1. Architecture overview of MD-SDL

In the requirement phase, *Project environment* provides a consolidated view on various variables of the project settings: *Role and responsibility* identifies and assigns security responsibility to specific person; *Technical setting* defines technical means used in the project; *Environment setting* defines project environment parameters such as textual project description and time plan, and all things help stakeholders to understand the status of the project. Within the *Software requirement model*, *Role model* is a basis for specifying access model, i.e. who should access what information and conduct which action. A role-based access model can be used here for modeling the organizational structure and responsibilities for running the application. The *Business process model* captures the business processes supported and/or implemented by the software. Standard modeling languages, such as Business Process Model and Notation (BPMN), can be integrated here to fulfill the task. *System & Threat model* is where technical system specification and threat analysis takes place. *System specification* uses informal system diagrams, as well as UML diagrams (with different views), to capture and represent the software and system architecture. It provides accurate and semantically rich information on the system in the requirement phase. *Threat model* uses Microsoft threat modeling approach [13], i.e., specify system in dataflow diagrams and enumerate all possible threats, estimate their impacts on the system, and optionally list their possible mitigations. In *Security requirement model*, security requirements are categorized and listed. The requirements are cross-referenced to the other models, such that the background and rationale for these requirements can be traced and communicated among stakeholders. In addition, a requirement is graphically indicated according to their status, i.e., "not implemented", "in work", or "implemented" to provide a quick overview of the progress of the requirement fulfillment throughout the project.

In the design phase, *Software architecture model* represents the aspects to software and system architecture in a "plain vanilla" view, i.e., without security functions and services. *Implementation control* includes documented specifications and guidelines on practices for developer to follow in the implementation phase. Such a control is a part of the effort to realize secure by design, in which possible implementation related security mistakes can be identified and avoided in an early stage. *Security concepts & pattern model* provides high level security concepts and security patterns in forms of models. A generic concept or pattern is constructed as a meta model and instantiated as a model in specific context. *Security architecture design model* uses UML or other diagrams to represent the security design in details. *Threat modeling* is performed again in the design phase with more design details such that the security design can be improved iteratively. *Attack surface model* assists the minimization of all possible vulnerabilities and explicitly expresses security improvements in design phase.

In implementation phase, *System artifacts* provides: (1) generate artifacts from models as proposed in many model-driven security approaches, e.g., access rules, security configuration files, or even running code; (2) reverse engineer artifacts to generate models such that the artifacts can be analyzed by corresponding models. Given the state-of-the-art, we expect that such an artifact-to-model generation cannot be fully realized in the near future. However, if it is possible, *threat modeling* can be used to conduct security analysis on the implementation at a more abstract model level. Otherwise, more traditional *Security review* will be used to ensure the implementation fulfills security requirements.

3.2 Realization

Although many tools are candidates to implement the MD-SDL framework, in our approach we choose the platform provided by the Open Model Initiative [1]. Beside our consideration to avoid proprietary software, the main reason for choosing this platform is that it provides a community-based working environment and hence applies the open source concept to modeling method engineering. With regard to modeling method engineering, the Open Model Initiative provides foundational material, tools, and platforms. It also supports different modeling method engineering with specification, implementation and deployment of modeling methods. Using the Administration Toolkit and Modeling Toolkit provided by Open Model, we can define meta models on an abstract level and then develop concrete instance models. On the Open Model platform, meta models are defined by classes and relation classes with certain attributes. The defined class is available as an object in the Modeling Toolkit and can be used for implementing model instanced. The graphical representation of objects is defined by GraphRep attributes of the class. Furthermore, we add semantics to the models by defining relation classes that control the objects' connectivities.

We adopt a hybrid approach to implement the MD-SDL framework, such that we can take advantage of the latest development in the field. Currently, the models serve as an integration framework that bundles and "glues" existing methods and tools. For example, for system modeling we use UML modeling techniques; for threat modeling we adopt the

Microsoft SDL Threat Modeling approach; and for code generation we use code generator developed in [8].

4 Conclusion

This paper proposed a new approach to integrate security activities into software development lifecycles based on modeling methods and the progress made in model-driven security. Our goal is to have a framework that is practical, extensible, and reproducible for different security-critical software development and research projects. Our next step is to apply the MD-SDL approach in our ongoing and upcoming security activities and promote the usage among our project partners. The results and feedbacks will be used to improve the framework and provide evidence on the feasibility of our approach.

Acknowledgment

This work was partly funded by the Austrian security-research programme KIRAS and by the Federal Ministry for Transport, Innovation and Technology.

References

[1] Open model initiative. `http://www.openmodels.at`

[2] Alam, M., Breu, R., Hafner, M.: Model-driven security engineering for trust management in SECTET. JSW 2(1), 47–59 (2007)

[3] Basin, D., Clavel, M., Egea, M.: A decade of model-driven security. SACMAT '11, ACM, New York, NY, USA (2011)

[4] Basin, D., Doser, J., Lodderstedt, T.: Model driven security: From UML models to access control infrastructures. ACM Trans. Softw. Eng. Methodol. 15, 39–91 (January 2006)

[5] Brooks, Jr., F.P.: No silver bullet essence and accidents of software engineering. Computer 20(4), 10–19 (Apr 1987)

[6] Howard, M., Lipner, S.: The Security Development Lifecycle. Microsoft Press (2006)

[7] Kissel, R., Stine, K., Scholl, M., Rossman, H., Fahlsing, J., Gulick, J.: Security consideration in the system development life cycle (October 2008)

[8] Ma, Z., Wagner, C., Bleier, T.: Model-driven security for web services in e-government system: Ideal and real. NWeSP'11. pp. 221 –226 (October 2011)

[9] Microsoft: Simplified implementation of the microsoft sdl (March 2011)

[10] Nakamura, Y., Tatsubori, M., Imamura, T., Ono, K.: Model-driven security based on a web services security architecture. SCC'05, Washington, DC, USA (2005)

[11] Object Management Group (OMG): MDA guide version 1.0.1. `http://www.omg.org/mda/mda_files/MDA_Guide_Version1-0.pdf` (June 2003)

[12] SSE-CMM project: Systems security engineering capability maturity model: Model description document (version 3.0) (June 2003)

[13] Swiderski, F., Snyder, W.: Threat modeling. Microsoft Press (2004)