A Model-driven Approach for Securing Software Architectures*

Mario Arrigoni Neri[1], Marco Guarnieri[2], Eros Magri[3], Simone Mutti[1] and Stefano Paraboschi[1]
[1]*Dip. di Ingegneria Informatica e Metodi Matematici, University of Bergamo, Bergamo, Italy*
[2]*Institute of Information Security, ETH Zürich, Zürich, Switzerland*
[3]*Comelit R & D, Comelit Group S.p.A, Rovetta S. Lorenzo, Italy*

Keywords: Access Control, Model-driven Security, Security Policy, Software Architectures.

Abstract: Current IT systems consist usually of several components and services that communicate and exchange data over the Internet. They have security requirements that aim at avoiding information disclosure and at showing compliance with government regulations. In order to effectively handle the security management of complex IT systems, techniques are needed to help the security administrator in the design and configuration of the security architecture. We propose a model-driven security approach for the design and generation of concrete security configurations for software architectures. In our approach the system architect models the architecture of the system by means of UML class diagrams, and then the security administrator adds security requirements to the model by means of *Security4UML*, a UML profile. From the model enriched with security requirements, the concrete security configuration is derived in a semi-automated way. We present a tool that supports this model-driven approach, and a case study that involves a distributed multi-user meeting scheduler application.

1 INTRODUCTION

IT systems are becoming more and more complex as the range of offered services increases and the capabilities of the systems themselves improve. Current IT systems consist usually of several components and services communicating and exchanging data over complex IT infrastructure through the Internet. These systems usually have strict and complex security requirements, which are needed in order to adequately protect sensitive data and to assure the availability of the systems themselves ((Patterson, 2002; Pertet and Narasimhan, 2005) analyze the costs of unavailability of applications). The management of security requirements is a critical task that has the goal of avoiding possible information disclosure and showing compliance with respect to the many regulations promulgated by governments.

The security of the global system arises both from the security of the components and from the correct configuration of the entire IT infrastructure. Managing the security of these systems is a tough task, because the high number of components, which usually need an access to the Internet, leads to a large attack surface, whilst the high level of interconnections between components increases the damage that an attack to one of the components may cause to the entire system. Recent analysis of information security breaches has found empirical evidence that highlights the fact that security management is a difficult and error prone task, (e.g., (7safe, 2010; Langevin et al., 2008)) showing that misconfigurations of the security infrastructure are one of the main causes of breaches.

A viable way for handling the complexity of security management is by using a model-driven approach. In a way similar to the evolution seen in the area of software development and database design, the model-driven engineering of security leads to the specification of security requirements at an abstract level, with the subsequent refinement of the abstract model toward a concrete implementation. It is then guaranteed that, when the high-level representation of the security requirements is correct, the security configuration of the system satisfies the requirements. The application of this approach requires the implementation of a rich collection of tools supporting, for each of the many components in the system, the refinement from the high-level representation of security requirements to the concrete security con-

*This work was partially supported by the EC within the 7FP, under grant agreement 257129 "PoSecCo" and 256980 "NESSoS", by the Italian Ministry of Research within the PRIN projects "PEPPER 2008", "GATECOM" and "GenData 2020".

Arrigoni Neri M., Guarnieri M., Magri E., Mutti S. and Paraboschi S..
A Model-driven Approach for Securing Software Architectures.
DOI: 10.5220/0004611305950602
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 595-602
ISBN: 978-989-8565-73-0

figuration. A significant investment in research and development is needed to realize this vision, in order to manage the heterogeneous collection of devices and security services that information and communication technology offers to system administrators. Still, this large investment promises to produce adequate returns, considering the importance that the correct management of security requirements presents in modern information systems.

The model-driven approach described above has several advantages: (1) the concrete configuration can be generated in a semi-automated way from the models (in this way the risk of misconfigurations is low), (2) the models can be analyzed for an early identification of anomalies and inconsistencies, (3) the models act as documentation of the security architecture of the system, (4) the modeling process requires that the security administrator defines the security properties of the system and explicits the assumptions behind the requirements, (5) it allows the adoption of a *defence-in-depth* approach that deploys security countermeasures at several layers, (6) changes in the security requirements can be effectively handled by modifying the initial model and then by generating the new concrete configuration.

Integrating this model-driven security approach with Architecture Description Languages (ADLs) can lead to great advantages. Software architectures are produced in the early steps of the development process, and thus integrating security properties in architecture design allows developers to integrate security early in the development process. In this way, developers consider security during the whole process and not only as an afterthought, and they can also analyze early security configurations in order to detect flaws. This fact leads to better security and reduces the costs of changes in the security configuration.

In this work, we present an approach that can be used to add security requirements to architectural models of distributed, data-intensive applications represented in the C2 architecture style (Taylor et al., 1996). The paper is organized as follows. In Section 5 a comparison is made with previous related work. Section 2 presents a brief overview of the C2 architecture style. Section 3 describes the Model-Driven Design process in detail. In Section 4 we present our UML profile, called *Security4UML*, that can be used to add security properties to architectural models. In Section 6 we present a case study describing how our approach can be used to model security requirements for a distributed multi-user meeting scheduler application. Finally, Section 7 draws a few concluding remarks and presents future work.

2 BACKGROUND

The C2 architectural style is a software architectural style that can be used to model user interface intensive systems (Taylor et al., 1996). It can thus be used to model the majority of data intensive applications, e.g., e-commerce web applications, by means of the C2 ADL (Medvidovic, 1996). In (Robbins et al., 1998), Robbins et al. present a way to represent C2 ADL by means of UML class diagrams. In the remainder of the paper, we use simply C2 to refer to both the architecture style and the ADL. C2 architectures are based on two main concepts: *components* and *connectors*.

Components are those parts of the architecture that maintain state and perform operations. They contain objects with defined interfaces. Each component exchanges messages by means of two interfaces, called "top" and "bottom". In a UML class diagram, a component can be modeled by means of a class with the stereotype $\ll C2Component \gg$. In C2, *components* are not allowed to directly exchange messages; they can exchange messages by means of *connectors*, which have the responsibility to route, filter and broadcast messages. In a UML class diagram, a connector can be modeled by means of a class with the stereotype $\ll C2Connector \gg$. Each interface of a *component* may be connected to only one *connector*, whereas a *connector* may be connected with one or more *components*.

3 MODEL-DRIVEN DESIGN PROCESS

The management of security aspects of current IT systems is a tough task, and misconfigurations may cause security breaches. The increasing complexity of this task may be effectively handled by using model-driven approaches for the definition and management of the security configurations of the system. Our model-driven process involves two stakeholders: (a) the system architect and (b) the security administrator. The system architect models the architecture of the system in the C2 architectural style. As shown in (Robbins et al., 1998), there is a way to represent C2 models by means of UML class diagrams, and thus, without loss of generality, we assume that the system architect models the architecture by means of UML diagrams.

Then, the security administrator receives the architectural model of the system and he/she can enrich it with a representation of the security requirements. The security administrator can extract information on the security requirements from two sources: (a) busi-

ness requirements, which express a high-level description of the expected behaviour of the global system (security administrators usually receive business requirements from the business and they cannot modify them), (b) the physical configuration of the system, i.e., servers, applications etc. (this aspect also cannot be usually influenced by the security administrator, at least in the short term). The security modeling process is divided into three different phases. In each phase the security administrator adds security relevant information to the model.

In the first phase of the process, the *Modeling Phase*, the security administrator models the security requirements and adds them to the architectural model. The security requirements are expressed by using the *Security4UML* profile defined in Section 4. Despite the fact that business requirements are usually expressed in natural language, the security administrator can easily define a first high level model that is close to the business requirements. Then, he/she refines the model in several steps, by adding details at each iteration. During this refinement process the administrator can execute automated analysis tools over the model, in order to detect conflicts and anomalies in the model itself. The results of the analysis can lead to further modifications to the model. The resulting model, called *Sec-Model*, is a platform independent model (PIM).

In the second phase of our process, the *Enrichment Phase*, the security administrator enriches the *Sec-Model* with context dependent information, i.e., he/she adds information about the concrete configuration of the system, e.g., information about type and version of DBMSs or web servers. In this phase the security administrator can extract the needed information from the *actual configuration* of the system. The result of this phase is an *Enriched Sec-Model* (ESec-Model) which contains the specification of the security requirements and information about the actual infrastructure. The *ESec-Model* is a platform specific model (PSM).

The last phase, called *Derivation Phase*, consists in the derivation and the deployment, in a semi-automated way, of the concrete security configuration of the system. During this phase, the *ESec-Model* is given as input to several derivation modules that generate and deploy the global concrete configuration. Each module generates and deploys part of the concrete configuration, e.g., in the example presented in Section 6 we have defined a module for *PostgreSQL*, which extracts the information needed from the *ESec-Model*. The platform dependent model has to contain enough information to allow the generation of the configuration.

A critical aspect of our process is the need of a dedicated derivation module for each kind of resource in the IT system under modeling. However, once the effort for the development of the adequate modules is done, the entire security architecture of the application can be managed, updated and modified without directly operating over the concrete configuration.

4 Security4UML

In order to adequately represent the large number and variety of entities involved in actual enterprise scenarios, we need a rich metamodel and a flexible language for expressing relationships between the entities of the model. In this section we are going to present a UML profile, called *Security4UML*, which can be used to model security properties. Due to space constraints we are going to present only a brief overview of the profile and in the remainder of the section we are going to present only the *Sec-Model* meta-model. The *ESec-Model* meta-model, not shown, is an extension of the *Sec-Model* meta-model that refines and enriches classes and associations with system dependent information. System dependent information is stored in tagged values associated with stereotypes.

Abstract Syntax. The meta-model of the *Security4UML Sec-Model* consists of six sub-meta-models, each one focusing on a particular aspect of the system.

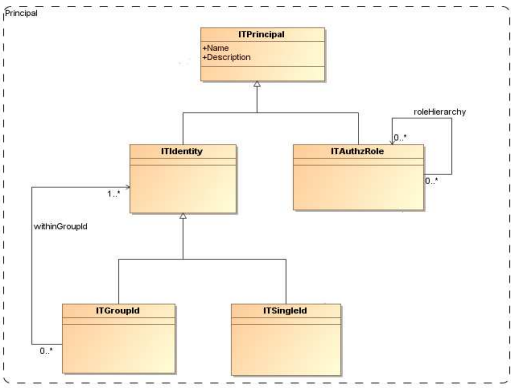


Figure 1: Principal meta-model.

Principal Meta-model: it can be used to represent all the individuals and entities that can appear as beneficiaries of authorizations or involved in an authentication rule. The meta-model is shown in Figure 1. It contains the abstract class *Principal*, which represents an abstraction of the principals in the system. We can represent single users by means of the *SingleId* class and groups of users using the *GroupId*

class. Both classes have a common abstract superclass called *Identity*. *SingleId*s are flexible, they may represent single users of the system, legal entitites (i.e., external companies or customers of the system under modeling) or software components. Over the identities we can define a hierarchy, because groups are organized in a taxonomy via the *contains* relationship. The class *Role* can be used to model roles. In RBAC, each role represents the binding between users and permissions associated to them. The meta-model allows the definition of the role hierarchy by means of the *roleHierarchy* relationship. *Role*s may be used to represent several concepts. For instance, they may represent *organizational roles* (i.e., specific enterprise functions), *collections of privileges* that may be activated by users or *system accounts* available on certain systems.

Privilege Meta-model: this meta-model contains the *Privilege* class, which can be used to represent the right of performing a certain action. When a privilege is associated with a specific resource, it can be modeled by means of the *ObjectPrivilege* class. The *Action* class represents the actions that the principals can execute.

Security Rule Meta-model: it is used to describe security rules and policies. The main class is *SecurityRule* which is an abstract class representing security rules. Each security rule is granted to a certain principal, as represented by the *grantedTo* relationship, by a particular single id, modeled by the *grantor* relationship. Our meta-model aims at handling several aspects of security (i.e., authorization and authentication) in a common way. For this reason the concrete subclasses of *SecurityRule* are: (a) the *RoleAuthorization* class which gives the privilege to enable a certain role (this fact is represented by the *enabledRole* relationship), (b) the *SystemAuthorization* class which associates the subject to which the authorization is granted with a privilege, (c) the *AuthenticationRule* which models the authentication configuration for a principal with respect to a certain resource. Security rules have a sign, and may be marked with a *grant option*. This meta-model contains the classes *Policy*, *PolicySet* and *Target* which can be used to model security policies in a way similar to what is done in XACML.

Authentication Meta-model: it can be used to describe the properties that the authentication service has to satisfy by means of the *AuthcFeature* class. The class *AuthcOption* can be used to specify the configuration options of the authentication feature.

Resource Meta-model: this meta-model can be used to model the entities of the IT system. Given the fact that our meta-model is quite general and aims at representing several aspects of security, we have several concrete subclasses of the *SecurityObject* abstract class: 1. the *Service* class can be used to represent services in a Service Oriented Architecture environment; 2. the *SoftwareModule* class describes nodes and software modules; 3. the *Link* class models communication channels that are concrete, e.g., network links, or abstract , e.g., software connectors; 4. the *Data* class can be used to represent static resources like files and directories in a file system or tables and views in a database. Protected data may be modeled by means of the *ProtectedData* class and its subclasses *EncryptedData* and *SignedData*, which contain the details about the encryption and signature processes. Protected communication channels, e.g., SSL connections, may be modeled by the *ProtectedLink* class (subclass of the *Link* class).

Table 1: Mapping between UML elements and the ESecModel metamodel.

UML type	Stereotype	Security4UML
Class	⟨⟨Principal⟩⟩	Principal
Class	⟨⟨Role⟩⟩	Role
Class	⟨⟨GroupId⟩⟩	GroupId
Class	⟨⟨SingleId⟩⟩	SingleId
Association	⟨⟨contains⟩⟩	contains
Association	⟨⟨roleHierarchy⟩⟩	roleHierarchy

Security Domain Meta-model: it contains only the *SecurityDomain* class which can be used to represent security domains and the contained principals, by means of the *principalSecurityDomain* relationship, and resources, using the *resourceSecurityDomain* relationship. Given the fact that *Security4UML* aims at modeling security architectures, which may consist of several IT systems, the concept of security domain is important, because it may be used to represent trust boundaries between systems.

Concrete Syntax. In order to define the concrete syntax of our *Security4UML* modeling language, we defined a UML profile based on the meta-model presented above. The mapping between UML types and stereotypes and the *Security4UML* meta-model is straightforward. Due to space constraints we present only an example. Table 1 shows how the *Principal meta-model* can be represented by using UML types and stereotypes.

Tool Support. A critical aspect in model-driven engineering approaches is the need of adequate tools that can be used by developers in order to handle, refine and transform the models. The availability of such tools is usually one of the main factors that may influence the success of the approach itself.

In order to help security administrators in the def-

inition and refinement of our models, we developed the *Security4UML Tool*. The tool provides the developer with three functionalities:

- Reasoning: *Security4UML Tool* provides several reasoning services that can be used to detect anomalies and inconsistencies in the model, and to report them to the user. In this way the user can modify the model according to the analysis' results, in order to remove inconsistencies. These reasoning services are implemented by using Semantic Web technologies, i.e., the *Sec-Model* is represented as an *OWL ontology* and the reasoning services are expressed by means of Semantic Web tools, e.g., *OWL-DL*, *SWRL* and *SPARQL*. The *Minimization* service can be used to detect redundancies in the *Sec-Model*, i.e., the model may contain authorizations which are implied by other authorizations and thus may be removed. The *Incompatibility* service detects authorizations that are incompatible, e.g., the *Sec-Model* may contain conflicting authorizations. The *SoD* service can be used to detect authorizations breaking Separation of Duty constraints expressed in the *Sec-Model*. A detailed description of these services can be found in (Arrigoni Neri et al., 2012; Arrigoni Neri et al., 2013).

- Enrichment: *Security4UML Tool* helps the developer in the definition of the *ESec-Model*. The platform independent *Sec-Model* is enriched and transformed in order to obtain the platform specific model. *Security4UML Tool* provides also reasoning capabilities that can be used to check the consistency of an instance of the *ESec-Model*.

- Code Generation: *Security4UML Tool* manages the process that leads to the concrete implementation and deployment of the security configuration. The tool supports the definition of several code generation modules that generate the concrete configuration for a particular element of the infrastructure, e.g., a particular version of a DBMS. Each module takes as input the *ESec-Model*, extracts the needed information and generates part of the concrete configuration.

We implemented *Security4UML Tool* on the basis of the Eclipse framework, because Eclipse is today one of the de-facto standard in terms of IDEs. Several model-driven engineering tools are based upon Eclipse, e.g., TopCased or IBM Rational Rose. *Security4UML Tool* can be integrated in one of such tools, and this fact lets us providing developers with reasoning, enrichment and code generation capabilities. The extension point mechanism allows an easy integration of new reasoning services, enrichment modules and code generation modules. A detailed description of part of the tool can be found in (Mutti et al., 2011; Guarnieri et al., 2012).

5 RELATED WORK

The last decade has seen a growing interest in model-driven techniques concerning security aspects (Basin et al., 2011), and several solutions have been proposed to formalize the development of secure systems.

Basin et al. have proposed *SecureUML* (Basin et al., 2006), a UML profile that can be used to model Role-Based Access Control (Sandhu, 1998) (RBAC) infrastructures. They have proposed the application of *SecureUML* in several contexts: from the definition of security aware GUIs (Basin et al., 2010) to process oriented systems (Basin et al., 2003). They have shown how *SecureUML* can be used for supporting a model-driven security approach that leads to the concrete implementation of IT systems. The *SecureUML* meta-model is primarily focused on access control systems, and more in detail on RBAC. On the contrary, our *Security4UML* profile can model several aspects of security. It supports the definition of access control policies, it can be used to model authentication properties, it also considers encrypted and signed resources and protocols such as SSL.

In (Jürjens, 2003), Jürjens proposed, as a security extension of UML, the *UMLSec* profile. In (Jürjens, 2005), he shows how *UMLSec* can be used for security analysis and formal security requirement verification. Although *UMLSec* supports the annotation of UML models with security requirements, we need a more extended meta-model, like *Security4UML* or *SecureUML*, to facilitate a model-driven security approach.

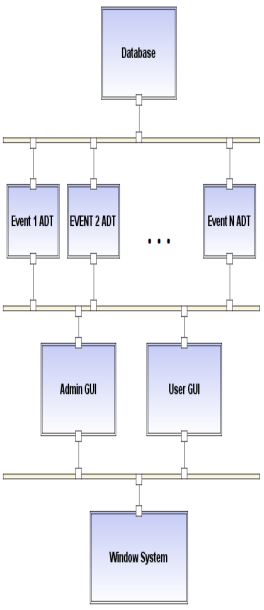
Some works present approaches for the integration of security concepts in ADLs. In (Mouratidis et al., 2005), Mouratidis et al. propose an ADL for agent systems that is able to express security properties by means of *protection objectives*, i.e., desiderable security properties that an agent might have. Each agent may own several *security mechanisms* to satisfy objectives. The ADL allows the definition of *security constraints* that may restrict goals and capabilities of agents. In (Ren and Taylor, 2005), Ren et al. extend the existing Architecture Description Language *xADL* with security concepts. They provide a new *SecureConnector* which can be used to model architectural access control. In (Oladimeji et al., 2007), Oladimeji et al. present a UML-based ADL that can express access control properties. On the contrary, our work can model several aspects of security, not only access control. Another difference is that we integrate security requirements in a well-known ADL, i.e., C2, instead of defining a new one.

Integrating security in software architectures may be a viable way of handling the increasing complexity

of IT systems. Although a lot of work has been done on model-driven security approaches, these works are not integrated in the current software architecture design process. On the other hand, some approaches propose ways of integrating security requirements in ADLs. Given the fact that these approaches tackle security at a very abstract level, they are not able to effectively support a model-driven security approach. In this work, we want to propose a model-driven security approach that can be used to integrate the *C2* ADL with security requirements.

6 CASE STUDY

The management of security requirements is a critical task specially in modern application. In order to aid the *Security Administrator* in this task, we implemented a prototype that permits to start from business security requirements, transforms them into a more concrete representation (i.e., policies), ties them to a representation of the elements in the software architecture, and produces a concrete configuration for each element involved in the policies. In order to point out the benefit of the tool we have defined the following case study. Figure 2 shows the software architecture of a distributed multi-user meeting scheduler. Usually, the architecture model is provided by the *System Architect*. Figure 3 shows the architecture modeled by means of UML.



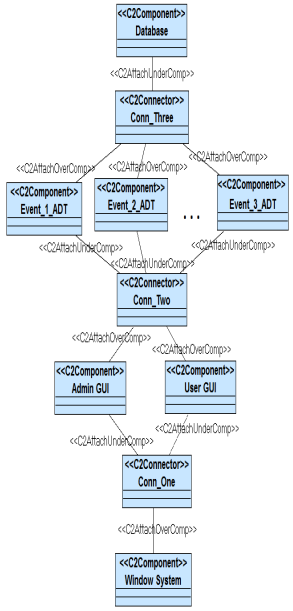


Figure 2: C2 Architecture. Figure 3: UML Architecture.

As described in Section 3 the architecture presented in Figure 2 can be easily translated into the corresponding UML class diagram with the use of the $\ll C2Component \gg$ and $\ll C2Connector \gg$ stereotypes. With this translation the *Security Administrator* has all the information needed to add the security requirements. According to the *ISAE 3402* standard, a business requirement that the application has to fulfill is the following: *Standards and policies exist to authenticate all administrative users*. This top level requirement exemplifies requirements related to authentication (and authorization) of administrative users. For example, this requirement can be refined into the following set of authorizations and authentications:

- access to *Admin GUI* requires authentication,
- the data exchanged through *Admin GUI* must be protected,
- the data exchanged between the web server and the Database must be protected,
- only the Administrative roles can create a new meeting,
- persistent data must be encrypted.

The requirements presented above lead also to the definition of a set of negative authorizations and authentications in order to make explicit the assumptions behind the requirement.

- the role *User* cannot create a new meeting,
- the *Admin GUI* is not accessible from the Internet,
- the role *User* must not access the *Admin GUI*,
- the data exchanged between the web server and the Database must be protected.

With the use of *Security4UML* the Security Administrator is able to represent the above policies in a more concrete way. Figure 4 shows the *ESec-Model* representing the following authorizations: (a) *only the Administrative roles can create a new meeting* (**authz-1**), (b) *the role User cannot create a new meeting* (**authz-2**) and (c) *the role User must not access the Admin GUI* (**authz-3**). These authorizations are not tied to the specific system, hence in order to add the security requirements to the software architecture, the Security Administrator has to refine the *Sec-Model* into the *ESec-Model* as described in Section 3. For instance, the Security Administrator may enrich the top-level action *Create* with information about the concrete action. If we assume that the new meeting corresponds to a new row in the *Meeting_Table*, then the corresponding concrete action will be an *Insert* into the database. Furthermore, in this phase, the Security Administrator can introduce additional properties in order to tie the authorizations and/or authentications with the technologies related to the architecture. For instance he/she can specify which cryptographic technique (e.g., 3DES, AES) is used for the encryption of the *Meeting_Table*.

The result of the Enrichment process above is a

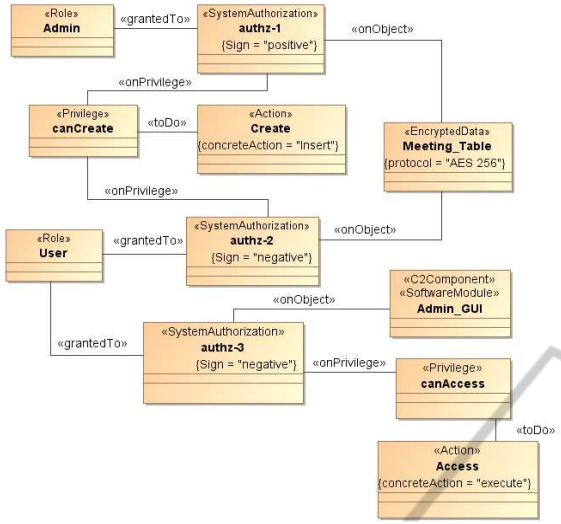


Figure 4: Security requirements.

set of policies related to the actual infrastructure and ready to be translated into the concrete configuration in order to fulfill the security requirements. After the *Enrichment phase* the Security Administrator can perform the *Derivation phase* in order to produce a set of scripts and configuration files in order to enforce the authorizations and authentications defined by means of *Security4UML*. The requirements *only the Administrative roles can create a new meeting* (**authz-1**), and *the role User cannot create a new meeting* (**authz-2**) are implemented by the following SQL scripts:

```
CREATE ROLE Admin LOGIN
CREATE ROLE User LOGIN
GRANT INSERT ON Meeting_Table TO Admin
REVOKE INSERT ON Meeting_Table TO User
```

7 CONCLUSIONS AND FUTURE WORK

The increasing complexity of current IT systems requires techniques that can be used to help security administrators and system architects in the design, implementation and deployment of secure software architectures. A viable way for handling this increasing complexity is by using model-driven security approaches, which can guide the user from the formal definition of security requirements to the concrete implementation of the security configuration.

In this paper we presented a model-driven security approach that can be used to introduce security requirements in software architectures, and to implement and deploy the security configuration in a semi-automated way. We have shown how a well-known ADL (C2) may be integrated with a representation of security requirements. We presented an overview of our *Security4UML* meta-model, and we have shown how this meta-model can be mapped on a UML profile. We also presented a case study in which we have modeled a distributed multi-user meeting scheduler application.

Our approach is currently based on UML class diagrams. This is not a restriction, because there are ways to represent Architectural Design Languages with UML models (Robbins et al., 1998). We plan in the future to integrate our model-driven approach in ADLs that are not UML-based.

REFERENCES

7safe (2010). UK security breach investigations report. Technical report, University of Bedfordshire.

Arrigoni Neri, M., Guarnieri, M., Magri, E., and Mutti, S. (2013). On the Notion of Redundancy in Access Control Policies. In *Proc. of SACMAT*.

Arrigoni Neri, M., Guarnieri, M., Magri, E., Mutti, S., and Paraboschi, S. (2012). Conflict Detection in Security Policies using Semantic Web Technology. In *Proc. of IEEE ESTEL - Security Track*.

Basin, D., Clavel, M., and Egea, M. (2011). A decade of model-driven security. In *Proc. of SACMAT*.

Basin, D., Clavel, M., Egea, M., and Schläpfer, M. (2010). Automatic generation of smart, security-aware GUI models. In *Proc. of ESSOS*.

Basin, D., Doser, J., and Lodderstedt, T. (2003). Model driven security for process-oriented systems. In *Proc. of SACMAT*.

Basin, D., Doser, J., and Lodderstedt, T. (2006). Model driven security: From UML models to access control infrastructures. *ACM Trans. Soft. Eng. Methodol.*, 15(1).

Guarnieri, M., Magri, E., and Mutti, S. (2012). Automated management and analysis of security policies using eclipse. In *Proc. of the Eclipse-IT* 2012.

Jürjens, J. (2003). *Secure Systems Development with UML*. Springer Berlin / Heidelberg.

Jürjens, J. (2005). Sound methods and effective tools for model-based security engineering with UML. In *Proc. of ICSE*.

Langevin, J., McCaul, M., Charney, S., and Raduege, H. (2008). Securing cyberspace for the 44th presidency. Technical report, DTIC Document.

Medvidovic, N. (1996). Formal modeling of software architectures at multiple levels of abstraction. In *Proc. of the California Software Symposium*.

Mouratidis, H., Kolp, M., Faulkner, S., and Giorgini, P. (2005). A secure architectural description language for agent systems. In *Proc. of AAMAS*. ACM.

Mutti, S., Neri, M. A., and Paraboschi, S. (2011). An eclipse plug-in for specifying security policies in modern information systems. In *Proc. of the Eclipse-IT 2011*.

Oladimeji, E., Supakkul, S., and Chung, L. (2007). A Model-driven Approach to Architecting Secure Software. In *Proc. of SEKE*.

Patterson, D. A. (2002). A simple way to estimate the cost of downtime. In *Proceedings of LISA, Usenix*.

Pertet, S. and Narasimhan, P. (2005). Causes of failures in web applications. *CMU Technical Report*.

Ren, J. and Taylor, R. N. (2005). A Secure Software Architecture Description Language. In *Proc. of SSATTM Workshop*.

Robbins, J., Medvidovic, N., Redmiles, D., and Rosenblum, D. (1998). Integrating architecture description languages with a standard design method. In *Proc. of ICSE*.

Sandhu, R. (1998). Role-based access control. *Advances in computers*, 46.

Taylor, R., Medvidovic, N., Anderson, K., Whitehead, E.J., J., Robbins, J., Nies, K., Oreizy, P., and Dubrow, D. (1996). A component- and message-based architectural style for GUI software. *IEEE Trans. on Soft. Eng.*, 22(6).