Using a variety of patterns in a secure software development methodology

Eduardo B. Fernandez
Dept. of CS and EE
Florida Atlantic University
USA
fernande@fau.edu

Nobukazu Yoshioka
GRACE Center
Nat.Inst. of Informatics
Japan
nobukazu@nii.ac.jp

Abstract— **Building secure software systems requires the application of a systematic methodology. A security methodology includes a security process and a conceptual security framework consisting of security artifacts such as patterns. In this work we consider systems designed using patterns. In previous work we proposed a secure systems development methodology that uses security patterns. This methodology applies security throughout the whole lifecycle and considers all architectural levels. As part of this work we have produced a variety of security patterns. As it is difficult for designers to select security patterns, we proposed SSFs (Security Solution Frames), which are hierarchical combinations of related patterns. We introduce now a new artifact, the Security Cluster, an application-oriented combination of SSFs which further facilitates the use of security patterns to build secure applications. We also present a metamodel to get a perspective of the use of these artifacts.**

Keywords— Systems security, secure software development, security patterns, software architecture, software security

I. INTRODUCTION

Building a secure system requires the application of a systematic methodology. A variety of methodologies to build secure systems have been proposed [27]. A security methodology includes a security process and a conceptual security framework consisting of security artifacts such as security patterns [26,29]. A pattern describes a solution to a recurrent software or systems problem in a given context; security patterns provide solutions to security problems. Security patterns provide a way for guiding system designers who are not experts on security to build secure systems. We have proposed a secure systems development methodology that uses security patterns and we have extended it recently [9, 29]. This methodology applies security throughout the whole lifecycle and considers all architectural levels. As a complement to this work we have produced a variety of security patterns [9].

Secure development methodologies apply security solutions throughout the whole development lifecycle. A number of those solutions have been proposed, including as artifacts security tactics, security patterns, aspects, arguments, formal methods, and others. We use security patterns and related types of patterns as our artifacts; however, the number of published security patterns is now close to 200 and it is hard for developers to select the patterns they need in a specific application. To help developers select patterns we proposed the concept of Security Solution Frame (SSF) [30], which groups together related patterns in horizontal and vertical sub-structures for a single high-level policy, e.g. authentication. We propose now a new artifact, a *Security Cluster* (SC), that goes further that SSFs by gathering a set of defense mechanisms, represented as patterns, to provide a conceptual unit that can be added to an application to make it secure. We provide an example of its use and we define a metamodel to relate the new artifact to existing artifacts we have used in our methodology to help its application. We consider the use of our patterns in possible Architectural Knowledge Management frameworks. The proposed SC can also be used in other methodologies.

Section II presents our view of building secure systems, indicating that we use a model-based approach and defining our variety of patterns. Section III describes some background. Section IV presents the Security Cluster, our main result. Section V considers the use of our artifacts in architecture repositories, while Section VI is a discussion of our ideas and describes related work. We end with conclusions in Section VII.

II. BUILDING SECURE SOFTWARE

From our analysis of the literature we have identified three basic approaches to build secure systems (Figure 1), which include:

- *Theoretical models*—attempt to verify security properties by using model checking but make many assumptions which may not be true in practice and are limited in the size of the systems that can be handled. *Cryptographic methods* are a variety of theoretical models but are effective only for specific aspects such as system or message authentication, secure transmission of messages, and storage protection. They cannot stop attacks based on code or design flaws.
- *Code-based methods* cannot find all vulnerabilities and many attacks exploit system interactions, not code flaws. Another problem is the complexity of code, large systems may have tens of millions of lines of code. Furthermore, the code changes more often than models.

- *Model-based security* tries to build a strong design structure where conceptual models describe the different units of a system and their interactions. Overall, the architecture of a system has a much larger effect on security than code vulnerabilities or the security of specific units, which makes these methods more effective in practice. Their use of abstraction is very valuable to handle complex systems.

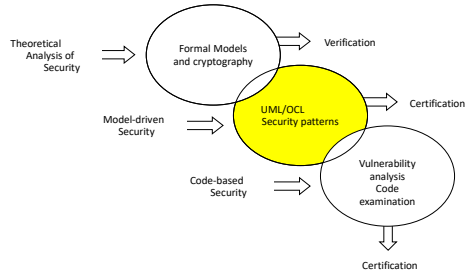

Figure 1. Approaches to security

We prefer to look for design vulnerabilities and structure a robust architecture instead of looking for code flaws. We believe that the best way to approach the security problem is using models. Because of their abstraction properties, models provide a way to apply a holistic approach to system security and they are useful to handle large and complex systems in a comprehensive and unified way. Of course, these approaches are complementary, but models should define the basic structure of any methodology. Note that the ovals in Figure 1 are not disjoint; i.e., a model-based approach may need to consider formal proofs for critical units and code-based analysis for heavily-used code sections.

III. SECURE SOFTWARE

A. *Applications and patterns*

An application is software intended to perform some business or useful function, such as student registration, managing accounts in a bank, and similar. An application implements a set of use cases, where each use case is composed of a set of activities, where each activity performs a specific action of the use case. For example, opening an account in a financial institution requires activities to provide customer information, followed by a manager creating an account and authorizations for the use of the account. These authorizations are materialized in the form of physical cards. In some places, an initial customer deposit is also required. Each activity may create or use some data, which constitute assets because they have a value for the institution. Reading or modifying these assets are usually the attacket goals.

The logics of the application can be described using Analysis patterns (APs), which can be used to build a conceptual model of the application functions. Another source for the semantics of the application is the use of Domain Models (DMs). A Domain Model (DM) is a conceptual model of an area of knowledge, e.g. finance, and has no software concepts. DMs are often described using ontologies [19] and we can think of a DM as a compound analysis pattern including several simpler analysis patterns that represent specific aspects of the domain. There is a good number of ontology patterns that describe a variety of semantic aspects in different domains [19, 23]. Ontology patterns can be converted into analysis patterns.

A pattern is a solution to a recurrent problem in a given context [3]. Patterns are described using a template composed of a set of structured sections. A problem section describes a general problem and forces that constrain and define guidelines for the solution, e.g., “some actions must be transparent to the users”. The solution is usually expressed using UML class, sequence, state, and activity diagrams (although we usually don’t need all these models). A set of consequences indicate what is the effect of the pattern and how well the forces were satisfied by the solution, including advantages and disadvantages of using the pattern. An implementation section provides hints on how to use the pattern in an application, indicating what steps are needed and possible realizations. A section on “Known uses” lists real systems where this solution has been used previously, i.e., a pattern is an abstraction of good practices. A section on related patterns indicates other patterns that complement the pattern or that provide alternative solutions. A pattern embodies the knowledge and experience of software developers and can be reused in new applications; carefully-designed patterns implicitly apply good design principles. Patterns are also good for communication between designers and to evaluate and reengineer existing systems. While initially developed for software, patterns can describe hardware, physical entities, and combinations of these, as well as non-technical processes such as teaching a course or organizing a conference. Pattern solutions are suggestions, not plug-ins or software components. In particular, security patterns can suggest solutions to designers who don’t have much security experience. Abstract Security patterns (ASPs) describe conceptual (no implementation aspects) security mechanisms that realize one or more security policies able to handle a threat or comply with a security-related regulation or institutional policy [10]. ASPs are used in the early lifecycle stages.

The use of Reference Architectures (RAs) can simplify the application of patterns and thus the construction and evaluation of secure systems. A Reference Architecture (RA) is a generic software architecture, based on one or more domains, with no implementation aspects [1, 25]. An RA is reusable, extendable, and configurable; that is, it is a kind of pattern for whole architectures and it can be instantiated into a specific software architecture by adding platform aspects [1]. We can build RAs using patterns. After adding security patterns to neutralize

identified threats in an RA we have a Security Reference Architecture (SRA), and we recently produced a SRA for clouds [11]. Complete RAs or SRAs can be catalogued and used in conjunction with appropriate development environments.

An intermediate construct between patterns, SSFs and RAs is a *Secure Semantic Analysis Pattern* (SSAP) [5]. A SSAP adds security patterns to a semantic unit based on a set of related use cases. We have built a few of these, including legal trials and medical records [9].

We use UML for describing patterns. UML is a semiformal language whose syntax is formally defined using a metamodel [21]. It is widely used, many tools support its use, it is an industry standard, and it is familiar to a wide segment of practitioners. It can also be complemented with formal methods, and its standard defines an associated formal language, OCL. Being a graphic language, it is highly intuitive and has a direct correspondence to code. There exists an extensive literature on design and security patterns, and most of them describe their solutions using UML.

B. Threat modeling

Table 1 shows examples of vulnerabilities and threats. They refer to attacks to unstructured cloud storage, such as AWS S3.

Table 1. Examples of vulnerabilities and threats

Vulnerability	Threat	Misuse
System misconfiguration	Access cloud storage	Unauthorized reading or writing of data
Expose access keys	Access cloud storage	Unauthorized reading or writing of data
Unrestricted cloud access	Store poisonous URLs in the cloud	Attacks to other web sites

Most methodologies, including [14], use misuse cases. We think that misuse cases are too coarse, a use case is not atomic but can have a good number of activities; we prefer to analyze each activity to see how it can be subverted [9].

In order to describe attacks we defined another type of pattern: A *misuse pattern* describes, from the point of view of an attacker, a generic way of performing a misuse (such as a violation of confidentiality or integrity) that takes advantage of the specific architecture and vulnerabilities of some environment [7]. Until now there is only one catalog of threat patterns [28] and several misuse patterns [9, 20], although the concept has been studied in some detail [7]. Misuse patterns define the environment where the attack is performed, countermeasures to stop it, and provide forensic information in order to trace the attack once it happens. For example, a security defense misconfiguration is a vulnerability, taking advantage of this vulnerability is a threat (potential attack) which can lead to reading unauthorized information (a misuse). In particular, misuse patterns are useful for developers because once they determine that a possible attack can happen in the environment, the pattern will indicate what security mechanisms are needed as countermeasures. Also, misuse patterns can be useful for forensic examiners to find evidence information after the attack has been performed. Finally, they can be used to evaluate if an existing system can handle specific threats. Note that a misuse pattern describes a complete attack, e.g., stealing information from a database [8], not just specific steps used to perform the attack, such as SQL injection or buffer overflow (both can be used in the same attack or individually in many attacks.

IV. Security Clusters

We introduce here the concept of Security Cluster, starting from Secure Solution Frames (SSFs), defined earlier. A SSF can help a designer select patterns for all the architectural levels of the application. We have described SSFs for Authorization [30], Cryptography, and Authentication. SSFs partition the solution space horizontally into *Pattern Families,* which are collections of related patterns. SSFs correspond to full realizations of security tactics [2] and can be related to other SSFs to secure a particular application. We only consider here SSFs that provide complementary defenses; other relationships are discussed in [30]. We draw SSFs using pattern diagrams [3]. A pattern diagram shows patterns as rounded rectangles where directed edges show the contribution of a pattern to another. Figure 2 shows a SSF for Authentication.

Another advantage of SSFs is that they can be used to classify security patterns. There are now about 200 security patterns but many are just renaming or translation of patterns to a different architectural level. SSFs can structure patterns of the same type by emphasizing their common concerns.

Security patterns can be organized as SSFs and combined with other artefacts. In particular, specific patterns in SSFs can be related to specific patterns in another SSF. We can analyze use cases to determine the threats to each application asset and define a set of specific SSFs to control these threats, then we can combine the corresponding SSFs.

A *Security Cluster* (SC) is a selection of patterns from differente SSFs. Formally: SCa= {SSFi.pa, SSFj.pb, SSFk.pc,…}, where cluster SCa combines patterns where SSF.pi denotes pattern i in a SSF. SCs can be catalogued by defining a start SSF and using it as index.

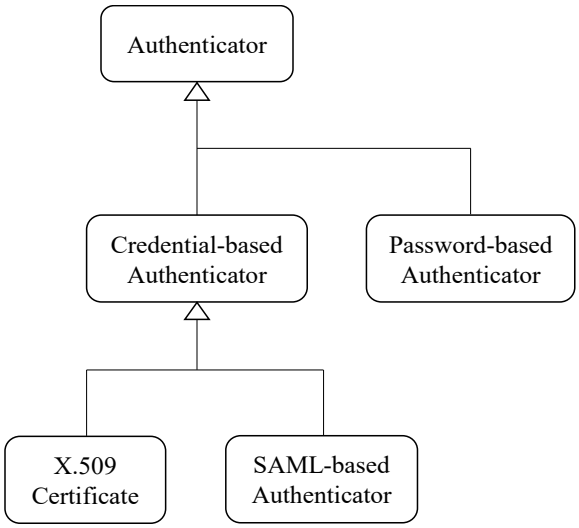

Figure 2. Authentication SSF

Fig. 3 (a set of pattern diagrams representing SSFs) shows the construction of a Security Cluster, SC1. To define SC1 the designer decided to use credentials as authentication artefact, then selected Attribute-Based Access Control (ABAC), for securing its communications she chose the Advanced Encryption Standard, a symmetric encryption algorithm, and finally used Distributed Logging. This specific selection was based on the analysis of the expected threats of this application, obtained by the method described in Section II. As we hve shown elsewhere we can map threats to security patterns that can stop them [9, 29].

Fig. 4 shows this SC used to secure an analysis pattern for Accounts where Customers can perform Transactions [6]. SC1 defines here credential-based authentication for Customers, ABAC for Customers accessing their accounts, a Distributed Logger/Auditor to record Transactions, and the use of Symmetric Cryptography using AES for the related communications.

In a catalog, each SC description should include recommended applications or include analysis patterns where it would fit, as shown in Figure 3. Conversely, each security pattern description could come with several SCs, each one appropriate for different environments; for example, the Account analysis pattern of Fig. 4 is appropriate for distributed, high security environments; another version could include an SC using passwords, RBAC, DES, and Centralized Logging. We can also make the SCs more complex, including patterns to protect security information, for filtering, or for secure storage. However, in this latter case, these SCs would be less reusable. SCs would be built by security experts and would make the work of software developers much simpler.

As indicated, we should not make the SC too specific. For example, there are many varieties of authorization models. To help designers choose in those cases we can use pattern diagrams that expand parts of SCs. As an illustration of how pattern diagrams can be used with SCs, Figure 5 shows some variations of access control models. The double-lined patterns show the intended selecions. The most basic access control model is the access matrix (Basic Authorzation). This model includes the tuple {s,o,t}, where s indicates a subject or active entity, o is the protected object or resource, and t indicates the type of access permitted. In that model users are allowed to delegate their rights (discretionary property, delegatable authorization), implying a tuple {s,o,t,f}, where f is a Boolean copy flag indicating if the right is allowed to be delegated or not. A predicate added to the basic rule allows content-based authorization, becoming {s,o,t,p,f}, where p is the predicate

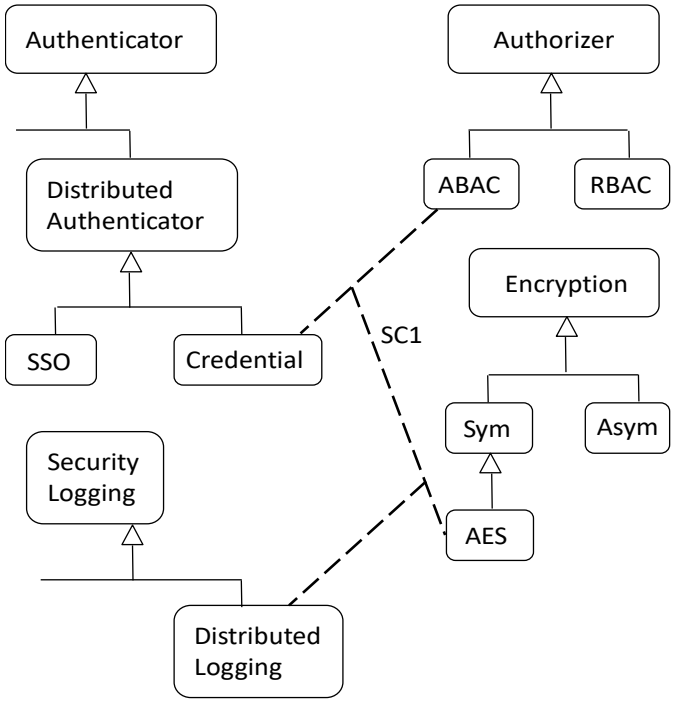

Figure 3. Defining a SC

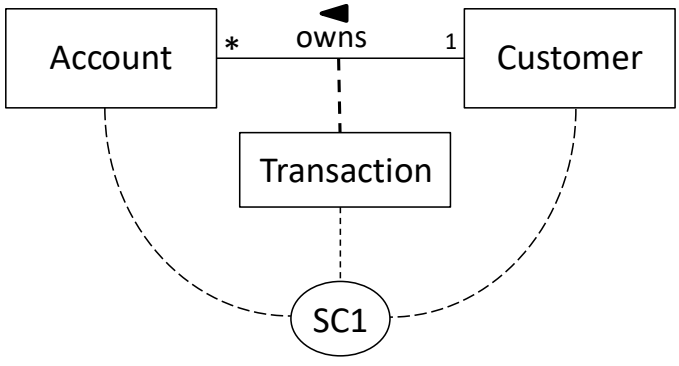

Figure 4. A use of SC1

(the predicate can also include environment variables). The rule could also include the concept of Authorizer (a), becoming {a,s,o,t,p,f} (Explicitly Granted Authorization). Role-Based Access Control (RBAC) can be considered a special

interpretation of the basic authorization model, where subjects are roles instead of individual users. Several variations and extensions of these models have appeared. Attribute-Based Access Control (ABAC), is the most flexible model, where any attribute values can be used to decide access. This diagram can be the starting point that allows a designer to select the type of access control he needs in his application.

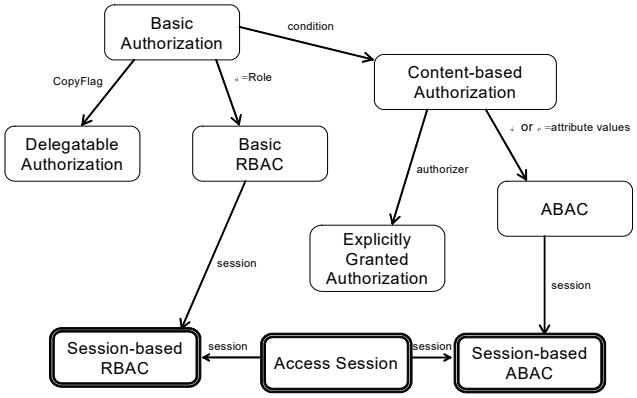

Figure 5. Access control patterns

V. ARCHITECTURAL KNOWLEDGE MANAGEMENT

Figure 6 shows a metamodel that places in perspective the use of the different artifacts we have proposed for building a secure application. The classes in blue describe the structure of the application as described in Section II. The second level of the metamodel (classes in red) describe the concepts related to security attacks: assets have vulnerabilities that can be exploited by attacks realizing threats. The third level (classes in yellow), describe the countermeasures to the threats. In all stages, classes in yellow identify artifacts that need to be catalogued to serve as a guideline to understand threats and apply defenses. Security patterns are realizations of policies and tactics [2] but these are not shown in Fig. 6.

To be effective all these artifacts must be organized as an Architectural Knowledge Management (AKM) tool [4]. As indicated earlier, security must be developed along the lifecycle together with the functional (semantic) aspects of the application. The application of specific patterns in an architecture defines some of the most important design decisions. The main uses of an AKM are: sharing of the development activities among the stakeholders; compliance of the design with specific quality requirements; discovery, where the designer is helped by the tool to apply specific artifacts, and traceability, to evaluate the impact of changes. There is now a variety of tools [4] but in general, they do not support security artifacts or a particular secure software development methodology.

A tool to support the security aspects of an AKM is needed. The repository of such a tool should follow the IEEE standard for software architecture to be compatible with existing tools. An important use of this tool is for assurance purposes. Assurance is a proof that a product is secure, according to some definition of security [22]. Cyber-physical systems require to consider also safety and reliability, which means that more artifacts should be included in such a tool [24]. The metamodels presented above can guide the structure of the repository. Such a tool should also support BPMN models which are important complements to describe requirements. Having an explicit record of the application of artifacts in an architecture makes this process much more systematic and convenient.

VI. DISCUSSION AND RELATED WORK

As indicated earlier, security patterns encapsulate solutions that can stop or mitigate specific threats and their consequent misuses. This means that each pattern added to the system may contribute to the total security of the system. However, adding security patterns that do not stop threats would lead to systems which are very slow, expensive, and hard to maintain. It is clear that security patterns need a guiding methodology to be effective and several pattern-based methodologies have been proposed [26]; we mentioned ours earlier, a few others are mentioned below.

A general methodology for developing security-critical software has been proposed in [15]. It makes use of an extension of the Unified Modeling Language (UML), UMLSec, to include security-relevant information. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLSec models against security requirements [16]. The analysis is based on model-checking specific portions of a system. This methodology annotates the model classes with security indications, it does not use special artifacts.

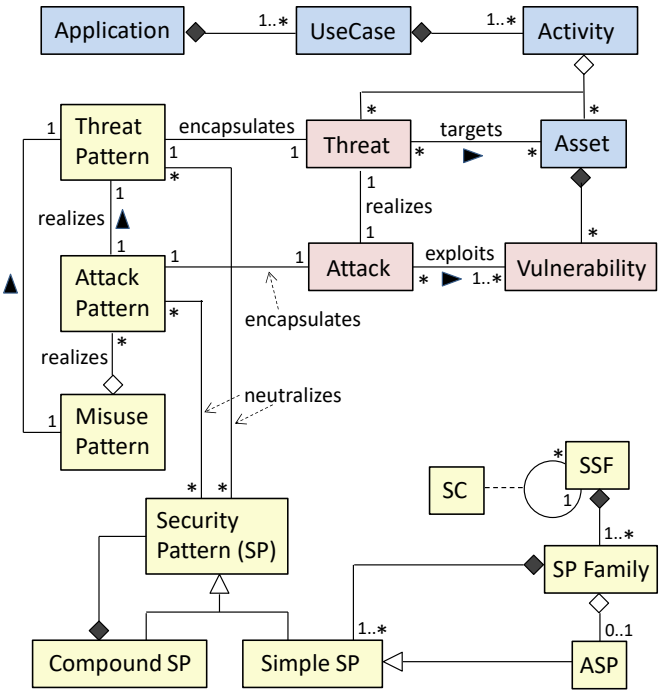

Figure 6. A metamodel for security concepts

Mouratidis and his group use a special methodology, Secure Tropos, to model security [17] Their work started

modeling requirements but they have also considered other stages; for example how to test security along the lifecycle. Instead of UML they use special diagrams and they use patterns described in their style.

Hazeyama et al. [14], include a variety of artifacts as part of a Security Knowledge Base. They use CLASP (the OWASP methodology) and their own methodology combined with a knowledge base. They use security patterns, attack patterns, and misuse cases as artifacts.

Hamid [13] built a methodology and repository of security artifacts. A metamodel for security and privacy was presented in [32]; that model is more general than ours in conceptual scope but less general in that it applies only to clouds. An attempt to add extensions to UML to describe security concerns is shown in [18], but it is mostly a notation, it does not include new artifacts. A metamodel is used in [12] to structure a repository for security patterns; their ontology includes threats and attackers but its only artifacts are security patterns. Nagaratman et al. [18] describe an AKM used by IBM, which uses security patterns to let developers manage development and monitor executing systems.

In all these works there is emphasis on the general use of the knowledge bases and on their implementation details but not much concern about having a variety of artifacts to support different aspects of the lifecycle or different architectural levels. We believe a greater variety of artifacts is valuable.

Examining several specific methodologies we have found that they use few artifacts, and they could be enhanced by using more varieties. In fact, in the final system each security pattern will become a COTS component. Security patterns are not intended to be coded except by the producers of the corresponding mechanisms. Well-defined patterns make this selection easier. In this sense, security patterns are quite different from design patterns.

Our approach is against some of the principles of agile development processes, which emphasize producing code with little or no use of models; while those methods are clearly faster they are not appropriate to build secure or complex systems.

VII. CONCLUSIONS

We introduced the concept of Security Clusters, which appear as a good way to facilitate the work of software developers who are not security experts. We have shown here a variety of artifacts but they must be catalogued systematically and stored in some knowledge base to be effective. Although we have used a specific methodology as example, these artifacts can be used in other methodologies using patterns [26], and can even complement methodologies that do not use patterns [27]. Note also that system programs such as operating systems and database systems can also be built using these methods; they are in fact, just specialized applications.

Applications must not only protect their data but also comply with regulations; compliance requires systematic and complete logging as well as access control to individual's information and artifact-based information makes compliance simpler and more transparent [33].

Future work will include the development of a tool including a repository to store and manage artifacts. We have written a partial catalog of security and misuse patterns [9] but we need to extend it and support it through software functions.

ACKNOWLEDGMENTS

The work of Eduardo Fernandez was partially supported by the National Institute of Informatics of Japan. The reviewers provided valuable comments.

REFERENCES

[1] P. Avgeriou, "Describing, instantiating and evaluating a reference architecture: A case study", Enterprise Architecture Journal, June 2003.
[2] L. Bass, P. Clements, and R. Kazman, Software architecture in practice (3rd Ed), Addison-Wesley 2012.
[3] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. Pattern-Oriented Software Architecture: A System of Patterns, Vol. 1. J. Wiley, 1996.
[4] R. Capilla, A. Jansen, A. Tang, P. Avgeriou, M.A. Babar, "Ten years of software architecture knowledge management: Practice and future", The J. of Systs. and Software, 2016, 191-205. DOI: 10.1016/j.jss.2015.08.054
[5] E.B. Fernandez and X. Yuan, "Semantic Analysis Patterns", Procs. 19th Int. Conf. on Conceptual Modeling, ER2000 , Salt Lake City, UT, October 2000. Lecture Notes in Computer Science, Volume: 1920 , 183-195
[6] E.B.Fernandez and Y.Liu, "The Account Analysis Pattern", Procs. of EuroPLoP (Pattern Languages of Programs, 2002.
[7] E.B. Fernandez, N. Yoshioka and H. Washizaki, "Modeling misuse patterns", Procs. of the 4th Int. Workshop on Dependability Aspects of Data Warehousing and Mining Applications (DAWAM 2009), in conjunction with the 4th Int.Conf. on Availability, Reliability, and Security (ARES 2009). March 16-19, 2009, Fukuoka, Japan.
[8] E B. Fernandez, E.Alder, R.Bagley, and S. Paghdar, "A Misuse Pattern for Retrieving Data from a Database Using SQL Injection" , RISE'12, Workshop on Redefining and Integrating Security Engineering, part of the ASE Int. Conf. on Cyber Security, Washington, DC, December 12-14, 2012.
[9] E.B.Fernandez, Security patterns in practice: Building secure architectures using software patterns". Wiley Series on Software Design Patterns. 2013
[10] E.B.Fernandez, N. Yoshioka, H. Washizaki, and J. Yoder, "Abstract security patterns for requirements specification and analysis of secure systems", Procs. of the WER 2014 conference, a track of the 17th Ibero-American Conf. on Soft. Eng.(CIbSE 2014), Pucon, Chile, April 2014
[11] E.B.Fernandez, Raul Monge, and Keiko Hashizume, "Building a security reference architecture for cloud systems", Requirements Engineering. Doi: 10.1007/s00766-014-0218-7, June 2016, Volume 21, Issue 2, pp 225-249
[12] Gymnopoulos, L., Karyda, M., Balopoulos, T., Dritsas, S., Kokolakis, S., Lambrinoudakis, C., Gritzalis, S.: Developing a Security Patterns Repository for Secure Applications De-sign. In: Proceedings of the 5th European Conf. on Information Warfare and Security (ECIW 2006), Helsinki, Finland (2006)
[13] B. Hamid, D. Weber, "Engineering secure systems: Models, patterns and empirical evaluation", Computers & Security, 77 (2018), 315-348. https://doi.org/10.1016/j.cose.2018.03.016
[14] A. Hazeyama et al., "Security requirements modeling support system using software security knowledge base", 42nd IEEE Int. Conf. on Comp. Software & Applications, 2018.
[15] J. Jurjens, Secure systems development with UML, Springer-Verlag, 2004.

[16] J. Jurjens, “Sound methods and effective tools for model-based security engineering with UML”, 27th International Conference on Software Engineering (ICSE 2005), ACM, 2005, pp. 322-331.
[17] H. Mouratidis, P. Giorgini, G. Manson. “When Security Meets Software Engineering: a Case of Modelling Secure Information Systems”. Inf. Syst., Elsevier Science Ltd., Oxford, UK, v. 30, n. 8, .609-629, 2005.
[18] N. Nagaratnam, A. Nadalin, M. Hondo, M. McIntosh, and P. Austel, “Business-driven application security: from modeling to managing secure applications”, IBM Systems Journal, 44(4), 2005, 847-867.
[19] http://Ontology design patterns.org
[20] J. Pelaez, E.B.Fernandez, and M.M. Larrondo-Petrie, "Misuse patterns in VoIP", Security and Communication Networks Journal. Wiley, Volume 2, Issue 6, November/December 2009, 635–653, DOI: 10.1002/sec.105
[21] J. Rumbaugh, I. Jacobson, and G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Boston, Mass., 1999.
[22] Reijo M. Savola: “Quality of security metrics and measurements”. Computers & Security 37: 78-90 (2013)
[23] L Sion, K. Yskout, et al., “MASC: Modelling architectural security concerns”, 2015 IEEE/ACM 7th. Int. Workshop on Modeling in Software Engoineering”, 36-41.
[24] I. Sljivo, B. Gallina, “Building multiple-viewpoint assurance cases using assumption/guarantee contracts”, ISSA’16, November 2016, Copenhaguen, Denmark. DOI:10.1145/1235
[251] R.N.Taylor,N. Medvidovic, and N.Dashofy. Software architecture: Foundation, theory, and practice, Wiley, 2010.
[26] A.V. Uzunov, E.B. Fernandez & K. Falkner (2012), "Securing distributed systems using patterns: A survey", Computers & Security, 31(5), 681 - 703. doi:10.1016/j.cose.2012.04.005
[27] A. V. Uzunov, E.B.Fernandez, and K. Falkner, “Engineering Security into Distributed Systems: A Survey of Methodologies”, Journal of Universal Computer Science, Vol. 18, No. 20, 2013, pp. 2920-3006 http://www.jucs.org/jucs_18_20/engineering_security_into_distributed
[28] A. V.Uzunov and E.B.Fernandez, “An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems”- Special Issue on Security in Information Systems of the Journal of Computer Standards & Interfaces. 2013. http://dx.doi.org/10.1016/j.csi.2013.12.008
[29] Anton Uzunov, E. B Fernandez, Katrina Falkner, “ASE: A Comprehensive Pattern- Driven Security Methodology for Distributed Systems”, J. of Comp. Standards & Interfaces, Vol. 41, Sept. 2015, 112-137
[30]Anton Uzunov, E. B Fernandez, Katrina Falkner, “Security solution frames and security patterns for authorization in distributed, collaborative systems”, *Computers & Security*, 55, 2015, pp. 193-234, doi: 10.1016/j.cose.2015.08.003
[31] Anton V. Uzunov, Eduardo B. Fernandez, Katrina Falkner, “Assessing and Improving the Quality of Security Methodologies for Distributed Systems”, accepted for the *Journal of Software: Evolution and Process*
[32] H. Washizaki et al. , “A metamodel for security and privacy knowledge in cloud services”, 2016 IEEE World Congress on Services (SERVICES) DOI**:** 10.1109/SERVICES.2016.30
[33] D. Yimam and E. B. Fernandez, “Building Compliance and Security Reference Architectures for Cloud Systems”, IEEE Int. Conf. On Cloud Engineering (IC2E) 2016, Berlin, April 4-8, 2016.