Revisiting architectural tactics for security

Eduardo B. Fernandez
Universidad Técnica Federico Santa María, Valparaíso, Chile
edfernan@inf.utfsm.cl
(On leave from Florida Atlantic University, Boca Raton, FL, USA)
Hernán Astudillo
Departamento de Informática, Universidad Técnica Federico Santa María, Valparaíso, Chile
hernan@inf.utfsm.cl
Gilberto Pedraza-García
Universidad de Los Andes, Bogotá, Colombia
g.pedraza56@uniandes.edu.co
Universidad Piloto, Bogotá, Colombia

ABSTRACT

Architectural tactics are design decisions intended to improve some system quality factor. Since their initial formulation, they have been formalized, compared with patterns and associated to styles. However, the initial set of tactics for security has only been refined once. We have examined the tactics set and classification from the viewpoint of security research, and concluded that some tactics would be better described as principles or policies, some are not needed, and others do not cover the functions needed to secure systems, which makes them not very useful for designers. We propose here a refined set and classification of architectural tactics for security, which we consider more appropriate than the original and the previously refined sets. We also suggest a possible realization for this modified set. Finally, we conclude that patterns can be complementary and not alternatives because they can be used together: patterns can realize tactics.

Keywords: Architecture tactics, security patterns, secure architectures, secure software development

I. Introduction

Secure systems are notoriously hard to build; like most global system quality criteria, a piecemeal approach based on securing system elements is simply inappropriate. Since design decisions about incorporating security mechanisms have a global effect, no local optimizations are possible. From a security research standpoint, in the lack of quantitative measures, a secure system is one that can be shown to withstand a variety of attacks, and although many approaches to build secure systems have been proposed [17], they usually focus on some specific aspect, e.g. authorization, that can control only a type of threats. Security-related decisions also interact with other quality attributes, e.g. availability, and separate optimization is not possible.

The security research community has considered many ways to secure specific parts of a system, to build secure systems, or to stop specific attacks, but few studies exist about how to make secure a whole system [7, 15, 22]. On the other hand, software architecture research has addressed security as yet another global quality property, and to a large degree they have ignored the work on security research. The best known software architecture approach to defining secure systems (and other global quality attributes) is based on tactics.

Originally introduced in 2003 [2], architectural tactics are described as "measures" or "decisions" taken to improve some quality factor, a definition later refined to "architectural building blocks from which architectural patterns are created" [3]. Each tactic corresponds to a design decision with respect to a quality factor. Architectural tactics codify and record best practices for achieving some quality attribute. However, no specific

justification or validation of the selected set of security tactics was given. No specific realization for those building blocks was given either.

Security differs from other quality factors like availability or scalability in its close tie to the application semantics; e.g., only an account owner can withdraw money from it, or a process cannot write outside its own virtual address space. Also, external regulations may prescribe (directly or indirectly) specific information protection needs, leaving no space for possible tradeoffs among different tactics.

Since their initial formulation, tactics have been formalized [1], compared with patterns [14], associated to the Common Criteria [12] and associated to styles [10]. However, the initial set of tactics for security [2, 3] has only been refined once [15]. We believe that for an effective use in building secure systems the original set needs revisiting but we do not agree with the results of the recent refinement of [15]. This article presents a reasoned examination, pruning and reclassification of architectural tactics for security, considering both the original set and the refined set of [15]. We also consider a possible realization using security patterns; without a convenient realization, tactics do not provide enough guidance to architects.

The remainder of the article is organized as follows. Section 2 discusses the use of tactics for building secure architectures; Section 3 defines security principles and policies, terms used for secure systems design; Section 4 examines the initial (and still used) tactics tree and prunes it; Section 5 presents some new tactics; Section 6 proposes a realization for tactics using security patterns, while Section 7 discusses related work. We end with some conclusions.

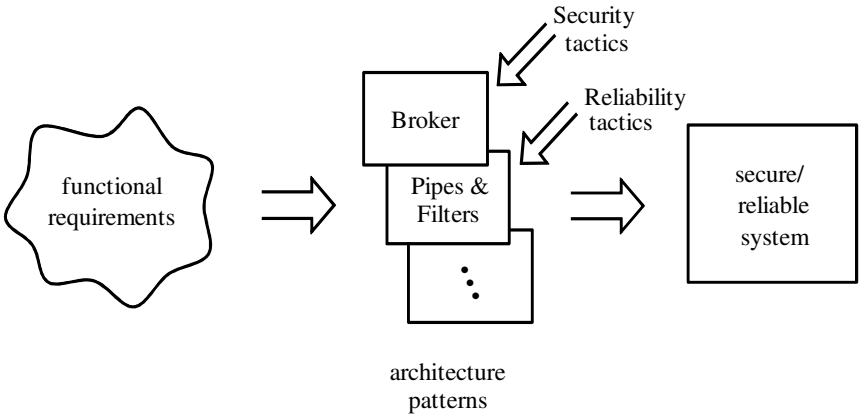

Fig. 1. Producing secure or reliable architectures from tactics.

II. Building Secure Architectures

The literature records several approaches to use architectural tactics to build secure systems. Harrison and Avgeriou [10] propose (see Figure 1) that the architecture be first defined using architecture patterns to determine the structural aspects of the functional requirements, and then apply tactics to introduce non-functional aspects such as security and reliability. The articles with this proposal do not evaluate the actual level of security or reliability thus obtained, or whether different realizations may yield unnecessary security mechanisms.

A more serious problem is the lack of guidance to complete the system implementation. Because tactics do not prescribe any realization, several researchers have introduced their own realizations. Kim [11] reifies tactics as reusable UML building blocks that can be plugged in the architecture according to a list of non-functional requirements (NFRs). A similar approach is used by Bagheri [1], with tactics like "ping echo", which is clearly a specific mechanism (realizable using patterns), to detect faults. These approaches are in line with [13] and imply a plug-in or template way of applying realizations, contrary to the idea of design and to the concept of security patterns, which are generic solutions that include several sections indicating the conditions to apply the pattern as well as its consequences. Patterns are not plug-ins, but guidelines. We are still in the requirements stage (or

coming out of it) and it is too early to commit to precise solutions, which leave out many alternative implementations. We propose a better approach in Section 6.

Another attempt to make tactics more precise is by formalizing them [1]. Formalizing a vague concept is misleading since there can be many ways to do it and the formalizer needs to make many assumptions. Real systems need to be designed. Design is not a mathematical or formal process but it requires experience and intuition from the designer. Often, system designers are not experts on security and selecting precise solutions is too hard for them. This means that premature formalization is not a good idea either. We can formalize specific parts, which are separable from the rest of the system; for the most part of the system, UML, which is a semi-formal approach, is a good choice, and patterns which are "suggested solutions" appear to be the right direction.

Patterns are encapsulated solutions to recurrent system problems and define a vocabulary that concisely expresses requirements and solutions without getting prematurely into implementation details [4]. Security patterns are a type of architecture pattern in that they usually describe global software architecture concepts, although some consider security patterns to be a type of design pattern as well. Finally, some security patterns are a type of analysis pattern in the sense that security constraints should be defined at the highest possible (semantic) level of the system. While there is no "official" template for security patterns, we use a variation of the POSA template [4], which is composed of a thumbnail of the problem it solves (a threat to the system), the context where the pattern is applicable, a brief description of how it solves the problem (stops or mitigates the threat), the static structure of the solution (usually UML class diagrams), the dynamic structure of the solution (usually UML sequence diagrams or possibly activity or state diagrams), and guidelines for the implementation of this pattern. The contents of all the sections of the template are fundamental for the correct use of the pattern. Patterns are not just solutions and are not plug-ins. However, their solutions require some level of concreteness to be useful.

The effect of a pattern on security, performance, or any other factor depends on how it is used; for example, applying authentication in many places in a system may increase security but reduces performance. There are tradeoffs when improving any quality factor. "Hide information" is a tactic that can be realized in at least two ways: cryptography and steganography. Depending on the application, one is more convenient than the other. If the architect is not experienced or knowledgeable about security, it is important to provide him with more detailed guidance.

III. Security Principles and Policies

Principles are general statements that define ways to produce good designs. The classical paper of Saltzer and Schroeder [16], defined a set of *security principles* that included among others: least privilege, separation of duty, and least common mechanism. Later, more principles have been added, including "Defense in depth", "Start from semantic levels", and others.

Policies are high-level guidelines defining how an institution conducts its activities in its business, professional, economic, social, and legal environment [9]. The *institution security policies* include laws, rules, and practices that regulate how an institution uses, manages and protects resources. *Regulations* are legal or government policies that must be reflected in the implemented system.

More concretely, policies are management instructions indicating a predetermined course of action or a way to handle a problem or situation. Every institution has a set of policies, explicit or implicit, some of which are security policies. *Security policies* are essential to build secure systems since they indicate what to protect and how much effort to invest in this protection. In general, policies come from regulations, institution practices, or just good principles of design, i.e. prescribing some quality aspects for the final product.

Policies are also used to indicate the way to avoid or mitigate threats; for example, a mutual authentication policy avoids impostors from either side. Each system uses a combination of policies according to its objectives and environment. As an example, two of the most common security policies used in practice are:

- *Open/closed systems* — In a closed system, nothing is accessible unless explicitly authorized, whereas in an open system everything is accessible unless explicitly denied. Institutions where information security is very important, such as banks, use a closed policy (e.g. "only an account's owner can access it"); institutions whose objective is disseminating information, such as libraries, use an open policy (e.g. "all books are accessible except rare books").
- *Least privilege (need to know)* — People or any active entity that needs to access computational resources must be given authorization only for those resources needed to perform their functions; e.g. "a secretary should not have access to product plans"; it also applies to the institution (e.g. "it should not collect more information than strictly necessary about its members"). This policy can be considered a refinement of the closed system policy.

Policies are prescriptive, and can be thought of as directions for designers. Policies can be structured into hierarchies, and more specific policies apply to the components elements of a system. It is possible to express individual policies using UML class diagrams and constraints.

In software architecture terms, policies are guidelines to apply tactics, which in turn prescribe a realization based (e.g.) on security patterns [7], as shown in Figure 2. Policies are applied as tactics which are implemented as patterns. The associations are many-to-many. For example, Figure 3 indicates a policy which prescribes that "only owners of accounts can access their accounts", which is translated into two more specific policies, for *Authentication* and for *Content-Dependent Authorization*, which can be realized by corresponding security patterns. In this example, the "content-dependent authorization" policy can prescribe the use of the tactic "Authorize actors", which would be realized by a "Content-dependent Authorizer" security patterns. Broad policies or tactics are usually obtained by combining several patterns; e.g. "Authorize users" can be obtained with the patterns *Authenticator*, *Authorizer*, and *Security Logger/Auditor* [7].

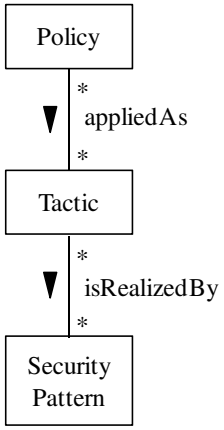

Fig. 2. From policies to security patterns.

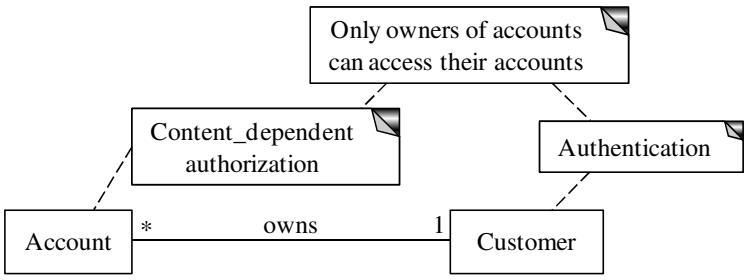

Fig. 3. Hierarchies of policies.

IV. Examining the Current Tactics

The original list of tactics is structured as a classification tree (see Figure 4); the tactics are the tree leaves and most of them are at the same level. Although security patterns can be classified [19] to cover all concerns, all the architectural levels of a system, and other facets with a multidimensional matrix, tactics are simpler and we will keep their tree structure. The branches of the current tree correspond to one of these dimensions. And we have changed "Resist attacks" to "Stop or mitigate attacks", which is closer to what security designers do. We start by removing some tactics considered not useful and later we add some new ones. This selection is based on our experience and knowledge of security so we are not sure if we have a complete set but we will obtain a set more appropriate to build secure systems than the original set. There is no formal way to prove that this is an optimal or minimal set but based on our recent work we can have some level of confidence that our set is appropriate [18]. We do not indicate how to perform the operations to realize tactics now, security patterns, discussed later, indicate possible ways of realizing, for example, "authenticate users".

Good security design requires that security is enforced at the system level. Policies about access must cover all applications using system resources. This means that, if there are several applications sharing the same platform all of them should share intrusion detection, firewalls, authentication, authorization enforcement, and logging. When a user of the application attempts to access some resource the system functions enforce that this access satisfies the security constraints of the whole system. Additional security constraints may be placed in applications but a base set of security mechanisms controls the execution of all applications. Even if there is only one application using the platform, this separation is important because of the need to decouple those aspects which are not intrinsic to a specific application. This was the same motivation that lead to aspect-oriented design [14]. All this means that an application architect only needs to verify that the system implements some tactics and does not need to include them in her design; only calls to those functions are needed so they are enforced during the application execution.

We start with the branch of Figure 4 that describes tactics to resist attacks (stop or mitigate attacks). "Identify actors" is not a tactic to resist attacks, it is a useful mechanism in distributed systems, necessary to implement authentication, authorization, and logging. In any case, it should be implemented as a global system function, not as part of a specific application.

Some tactics are really *principles*. Principles are not specific enough to become patterns or even tactics; there may be millions of solutions that satisfy a principle. This means that tactics that correspond to principles are not useful; the designer has no concrete guide about its realization. In the set of Figure 4 we can then eliminate as tactics: "Limit exposure", "Limit access", and "Separate entities". These three tactics are good recommendations for designers, but there are millions of ways to implement them. They are also not complete, why only these principles, "Need-to-know (least privilege)" is another very important security principle, but it is not included, "use a closed system" is another basic security principle not included. We also eliminated "Detect message delay" which is a way to detect some attacks, but if we include it we also need to include "detect abnormal behavior", "match traffic to known attacks", and many others. In other words, its inclusion in the original set is arbitrary.

As indicated, some proposed tactics are functions that apply to all applications, not to specific ones. As such, they don't need to be incorporated in each new application. Some like "Verify message integrity" could be left if the application needs to have its own way of applying cryptography. What security mechanism to put in an application and which ones should be left to the system requires experience and depends on the specific application, to imply that all applications need to incorporate these tactics is misleading.

Similarly, the proposed tactics for reacting to attacks are always system functions, which are implemented independently of any application, and don't need to be added to specific applications. The tactics to recover from attacks are also system functions: "Maintain audit trail" is clearly a system function that can be used to detect attacks and to recover from attacks. "Change default settings" is about reconfiguration and boot up; these are operational system functions, and do not belong in an application design.

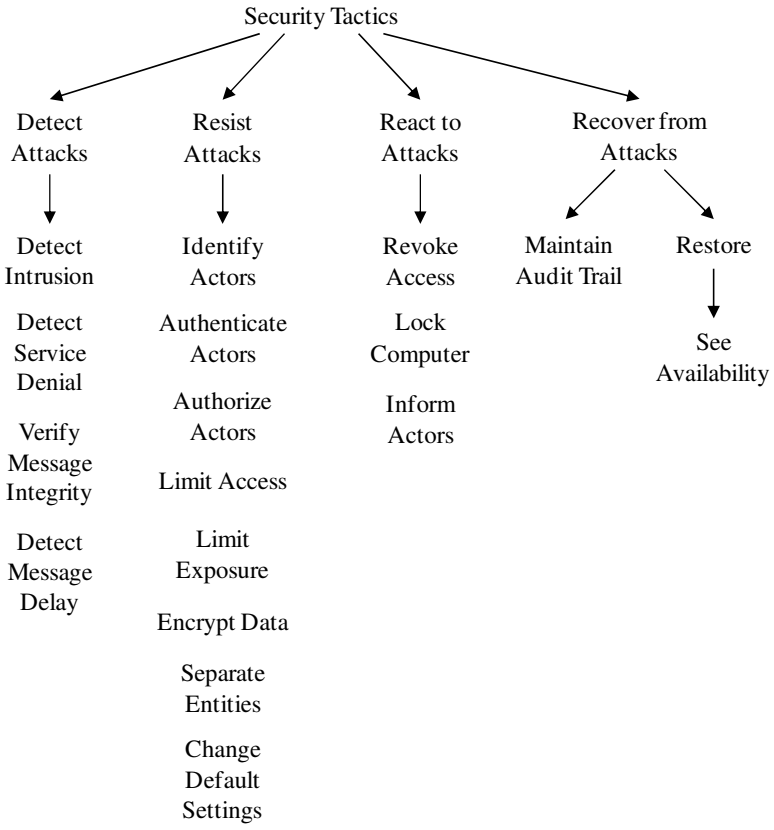

Fig. 4. Classification of security tactics [3].

V. New or Modified Tactics

A tactic should not be too general or too specific. If too general the designer has no guide which is the reason we eliminated principles; if too specific we have mechanisms instead of tactics. Some tactics must be applied together in some order, e.g. we have to apply Authentication, then Authorization, and then Access Control, in that order. Establish secure channel is required before we can hide data.

We have deleted all the tactics considered unnecessary and we need to add new tactics for the missing security aspects. These include: "Detect the origin of messages. "Encrypt data" should be changed to "Hide data" with two varieties: "Use cryptography" and "Use steganography". To enforce the rules defined in "Authorize users" we need a "Control access" tactic that would correspond to the security concept of Reference\Monitor [9].

We have also added in *Detect attacks* the tactic "Verify storage integrity", which indicates the need to define measures to make sure that databases have not been modified. In this branch, we also split "Identify intrusions" into "by signature" and "by behavior", the two standard ways to apply intrusion detection.For *React and recover from attacks*, the specific functions depend on institution policies, should be performed by the system, and it does not make sense to define general functions.

"Establish secure channel" is needed to provide secure communications in a distributed system [18]. Finally, we add the tactic "Manage security data", which includes the management of keys for cryptography, the secure storage of authorization rules, and other ways to handle security information. Some security patterns for this purpose are given in [18]. Obviously, this is a fundamental tactic to have a secure system and it should be performed by the system.

We also changed the word "user" or "actor" for "subject". According to standard security terminology [9], subject is an active entity that can request resources and includes humans and executing processes.

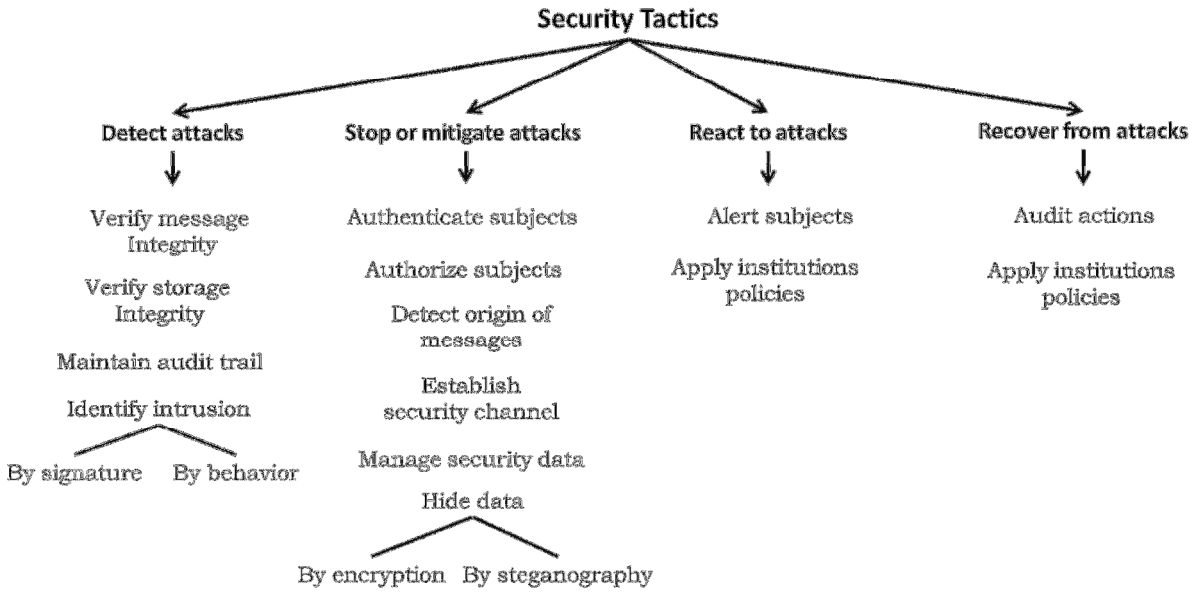

Fig. 5. A new set of tactics.

Figure 5 shows our final set of tactics. A very important aspect not mentioned in software architecture books or papers is that all of these tactics are to be implemented as shared system functions supporting all applications and should not be in general implemented in specific applications. A well known principle of security asserts that applications should not enforce their own security; this results in incomplete security with inconsistent enforcement [9, 20].

VI. Complementing Tactics with Security Patterns

After the previous discussion we can see that patterns and tactics are not parallel; patterns are well-defined structured entities while tactics are just a name. Moreover, tactics have a much broader scope than patterns because they can be applied simultaneously in multiple models and use several forms of implementation such as: introduce a new hardware or software security technology, add operational procedures to support secure operation or modify existing structures. They are really different concepts, not alternatives. As indicated, we can see tactics as a step leading to patterns but the selection of the right pattern takes more knowledge, some of which is in the sections of the pattern and some is in the classification of patterns. In fact, several methods exist to help designers use them appropriately [17]. Other realizations for tactics exist, e.g., generic security architectures and components [11], aspects [13], and S&D patterns [8]. Based on our experience, we believe that the most convenient realization of tactics is by using security patterns but more uses of them are needed to prove this point.

Tactics have value for other purposes and we should consider this value as well. Cañete uses tactics to annotate Jackson's problem frames [5]. The annotations are intended to provide arguments for the satisfaction of quality factors. [10] also uses them for annotations about design decisions, which appears as a valuable use.

Another possibility is to use tactics to build a secure development approach simpler and faster than methodologies based on threats. The methodology of [6] uses a process based on the Rational Unified Process. This process is considered too complex by many practitioners and a variety of agile methods have appeared. We want to see if it is possible to combine tactics with some steps of this methodology to produce an agile secure development

methodology to build secure systems. The methodology will also be enhanced using the new catalog of tactics to facilitate the selection of patterns by inexperienced designers. We will evaluate the new methodology in a new experiment where inexperienced designers and security experts will work together to produce a more secure system in a shorter time. The idea is to use a minimum of security experts. Evaluation of the final system will be based on analyzing how the final system can handle all the identified threats discovered in the use cases.

VII. Related Work and Discussion

Several authors [10, 11, 18, 19] have found relationship between security tactics and security patterns.

Kim et al. [11] proposes architectural tactics as reusable UML architectural building blocks that offer generic solutions to specific problems related to quality attributes, and tactics are represented as feature models to support decision making for non-functional requirements through a set of explicit solutions. This approach is too rigid and does not allow the designer the freedom provided by patterns; the choice of blocks is also limited since no catalogs of these building blocks exist.

Ryoo et al [14] and VanHilst et al [19] propose tactics as an intermediate architectural concept between high-level decisions and patterns of architecture, so architectural patterns implement architectural tactics. [14] defines a methodology to extract tactics from security patterns through activities such as reclassification of architectural patterns, decomposition of patterns, derivation of tactics hierarchies applying reasoning and intuition over patterns, and realization or instantiation of existing tactics. They evaluated several sets of security architectural patterns and applied a Delphi technique to produce a new security tactics hierarchy. Statistical approaches cannot produce appropriate tactics because they do not consider security knowledge. Their results include as tactics “Identify actors”, “Limit exposure”, and “Limit access”, which we removed from the updated list of tactics. Some redundancies remain: tactic “Maintain confidentiality” is as shown as separate, but Confidentiality requires using “Authentication” and “Authorization”, which are also tactics. They say that some "tactics have been misidentified as patterns" but the structure of tactics and patterns are very different, since tactics are building blocks but patterns include information on use. We prefer to stress their difference and find ways of using them together.

VIII. Conclusions

There is significant confusion in terminology, conceptual definitions, and use of patterns and tactics. This has led to methodologies that are difficult to combine with others and with unclear security results. We have tried here to make these concepts clearer by questioning the original set of tactics according to established security knowledge and showing the need for appropriate ways to realize tactics. Precise definition of these concepts should lead to better architectural knowledge and better methodologies to build secure systems. Our tactics are derived from our architectural and security knowledge; we do not claim they are complete but they are useful, especially when complemented with a good catalog of security patterns.

We believe that current methodologies to improve architectural quality using tactics cannot produce secure or reliable systems, unless they are complemented with appropriate realizations. We need an approach where we can specify these NFRs from the beginning of software development, considering the semantics of the application and where iterations are done between requirements and design. Only a methodology of this type can produce secure/reliable systems; it does not have to use patterns but it must consider all lifecycle stages and all architectural levels. There are many approaches to build secure systems, in [17] we identified over 17 methodologies but none of them uses tactics explicitly. By methodology we mean a complete approach to develop security applications. We are considering how to enhance our own methodology [6] with the use of tactics and how to produce a lightweight version of it.

Acknowledgments

This work was partially supported by UTFSM (grant DGIP 24.12.50, CCTVal (Basal FB0821), and AGCI Chile Student Mobility Platform of the Pacific Alliance (PhD research internship 2013).

References

1. H. Bagheri and K. Sullivan, "A formal approach for incorporating architectural tactics into the software architecture", *Procs. of SEKE 2011*, 770-775.
2. L. Bass, P. Clements, and R. Kazman, *Software architecture in practice* (2nd Ed), Addison-Wesley 2003.
3. L. Bass, P. Clements, and R. Kazman, *Software architecture in practice* (3rd Ed), Addison-Wesley 2012.
4. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal, *Pattern-oriented Software Architecture*, Wiley, 1996.
5. J. M. Cañete, "Annotating problem diagrams with architectural tactics for reasoning on quality requirements", Information Proc. Letters, 112, 2012, 656-661.
6. E. B. Fernandez, M. M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "*Integrating security and software engineering: Advances and future vision*", H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.
7. E. B. Fernandez, *Security patterns in practice - Designing Secure Architectures Using Software Patterns*, Wiley Series on Software Design Patterns, June 2013.
8. B. Gallego, A. Muñoz, A. Maña, D. Serrano, "Security patterns, towards a further level", *Procs. SECRYPT 2009*, 349-356
9. D. Gollmann, *Computer security* (2nd Ed.), Wiley, 2006.
10. N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? A model and annotation", *The Journal of Systems and Software*, 83, 2010, 1735-1758.
11. Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, Sooyong Park, "Quality-driven architecture development using architectural tactics", *Journal of Systems and Software*, 2009
12. Christopher Preschern, "Catalog of Security Tactics linked to Common Criteria Requirements", *Procs. of PLoP 2012*
13. Indrakshi Ray, R. B. France, N. Li, G. Georg, "An aspect-based approach to modeling access control concerns". *Inf. & Soft. Technology*, (9): 575-587 (2004)
14. J. Ryoo, P. Laplante, and R. Kazman, "A methodology for mining security tactics from security patterns", *Procs. of the 43rd Hawaii International Conference on System Sciences*, 2010, http://doi.ieeecomputersociety.org/10.1109/HICSS.2010.18
15. J. Ryoo, P. Laplante, and R. Kazman, "Revising a security tactics hierarchy through decomposition, reclassification, and derivation", *2012 IEEE Int. Conf. on Software Security and Reliability Companion*, 85-91.
16. J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems", *Procs. of the IEEE*, vol. 63, No 9, Sept.1975, 1278-1308
17. 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, 2920-3006.
18. A. Uzunov and E. B. Fernandez, "Cryptography-based security patterns and security solution frames for networked and distributed systems", submitted for publication
19. M. VanHilst, E. B. Fernandez, and F. Braz, "A multidimensional classification for users of security patterns", *Journal of Res. and Practice in Information Technology*, vol. 41, No 2, May 2009, 87-97

Authorization and Disclaimer

Authors authorize LACCEI to publish the paper in the conference proceedings. Neither LACCEI nor the editors are responsible either for the content or for the implications of what is expressed in the paper.