1st LACCEI International Symposium on Software Architecture and Patterns (LACCEI-ISAP-MiniPLoP'2012), July 23-27, 2012, Panama City, Panama.

Should we use tactics or patterns to build secure systems?

Eduardo B. Fernandez[1] and Hernan Astudillo[2]
1 Department of Computer & Electrical Engineering & Computer Science,
Florida Atlantic University, Boca Raton FL
2 Department of Informatics, Universidad Tecnica Federico Santa Maria,
Valparaiso, Chile

Abstract

Quality attributes are the hardest aspect of designing software architectures. Security has become one of the most important of them. Software architecture textbooks and literature imply that we can build secure systems by adding tactics, which are defined as measures to improve quality factors, to the system architecture. We believe that security must be applied throughout the complete lifecycle and that working with architectures is not enough. To prove our point we consider our own secure systems methodology as an example of an approach based on security patterns where security is applied from the requirements stage.

1. Introduction

Quality attributes are the hardest aspect of designing software architectures. They are difficult because they must be applied in a global way; no local optimizations are possible. Being global, their full impact is only known when the system is completed. They also interact with each other and separate optimization is not possible. All this make decisions about quality attributes to be the most important design decisions when building new applications. It is important then to spend time and effort in doing these stages right. Design decisions should be modeled and recorded as first-class entities so the same mistakes are not repeated [Bos04].

There are several quality attributes; security has become one of the most important of them. Reliability is another important attribute closely related to security. We analyze here how some authors have tried to improve security and reliability when building applications and we compare their approaches to ours. Our approach should apply to all quality attributes but we have only explored in detail security and to less extent reliability.

Patterns are considered a good way to handle quality aspects in system architectures. [Har07a] proposed a systematic use of patterns to help the designer. Since patterns can correspond to architectural decisions, they can help understand the impact of each architectural decision. In particular, patterns contain a section of "Consequences" which enumerates the positive and negative aspects of using a pattern. This section helps the architect when deciding if to use a specific pattern or when selecting a pattern from a set of alternatives. However, [Har07a] criticizes existing pattern descriptions because their consequences do not fully describe their interactions with other patterns and their effect on quality attributes. This is a valid criticism and an obvious improvement is to make quality aspects explicit in the application of the patterns and to apply patterns earlier in the lifecycle..

A way to apply security to architectures is based on the idea of tactics. Tactics are described as "measures" or "decisions" taken to improve some quality factor. Another definition is: "architectural building blocks from which architectural patterns are created [Bas03]. Each tactic corresponds to a design decision with respect to a quality factor. Architectural tactics can also be seen as decisions that codify and record best practices for achieving some quality attribute [Bas03]. Tactics are usually expressed in words, but there has been some attempts to formalize them [Bag11].

[Bas03] and [Har10] indicate that with the use of tactics we can produce secure architectures. However, architectures are just one step in a system development cycle which includes some stages that need to be performed before defining a security architecture. We believe that methods based on tactics are not sufficient to build secure or reliable architectures. We believe that we need to apply a complete lifecycle approach to security, not just apply security in the architecture and we try to prove this point here.

Section 2 describes the idea behind the use of tactics for security architectures. Section 3 shows our security methodology based on patterns. A discussion and comparison is presented in Section 4. Related work is presented in Section 5, while Section 6 presents some conclusions.

2. Building secure systems using tactics

In the approach of [Har08, Har10a, Har10b] architectures are built out of architectural patterns, e.g. Brokers, Pipes and Filters, etc. Those patterns implement the high level structural representation of the functional aspects of the application. Then they add tactics to each pattern individually, e.g. a Broker is made secure by the addition of specific security patterns which realize the corresponding tactics. Supposedly, if we add security tactics to all the patterns in the architecture, the system will be secure, and similarly for reliability. Figure 1 illustrates their approach: first build the architecture using architecture patterns in order to implement the structural aspects of the functional requirements. Once these blocks are in place we apply tactics to introduce nonfunctional aspects. Hopefully, the resulting architecture should be secure or reliable. Alternatively, [Kim09] adds tactics directly to the architecture based on a list of Non-functional Requirements (NFRs).

Works that use tactics assume that somehow we know what we want about nonfunctional aspects; that is, we have requirements that prescribe if we need security or performance for example. Their only problem is how to satisfy these requirements with the judicious application of tactics. Threats or failures are not considered in their approach.

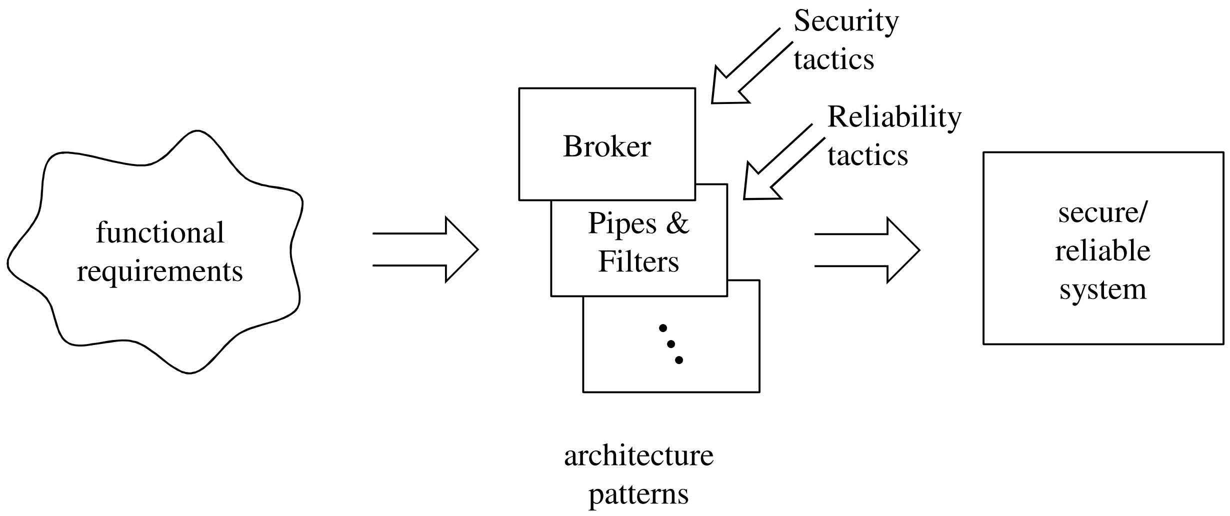

Figure 1. Producing secure or reliable architectures from tactics.

3. Building secure systems using patterns

Several approaches have been proposed to build secure systems [Uzu12]. A few of them use patterns. We have proposed a methodology based on patterns and encompassing all development stages and all architectural levels [Fer04, Fer06c]. Our approach is based on the belief (shared by most security researchers) that security must be applied throughout the complete lifecycle. We start from the use cases of the application, we analyze each action in its activity diagram and enumerate all possible threats as goals of the attacker [Fer06a, Bra08]. From the threats we apply policies to neutralize them, which are realized as security patterns. This results in a conceptual model where all the identified threats are avoided or mitigated though a security pattern. This secure conceptual model is then refined into a secure architecture. In the architecture we use the standard architectural patterns, e.g. brokers, and we add the prescribed security patterns from the conceptual model. There can be additional threats at this stage and in [Fer06b] we proposed adding security to middleware patterns by considering their specific threats. This part is similar to the proposal in [Har07a], which adds quality attributes to specific architectural patterns, but we use the secure middleware components to refine conceptual models where security has already been specified, not as a starting point to build secure applications.

The main difference in our approach with respect to most of the literature about architectural quality is the starting point of architectural decisions. In our approach, we assume that although we know general objectives for an architecture, as developed in the inception phase, we do not know exactly what we want until we develop precise requirements. To know that we need confidentiality for example, is not enough, we need to know which components of the architecture need confidentiality and what kind of confidentiality. We need to identify for example, which are the sensitive assets that need to be protected. Confidentiality is usually obtained by combining several patterns, there is no one pattern that provides it for a whole unit. We have a way to define precisely where to apply confidentiality and what specific mechanisms are the most suitable to get this objective.

4. Discussion

An important aspect is the scope of the system we are building; a system can be a new system (green design) or a legacy system (brownfield design). Also, are we building only the software or both software and platform (OS plus hardware)? Are we building one application or multiple applications, or a software product line? These differences are important because depending on the type of project, we may need to consider some or all the architectural levels of the system.

The main flaw we see in most works on software architecture is the lack of connection to the earlier stages and the lack of environment definition. They assume that the requirements are given and indicate general aspects such as "this architecture needs confidentiality" or "this architecture must be fault tolerant". With those vague requirements it is impossible to define precisely where to add security or reliability mechanisms. As indicated earlier, the precise specification of where in the system we need to apply security patterns can only be done in the analysis stage and propagated to the design stage, i.e. to the architecture.

Architectures have several layers but most works on software architecture consider only the high-level layer. For example, a Secure Broker must enforce access control using its proxies, which correspond to design patterns in a lower level. Security requires securing all levels, not just the highest one.

Adding tactics to individual pattern instantiations cannot make a system secure. Reliability and security are global properties, reinforcing specific components does not make the system secure or reliable. Security and reliability must start from the semantics of the applications that will execute in the architecture. This is the same reason why security cannot be applied successfully using agile methodologies. They work piecemeal, building and testing individual modules. Each module could be secure on its own but when combined with other modules the system may not be secure as a whole.

Our approach is also more general. Not all architectures may include patterns, many are just Big-Balls-of-Mud [Foo00]. Even in those cases we can reconstruct the domain model and apply the constraints to their architectural levels. In fact, we should trace back all the way to the domain level and after we discover the semantics of the application we can apply security constraints which are propagated to the architectural levels.

Architecture patterns can be defined at several levels and a high-level pattern may have several implementations. Code changes may affect all architectural levels, not just the highest level, as it is assumed in many papers.

There are different degrees of satisfaction of a quality attribute and their satisfaction requires global measures, not local optimizations. The ultimate goal of architectural evaluation is to show that an architecture has reached a given level of security or reliability. For example, [Zha11] does not evaluate the "effectiveness" of their architectural recommendations. When we use patterns we can provide some measure of the security level that has been reached. We can, for example [Fer10], check if all the found threats have been handled by some pattern.

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 policy; it can be realized in at least two ways: cryptography, and steganography. Depending on the application, one is more convenient than the other. Figure 2 shows the fact that an architecture may instantiate several patterns each of which is realized in a specific way; for example, an architecture may require Authentication, Authorization, and Logging. In this figure, a pattern includes several possible realizations, where each realization implies a set of classes and associations. For example, Authentication may be realized by a FingerRecognition pattern, which is the way to realize the Authentication needed in the architecture.

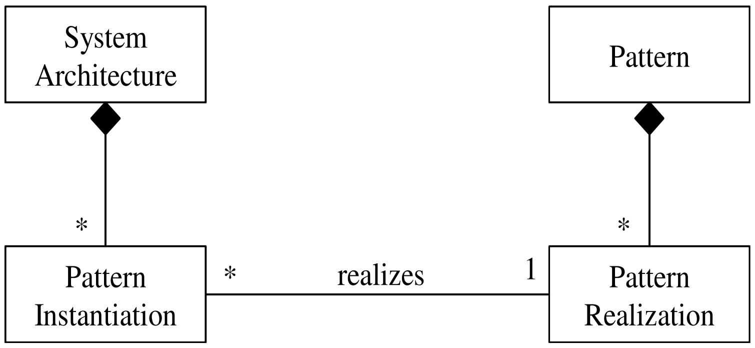

Figure 2. An architecture as a set of patterns.

5. Related work

[Kim09] defines tactics as reusable architectural building blocks that provide generic solutions to address quality attributes. This definition implies a plug-in or template approach, contrary to the idea of patterns, which include several sections indicating the conditions to apply the pattern as well as its consequences. [Ryo10] discusses a similar idea where if a generic model addresses a single architectural force it is a tactic, otherwise it is a pattern. In both cases, the emphasis is on the model as a building block, not on the whole meaning of a pattern.

[Zha11] considers quality attributes to be a subset of NFRs and proposes a metamodel for design decisions. The metamodel is used in an architectural design methodology intended to satisfy quality requirements. They do not use tactics or patterns but talk of "design issues", similar to tactics.

[Haf11] tries to improve the security of a system by applying program code transformations. It is clear that code manipulation is not enough to make a system secure. Security is a systems problem, not a code problem. The system must enforce security constraints that apply to all applications. Hafiz works at the vulnerability level trying to find generic threats.

The work of Lopez and Astudillo [Lop05b] complements our work in the direction of converting patterns into COTS components, an important aspect for a practical realization of secure architectures. Their work has also pointed out the need to start from use cases to define precisely nonfunctional requirements [Lop05a].

Traceability is important to mitigate architectural degradation, where changes made directly in the code may have strong effects on the initial architecture. Traceability establishes links between design decisions and components. Patterns have been proposed for tracing architectural aspects [Mir11], where tactics are used to relate architectural goals to components. This could be a basis for traceability of NFRs.

6. Conclusions

We believe that general methodologies to improve architectural quality using tactics cannot produce secure or reliable systems. We need an approach where we can specify these NFRs from the beginning of software development, considering the semantics of the application. Only a methodology of this type can produce secure/reliable systems; they don't have to use patterns but they must consider all lifecycle stages and all architectural levels. There are many approaches to build secure systems, most of which are surveyed in [Uzu12]; we identified over 17 methodologies but none of them uses tactics explicitly. By methodology we mean a complete approach to develop security applications. It seems that tactics are useful, however, as an architectural notation tool, to record architectural decisions about NFRs, as done in [Har10a].

An architecture based on patterns has other advantages. We can use security patterns as intermediaries for traceability, since we start from high-level patterns and we refine them along levels. We believe that patterns are more convenient for traceability than tactics but we have not proven this here.

Acknowledgements

We thank the referees who provided valuable suggestions and Anton Uzunov who provided some references and discussion.

References

[Bag11] H. Bagheri and K. Sullivan, "A formal approach for incorporating architectural tactics into the software architecture", *Procs. of SEKE 2011*, 770-775.

[Bas03] L.Bass, P. Clements, and R.Kazman, *Software architecture in practice* (2nd Ed), Addison-Wesley 2003.

[Bra08] F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" *Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'08).* In conjunction with the *4th International Conference onTrust, Privacy & Security in Digital Busines(TrustBus'08)*, Turin, Italy, September 1-5, 2008. 328-333.

[Fer04] E.B.Fernandez, "A methodology for secure software design", *2004 Intl. Conference on Software Engineering Research and Practice (SERP'04),* Las Vegas, NV, June 21-24, 2004.

[Fer06a] E. B. Fernandez, M. VanHilst, M. M. Larrondo Petrie, S. Huang, "Defining Security Requirements through Misuse Actions", in *Advanced Software Engineering: Expanding the Frontiers of Software Technology,* S. F. Ochoa and G.-C. Roman (Eds.), International Federation for Information Processing, Springer, 2006, 123-137.

[Fer06b] E. B. Fernandez and M. M. Larrondo-Petrie, "Developing secure architectures for middleware systems", *Procs. of CLEI 2006. (XXXII Conferencia Latinoamericana de Informática).*

[Fer06c] 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.

[Fer10] E.B.Fernandez, N. Yoshioka, H. Washizaki, and M. VanHilst, "Measuring the level of security introduced by security patterns", *Procs. of the 4th workshop on Secure systems methodologies using patterns (SPattern 2010*), in conjunction with ARES 2010, Krakow, Poland, February 2010.

[Foo00] Brian Foote and Joseph Yoder, "*Big Ball of Mud",* Fourth Conference on Patterns Languages of Programs (PLoP'97) Monticello, Illinois, September 1997. Also in *Pattern Languages of Program Design*, edited by Neil Harrison, Brian Foote, and Hans Rohnert. Addison Wesley, 2000

[Haf11] M. Hafiz, P. Adamczyk, and R. Johnson, "Patterns transform architectures", *Procs. of WICSA 2011.*

[Har07a] N. Harrison and P. Avgeriou, "Leveraging Architecture Patterns to Satisfy Quality Attributes", *First European Conference on Software Architecture,* Madrid, Spain, September 24-26, 2007, Springer Lecture Notes in Computer Science.

[Har07b] N.B. Harrison, P. Avgeriou, and U. Zdun, "Using patterns to capture architectural decisions", *IEEE Software*, July/August 2007, 38-45.

[Har10a] 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.

[Har10b] N. Harrison, P. Avgeriou, and U. Zdun, On the Impact of Fault Tolerance Tactics on Architecture Patterns, 2nd International Workshop on Software Engineering for Resilient Systems (SERENE), April 13-16, 2010, London, UK, ACM CS Press.

[Jan05] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions", *Procs. of 5th Working IEEE/IFIP Conf. on Software Architecture*, 2005, 109-120.

[Kim09] Suntae Kim, Dae-Kyoo Kim, Lunjin Lu, Sooyong Park, "Quality-driven architecture development using architectural tactics", *J. of Systems and Software*, 2009, doi:10.1016/
j.jss.2009.03.102, http://www.secs.oakland.edu/~kim2/papers/JSS2009.pdf

[Lop05a] Claudia López, Hernán Astudillo, "Use case- and scenario-based approach to represent NFRs and architectural policies, http://www.ie.inf.uc3m.es/wuscam-05/4-WUsCaM.pdf

[Lop05b] Claudia López, Hernán Astudillo: Explicit Architectural Policies to Satisfy NFRs Using COTS. MoDELS Satellite Events 2005: 227-236

[Mir11] M. Mirakhorli and J. Cleland-Huang, "A pattern system for tracing architectural concerns", *Procs. of the Conference on Patterns Languages of Programs* (*PLoP 2011).*

[Ryo10] J. Ryoo, P. Lapalnte, 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

[Uzu12] A. Uzunov, E.B. Fernandez, and K. Falkner, "Engineering Security into Distributed Systems: A Survey of Methodologies", submitted for publication.

[Zha11] L. Zhang, Y. Sun, Y. Peng, X. Cui, and H. Mei, “Towards quality based solution recommendation in decision-centric architecture design”, *Procs. of SEKE 2011*, 776-781.

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.