A Qualitative Evaluation of Security Patterns

Spyros T. Halkidis, Alexander Chatzigeorgiou, and George Stephanides

Department of Applied Informatics, University of Macedonia,
Egnatia 156, GR-54006 Thessaloniki, Greece
{halkidis,achat,steph}@uom.gr

Abstract. Software Security has received a lot of attention during the last years. It aims at preventing security problems by building software without the so-called security holes. One of the ways to do this is to apply specific patterns in software architecture. In the same way that the well-known design patterns for building well-structured software have been used, a new kind of patterns, called security patterns have emerged. The way to build secure software is still vague, but guidelines for this have already appeared in the literature. Furthermore, the key problems in building secure software have been mentioned. Finally, threat categories for a software system have been identified. Based on these facts, it would be useful to evaluate known security patterns based on how well they follow each guideline, how they encounter with possible problems in building secure software and for which of the threat categories they do take care of.

1 Introduction

Information systems security has been an active research area since decades [7, 13]. The wide applicability of information systems security techniques has been acknowledged due to the wide spread of computer communication technologies and the Internet. Network architecture techniques for building secure intranets have been developed.

Though, only recently it has been recognized that the main source of attacks questioning the security characteristics of information systems is in most cases software poorly designed and developed. Specifically, designed and developed without security being in the minds of people involved [15, 9, 18]. Through practical examples from attacks to businesses and universities it can be shown that the main source of security related attacks are in fact so-called software holes. With this in mind, a new field of research called software security has emerged during the last years.

In analogy to design patterns for building well-structured software, architectural patterns for building secure systems have been proposed. These patterns, called security patterns, have been an active research area since the work by Yoder and Barcalow [23]. Though, until now no qualitative evaluation of the security properties of these patterns does exist.

In this paper we try to investigate this field by providing an evaluation of the patterns based on three main criteria categories. First of all, guidelines for building security software exist [15]. Secondly, main software hole categories that offer seedbed for possible attacks have been identified [15,9]. Thirdly, categories of possible attacks to a system have been analyzed [9]. In this paper we evaluate known security patterns based on how well they confront to the aforementioned guidelines, how well they guide the software to be designed without any software holes and how well a software

J. López, S. Qing, and E. Okamoto (Eds.): ICICS 2004, LNCS 3269, pp. 132–144, 2004.

system using a specific security pattern might respond to each category of possible attacks.

The remainder of the paper is organized as follows. Section 2 makes a short overview of existing security patterns. Section 3 describes the qualitative criteria for the evaluation. Section 4 is the main part of the paper, where the security patterns are evaluated, based on these qualitative criteria. Finally, in Section 4 we make some final conclusions and propose future directions for research.

2 A Short Review of Existing Security Patterns

Since the pioneer work by Yoder and Barcalow [23] several security patterns have been introduced in the literature. Though, there exists no clear definition of a security pattern because different authors refer to security patterns in a different context.

For example, Ramachandran [18] refers to security patterns as basic elements of security system architecture in analogy to the work of Buschman et. al. [4] and Kis [12] has introduced security antipatterns. Romanosky [19, 20, 21] deals with security patterns from different viewpoints. Several authors describe security patterns intended for specific use, such as security patterns for Web Applications [22,11], security patterns for agent systems [17], security patterns for cryptographic software [2], security patterns for mobile Java Code [14], metadata, authentication and authorization patterns [6,3] and security patterns examined at a business level [10]. Furthermore, the same security patterns appear in the literature with different names.

Based on these facts, the Open Group Security Forum started a coordinated effort to build a comprehensive list of existing security patterns with the intended use of each pattern, all the names with which each security pattern exists in the literature, the motivation behind designing the pattern, the applicability of the pattern, the structure of the pattern, the classes that comprise the pattern, a collaboration diagram describing the sequence of actions for the use of the pattern, guidelines for when to use the pattern, descriptions of possible implementations of the pattern, known uses of the pattern and finally, related patterns [1]. The notion of a security pattern in the related technical guide published by the Open Group in March 2004 is completely in analogy with the notion of Design Patterns as originally stated by Gamma et. al. [8].

Our work is based on this review by Blakley et. al. [1] since this is the most comprehensive guide currently reviewing existing security patterns. For the sake of clarity, we will include in this paper the names of the patterns together with their intended use. We will also include a class diagram of the patterns.

Blakley et. al. [1] divide security patterns in two categories. The first category is *Available system patterns*, which facilitate construction of systems that provide predictable uninterrupted access to the services and resources they offer to users. The second category is *Protected system patterns*, which facilitate construction of systems that protect valuable resources against unauthorized use, disclosure or modification.

2.1 Available System Patterns

The intent of the Checkpointed System pattern is to structure a system so that its state can be recovered and restored to a known valid state in case a component fails. A class diagram of the pattern is shown in Figure 1.

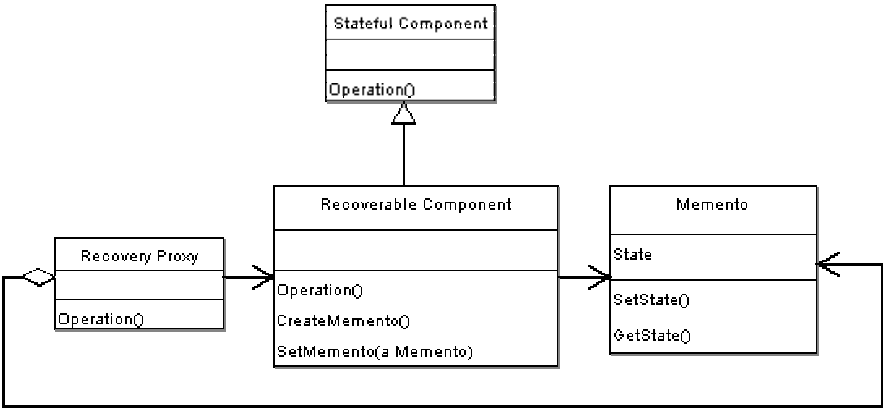

Fig. 1. Class Diagram of the Checkpointed System Pattern

The intent of the Standby pattern is to structure a system so that the service provided by one component can be resumed from a different component. A class diagram of the pattern is shown in Figure 2.

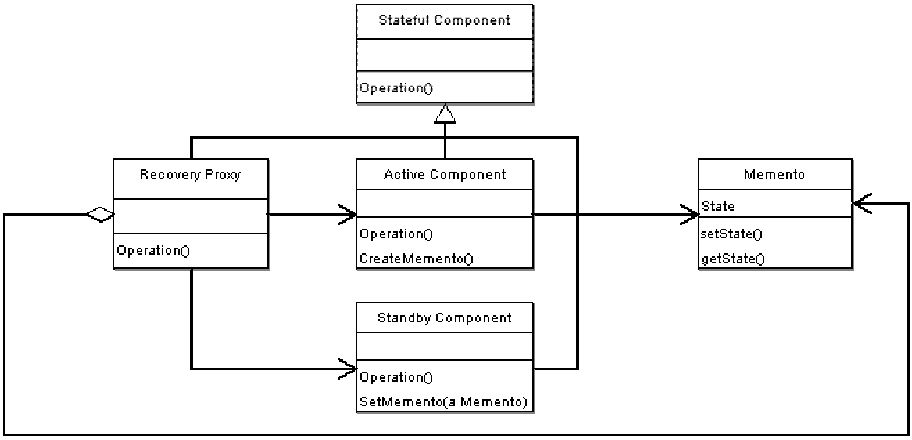

Fig. 2. Class diagram of the Standby pattern

The intent of the Comparator-Checked Fault Tolerant System pattern is to structure a system so that an independent failure of one component will be detected quickly and so that an independent single-component failure will not cause a system failure. A class diagram of the pattern is shown in Figure 3.

The intent of the Replicated System pattern is to structure a system that allows provision from multiple points of presence and recovery in the case of failure of one or more components or links. A class diagram of the pattern is shown in Figure 4.

The intent of the Error Detection/Correction pattern is to add redundancy to data to facilitate later detection of and recovery of errors. A class diagram of the pattern is shown in Figure 5.

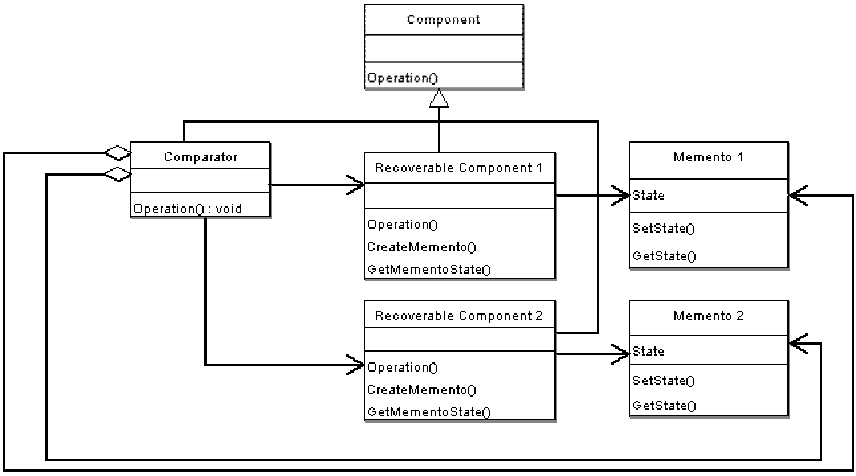

Fig. 3. Class diagram of the Comparator-Checked Fault-Tolerant System Pattern

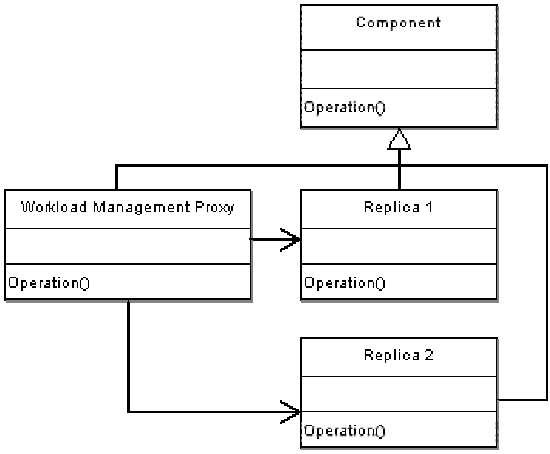

Fig. 4. Class diagram of the Replicated System pattern

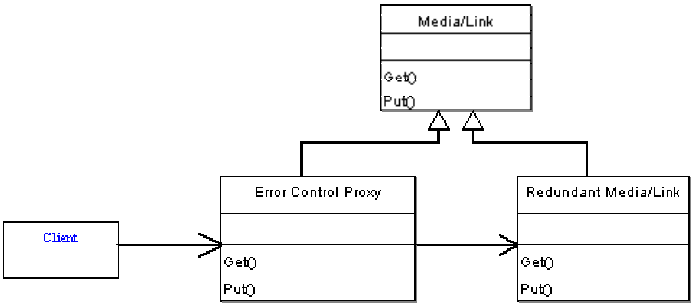

Fig. 5. Class diagram of the Error Detection/Correction pattern

2.2 Protected System Patterns

The intent of the Protected System pattern is to structure a system so that all access by clients is mediated by a guard that enforces a security policy. A class diagram of the pattern is shown in Figure 6.

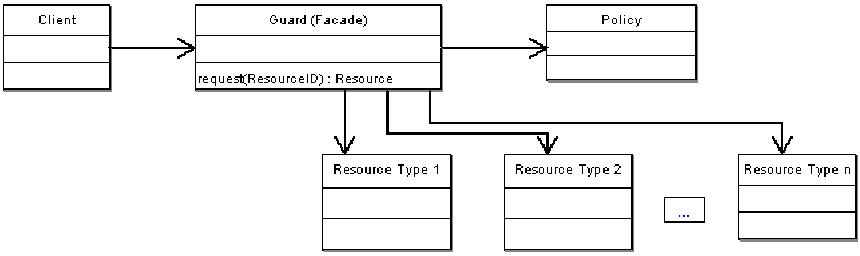

Fig. 6. Class diagram of the Protected System pattern

The intent of the Policy pattern is to isolate policy enforcement to a discrete component of an information system and to ensure that policy enforcement activities are performed in the proper sequence. A class diagram of the pattern is shown in Figure 7.

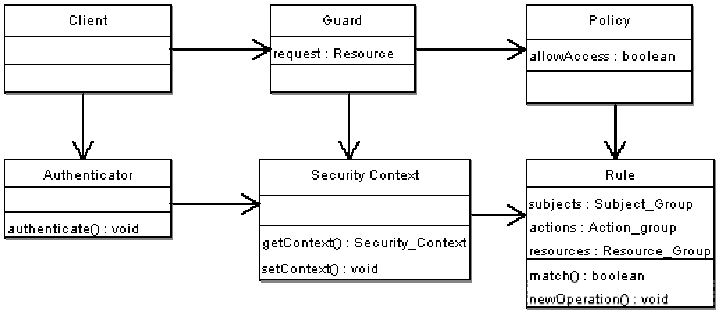

Fig. 7. Class diagram of the Policy pattern

The intent of the Authenticator pattern [3] is to perform authentication of a requesting process, before deciding access to distributed objects. A class diagram of the pattern is shown in Figure 8.

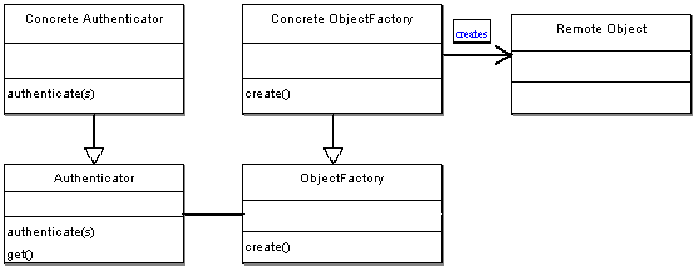

Fig. 8. Class diagram of the Authenticator pattern

The intent of the Subject Descriptor pattern is to provide access to security-relevant attributes of an entity on whose behalf operations are to be performed. A class diagram of the pattern is shown in Figure 9.

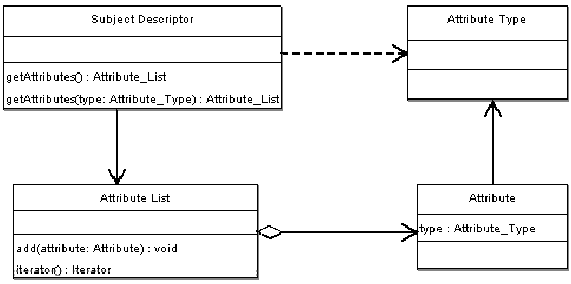

Fig. 9. Class diagram of the Subject Descriptor Pattern

The intent of the Secure Communication Pattern is to ensure that mutual security policy objectives are met when there is a need for two parties to communicate in the presence of threats. A class diagram of the pattern is shown in Figure 10.

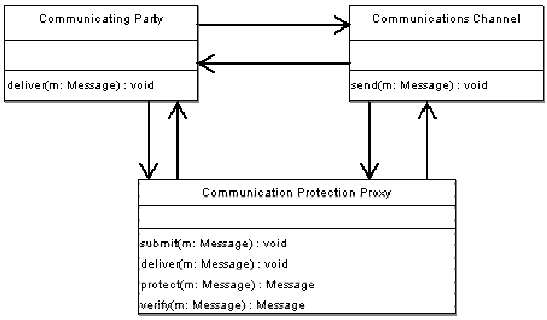

Fig. 10. Class diagram of the Secure Communication Pattern

The intent of the Security Context pattern is to provide a container for security attributes and data relating to a particular execution context, process, operation or action. A class diagram of the pattern is shown in Figure 11.

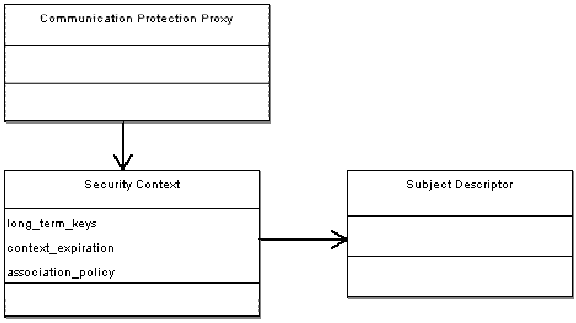

Fig. 11. Class diagram of the Security Context pattern

The intent of the Security Association pattern is to define a structure which provides each participant in a Secure Communication with the information it will use to protect messages to be transmitted to the other party and with the information it will use to understand and verify the protection applied to messages received from the other party. A class diagram of the pattern is shown in Figure 12.

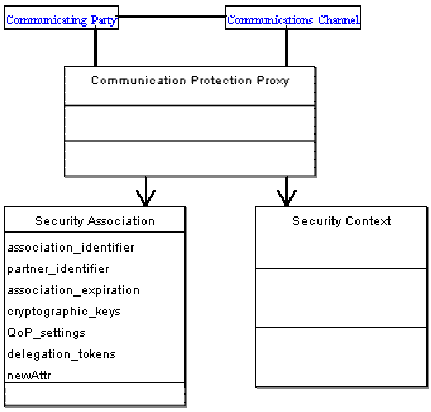

Fig. 12. Class diagram of the Security Association Pattern

Finally, the intent of the Secure Proxy pattern is to define the relationship between the guards of two instances of Protected System, in the case when one instance is entirely contained within the other. Figure 13 shows a class diagram of the pattern.

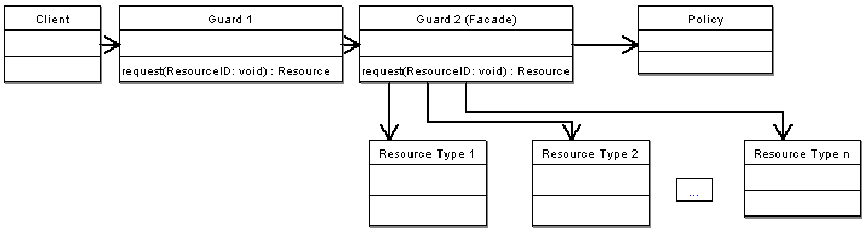

Fig. 13. Class diagram of the Secure Proxy pattern

3 Description of the Qualitative Criteria for the Evaluation

The criteria we use for the evaluation of the security patterns are based on previous work done in the field of software security. Specifically we examine how well the security patterns follow the guiding principles stated by McGraw [15], something that has been also done for some security patterns by Cheng et. al. [5], how well they deter the developer from building software that might contain security holes and finally how well software built based on a specific security pattern might respond to the STRIDE model of attacks described by Howard and Leblanc [9]. We are going to briefly describe these qualitative criteria.

McGraw [15] describes ten guiding principles for building secure software. Principle 1 states that we should secure the weakest link since it is the place of a software system where it is most likely that an attack might be successful. Principle 2 states that we should practice defense in depth, which means that we should have a series of defenses so that, if an error isn't caught by one, it will be caught by another. Principle 3 states that the system should fail securely, which means that the system should continue to operate in secure mode in case of a failure. Principle 4 states that we should follow the principle of least privilege. This means that only the minimum access necessary to perform an operation should be granted, and the access should be granted only for the minimum amount of time necessary. Principle 5 advises us to compartmentalize, which means to minimize the amount of damage that can be done to a system by breaking up the system into as few units as possible while still isolating code that has security privileges. Principle 6 states that we should keep the system simple since complex systems are more likely to include security problems. Principle 7 states that we should promote privacy, which means that we should protect personal information that the user gives to a program. Principle 8 states that we should remember that hiding secrets is hard, which translates into building a system where even insider attacks are difficult. Principle 9 states that we should be reluctant to trust, which means that we should not trust software that has not been extensively tested. Finally, principle 10 states that we should use our community resources, which means that we should use well-tested solutions. From the above descriptions it is obvious that there are some principles that conflict and that there are tradeoffs in designing a software system. For example the principle of keeping the system simple conflicts with the principle of practicing defense in depth. Though, a good solution to this might be to build systems where different parts of them adhere to different sets of principles, so that different parts supplement each other.

The second set of criteria describes how well a security pattern deters the software developer from building a system that contains common software security holes, as they are described by McGraw [15]. In this paper we focus on three pure software development problems that might be encountered which are buffer overflows, poor access control mechanisms and race conditions and don't study problems related to cryptography such as poor random number generation.

The last set of criteria can be described as how well a specific security pattern might respond to different categories of attacks as they are described by Howard and Leblanc [9]. To describe the different categories of attacks that are possible in a software system Howard and Leblanc propose the so-called STRIDE model. The first category of attacks consists of the Spoofing identity attacks. The second category of attacks consists of the Tampering with data attacks. The third category of attacks consists of the Repudiation attacks. The fourth category of attacks consists of the Information disclosure attacks. The fifth category of attacks consists of Denial of Service attacks. Finally, the sixth category of attacks consists of the Elevation of privilege attacks.

4 Qualitative Evaluation of the Security Patterns

In many cases we cannot make judgment about specific criteria because in some cases the security properties of the system are not dependent on the security pattern but in

its specific implementation. In these cases we do not mention the criteria for the pattern we are considering.

We first discuss which of the qualitative properties we described previously exist in the so-called Available System Patterns. We first note that the basic aim of these security patterns is to make systems robust in the case of failure. So, the first general observation we can do is that these patterns are designed in order for a system to fail securely. Furthermore, by looking at the class diagrams of these patterns we can conclude that the Checkpointed System pattern, the Standby pattern and the Error Detection Correction pattern are designed in such a way that they are kept simple. All the Available System patterns, due to the purpose they serve have protection from Denial of Service attacks because they can detect such situations as failure cases. The more complex of them, namely the Comparator-Checked Fault Tolerant System pattern and the Error Detection/Correction pattern have improved protection from Denial of Service Attacks, since they consist of Multiple Recoverable Components or Replicas respectively. That implies that in case a part fails not only can it be replaced by another part, but also in case the second part fails it can be replaced too by another part and so on.

We describe the qualitative properties of Protected System Patterns in more detail since they differ from each other.

The Protected System pattern aims at protecting access to some resources from clients accessing them without control by setting a guard between them. It implements the principle of least privilege, since the access to the resources is controlled. It can follow the principle of using community resources, by choosing appropriate software solutions for the guard. It works against the principle of compartmentalization, since one guard protects all the resources. It works against the principle of practicing defense in depth since there exists only one level of protection. Considering the second set of previously described criteria for avoiding software holes we can note that by using a Stackguard [18] as part of the guard design of the pattern we could prevent clients producing buffer overflows to the system. Furthermore, the guard could perform good access control satisfying the second criterion for deterring the system from having software holes. Race conditions could be prevented by not letting different clients competing for the same resource. Regarding the third set of criteria we can estimate that the guard could protect the system from spoofing, information disclosure, tampering and elevation of privilege attacks through the implementation of a good authentication and authorization mechanism as part of its functionality.

The Policy pattern aims at applying a specified security policy to a discrete component of an information system. It uses both an Authenticator and a Guard class. So, it achieves practicing defense in depth. Furthermore, it follows the principle of least privilege and the principle of promoting privacy by proper design of the Authenticator class. It could follow the principle of using community resources by choosing tested solutions for the Guard and the Authenticator. It has simple design, so it follows the principle of keeping the system simple. Regarding the second and third sets of criteria the same things as for the Protected System pattern hold, for the same reasons. Additionally, it protects from repudiation attacks due to the Authenticator class.

The Authenticator pattern [3] performs authentication of a requesting process before deciding access to distributed objects. Through the Authenticator class, it applies the principle of least privilege and the principle of promoting privacy. By requesting authentication from the same Authenticator for every object of the server [3], this pattern works against the principle of compartmentalization. Due to its simple design

it follows the principle of keeping the system simple. About the third set of criteria we can conclude that it has the same properties with the Policy Pattern for the same reasons.

The Subject Descriptor pattern aims at providing access to security-relevant attributes an entity. It promotes the principle of keeping the system simple due to its design. It can promote security properties only in association with other security patterns, like the Protected System Pattern. In its own it offers no protection from STRIDE attacks.

The Secure Communication pattern aims to ensure that mutual security policy objectives are met when there is a need for two parties to communicate in the presence of threats. It follows the secure weakest link principle, since the communication link is the weakest link of the system in this case. It follows the principle to compartmentalize since a separate Communication Protection Proxy protects each link. It follows the principle of promoting privacy since the pattern protects from unauthorized use of the communications channel. The presence of software holes is dependent on the quality of the Communication Protection Proxy software. Specifically, the Communication Protection Proxy software can protect from buffer overflows and perform good access control to the communications channel. Regarding the third set of criteria, this pattern could protect from all types of attacks, since it can perform good access control to the communication link, confirm that each communicating party is the one it claims to be and finally the Communication Protection Proxy could cater for the protection from Denial of Service attacks.

The Security Context pattern aims at providing a container for security attributes or data. It follows the principle of least privilege and promotes privacy, since the security attributes and data are protected by a Communication Protection Proxy class. Regarding the protection from software security holes we can estimate that a Communication Protection Proxy Software of good quality can protect from all three basic types of software security holes. Regarding possible attacks the Communication Protection Proxy can protect from Tampering, Information disclosure and Elevation of Privilege attacks.

The Security Association pattern aims at defining a structure that provides each participant in a Secure Communication with the information it will use to protect messages to be transmitted to the other party and with the information it will use to understand and verify the protection applied to messages received from the other party. As a general note we can observe that this pattern has meaning only in association with the Secure Communication pattern. It follows the principle of securing the weakest link since it aims at protecting the communication channel. It follows the principle of practicing defense in depth, since it provides a second mechanism for protecting the communication channel. It follows the principles of least privilege and of promoting privacy through the use of the Communication Protection Proxy. It has simple design and consequently follows the principle of keeping the system simple. Regarding the second set of criteria the same as with the Security Context pattern holds for the same reasons. It protects from spoofing identity attacks and repudiation attacks through the use of the Communication Protection Proxy. It protects from Tampering, Information Disclosure and Elevation of Privilege attacks through the use of the Communication Protection Proxy.

The Secure Proxy pattern aims at defining the relationship between the guards of two instances of the Protected System when one instance is entirely contained within the other. It practices defense in depth since it uses multiple levels of protection for

the resources. It promotes privacy and follows the principle of least privilege through the use of the guards. Regarding the second set of criteria the same as with the Protected System pattern holds for the same reasons. This pattern can protect from the same type of attacks as the Protected System for the same reasons.

The evaluation based on the first set of criteria can be summarized in table 1:

Table 1. Summary of the evaluation of the security patterns based on the ten guiding principles by McGraw. (Explanations, Y: Yes, A: Against, P: Possible)

	Principles							
Pattern Name	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**10**
Checkpointed System			Y			Y		
Standby			Y			Y		
Comparator Checked Fault Tolerant System			Y					
Replicated System			Y					
Error Detection/Correction			Y					
Protected System		A		Y	A			P
Policy		Y		Y		Y	Y	P
Authenticator				Y	A	Y	Y	
Subject Descriptor						Y		
Secure Communication	Y			Y		Y		
Security Context				Y		Y		
Security Association	Y	Y		Y		Y	Y	
Security Proxy		Y		Y			Y	

A summary of the evaluation of the patterns based on the second set of criteria appears in Table 2. The security patterns, which are not present in the table, do not offer protection from any of the categories listed.

Table 2. Summary of the evaluation of security patterns based on the second set of criteria. (Explanations, P:Possible)

Pattern Name	**Protection from Buffer Overflows**	**Good Access Control**	**Protection from Race Conditions**
Protected System	P	P	P
Policy	P	P	P
Secure Communication	P	P	
Security Context	P	P	P
Security Association	P	P	P
Secure Proxy	P	P	P

Finally, Table 3 summarizes the evaluation of the security patterns based on the third set of criteria.

5 Conclusions and Future Work

As it is well known in the security patterns community no security pattern in its own has all the desired characteristics. So, a good combination of the existing security patterns when designing a software system is required in order for it to be secure enough. The qualitative evaluation presented in this paper can aid in choosing good combinations of security patterns in order to build a secure software system. Secondly, we could note that beyond the qualitative evaluation of security patterns a

Table 3. Summary of the evaluation of security patterns, based on the third set of criteria. (Explanations, P: Protection Exists, I: Improved Protection)

Pattern Name	S	T	R	I	D	E
Checkpointed System					P	
Standby					P	
Comparator-Checked Fault Tolerant System					I	
Replicated System					P	
Error Detection/Correction					I	
Protected System	P	P		P		P
Policy	P	P	P	P		P
Authenticator	P	P	P	P		P
Subject Descriptor						
Secure Communication	P	P	P	P	P	P
Security Context		P		P		P
Security Association	P	P	P	P		P
Secure proxy	P	P		P		P

quantitative approach to evaluating the security of software systems would be desirable. This is also noted in [16]. In order for this goal to be achieved, one possible approach would be to combine software metrics techniques with the use of security patterns so that software designs could be quantitatively evaluated in terms of security.

References

1. Blakley, B., Heath, C. and Members of the Open Group Security Forum, Security Design Patterns, Open Group Technical Guide (2004)
2. Braga, A., Rubira, C., and, Dahab R., Tropyc: A Pattern Language for Cryptographic Software, in Proceedings of the 5th Conference on Pattern Languages of Programming (PloP '98) (1998)
3. Lee Brown, F., Di Vietri, J., Diaz de Villegas, G., and Fernandez, E., The Authenticator Pattern, in Proceedings of the 6th Conference on Pattern Languages of Programming (PloP '99) (1999)
4. Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., and Stahl, M., Pattern Oriented Software Architecture – A System of Patterns, John Wiley and Sons (1996)
5. Cheng, B., Konrad, S., Campbell, L. and Wassermann, R., Using Security Patterns to Model and Analyze Security Requirements, In Proceedings of the High Assurance Systems Workshop (RHAS '03) as part of the IEEE Joint International Conference on Requirements Engineering (2003)
6. Fernandez E., Metadata and authorization patterns, http://www.cse.fau.edu/~ed/MetadataPatterns.pdf (2000)
7. Fites, P., and Kratz, M., Information Systems Security: A Practitioner's Reference, International Thomson Computer Press, (1996)
8. Gamma, E., Helm, R., Johnson, R. and Vlissides, J., Design Patterns, Addison Wesley, (1995)
9. Howard, M., and LeBlanc, D., Writing Secure Code, Microsoft Press (2002)
10. IBM, Introduction to Business Security Patterns, IBM White Paper (2003)
11. Kienzle, D., and Elder, M., Security Patterns for Web Application Development, Univ. of Virginia Technical Report (2002)
12. Kis, M., Information Security Antipatterns in Software Requirements Engineering, In Proceedings of the 9th Conference on Pattern Languages of Programming (PLoP '02) (2002)

13. Krause M. and Tipton H. editors, Information Security Management Handbook, Fourth Edition, CRC Press – Auerbach Publications (1999)
14. Mahmoud, Q., Security Policy: A Design Pattern for Mobile Java Code, in Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP '00) (2000)
15. McGraw, G., Building Secure Software, How to Avoid Security Problems the Right Way, Addison Wesley (2002)
16. McGraw, G., From the Ground Up: The DIMACS Software Security Workshop, IEEE Security and Privacy, March/April 2003, 2-9
17. Mouratidis, H., Giorgini, P., and Schumacher, M., Security Patterns for Agent Systems, in Proceedings of the Eighth European Conference on Pattern Languages of Programs (EuroPLoP '03) (2003)
18. Ramachandran, J., Designing Security Architecture Solutions, John Wiley and Sons (2002)
19. Romanosky, S., Security Design Patterns, http://www.romanosky.net/papers/securityDesignPatterns.html (2002)
20. Romanosky, S., Enterprise Security Patterns, http://www.romanosky.net/papers/EnterpriseSecurityPatterns.pdf (2002)
21. Romanosky, S., Operational Security Patterns, http://www.romanosky.net (2003)
22. Weiss, M., Patterns for Web Applications, in Proceedings of the 10th Conference on Pattern Languages of Programming (PLoP '03) (2003)
23. Yoder, J., and, Barcalow, J., Architectural Patterns for enabling application security, in Proceedings of the 4th Conference on Pattern Languages of Programming (PLoP '97) (1997)