available at www.sciencedirect.com

journal homepage: www.elsevier.com/locate/cose

Computers & Security

A qualitative analysis of software security patterns

Spyros T. Halkidis, Alexander Chatzigeorgiou, George Stephanides*

Department of Applied Informatics, University of Macedonia, Egnatia 156, GR-54006 Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 5 May 2005
Revised 22 December 2005
Accepted 6 March 2006

Keywords:
Security patterns
Software security
Design patterns
Security architecture
Software architecture

ABSTRACT

Software security, which has attracted the interest of the industrial and research community during the last years, aims at preventing security problems by building software without the so-called security holes. One way to achieve this goal 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 defined, a new kind of patterns called security patterns have emerged. These patterns enable us to incorporate a level of security already at the design phase of a software system. There exists no strict set of rules that can be followed in order to develop secure software. However, a number of guidelines have already appeared in the literature. Furthermore, the key problems in building secure software and major threat categories for a software system have been identified. An attempt to evaluate known security patterns based on how well they follow each principle, how well they encounter with possible problems in building secure software and for which of the threat categories they do take care of, is performed in this paper. Thirteen security patterns were evaluated based on these three sets of criteria. The ability of some of these patterns to enhance the security of the design of a software system is also examined by an illustrative example of fortifying a published design.

© 2006 Elsevier Ltd. All rights reserved.

1. Introduction

Information systems security has been an area of interest to researchers since decades (Fites and Kratz, 1996; Tipton and Krause, 1999) and nowadays is also a major concern for the software industry. The high importance of information systems security techniques is obvious, due to the widespread use of computer communication technologies and the Internet.

However, only recently it has been recognized that the main source of attacks questioning the security characteristics of information systems, is in most cases on software poorly designed and developed. Specifically, software is often designed and developed without security being in the mind of the developers (Howard and LeBlanc, 2002; Ramachandran, 2002; Viega and McGraw, 2002). Through practical examples from attacks to businesses and universities, it has been shown that almost all security related attacks in fact take advantage of so-called software holes. (Software holes are parts of software written in such a way that they can be exploited to perform an attack that compromises the security of the corresponding system.) As a result, a new field of research, namely software security has emerged during the last decade.

In analogy to design patterns for building well-structured and maintainable software (Gamma et al., 1995), architectural patterns aiming at building secure systems have been proposed. These patterns, called security patterns, have been an active research area since the work by Yoder and Barcalow (1997). This approach to ensuring the security of a software system differs significantly from the approach that is followed from the dependability community (Nicol et al., 2004). People doing research in dependability try to extend techniques developed to ensure system dependability, in order to ensure system security. Although the approach of using security

* *Corresponding author.*
E-mail addresses: halkidis@java.uom.gr (S.T. Halkidis), achat@uom.gr (A. Chatzigeorgiou), steph@uom.gr (G. Stephanides).
0167-4048/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2006.03.002

patterns differs, this approach is also very promising and we estimate that these two approaches will converge in the near future. The security patterns are of high importance to the security of a software system, since they allow us to incorporate a level of security to it already at the design phase of the system. Despite the importance of security patterns, until now no qualitative evaluation of the security properties of these patterns has appeared in the literature.

In this paper an attempt to investigate the qualitative features of the security patterns is performed by providing an evaluation of each pattern based on three main criteria. First of all, guidelines regarding how to build secure software exist (Viega and McGraw, 2002). Secondly, main software-hole categories that offer seedbed for possible attacks have been analyzed (Howard and LeBlanc, 2002; Viega and McGraw, 2002). Thirdly, categories of possible attacks to a system have been identified (Howard and LeBlanc, 2002). In this paper we evaluate known security patterns based on how well they conform to the aforementioned guidelines, how well they guide the software to be designed without software holes and how well a software system using a specific security pattern might respond to each category of possible attacks. A brief description of the intent and the structure of 13 well-known security patterns, as well as a discussion of the qualitative criteria is also made. A preliminary version of this paper can be found in Halkidis et al. (2004).

The remainder of the paper is organized as follows. Section 2 includes a short overview of existing security patterns. Section 3 describes the qualitative criteria for the evaluation and in Section 4 the security patterns are evaluated against each set of criteria. Section 5 describes a practical example illustrating the application of security patterns in a real system. Finally, in Section 6 some final conclusions and future directions for research are discussed.

2. A short review of existing security patterns

Since the pioneer work by Yoder and Barcalow (1997) several security patterns have been introduced in the literature. Though, there exists no clear definition of a security pattern, since different authors refer to security patterns in different contexts.

For example, Ramachandran (2002) refers to security patterns as basic elements of security system architecture, in analogy to the work of Buschmann et al. (1996). Kis (2002) has introduced security antipatterns as common security related pitfalls. Romanosky (2002) aims to investigate some security patterns using a specific format, in analogy to the examination of software design patterns (Gamma et al., 1995). Several authors describe security patterns intended for special purposes, such as security patterns for Web Applications (Weiss, 2003; Kienzle and Elder, 2002), security patterns for agent systems (Mouratidis et al., 2003), security patterns for cryptographic software (Braga et al., 1998), security patterns for mobile Java Code (Mahmoud, 2000), and finally metadata, authentication and authorization patterns (Fernandez, 2000; Lee Brown et al., 1999). Furthermore, similar security patterns appear in the literature with different names.

Based on all these facts, the Open Group Security Forum has initiated 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 its use, guidelines for when to use the pattern, descriptions of possible implementations, known uses and finally, related patterns (Blakley et al., 2004). 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. (1995). A review of the implemented security patterns in qmail can be found in Hafiz et al. (2004).

The work presented in this paper is based on the review by Blakley et al. (2004) 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 for each pattern. For those not familiar with class diagrams we propose the book by Fowler (2003). These patterns are not used only in object oriented systems. For example the widely known qmail program (Hafiz et al., 2004) is implemented in a language that is not object oriented and contains security patterns. However, security patterns are more easily incorporated into object oriented systems and our focus in this paper is on how security patterns can be used in object oriented applications.

Blakley et al. (2004) divide security patterns into two categories. The first category includes *Available System patterns*, which facilitate the construction of systems that provide predictable uninterrupted access to the services and resources they offer to users. The second category consists of *Protected System patterns*, which facilitate construction of systems that protect valuable resources against unauthorized use, disclosure or modification.

2.1. *Available System patterns*

An explanation of concepts used here, related to fault tolerance can be found in Avizienis et al. (2004).

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 Fig. 1. The Checkpointed System pattern offers protection from loss or corruption of state information in case a component fails (e.g. due to a broken SMTP connection in a mail system (Hafiz et al., 2004)). The Recovery Proxy shown in the diagram consists of one or more Mementos. It periodically checks the Recoverable Component's state and if it has changed from the last check, it initiates the creation of a Memento with the new state. Furthermore, the Recovery Proxy can detect failures. If a failure is detected, it initiates state recovery by instructing Recoverable Component to restore state from Memento. From the function of the Checkpointed System pattern it can be concluded that if we use multiple Mementos, we can counterbalance the failure of a Memento itself.

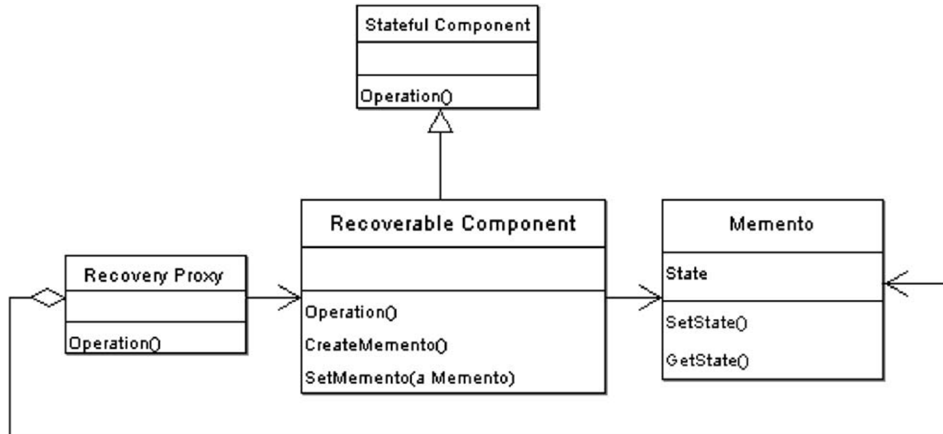

Fig. 1 – Class diagram of the Checkpointed System pattern.

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 Fig. 2. The Standby pattern can be used in cases where failed components may not be recoverable but a similar or identical backup component is available. The Recovery Proxy does also in this case periodical checks of the Recoverable Component's state and if it has changed from the last check, it initiates the creation of a Memento with the new state. If the Recovery Proxy detects a failure, it activates the Standby Component, which restores state from a Memento. From this point on all requests are routed to the Standby Component. We can easily conclude that this security pattern can be used in cases where loss of a small number of transactions is allowed, since it takes some time until the Standby Component restores the saved state and is activated.

The intent of the *Comparator-Checked Fault Tolerant System* pattern is to structure a system, so that an independent failure of one component (i.e. a failure of a component that does not affect other components at all) 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 Fig. 3. The use of this pattern is more effective compared to the Checkpointed System pattern and the Standby pattern since it supports detection of faults, which have not caused a failure yet. The Comparator shown in the diagram routes each request to Recoverable Components each of which creates a Memento saving the state after completion of the operation. Mementos are checked by the Comparator for whether they match. If not, the Comparator takes corrective action. The most effective corrective action is to detect the failed component and automatically correct it and restart it. If this is not possible, the failed pair of Recoverable Components may be taken offline. It can be easily concluded that use of this security pattern is advised when failure of one component is not expected to be strongly correlated with similar or identical failures in another component. Furthermore, it should be feasible to compare the internal states of components and duplicating components should be economical.

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 Fig. 4. The way it works is that the Workload Management Proxy assigns requests to the components called Replicas in the diagram. If a component fails the other one is chosen to complete the task. If no component fails the component with the smallest workload is chosen to complete the task.

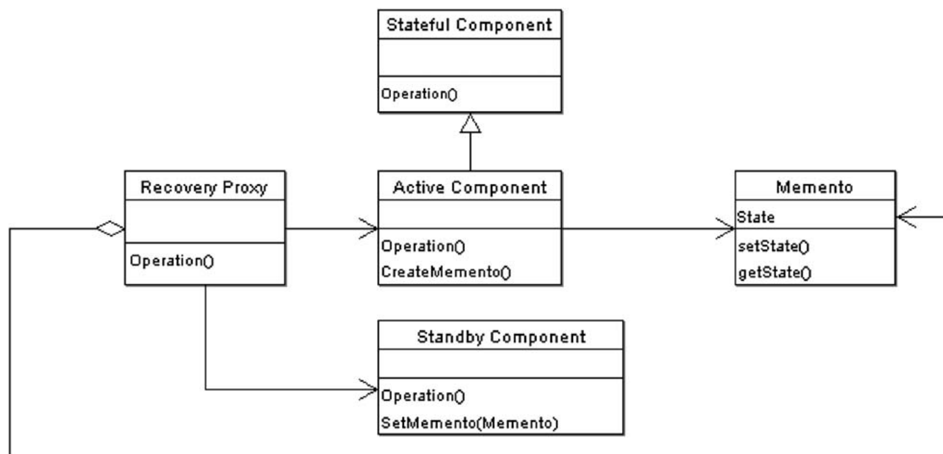

Fig. 2 – Class diagram of the Standby pattern.

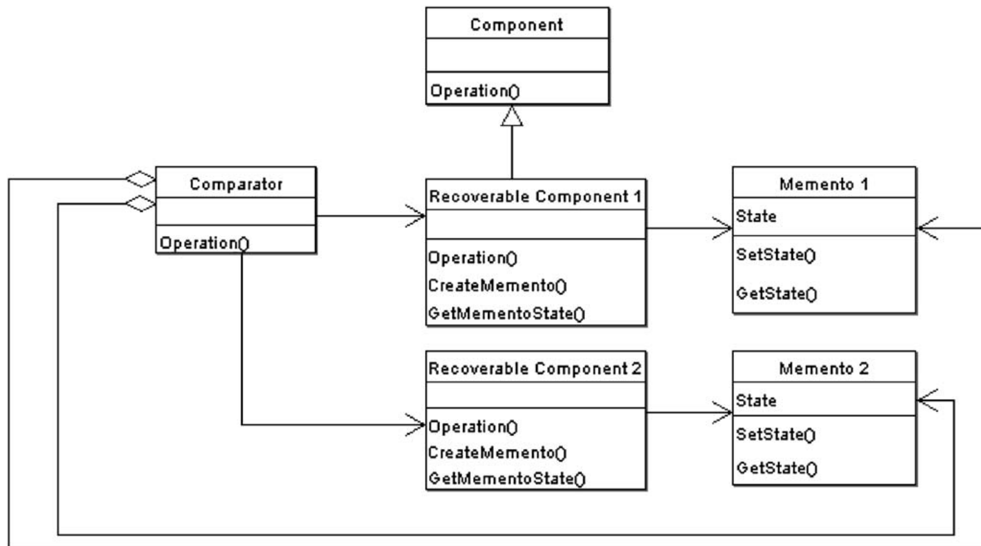

Fig. 3 – Class diagram of the Comparator-Checked Fault Tolerant System pattern.

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 Fig. 5. The Error Control Proxy adds redundancy to the data provided by the Client. These data that include redundancy are saved to Redundant Media/Link. If the Client does a read request, the Error Control Proxy forwards this request to the Redundant Media/Link and after the data are read it checks their integrity. If a problem occurs, the Error Control Proxy may repair the integrity of the data before they are returned to the Client. If this is not possible the Error Control Proxy notifies the Client of the Problem.

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 Fig. 6. The Guard controls access requests to resources according to a predefined policy. Of course the Guard itself must be robust to malicious code attacks. A similar pattern that was introduced by Yoder and Barcalow (1997) is the Single Access point pattern.

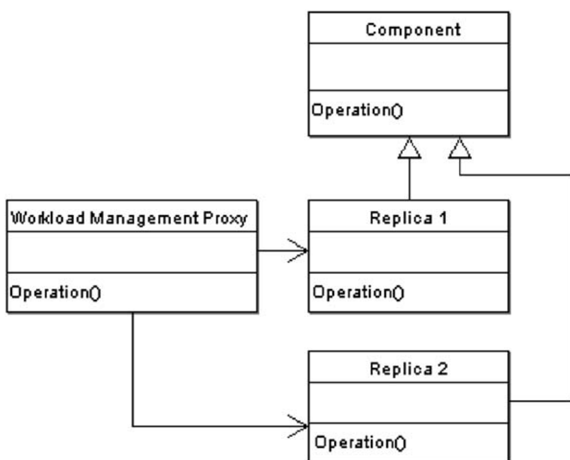

Fig. 4 – Class diagram of the Replicated System pattern.

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 Fig. 7. The way it works is that Policy enforces rules that are to be applied by the Guard for possible authentication. The policy enforcement functions are invoked every time access is attempted to a resource, which is subject to the policy. If all the constraints are satisfied, access to resources is allowed by the guard. In more detail, the first step of the function of this pattern is the authentication of the Client. If this step is successful, Security Context attributes are set. After that, the Security Context is read from the guard and the guard requests a policy decision according to the rules.

The intent of the *Authenticator* pattern (Lee Brown et al., 1999) is to perform authentication of a requesting process, before deciding access to distributed objects. A class diagram of the pattern is shown in Fig. 8. If the authentication process performed by the Authenticator is successful, the Authenticator forwards a request for the creation of a Remote Object to the ObjectFactory.

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. Specifically, the Subject Descriptor pattern is used in conjunction with other security patterns to control the conditions under which authorization is to be performed. In more depth, this pattern is used to represent authorization subjects as sets of predicates or assertions on attribute and property values. A Subject Descriptor consists of several attribute conditions (e.g. when authorizing a user for entering a part of the system of a company, the conditions could be that he/she belongs to the computing department of the company and works for at least five years for it), which can possibly correspond to several real subjects. A class diagram of the pattern is shown in Fig. 9. The Subject Descriptor

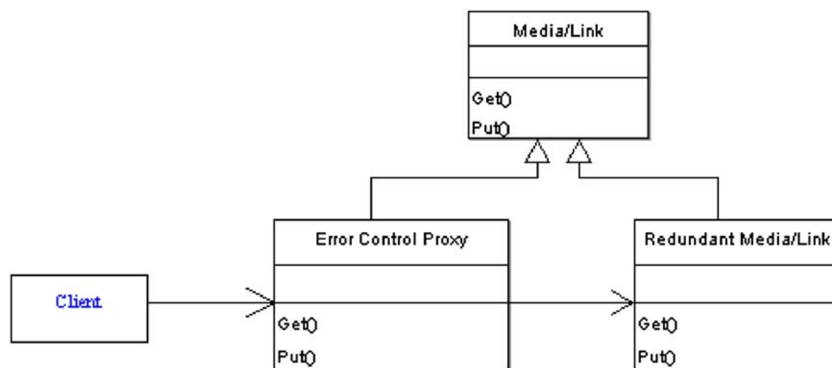

Fig. 5 – Class diagram of the Error Detection/Correction pattern.

iterates through the Attribute List, until the desired security-relevant attribute is found. A similar pattern is the Roles pattern, which was introduced by Yoder and Barcalow (1997).

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 Fig. 10. The Secure Communication pattern protects the communication channel. The Communication Protection Proxy acts as an inline proxy that controls traffic, i.e. it checks any message the Communicating Party wishes to deliver, before it reaches the Communications Channel. If the sender wants to send a message, the Communication Protection Proxy of the sender applies appropriate protection to the message. Then it uses the Communications Channel to transmit the message to the Communication Protection Proxy of the receiving Communicating Party, which verifies protection. If verification is successful the message is delivered to the receiver. Through the use of cryptography, data origin authentication and promotion of data integrity and confidentiality are possible.

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 Fig. 11. After a process becomes active, an instance of Security Context is created by a Communication Protection Proxy and populated with the necessary information about the process. Authentication of the user initiating the process may be applied by the Communication Protection Proxy.

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. Furthermore, it provides the participants 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 Fig. 12. The Security Association pattern enables an instance of Secure Communication to protect more than one message.

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. Fig. 13 shows a class diagram of the pattern. The first guard checks the request of the Client, according to some of the rules enforced by Policy. If the first check is successful, the second guard checks the request according to the rest of the rules. If the second check is successful, access to the resources is allowed. The guards may also check both on all the rules enforced by Policy, in order to achieve increased protection in case a problem in the first guard occurs.

3. Description of the qualitative criteria for the evaluation

The criteria 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 Viega and McGraw (2002), something that has been also examined for some security patterns by Cheng et al. (2003), how well they deter the developer from building software that might contain security holes and

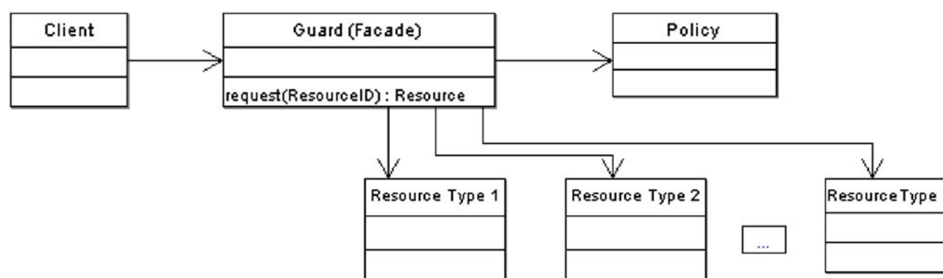

Fig. 6 – Class diagram of the Protected System pattern.

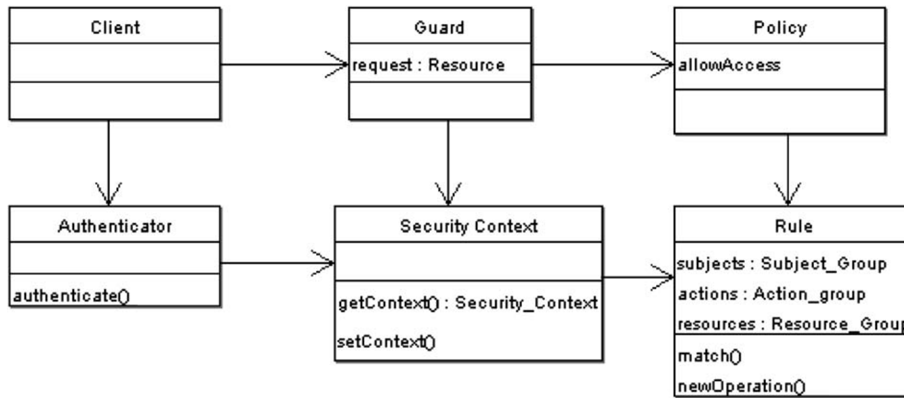

Fig. 7 – Class diagram of the Policy pattern.

finally how well software built based on a specific security pattern, might respond to the STRIDE model of attacks described by Howard and LeBlanc (2002). We are going to briefly describe these qualitative criteria.

Viega and McGraw (2002) describe 10 guiding principles for building secure software. Principle 1 states that we should secure the weakest link, since the weakest link 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 is not 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 many 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 contradicts 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 Viega and McGraw (2002). In this paper we focus on three pure software development problems that might be encountered, which are *buffer overflows* (for example when an attacker corrupts the execution stack through writing past memory), *poor access control* mechanisms (for example when the user is easily allowed to get administrator rights) and *race conditions* (encountered in environments in which there are multiple threads or processes) and do not 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 (2002).

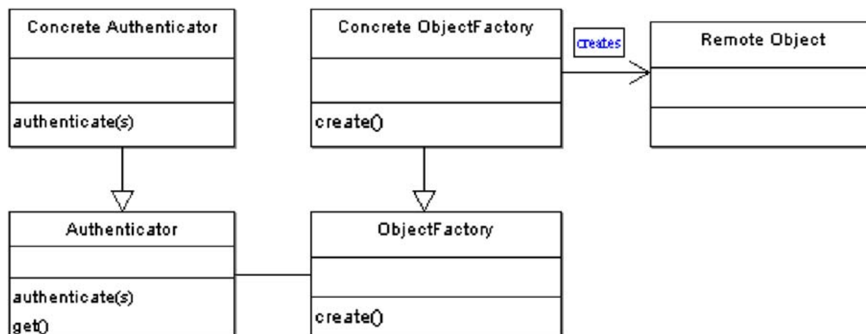

Fig. 8 – Class diagram of the Authenticator pattern.

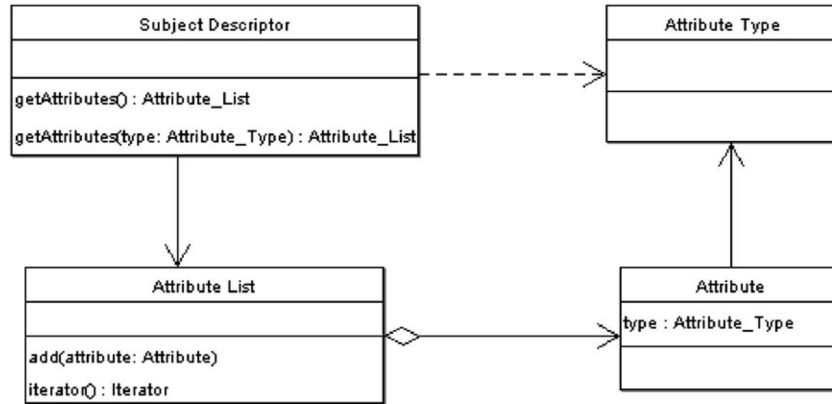

Fig. 9 – Class diagram of the Subject Descriptor pattern.

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. Identity spoofing is illegally accessing and then using another user's authentication information. The second category of attacks consists of the *Tampering with data* attacks. Data tampering involves malicious modification of data. The third category of attacks consists of the Repudiation attacks. *Repudiation* attacks are associated with users who deny performing an action without other parties having a way to prove otherwise. The fourth category of attacks consists of the *Information disclosure* attacks. Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it. The fifth category of attacks consists of *Denial of Service* attacks. Denial of Service (DoS) attacks deny service to valid users. Finally, the sixth category of attacks consists of the *Elevation of privilege* attacks. In this type of attack, an unprivileged user gains privileged access and therefore has sufficient access to compromise or destroy the entire system, if only one level of privilege is used.

4. Qualitative evaluation of the security patterns

In many cases it is not possible to decide about to which extent specific criteria are satisfied, because in some cases the security properties of the system do not depend on the security pattern but on its specific implementation. For example, the Protected System pattern may offer protection from buffer overflows, but only if the Guard performs proper input checking (or if the software of the Guard is implemented in a programming language such as Java that protects from buffer overflows itself). In these cases we state that the existence of the related security property is possible. In other cases the scope of the security pattern under consideration is irrelevant to a specific security property. For example, the Available System patterns are not related at all to access control. In these cases the corresponding criteria for the pattern that is being considered are not mentioned.

We first discuss which of the qualitative properties that were previously described exist in the so-called Available System patterns. Someone could first make the observation that the basic aim of these security patterns is to make systems robust in the case of failure. So, we could first note that these patterns are designed in order for a system to fail in such a way that, after recovery no security problem has emerged due to the failure that occurred. Furthermore, by looking at the class diagrams of these patterns someone 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, protect 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

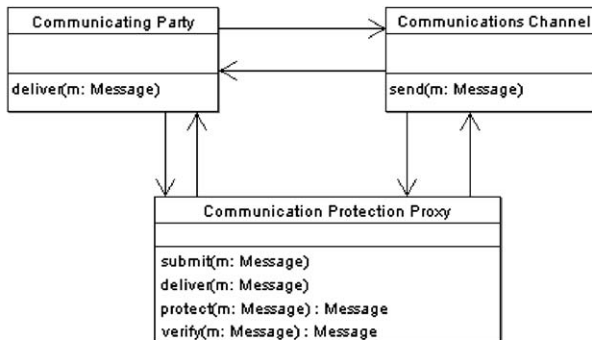

Fig. 10 – Class diagram of the Secure Communication pattern.

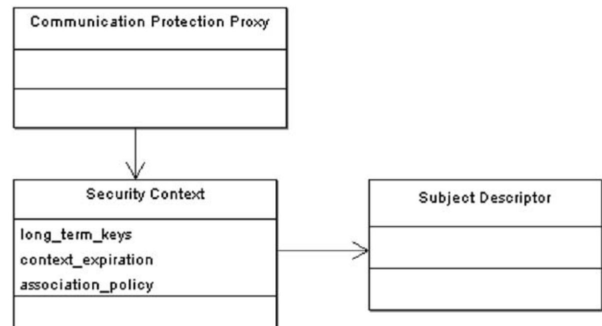

Fig. 11 – Class diagram of the Security Context pattern.

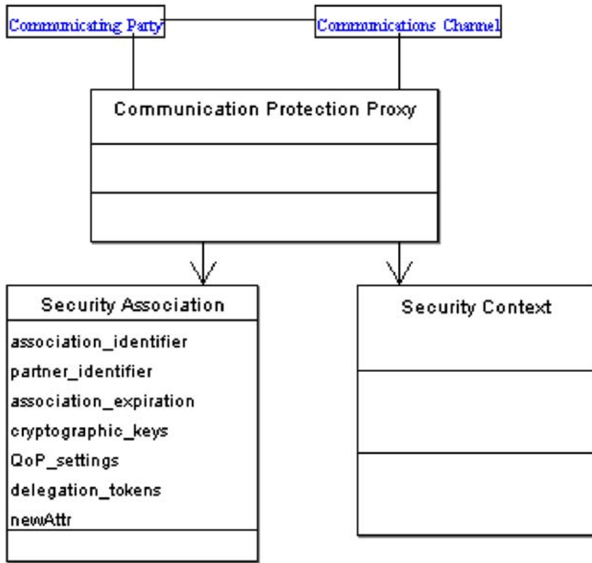

Fig. 12 – Class diagram of the Security Association pattern.

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. In order to detect Denial of Service attacks, an algorithm similar to the one proposed by Castro and Liskov (2000) could be used.

We describe the qualitative properties of the 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. It secures the weakest link, which in this case are the resources protected, since authentication is required to access them. Furthermore, its design is simple. Considering the second set of previously described criteria for avoiding software holes, we can pinpoint that by using an input checking mechanism 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 to deter 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, except that it does not protect from race conditions due to Time of Check versus Time of Use problems.

The Authenticator pattern (Lee Brown et al., 1999) 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 (Lee Brown et al., 1999), 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 Protected System pattern for the same reasons.

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

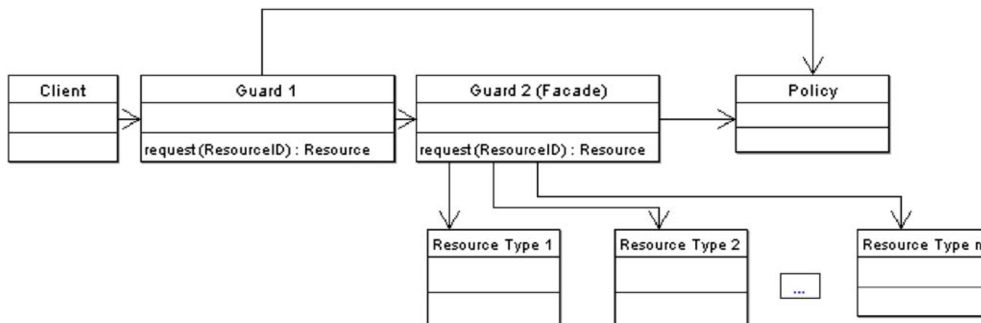

Fig. 13 – Class diagram of the Secure Proxy pattern.

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 principle of securing the weakest link, since the communication link, which is the weakest link of the system in this case, is protected. It follows the principle of compartmentalization, 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. Its design is simple. 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 all three basic types of software security holes (buffer overflows, poor access control, and race conditions). Regarding the third set of criteria, this pattern could protect from all types of attacks, except Repudiation 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. For protection from Repudiation attacks, an additional logging mechanism or a digital signature mechanism would be required.

The Security Context pattern aims to provide 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 and each action is subject to the constraints provided by security attributes. Furthermore, it has simple design. Regarding the protection from software security holes, we can estimate that the same as with the Secure Communication pattern holds for the same reasons. Regarding possible attacks, the Communication Protection Proxy can protect from Tampering, Information disclosure and Elevation of privilege attacks.

The Security Association pattern aims to define 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. Furthermore, it provides the participants with the information it will use to understand and verify the protection applied to messages received from the other party. Firstly, someone can make the general observation 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 (the weakest link in this specific pattern). 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 Secure Communication pattern holds for the same reasons. It protects from Spoofing identity 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 to define the relationship between the guards of two instances of the Protected System, where each instance ensures that specific security policies are followed. 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

Table 1 – Summary of the evaluation of the security patterns based on the 10 guiding principles by McGraw

Pattern name	Principles							
	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	Y	A		Y	A			P
Policy		Y		Y		Y	Y	P
Authenticator				Y	A	Y	Y	
Subject Descriptor				Y		Y	Y	
Secure Communication	Y				Y	Y	Y	
Security Context				Y		Y	Y	
Security Association	Y	Y		Y		Y	Y	
Security Proxy	Y	Y		Y			Y	

Explanations: Y, yes; A, against; P, possible.

Table 2 – Summary of the evaluation of security patterns based on the second set of criteria

Pattern name	Protection from buffer overflows	Good access control	Protection from race conditions
Protected System	P	P	P
Policy	P	P	
Secure Communication	P	P	P
Security Context	P	P	P
Security Association	P	P	P
Secure Proxy	P	P	P

Explanations: P, possible.

Table 3 – Summary of the evaluation of security patterns, based on the third set of criteria

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
Authenticator	P	P		P		P
Subject Descriptor						
Secure Communication	P	P		P	P	P
Security Context		P		P		P
Security Association	P	P		P		P
Secure Proxy	P	P		P		P

Explanations: P, protection exists; I, improved protection.

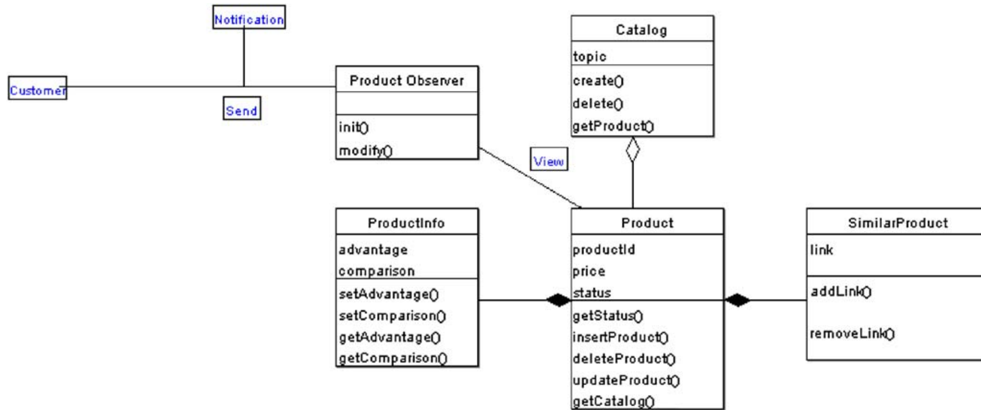

Fig. 14 – Class diagram of the part of an Internet Shop that is related to the catalog of the products.

guards. It secures the weakest link, since the weakest link in this case is the resources protected. 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.

All security patterns work neither for nor against principles 8 and 9.

The evaluation based on the first set of criteria can be summarized in Table 1.

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.

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

5. Case study

To illustrate the use of security patterns in order to enhance the security of a real design, we apply security patterns in parts of an Internet Shop as they are described in Fernandez

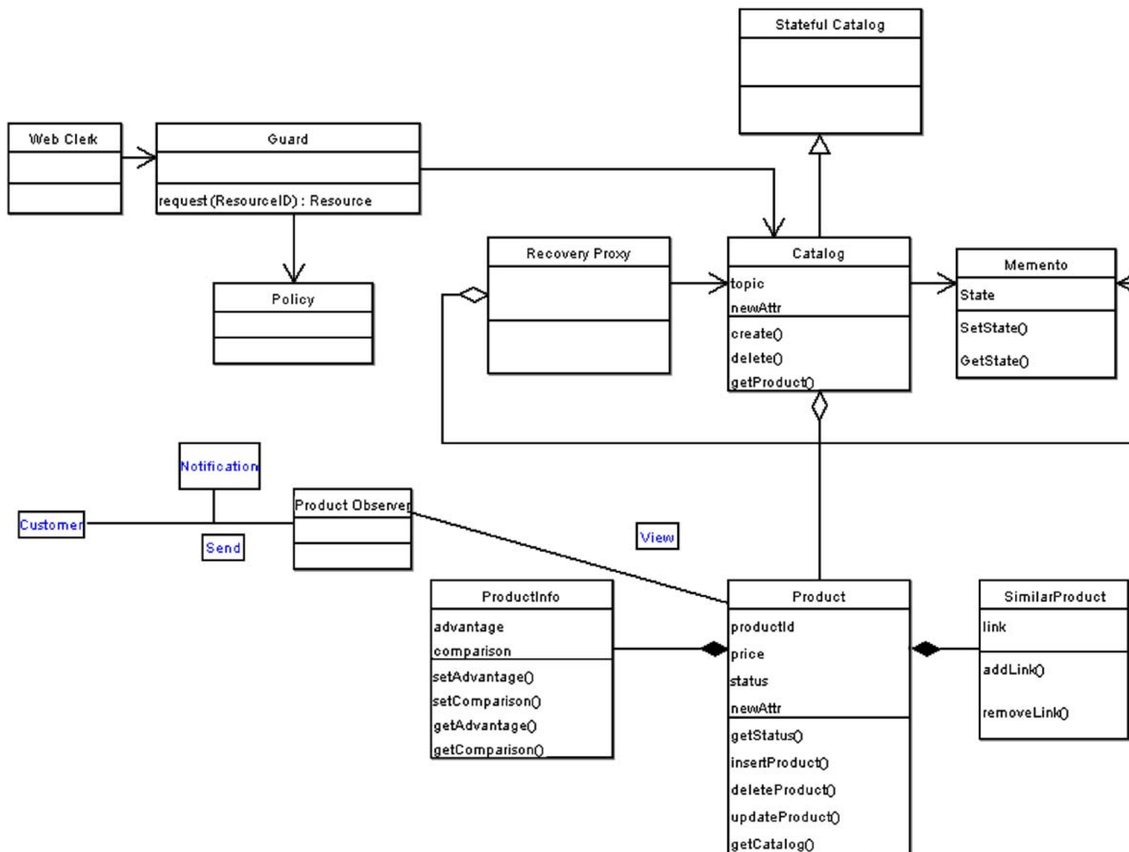

Fig. 15 – Class diagram of the security enhanced product catalog part of an Internet Shop.

et al. (2001). In the same way that design patterns generate well-structured architectures, as noted in Beck and Johnson (1994), we note that security patterns enhance the security of existing architectures and thus generate secure architectures. We have chosen a small scale system to illustrate the approach, because the security patterns existing in large and medium scale object oriented systems are not documented.

5.1. *The product catalog part*

First of all, we examine the class diagram related to the catalog of the products that is shown in Fig. 14. A *Catalog* is a collection of products. The *Product* class defines the type of product being sold. The *ProductInfo* class provides more detailed information about a product. Class *SimilarProduct* provides links to similar products. Modifications to the products are notified to the customers by e-mail, to let them know there is some new product or some important change to a product already provided by the Internet Shop. To achieve this, an Observer pattern indicated by *Product Observer* is used.

After this short description of the product catalog part diagram, we examine this design according to the 10 guiding principles by Viega and McGraw (2002). According to Principle 1 the weakest link should be secured. In this case, the weakest link, namely the product catalog is not properly protected. According to Principle 2 we should have at least two lines of defense. According to Principle 3 the system should be robust in the presence of failures. According to Principle 4 we should perform some authentication, in order for every operation to be done with the least privilege. Principle 5 to compartmentalize is followed in this case, since this part of the system is small enough. The system is simple enough to adhere to Principle 6. Principle 7, to promote privacy, can be achieved by performing some authentication. Principle 8, which advises us to protect the system from insider attacks can be achieved only in part by having incorporated some fault-tolerance mechanism in the system to protect from Denial of Service attacks. Principles 9 and 10, which advise us to use only software that has been already extensively tested, depend on the specific implementation of the system.

Table 4 – Evaluation of the security enhanced product catalog part of an Internet Shop

Security principles by McGraw	Principle 1	Y
	Principle 2	Y
	Principle 3	Y
	Principle 4	Y
	Principle 5	Y
	Principle 6	IP
	Principle 7	Y
	Principle 8	IP
Protection from software vulnerabilities based attacks	Protection from buffer overflows	P
	Good access control	P
	Protection from race conditions	P
Protection from STRIDE attacks	Protection from Spoofing identity attacks	P
	Protection from Tampering with data attacks	P
	Protection from Repudiation attacks	N
	Protection from Information disclosure attacks	P
	Protection from Elevation of privilege attacks	P

Explanations: Y, yes; N, no; IP, in part; P, possible.

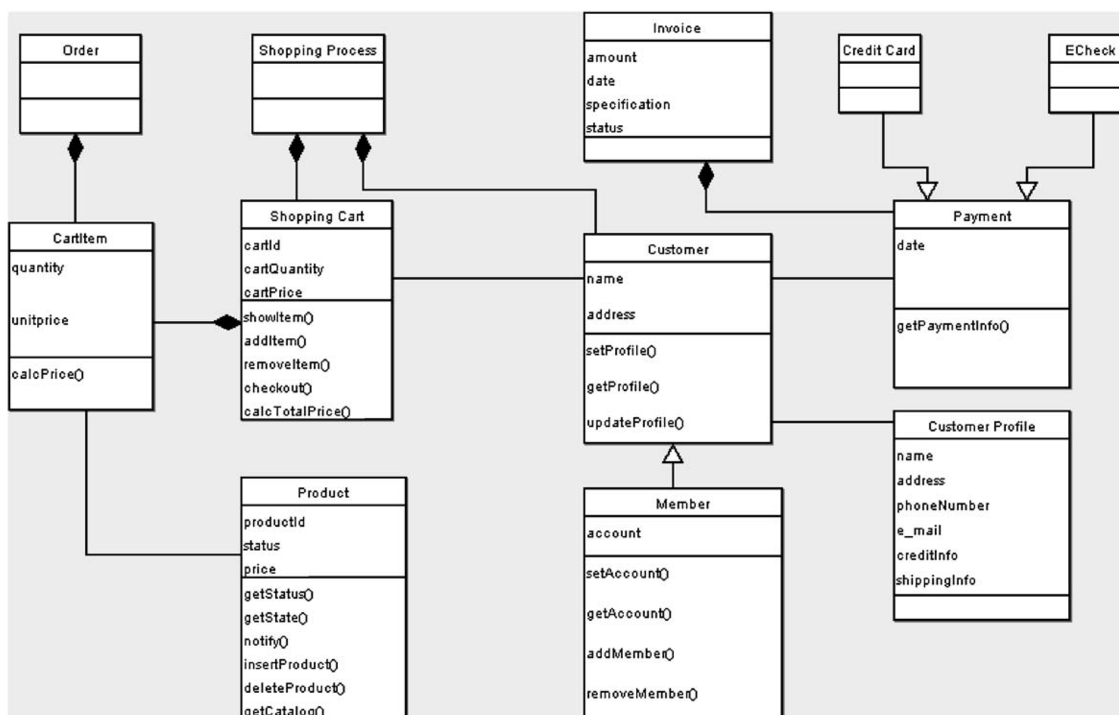

Fig. 16 – Class diagram of an Internet Shop that is related to the shopping process.

Next we examine the system according to the second set of criteria. As it is noted in Fernandez et al. (2001), there exists one actor in this system that has the task of the administration of the product catalog namely the Web Clerk. Buffer overflow attacks can occur only from the input provided by this actor since the Customer is only notified about changes to the product catalog. Furthermore, as it can be seen from Fig. 14, there exists no access control to the system and also race conditions may occur when more than one Web Clerks try to access the system.

By examining the system according to the third set of criteria, someone can note that the system is not protected from Spoofing, Tampering with data, Information disclosure and Elevation of privilege attacks since no authentication and-authorization mechanism exists in it. Furthermore, the system is not protected from Repudiation attacks, for which a logging mechanism is required and also not protected from Denial of Service attacks, which can be confronted by a fault-tolerance mechanism.

By taking into account all these observations, we propose the security enhancement of the system through its combination with a Protected System pattern and a Checkpointed System pattern as shown in Fig. 15.

First we examine the improved system according to the 10 guiding principles by McGraw. Principle 1 is followed, because we can see that the resource that we are trying to protect, namely the product catalog, is secure enough through the use of the Protected System pattern. Principle 2, to practice defense in depth is achieved through the combination of two patterns that do not offer this characteristic by themselves. Principle 3, to fail securely, is achieved through the use of the fault tolerant Protected System pattern. Principle 4, to follow the principle of least privilege, and Principle 7 to promote privacy, can be achieved through the good use of authentication and authorization mechanisms by the Guard of the Protected System pattern. Principle 6, to keep the system simple does apply only in part in the new design since the system is now more complicated. Principle 8, to protect the system

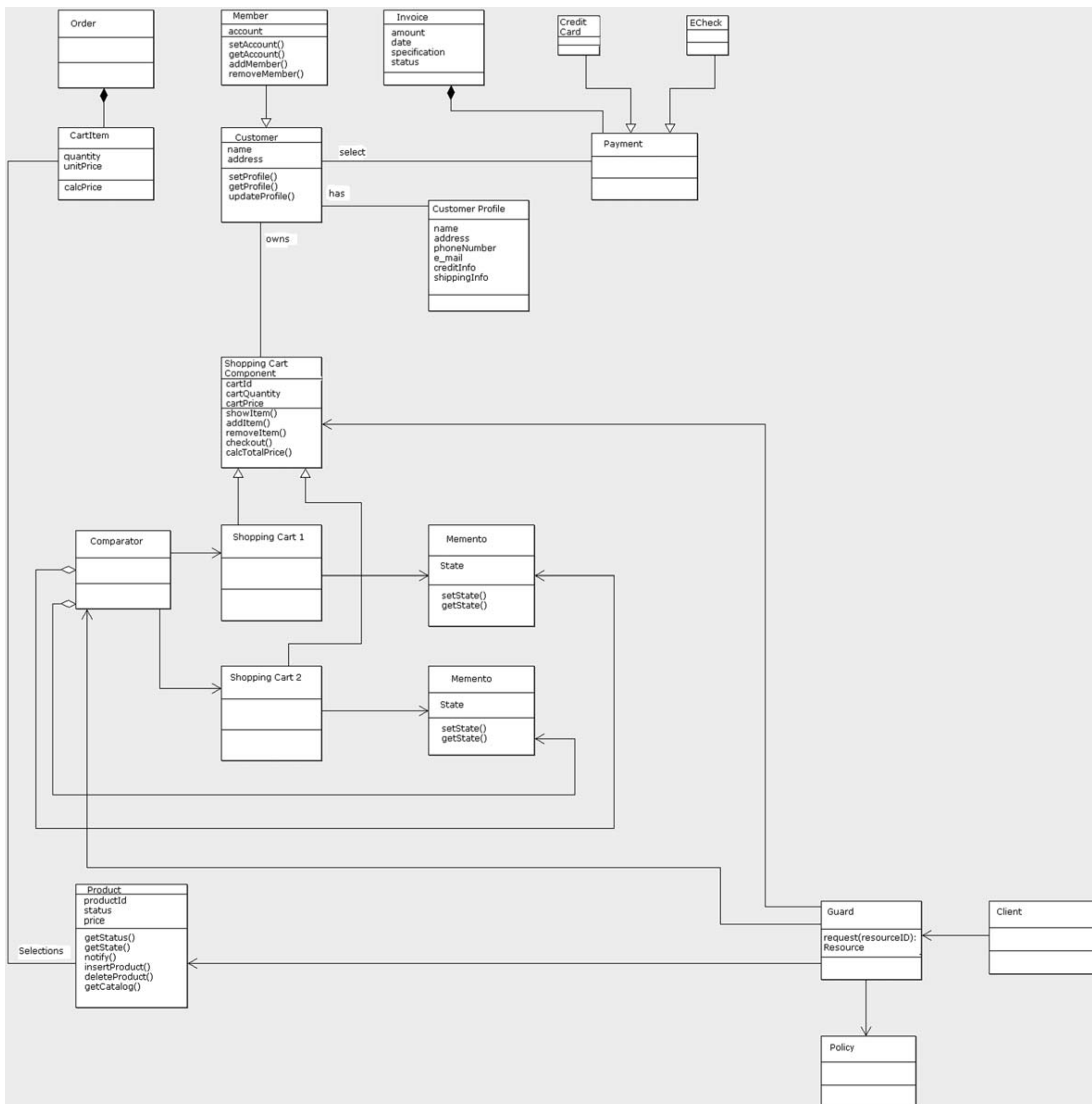

Fig. 17 – Class diagram of the more secure design for the part of an Internet Shop that is related to the shopping process.

even from insider attacks is also achieved only in part through the use of the Checkpointed System pattern.

The system can be protected from three main sources of software attacks described in the second set of criteria through the good operation of the Guard of the Protected System pattern.

According to the possible attacks to the system, the Protected System pattern can offer protection from Spoofing, Tampering, Information disclosure, and Elevation of privilege attacks, while protection from Denial of Service attacks can be achieved through the use of the Checkpointed System pattern. For protection from Repudiation attacks, an additional logging mechanism should be used.

The evaluation of the security enhanced product catalog part of the Internet Shop is summarized in Table 4. Principles 9 and 10 depend on the implementation of the system as was previously noted.

5.2. *The shopping process part*

Secondly, we examine the part of the system that is related to the shopping process that is shown in Fig. 16. The class *Shopping Process* is the entry point to the complete process. The *Shopping Cart* class collects information about all the products a customer has selected. The aim of the *CartItem* class is to indicate the quantity and the product selected by a customer. When a customer performs a selection operation of some kind of product, a new *CartItem* object is created and added to the cart. A customer besides adding products to the cart, can also query them and remove items from it. Payment through *Credit Card* or *ECheck* is possible. When a customer checks out a cart, the prices of all products selected are calculated, the customer billing and shipping information is retrieved and finally an *Order* and an *Invoice* are generated.

The same observations that we did for the product catalog part of the Internet Shop apply also in this case. The only difference is that in this case both the product and the shopping cart must be protected. Furthermore, it must be ensured that the shopping cart can always return to the state before possible failure. (For example, a customer should not have to repeat all prior steps or selections.) To achieve this we use a duplicate shopping cart and the Comparator-Checked Fault Tolerant System pattern instead of the Checkpointed System pattern. Additionally, since we use this pattern, we must also protect from unprivileged access the Comparator of the pattern. The more secure design is presented in Fig. 17.

The same observations that we did for the more secure product catalog part apply also in the case of the more secure shopping process part, with the only difference that this design does not follow the principle of keeping the system simple.

The evaluation of the more secure design for the part of an Internet Shop that is related to the shopping process is summarized in Table 5.

Table 5 – Evaluation of the more secure design of the part of an Internet Shop that is related to the shopping process

Security principles by McGraw	Principle 1	Y
	Principle 2	Y
	Principle 3	Y
	Principle 4	Y
	Principle 5	Y
	Principle 6	N
	Principle 7	Y
	Principle 8	IP
Protection from software vulnerabilities based attacks	Protection from buffer overflows	P
	Good access control	P
	Protection from race conditions	P
Protection from STRIDE attacks	Protection from Spoofing identity attacks	P
	Protection from Tampering with data attacks	P
	Protection from Repudiation attacks	N
	Protection from Information disclosure attacks	P
	Protection from Elevation of privilege attacks	P

Explanations: Y, yes; N, no; IP, in part; P, possible.

6. Conclusions and future work

As in any other engineering discipline, no patterned solution in its own has all the desired characteristics. Thus, 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 that can offer a reasonable protection against the most common attacks.

We believe that beyond the qualitative evaluation of security patterns, a quantitative approach to evaluating the security of software systems would be desirable. A quantitative measure of security would be valuable not only for comparing several alternatives but also to assess the evolution of a design through successive generations. 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.

The most important observation though, is that although the security offered by a software system depends on the specific implementation, the architecture of the system and more specifically the exploitation of the most suitable software security patterns can lead to higher reliability with regard to security.

REFERENCES

Avizienis A, Laprie JC, Randell B, Landwehr C. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 2004;1(1):11–33.

Beck K, Johnson R. Patterns generate architectures. Lecture Notes in Computer Science 1994;821:139–49.

Blakley B, Heath C, and Members of the Open Group Security Forum. Security design patterns. Open Group technical guide; 2004.

Braga A, Rubira C, Dahab R. Tropyc: a pattern language for cryptographic software. In: Proceedings of the fifth conference on pattern languages of programming (PLoP '98); 1998.

Buschmann F, Meunier R, Rohnert H, Sommerland P, Stahl M. Pattern oriented software architecture – a system of patterns. John Wiley and Sons; 1996.

Castro M, Liskov B. Proactive recovery in a byzantine-fault-tolerant system. In: Proceedings of OSDI 2000; 2000.

Cheng B, Konrad S, Campbell L, 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.

Fernandez E. Metadata and authorization patterns, <http://www.cse.fau.edu/~ed/MetadataPatterns.pdf>; 1996.

Fernandez E, Liu Y, Pan RY. Patterns for internet shops. In: Proceedings of the eighth conference on pattern languages of programming (PLoP '01); 2001.

Fites P, Kratz M. Information systems security: a practitioner's reference. International Thomson Computer Press; 1996.

Fowler M. UML distilled: a brief guide to the standard modeling language. 3rd ed. Addison Wesley; 2003.

Gamma E, Helm R, Johnson R, Vlissides J. Design patterns. Addison Wesley; 1995.

Hafiz M, Johnson RE, Afandi R. The security architecture of qmail. In: Proceedings of the 11th conference on pattern languages of programming (PLoP '04); 2004.

Halkidis ST, Chatzigeorgiou A, Stephanides G. A qualitative evaluation of security patterns. In: Proceedings of the sixth international conference on information and communications security (ICICS '04). Lecture Notes in Computer Science, vol. 3269; 2004.

Howard M, LeBlanc D. Writing secure code. Microsoft Press; 2002.

Kienzle D, Elder M. Security patterns for web application development, University of Virginia technical report; 2002.

Kis M. Information security antipatterns in software requirements engineering. In: Proceedings of the ninth conference on pattern languages of programming (PLoP '02); 2002.

Lee Brown, F, Di Vietri J, Diaz de Villegas G, Fernandez E. The authenticator pattern. In: Proceedings of the Sixth conference on pattern languages of programming (PLoP '99); 1999.

Mahmoud Q. Security policy: a design pattern for mobile Java code. In: Proceedings of the seventh conference on pattern languages of programming (PLoP '00); 2000.

Mouratidis H, Giorgini P, Schumacher M. Security patterns for agent systems. In: Proceedings of the eighth european conference on pattern languages of programs (EuroPLoP '03); 2003.

Nicol DM, Sanders WH, Trivedi KS. Model-based evaluation: from dependability to security. IEEE Transactions on Dependable and Secure Computing 2004;1(1):48–65.

Ramachandran J. Designing security architecture solutions. John Wiley and Sons; 2002.

Romanosky S. Security design patterns, <http://www.romanosky.net/papers/securityDesignPatterns.html>; 2002.

Tipton H, Krause M, editors. Information security management handbook. 4th ed. CRC Press – Auerbach Publications; 1999.

Viega J, McGraw G. Building secure software, how to avoid security problems the right way. Addison Wesley; 2002.

Weiss M. Patterns for web applications. In: Proceedings of the 10th conference on pattern languages of programming (PLoP '03); 2003.

Yoder J, Barcalow J. Architectural patterns for enabling application security. In: Proceedings of the 4th conference on pattern languages of programming (PLoP '97); 1997.

Spyros T. Halkidis received the BS degree and the MS degree in Computer Science from the University of Crete, Greece, in 1996 and 1998, respectively. He also received an MBA from the University of Macedonia, Greece, in 2000. Since 2003 he is a PhD candidate in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. His current research interests include secure software and security patterns.

Alexander Chatzigeorgiou received the Diploma in electrical engineering and the PhD degree in computer science from the Aristotle University of Thessaloniki, Greece, in 1996 and 2000, respectively. He is a lecturer in software engineering in the Department of Applied Informatics at the University of Macedonia, Thessaloniki, Greece. From 1997 to 1999, he was with Intracom S.A. Greece as a telecommunications software designer. His research interests are in object-oriented design metrics, pattern detection, and software security. He is a member of the IEEE Computer Society.

George Stephanides received the PhD degree in applied informatics from the University of Macedonia. He is an assistant professor in the Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece. His current research and development activities are in the applications of mathematical programming, security and cryptography and application specific software. He is a member of the IEEE Computer Society, ACM and SIAM.