ISSN No. 0976-5697

Volume 2, No. 4, July-August 2011

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

A Survey on the Existing Safety Patterns in Software Security

Salar Azimi Haredasht*
Department of Computer Engineering
Islamic Azad University, Qazvin Branch
Qazvin, Iran
Salar.azimi1@gmail.com

Hassan rashidi
Department of Computer Engineering
Islamic Azad University, Qazvin Branch
Qazvin, Iran
hrashi@gmail.com

Abstract: Security is one of the basic aims in software development. To discover safety problems, the software developers must consider the security control management in software life-cycle. Considering the security foundation in design process software reduces maintenance costs and achieving takes conveniently and quickly. Meanwhile designing safe patterns and implementation of software in the basis of them is very important. With combination of existing safety patterns, new pattern can be created with high security feature. In this paper the final cost of software security is shown in software life-cycle and features of safety patterns are discussed. Also security and designing of safety patterns are examined, advantages and disadvantages of the patterns are analyzed.

Keywords: system requirements; security engineering; software life-cycle; security pattern.

I. INTRODUCTION

For the last 30 years, due to increasing production software and growing software size, security has been one of the main challenge in developing software [1]. With genesis network, web and web-based software pay attention to software security has found very importance. Whatever access to software is wider accordingly, its vulnerability against attacks is more. Attention to web-based software security due to their accessible is in the first priority. With the Moore's Law, computing processing power doubles every eighteen months, and as a consequence, software size and complexity grow rapidly to consume all available memory and processing power. In section 2 of this paper, security engineering are described. In section 3, security pattern for software security are presented. Section 4 shows the strengths and weaknesses of each model and section 5 contains conclusions.

II. SECURITY ENGINEERING

Security engineering is mandatory part of software engineering that by using of tools, process and techniques keeps system reliable against errors and abuse. Because of the security importance in developing software of major projects, security engineering must be separated from system analyst and programming group. Also security engineering focuses on the tools and methods needed to design, implement and test complete systems, and to adapt existing systems are their environment evolves [2]. Software engineering has been relatively successful in programming in the large, producing large software effectively, but it has been so successful in producing secure software.

A. System Requiremenrs

A system requirement is divided in two main parts: functional requirements (FRs) and non-functional requirements (NFRs). Prior is depending of programming language that really determine which part of software must be run [5]. Second consist of internal system requirements such as reliability maintainability, performance, reusability, security and so on. In short, FRs determines which piece of software should be implemented but NFRs describe how that software should perform those tasks. Software developers not enough attention to NFRs. One reason for this is that product development by necessity must optimize the use of limited resources such as time, funds, personnel, etc.

B. Software Life-Cycle

Software security needs to be considered from beginning of software development life-cycle because adjourn the security consideration after the production software makes more resolving the cost of software errors. Therefore based on published research by B.Boehm and V.Basili (IEEE 2005), to resolve an error after installing it, is hundred times more than when that error discover and resolve in early the developed it, figure 1. About security errors this digit will go higher because in addition to cost of resolving error, to atone due to abuse of this security flaw in order to sabotage, steal information and other attacks, software developers are responsible.

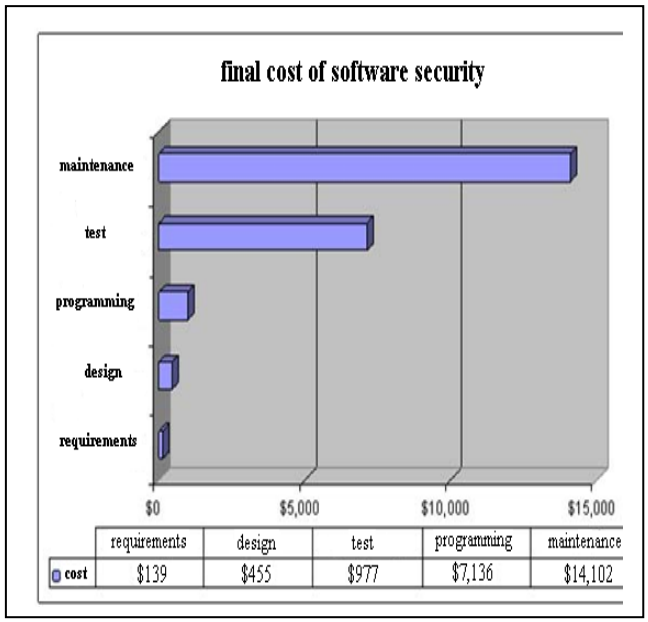

Figure: 1 Final cost of software security

III. EXISTING SECURITY PATTERNS

Information systems security have been broadly considered by researchers and it has strong link with software industry. Recent researches show that a main source attack to software security properties arises from the weakness of developing software. Generally software is designed and developed without security being in the mind of the developers. 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 part of software written in such away that they can be exploited to perform an attack.) in this paper existing and popular security patterns are evaluated. How security design patterns lead software without holes and how one good system software uses one suitable security pattern that be able to respond each possible attacks.

A. A Short Review of Existing Security Patterns

In field of software security patterns, first of all, Yoder and Baralow represent several security patterns in 1997, but their research has not clear and precise definition [6]. Each presented papers refer to security patterns in different perspective. For example, security patterns as basic elements of security system architecture [8], security patterns for cryptographic software and security patterns for agent systems [3]. All researchers have a coordinated effort for providing a comprehensive list of safety patterns available with the application of each patterns. Safety patterns idea fully released different with design pattern in 2004 by a group.

B. Software Security 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. Figure 2 illustrates class diagram of the checkpoint system pattern. The Checkpointed System pattern offers protection from loss or corruption of state information in case a component fails. 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[7]. 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. 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. Figure 3 illustrates class diagram of the Standby pattern. The Standby pattern can be used in cases where failed components may not be recoverable but a similar or identical backup component is available. 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 Standby pattern can be used in cases where failed components may not be recoverable but a similar or identical backup component is available.

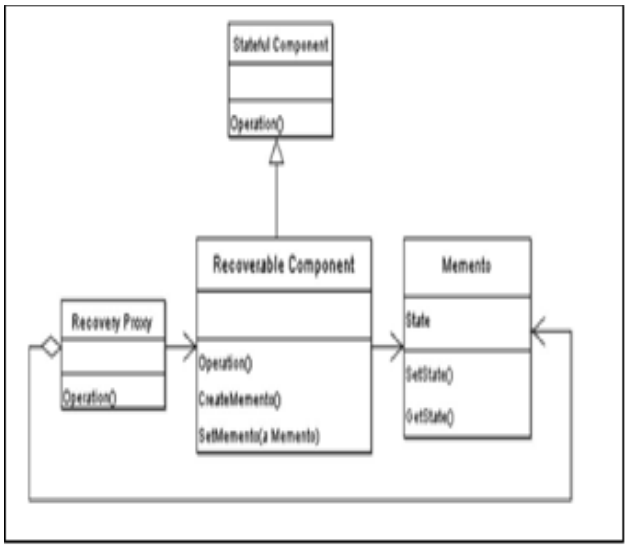

Figure: 2 Class diagram of checkpointed system pattern

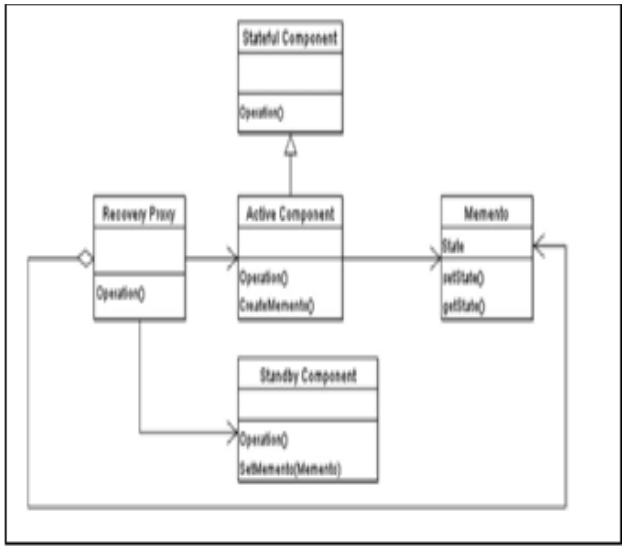

Figure: 3 Class diagram of the Standby pattern.

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. Figure 4 illustrates class diagram of the Comparator-Checked Fault Tolerant System pattern. 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[7]. The intent of the Error Detection/Correction pattern is to add redundancy to data to facilitate later detection of and recovery of errors. Figure 5 illustrates a class diagram for this pattern. 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.

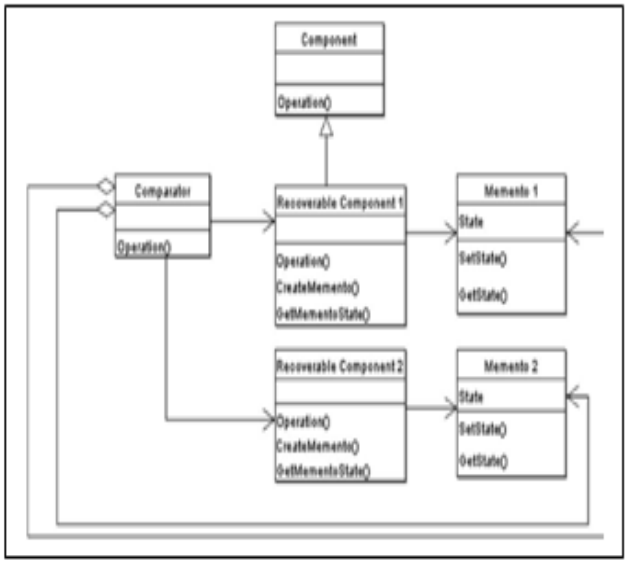

Figure 4. Class diagram of the comparator-cheched fault tolerant system patter.

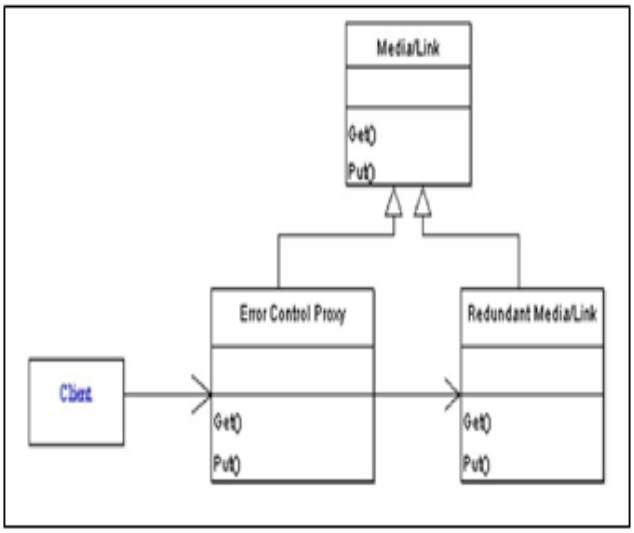

Figure 5. Class diagram of the error detection/correction pattern

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. Figure 6 illustrates class diagram of this pattern. The Guard controls access requests to resources according to a predefined policy. Of course the Guard itself must be robust to malicious code attacks. 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 [10]. Figure 7 illustrates class diagram of this pattern. The way it works is that Policy enforces rules that are to be applied by the Guard for possible authentication.

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 is to perform authentication of a requesting process, before deciding access to distributed objects. Figure 8 illustrates a class diagram for this pattern. If the authentication process performed by the Authenticator is successful, the Authenticator forwards a request for the creation of a Remote Object to the Object Factory [9].

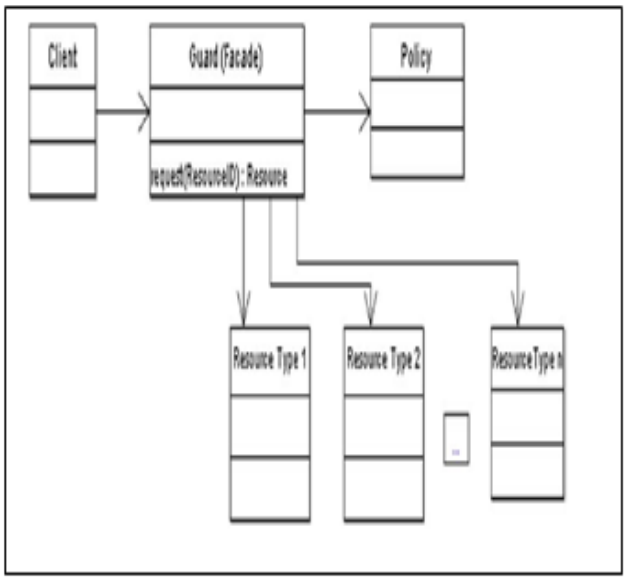

Figure 6. Class diagram of the protected system pattern

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. The Secure Communication pattern protects the communication channel. Figure 9 illustrates class diagram of this pattern. 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.

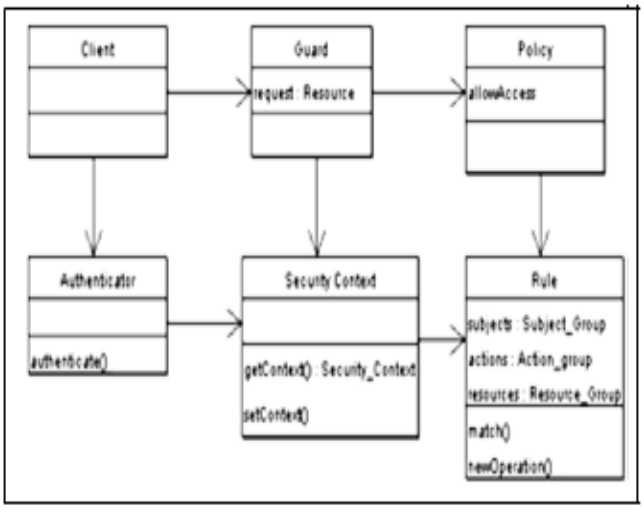

Figure 7. Class diagram of the policy pattern

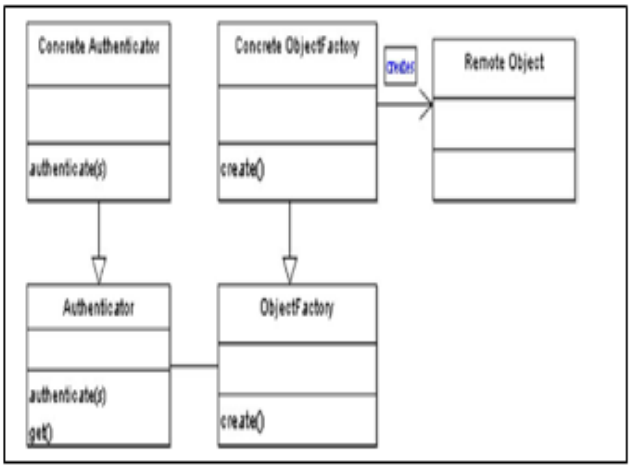

Figure 8. Class diagram of the authenticator pattern

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 10 shows a class diagram of this 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.

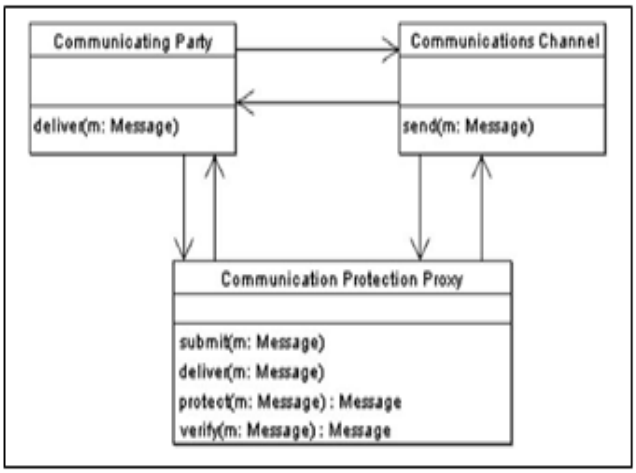

Figure 9. Class diagram of the secure communication pattern

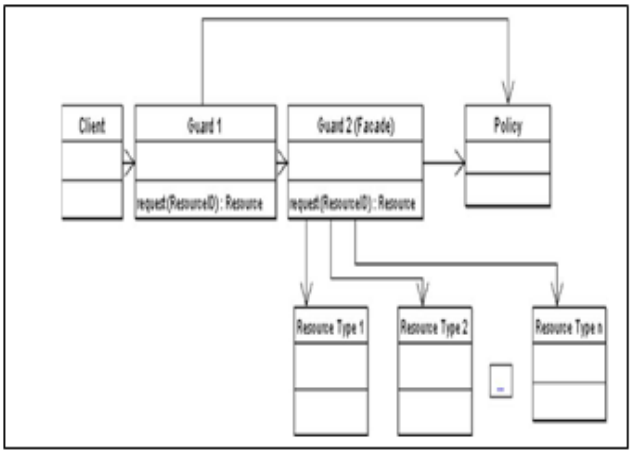

Figure 10. Class diagram of the secure proxy pattern

IV. STRENGTHS AND WEAKNESSES OF SAFTY PATTERNS

In this section strengths and weaknesses of security patterns are investigated and represented in table 1. Main properties of an security pattern are error detection and repair capabilities, data redundancy, speed and implementation.

Table I. strengths of the safty patterns

Security pattern	strengths
checkpoint	In failed state it transfers system to the valid case. It is Easy to implementation.
standby	In failed state it was down by same and similar component.
Comparator-checked Fault tolerant	It recognizes errors that still leading to failure yet.
Error detection / correction	It recognizes next errors and repair them.
Protected	Its reliability and security is high. Security policy is run by guard.
Policy	Guard and confirming validity implement the security rules. Its reliability is high.
Authenticator	Authentication of request process is down before accessing to resources.
Secure communication	Development of health data is possible.
Secure proxy	Its speed operation is higher. If not successful review by the request access of first guard to resources is canceled.

Table II. weaknesses of the safty patterns

Security pattern	weaknesses
checkpoint	It recognize errors that still happened yet.
standby	It has data redundancy. There should be a similar component.
Comparator- checked Fault tolerant	Its implementation is easier than the tow above patterns.
Error detection / correction	It has data redundancy.
Protected	Guard must be robust against attacks.
Policy	Guard must be robust against attacks. Its implementation is complex.
Authenticator	Its reliability and security is low.
Secure communication	It has data redundancy.
Secure proxy	Its implementation is complex.

V. CONCLUSION

Generally, software security first of all depends on security patterns with high security feature. Pattern is suitable that have high error detection and repair capabilities, minimum data redundancy and easy implementation. Meanwhile speed of security pattern is an important factor. To secure one software system a good combination of safety patterns with high security is required when system is designed. In web-based software due to their extensive access, security is important to note. Also it must be

considered in the software life-cycle. In the design process reasonable protection against attacks must be considered.

VI. REFERENCES

[1] Andy Ju An Wang "Security Testing in Software Engineering Courses", Department of Software Engineering School of Computing and Software Engineering Southern Polytechnic State University Marietta,, October 20 – 23, IEEE, 2004.

[2] Anderson, R.," Security Engineering: A Guide to Building Dependable Distributed Systems", Wiley 2001.

[3] Devanbu, P. T.; S. Stubblebine, "Software Engineering for Security": a roadmap, proceeding of the Conference on The Future of Software Engineering, p 227-239 ACM, Vol 46, Issue 6, p 75-78, ACM Press, 2003.

[4] Kienzle D, Elder M. "Security patterns for web application development', University of Virginia technical report; 2002.

[5] 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.

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

[7] Spyris T. Halkidis, Alexander Chatzigeorgiou, George Stephanidis," A qualitative analysis of software security patterns", Department of applied informatics, university of Macedonia, Egnatia 156, GR-54006 Thessaloniki, Greece, Elsivier ltd,2006

[8] Ramachandran J, John Wiley and Sons, "Designing security architecture solutions', 2002.

[9] Lee Brown, F, Di Vietri J, Diaz de Villegas G, Fernandez E. Theauthenticator pattern. In: Proceedings of the Sixth conferenceon pattern languages of programming (PLoP '99); 1999.

[10] Mahmoud Q. Security policy: a design pattern for mobile Javacode. n: Proceedings of the seventh conference on patternlanguages of programming (PLoP '00); 2000.