Quantitative Evaluation of Systems with Security Patterns Using a Fuzzy Approach

Spyros T. Halkidis, Alexander Chatzigeorgiou, and George Stephanides

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

Abstract. The importance of Software Security has been evident, since it has been shown that most attacks to software systems are based on vulnerabilities caused by software poorly designed and developed. Furthermore, it has been discovered that it is desirable to embed security already at design phase. Therefore, patterns aiming at enhancing the security of a software system, called security patterns, have been suggested. The main target of this paper is to propose a mathematical model, based on fuzzy set theory, in order to quantify the security characteristics of systems using security patterns. In order to achieve this we first determine experimentally to what extent specific security patterns enhance several security aspects of systems. To determine this, we have developed two systems, one without security patterns and one containing them and have experimentally determined the level of the higher robustness to attacks of the latter. The proposed mathematical model follows.

Keywords: Software Security, Security Patterns, Fuzzy Risk Analysis.

1 Introduction

The importance of software security has been evident since the discovery that most attacks to real software systems are initiated by software poorly designed and developed [34, 32, 15, 16]. Furthermore, it has been shown that the earlier we incorporate security in a software system the better [34]. Therefore, in analogy to design patterns [13], which aim at making software well structured and reusable, Security Patterns [33, 4] have been proposed, targeting at imposing some level of security to systems already at the design phase.

In this paper, we try to propose a mathematical model for the security of systems using security patterns. To achieve this, we first investigate to what extent specific security patterns reinforce several aspects of software systems security. To determine this experimentally we have built two software systems, which are the implementations of web applications, one without security patterns and one where security patterns were added to the former. We studied all applications under known categories of attacks to web applications [29]. To perform our analysis we have used the AppScan Web Application Penetration Testing tool, and organized a contest to study other approaches for evaluating software systems for vulnerabilities. We have estimated experimentally to what extent the system using security patterns is more

R. Meersman, Z. Tari, P. Herrero et al. (Eds.): OTM Workshops 2006, LNCS 4277, pp. 554 – 564, 2006.

robust to attacks compared to the one that does not use them. Furthermore, initiated by the findings, we propose expressions for the resistance to STRIDE attacks [16] for the patterns examined. Finally, we use results from fuzzy reliability [7] and the application of fault trees [6, 1], to examine the security properties of systems using security patterns and illustrate the application of the related results to a system properly using the security patterns examined.

The remainder of the paper is organized as follows. In Section 2 we briefly review work on security patterns. Section 3 is a description of the systems under examination. In Section 4 we describe the results of our evaluation. Section 5 proposes a mathematical model for systems using security patterns using fuzzy numbers and fuzzy fault trees. Finally, in Section 6 we make some conclusions and propose future research directions.

2 Security Patterns

Since it has been evident that it is desirable to incorporate security already at the design level [34, 16], various efforts to propose security patterns, that serve this aim, have been done.

Yoder and Barcalow were the first to propose security patterns [35] in 1997. Since then, various security patterns were introduced. Patterns for enterprise applications [27], patterns for authentication and authorization [11, 20], patterns for web applications [18, 36], patterns for mobile java code [23], patterns for cryptographic software [5] and patterns for agent systems [24]. Though, all these efforts did not share some common terminology.

The first effort to provide a comprehensive review of existing security patterns was done by the OpenGroup Security Forum [4]. In this work, security patterns are divided into Available System Patterns, which are related to fault tolerance [25] and Protected System Patterns, which aim at protecting resources.

In an earlier work [14] we have performed a qualitative evaluation of these security patterns.

Recently, a summary of security patterns has appeared in the literature [33]. In this text security patterns are divided into web tier security patterns, business tier security patterns, security patterns for web services, security patterns for identity management and security patterns for service provisioning. In this paper we focus on web tier security patterns and business tier security patterns.

3 Description of the Systems Under Examination

In order to perform our security analysis, we have used two systems. Specifically, we have developed a simple e-commerce application without security patterns, hereafter denoted as “first” application, and a second application where security patterns were added to it, hereafter denoted as “second” application.

The first application under consideration is a typical J2EE (Java 2 Enterprise Edition, now referred to as Java EE) application with no security patterns. We have chosen J2EE as a platform for both applications since the J2EE platform is widely

used in business applications and is useful from the security point of view [33, 3]. In our systems we have used JBoss 4.0.3 as an application server that encompasses the web and business tier, and MySQL 5.0 for the database tier.

The first system consists of 46 classes. It has 16 servlets and 7 EJBs. One EJB works as a web service endpoint [26].

We have left on purpose on this system so-called "security holes" that attackers can exploit.

First of all, several sources for SQL injection [29, 2, 31, 12] were included. An SQL injection attack occurs when an attacker is able to insert a series of SQL statements into a query that is formed by an application, by manipulating data input that is not properly validated [2]. SQL injection attacks can cause unauthorized viewing of tables, database table modification or even deletion of database tables.

Furthermore, several sources for cross-site scripting were included. Cross site scripting [29, 10, 30, 17], also known as XSS, occurs in a web application when data input in one page which are not properly validated, are shown in another page. In this case, script code can be input in the former page that is consequently executed in the latter. In this way it is easy to perform an Information Disclosure attack [16] for example by using Javascript code that shows the cookie values of sensitive information.

Additionally, several sources for HTTP Response Splitting [19], were included in the application. HTTP Response Splitting attacks can occur when user data that were not properly validated are included in the redirection URL of a redirection response, or when data that were not properly validated are included in the cookie value or name, when the response sets a cookie. In these cases, by manipulating http headers, it is easy to create two responses instead of one where in the second response an XSS attack can be performed. Variants of this attack include Web Site Defacement, Cross User page defacement, Hijacking pages with user specific information and Browser Cache Poisoning [19].

Furthermore, in the first application no SSL connection was used and therefore sensitive information such as credentials and important information in cookies could be eavesdropped.

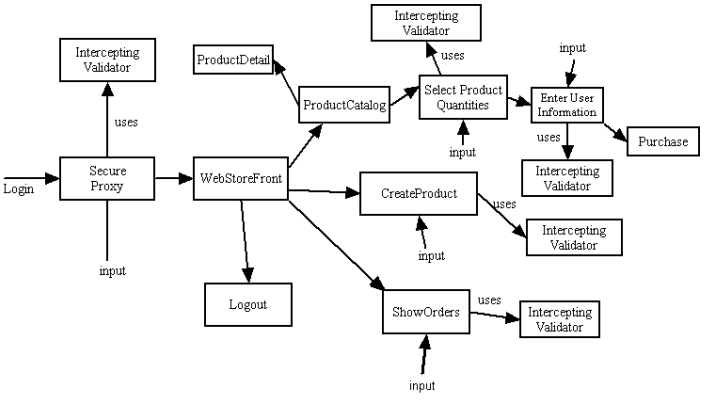

Fig. 1. Block diagram of the second application under examination

Finally, servlet member variables race conditions were included, which could be exploited by a number of users acting simultaneously.

In the second application we have built, the sources for attacks were not removed, but security patterns were used with the aim of protecting against them. The second application consists of 62 classes. It has 17 Servlets and 9 EJBs where one EJB again serves as an endpoint for the web service. The security patterns used in this system are the Secure Proxy pattern, Login Tunnel variant [4], the Secure Pipe pattern, the Secure Logger pattern, Secure Log Store Strategy, the Intercepting Validator pattern and the Container Managed Security pattern [33]. In Figure 1 we show a block diagram that consists of the main components of the second application with some of the security patterns used. Solid arrows show the flow of information.

4 Evaluation of the Systems with Regard to Attacks

In order to evaluate the systems with regard to attacks, we have used Watchfire's AppScan web application penetration testing tool. Furthermore, we have initiated a web application security contest, which was won by Benjamin Livshits from Stanford University. Livshits used static analysis tools to find the security flaws which are described in several papers [21, 22].

Both approaches found the major security flaws of the applications, meaning SQL Injection and Cross Site Scripting vulnerabilities. However both approaches had several false positives. AppScan for example found sources for buffer overflows, while java was used and the static analysis approach found sources for SQL injection in the second application, by examining the code for the EJBs, while proper input validation was done at the Web Tier. Race conditions for servlet member variables were found only by the static analysis approach. Several application errors of low severity not found by the static approach, were found by AppScan (checking for proper session variable values, that though not cause security risks). AppScan found the unencrypted login request flaw in the first application that did not use SSL. AppScan also found unencrypted SSL parameter flaws in the second application, which are of low severity. HTTP response splitting attacks in the first application as well as race conditions existing in the third application were found by neither of the approaches.

Additionally, the security flaws found by both approaches, were fewer in the case of the second application in comparison to the first one. The difference between the number of flaws found for the first and the second application was much more prominent in the set of high-risk flaws.

After careful analysis of the results we concluded that proper use of the security patterns leads to remediation of all the security flaws, except flaws that are of minor risk, like unencrypted SSL parameters (of course this flaw is of minor risk only when the unencrypted parameters are not crucial like in our case). These flaws that remain even after the use of security patterns, are due to the degrees of freedom left to the programmer even after using them imposes some level of security. Furthermore, current security patterns impose no rules for the use of servlet member variables and therefore race conditions may remain in a system using security patterns.

The Intercepting Validator pattern, when used for all input, including session variables, and variables that are not input by the user but still posted, protects from SQL Injection, Cross-Site scripting, and HTTP Response Splitting attacks. It offers therefore very high resistance to Tampering with Data and Information Disclosure Attacks [16].

The Secure Proxy pattern, Login Tunnel variant, has two levels of authentication in order to protect from Spoofing Identity, Elevation of Privilege and Information Disclosure attacks. Its resistance to related attacks can be estimated by considering it to be the equivalent the protection of two guards [4] connected in a series. The resistance of both of these patterns to attacks is dependent to the robustness of the authentication mechanism to dictionary attacks. Recent studies [37, 28] have shown that dictionary attacks, with a usual distribution of the complexity of the passwords selected, succeed 15-20% of the times. The authentication mechanism of the Protected System pattern can still be marked as of high security. All authentication patterns and consequently these two patterns examined here should be resistant to eavesdropping attacks to serve their purpose. Therefore, they should always be used in combination with the Secure Pipe pattern that provides SSL encryption.

The Secure Pipe pattern offers protection from information disclosure attacks. The programmer can still use unencrypted parameters in an SSL request, but usually, when these parameters are not of crucial importance this kind of flaw is of minor risk.

The Container Managed Security Pattern implements an authorization mechanism. It protects from Elevation of Privilege, Information Disclosure and partly from Spoofing Identity attacks, since anyone who belongs to the Role allowed to access the EJBs could do so.

Table 1. Resistance of the security patterns examined to STRIDE attacks

	S	T	R	I	D	E
Intercepting Validator		very high		very high		
Guard of Secure Proxy with Secure Pipe	high			high		high
Container Managed Security	medium			very high		very high
Secure Logger		very high				

Finally, the Secure Logger pattern protects from tampering the log created.

The evaluation of these security patterns with respect to the STRIDE (Spoofing Identity, Tampering with Data, Repudiation, Information Disclosure, Elevation of Privilege) model [16] is summarized in Table 1. The irrelevant entries are left blank.

5 Fuzzy Mathematical Model for Systems Using Security Patterns

One of the targets in our research was to build a mathematical model for systems that use security patterns, based on our findings for the level of security each pattern

offers. The most appropriate models for our purpose seem to be risk analysis models [1]. We have chosen to use a fuzzy risk analysis model because it is impossible to determine security characteristics of software systems using exact numbers. As Hoglund and McGraw [15] note, in software risk analysis exact numbers as parameters work worse than having values such as high, medium and low. These kinds of values can be termed as fuzzy.

Risk analysis techniques for estimating the security of systems have been proposed earlier [1]. The differences in our approach are that we apply risk analysis already at the design phase of a software system using a security pattern centric approach, that we make use of the newer STRIDE model of attacks [16] and that we use fuzzy terms.

When performing risk analysis for a system, a common formula used by the risk engineering community is the following [8]:

$$R = LEC\ . \tag{1}$$

where L is the likelihood of occurrence of a risky event, E the exposure of the system to the event, C the consequence of the event and R the computed risk. Examining this equation in comparison to the risk analysis performed by Hoglund and McGraw [16] in our case the likelihood L is the likelihood of a successful attack, the exposure E is a measure of how easy is to carry out the attack and C is the impact of the attack. As we explained earlier we have chosen that the terms in our risk analysis model are fuzzy.

Table 2. Mapping of linguistic terms to generalized fuzzy numbers

Linguistic Term	Generalized Fuzzy Number
absolutely-low	(0.0, 0.0, 0.0, 0.0; 1.0)
very-low	(0.0, 0.0, 0.02, 0.07; 1.0)
low	(0.04, 0.1, 0.18, 0.23; 1.0)
fairly-low	(0.17, 0.22, 0.36, 0.42; 1.0)
medium	(0.32, 0.41, 0.58, 0.65; 1.0)
fairly-high	(0.58, 0.63, 0.80, 0.86; 1.0)
high	(0.72, 0.78, 0.92, 0.97; 1.0)
very-high	(0.93, 0.98, 1.0, 1.0; 1.0)
absolutely-high	(1.0, 1.0, 1.0, 1.0; 1.0)

The applicability of fuzzy techniques to security problems has already been proposed [8] and the use of fault trees for security system design has also been suggested [6, 1]. In this paper we perform an analysis of the security of systems using security patterns, using results from fuzzy set theory [38] and fuzzy fault trees [7]. Specifically, we perform fuzzy risk analysis for a system that has properly added security patterns to the initial system under examination.

Our analysis uses generalized fuzzy numbers [9] and the similarity metric proposed by Chen and Chen [9]. We have chosen generalized fuzzy numbers instead of other existing approaches because the similarity measure for generalized fuzzy numbers has been proven to be robust in the cases where both crisp and fuzzy numbers are to be compared [9].

We used the mapping from linguistic terms to generalized fuzzy numbers shown in Table 2 adapted from [9]:

These fuzzy numbers (except absolutely-low and absolutely-high) are shown in Figure 2.

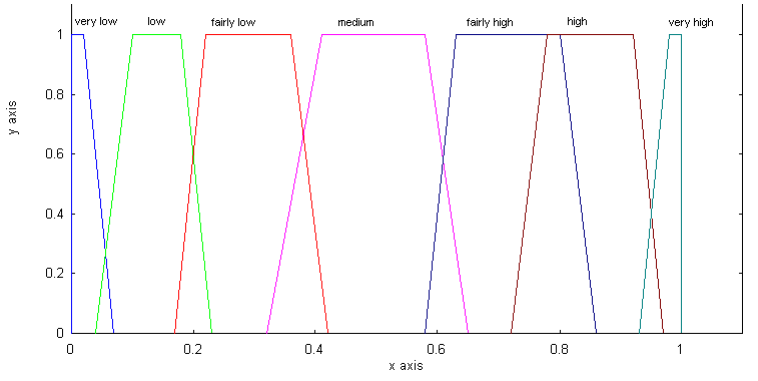

Fig. 2. Fuzzy numbers that correspond to the linguistic terms used in our analysis

Table 3. Analysis of primary attack events for a system properly using security patterns

Primary Event	Likelihood of occurrence	Exposure	Consequences	Categories of Attacks
Event 1. Dictionary attack to a guard of Secure Proxy is successful	low	high	very high	S, E, I
Event 2. Variable value is used unencrypted in SSL request	high	very high	low	I
Event 3. Variable value is read from wsdl file	medium	very high	low	I
Event 4. Input validation is bypassed	absolutely low	high	very high	T, I
Event 5. Unauthorized access to servlet member variables is allowed by exploiting race conditions	high	low	low	I

We then identified the primary events for the fault trees and the categories of attacks related to the STRIDE model [16] they belong to. A dictionary attack to the Secure Proxy pattern is successful only if both guards are compromised and causes a Spoofing Identity, Elevation of Privilege and Information Disclosure. An attack to a guard of this pattern can be performed using automated tools and therefore the exposure for this attack is high. The likelihood of such attack is low since the guard has high resistance to dictionary attacks. If such an attack is successful the consequences are very high. By performing a similar likelihood-exposure-consequence analysis for all primary events we obtain Table 3.

The Tampering with data attack does not exist practically for this system, since the only primary event that causes it has absolutely low likelihood of occurrence. The Spoofing Identity and Elevation of Privilege attacks occur for the same primary event.

The resulting fuzzy fault tree for Information Disclosure attacks is shown in Figure 3. The fault tree for Spoofing Identity and Elevation of Privilege attacks can be built using the same technique.

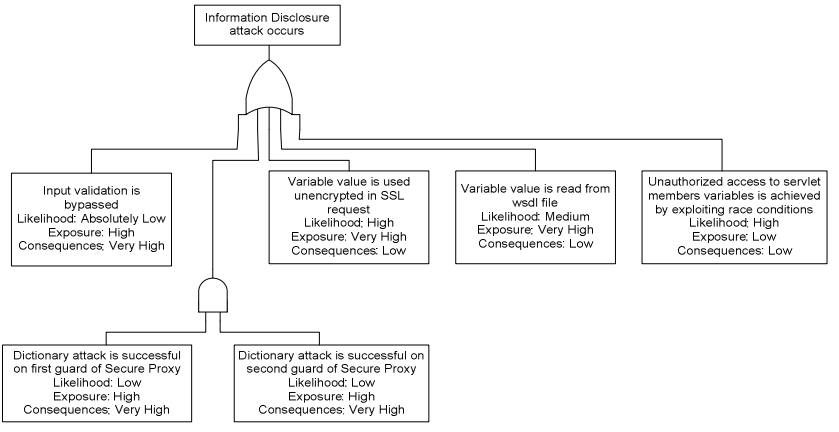

Fig. 3. Fault tree for Information Disclosure attacks

The methodology we use to derive the risk for the top event is outlined in the following steps:

1) We first identify the values of likelihood, exposure and consequences for the primary events.
2) We then perform the logical composition of values, according to rules for the gates of fault trees, starting from the values of primary events and ending at the computation of the risk for top event.
3) Finally we compare the risk for the top event computed in step 2, with the values in Table 2 using the similarity metric from [9].
4) The linguistic term with the highest similarity is chosen as the result.

This is a typical fuzzy risk analysis approach [7] where the terms for the events depend though on the security patterns used in the system examined. Furthermore, we use in our analysis generalized fuzzy numbers adapted from [9] as well as the similarity metric from [9].

Table 4. Summary of risks computed for different types of attacks for a system without security patterns and a system properly using them

	Spoofing	**Tampering with Data**	**Information Disclosure**	**Elevation of Privilege**
System without security patterns	fairly high	fairly high	high	fairly high
System properly using security patterns	very low	absolutely low	low	very low

After performing the necessary computations for the system properly using security patterns we come to the result that the risk for the fault tree corresponding to Spoofing and Elevation of Privilege attacks is *very low* and the risk for the fault tree corresponding to Information Disclosure attacks is *low*. These trees correspond to the system that properly uses security patterns. The risk for Tampering with data attacks is zero (*absolutely low*).

On the contrary, for the system that does not employ security patterns, the risk values according to the proposed model, for the same types of attacks are *fairly high* for Spoofing Identity and Elevation of Privilege attacks, *high* for Information Disclosure attacks and *fairly high* for Tampering with data attacks. Table 4 summarizes these results and quantifies the difference between the two systems.

The methodology described thus allows us to derive results about the *total* security of systems employing security patterns, already at the design, in terms of fuzzy linguistic variables.

7 Conclusions and Future Work

The results of the evaluation of the attacks as well as the fuzzy methodology used show that systems that use security patterns properly are highly secure and robust to attacks. This robustness to attacks has been also quantified in this work and a mathematical model has been proposed. Future work includes the introduction of new security patterns that solve the issues not addressed by existing ones and a software tool that automates the security evaluation process we described in this paper.

Acknowledgements

We would like to thank the Web Application Security mailing list of SecurityFocus and the comp.lang.java.security mailing list, for letting us organize a contest. Furthermore, we would like to thank Benjamin Livshits, from Stanford University, the winner of the contest and Watchfire Corporation for providing us an evaluation license for AppScan.

References

1. Amoroso, E., Fundamentals of Computer Security Technology, Prentice Hall (1994)
2. Anley, C., Advanced SQL Injection in SQL Server Applications, NGSSoftware whitepaper (2002)
3. Berry, C. A., Carnell, J., Juric, M.B., Kunnumpurath, M. M., Nashi, N. and Romanosky, S., J2EE Design Patterns Applied, Wrox Press (2002)
4. Blakley, B., Heath, C. and Members of the Open Group Security Forum, Security Design Patterns, Open Group Technical Guide (2004)
5. 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)
6. Brooke, P. J., and Paige, R. F., Fault Trees for Security System Design and Analysis, Computers and Security, vol. 22, No. 3, pp. 256-264 (2003)

7. Cai, K.-Y., Introduction to Fuzzy Reliability, Kluwer Academic Publishers (1996)
8. Cai, K.-Y., System Failure Engineering and Fuzzy Methodology, An Introductory Overview, Fuzzy Sets and Systems, Vol. 83 pp. 113-133 (1996)
9. Chen, S.-J., and Chen, S.-M., Fuzzy Risk Analysis Based on Similarity Measures of Generalized Fuzzy Numbers, IEEE Transactions on Fuzzy Sets and Systems, Vol. 11, No. 1 (2003)
10. Cgisecurity.com, Cross Site Scripting questions and answers, http://www.cgisecurity.com/articles/xss-faq.shtml
11. Fernandez E., Metadata and authorization patterns, http://www.cse.fau.edu/~ed/MetadataPatterns.pdf (2000)
12. Friedl, S., SQL Injection Attacks by Example, http://www.unixwiz.net/techtips/sql-injection.html
13. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley (1995)
14. Halkidis, S. T., Chatzigeorgiou, A., and Stephanides, G., A Qualitative Evaluation of Security Patterns, in Proceedings of the 6th International Conference on Information and Communications Security (ICICS '04) (2004)
15. Hoglund, G. and McGraw, G., Exploiting Software, How to Break Code, Addison Wesley (2004)
16. Howard, M. and LeBlanc, D., Writing Secure Code, Microsoft Press (2002)
17. Hu, D., Preventing Cross-Site Scripting Vulnerability, SANS Institute whitepaper (2004).
18. Kienzle, D., and Elder, M., Security Patterns for Web Application Development, Univ. of Virginia Technical Report (2002)
19. Klein, A., "Divide and Conquer", HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper (2004)
20. Lee Brown, F., Di Vietri J., Diaz de Villegas G., Fernandez, E., The Authenticator Pattern, in Proceedings of the 6th Conference on Pattern Languages of Programming (PLoP '99) (1999)
21. Livshits B., and Lam, M. S., In Proceedings of the 14th USENIX Security Symposium (2005)
22. Livshits, B., and Lam, M. S., Finding Security Vulnerabilities in Java Applications with Static Analysis, Stanford University Technical Report (2005)
23. 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)
24. 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)
25. Pullum, L. L., Software Fault Tolerance Techniques and Implementation, Artech House Publishers (2001)
26. Roman, E., Sriganesh, R. P. and Brose G., Mastering Enterprise JavaBeans, Third Edition, Wiley Publishing (2005)
27. Romanosky, S., Enterprise Security Patterns, http://www.romanosky.net/papers/EnterpriseSecurityPatterns.pdf (2002)
28. Ross, B., Jackson, C., Miyake, N., Boneh, D., and Mitchell, J.C., Stronger Password Authentication Using Browser Extensions, In Proceedings of the 14th USENIX Security Symposium (2005)
29. Scambray, J., and Shema, M., Hacking Exposed Web Applications, McGraw-Hill (2002)
30. Spett, K., Cross-Site Scripting, Are your web applications vulnerable?, SPI Labs whitepaper

31. SPI Labs, SQL Injection, Are Your Web Applications Vulnerable?, SPI Labs whitepaper.
32. Spinnelis, D., Code Quality : The Open Source Perspective, Addison Wesley (2006)
33. Steel, C., Nagappan R., and Lai, R., Core Security Patterns, Best Practices and Strategies for J2EE, Web Services, and Identity Management, Prentice Hall (2006)
34. Viega, J., and McGraw, G., Building Secure Software, How to Avoid Security Problems the Right Way, Addison Wesley (2002)
35. 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)
36. Weiss, M., Patterns for Web Applications, in Proceedings of the 10th Conference on Pattern Languages of Programming (PLoP '03) (2003)
37. Wu, T., A Real-World Analysis of Kerberos Password Security, In Proceedings of the 1999 Network and Distributed System Symposium (1999)
38. Zimmerman, H.-J., Fuzzy Set Theory and its Applications, Third Edition, Kluwer Academic Publishers (1996)