A Practical Evaluation of Security Patterns

Spyros T. HALKIDIS, Alexander CHATZIGEORGIOU, George STEPHANIDES

Department of Applied Informatics
University of Macedonia, Thessaloniki, Greece
halkidis@java.uom.gr, {achat,steph}@uom.gr

Abstract. Software security has attracted the attention of researchers in the area of security during the last years due to the proven fact that most attacks to businesses and organizations exploit software vulnerabilities. Moreover, the need to impose some level of security already at the design phase has been recognized. Therefore, software design patterns with the target of enhancing the security of software systems, already at design, have been proposed. These patterns are called security patterns. In this paper we evaluate common security patterns with respect to the STRIDE model of attacks by examining the attacks performed to two different systems: one without security patterns and one properly using them.

Keywords: Software Security, Security Patterns
Math. Subjects Classification 2000: 94A60, 14G50, 68Q99

1 INTRODUCTION

The high importance of software security to the process of ensuring some level of security to real systems has been evident since it has been discovered that most attacks to the systems exploit software vulnerabilities [22, 8, 7]. These vulnerabilities stem from software poorly designed and developed. Furthermore, it has been shown that the earlier we incorporate security into a system the better [22]. Therefore, the incorporation of a level of security already at the design phase is desirable. To achieve this, in analogy to design patterns [6] in software engineering, that aim to make software well structured and reusable, security patterns [20, 2] have been proposed. Security patterns are reusable solutions to common security problems that aim at imposing some level of security to software systems, already at the design phase.

In this paper, we try to practically examine the resistance to STRIDE [8] attacks of a small subset of security patterns that are commonly used in web applications. To perform this evaluation, we have built two systems one without security patterns and one using them, studied these systems under known categories of attacks to web applications [17] and determined which aspects of security are enhanced through the use of each security pattern used in the second system. To study these systems under known attacks, we have used two different approaches. First of all, we used the AppScan web application penetration testing tool [23] and secondly we have organized a contest to study other approaches that aim to evaluate software systems in terms of security vulnerabilities. Finally, based on our findings we determined to what extent each security pattern protects us from each category of STRIDE [8] attacks. Based on the fact that we can not quantify in strict terms the security of a system [7] levels of security ranging from absolutely low to absolutely high have been used in the analysis instead of exact numbers. The experimental evaluation shows that each security pattern protects us from different categories of attacks and therefore a smart combination of these security patterns, based on the resistance of each pattern to each category of attack, can lead to systems that are secure enough already from their design.

2 DESCRIPTION OF THE SYSTEMS UNDER EXAMINATION

In order to practically examine the robustness of various security patterns to known attacks we have developed two systems. The first system, hereafter denoted as "first" application, is a typical e-commerce application with no usage of security patterns where various sources for attacks were deliberately included. The second system, hereafter denoted as "second" application is a variant of the first application, where the sources for attacks where not removed, but security patterns were used to protect against the attacks.

Both applications under examination are typical J2EE (Java 2 Enterprise Edition, now referred to as Java EE) [15]. We have chosen J2EE as a platform for both applications since J2EE is widely used for business applications and is useful from the security point of view [20, 3].

The architecture of a typical J2EE system is shown in Figure 1.

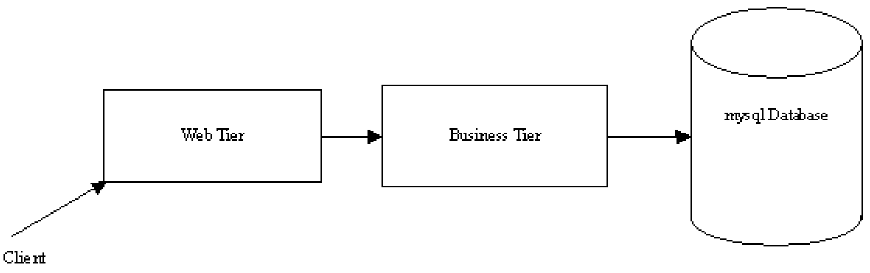

Fig. 1. A typical J2EE Architecture

The client, typically a web browser, accesses the Web Tier where servlets reside. Servlets can forward requests to Enterprise Java Beans (EJBs), that when needed access the database. We have used JBoss 4.0.3 [10] as an application server, for the web and business tier and MySQL 5.0 [14] for the database tier.

The first system consists of 46 class. It has 16 servlets and 7 EJBs.

We have deliberately included in this system several sources for attacks.

First of all three sources for SQL injection were included in this application [17, 1, 19, 5]. An SQL injection attack occurs when an attacker is able to insert a series of SQL statements in a query formed in an application, by exploiting non existence or of validation of data or improper validation of data [1]. An SQL injection attack can cause unauthorized viewing of database data and database modification.

Additionally, eleven sources for cross-site scripting were included. A cross-site scripting attack [17, 4, 18, 9], also known as an XSS attack, occurs when not properly validated data input in one page are shown in another. In this case, script code can be input in the former page to be consequently executed in the latter. In this way it is easy to perform and Information Disclosure attack [8] for example by inserting javascript code in the former page, that shows cookie values containing sensitive information, such as credit card numbers in the latter.

Furthermore, a source for HTTP Response Splitting [11] was included. HTTP Response Splitting attacks occur when user data that were not properly validated are included in the redirection URL of a redirection response, or when improperly validated data are included in the cookie value or name, when the response sets a cookie. In both cases it is easy to create two responses instead of one by manipulating headers. In the second response, an XSS attack can be performed [11].

Furthermore, there was no SSL connection used in the first application, having as result that the credentials and a cookie value containing credit card information could be eavesdropped.

Finally, six servlet member variables race conditions were included, which could be exploited by having a number of users acting simultaneously. However, the information contained in these variables was not sensitive and therefore the merits of such an attack would not be high. A summary of the security vulnerabilities present in the first application is shown in Figure 1.

Type of vulnerability	**Number of sources for attack**
SQL Injection	3
Cross-Site Scripting	11
HTTP Response Splitting	1
Servlet member variable race conditions	6
Eavesdropping	3

Table 1. Summary of Vulnerabilities present in the first application.

In the second application, the sources for attacks were not removed, but security patterns were added to the application instead. The second application consists of 62 classes. It has 17 Servlets and 9 EJBs. The security patterns that were added in this system are one instance of the Secure Proxy pattern, Login Tunnel variant [2], one instance of the Secure Pipe pattern, seventeen instances of the Secure Logger pattern, Secure Log Store Strategy, a twenty one instances of the Intercepting Validator pattern, and nine instances of the Container Managed Security pattern [20]. In Figure 2. we show a block diagram of the main components of the system, the security patterns used and the points where the user can login or inputs data, since these are the most crucial points in terms of security. The diagram does not show the Secure Pipe pattern, because it encompasses the whole application, since the whole application uses SSL. Additionally, it does not show the Secure Logger pattern because all servlets use it, nor the Container Managed Security Pattern since all EJBs make use of it.

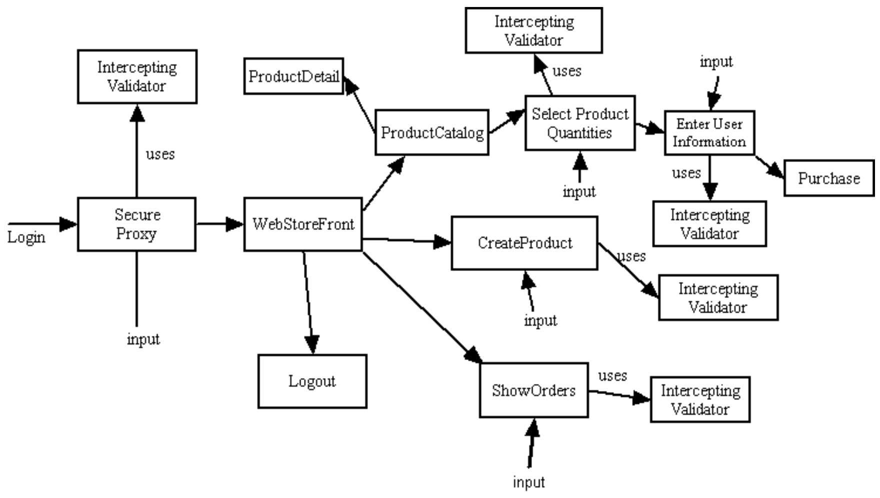

Fig. 2. Block diagram of the second application under examination

3 EVALUATION OF THE SYSTEMS AND THE SECURITY PATTERNS USED WITH REGARD TO ATTACKS

In order to evaluate the two systems under known attacks we have used different approaches. First of all we have used Watchfire's AppScan [23] web application penetration testing tool. Secondly, we have initiated a contest in various newsgroups, where the participants performed attacks to the systems we described earlier. This contest was was by Benjamin Livshits from Stanford University, who used static analysis tools described in several papers [12, 13] to find the security flaws existing in the applications.

Both approaches found the major security flaws of the first application, meaning the three SQL Injection and the eleven Cross-Site Scripting vulnerabilities. However, both approaches had several false positives. AppScan for instance 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 by patterns in the Web Tier. Race conditions for servlet member variables were found only by the static analysis approach. Minor application errors that pose no threat to security not found by the static approach, were found by AppScan (like lack of checking for proper session variable value ranges). 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 in our case are of minor importance, because the unencrypted parameters in the URLs are not crucial. The HTTP reponse splitting source in the first application was found by neither of the approaches.

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

The previous analysis of the results shows that proper use of the security patterns leads to the remediation of all major security flaws. The flaws that are not confronted are those of minor risk, like unencrypted SSL parameters (this flaw is of minor risk only when the unencrypted parameters are not crucial like in our case), and servlet member variable race conditions (this flaw is of minor risk when the variables for which the race conditions occur are not crucial). These flaws that remain even after the use of security patterns, exist because existing security patterns do not confront these kind of problems.

The Intercepting Validator pattern, when used for all input, including session variables that are not input by the user but still posted, protects from SQL Injection, Cross-Site Scripting and HTTP Response Splitting attacks. Therefore, it offers high protection against Tampering with Data and Information Disclosure attacks [8].

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 equivalent to the existence of two guards [2] connected in a series. The resistance of this pattern to attacks is dependent to the resistance of each guard to dictionary attacks. Specifically, in order for both guards to be compromised, two consecutive dictionary attacks to the authentication mechanism of a guard must succeed. Recent studies [24, 16], 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 a guard can still be marked as of high security.

All authentication patterns and consequently the Protected System [2] and the Secure Proxy pattern should be resistant to eavesdropping attacks to serve their purpose. Thus, they should always be used together with the Secure Pipe pattern that provides an implementation of the SSL protocol [20]. 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 of minor 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 EJB protected can do so [20].

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

Based on the above analysis that offers us a practical a practical examination of attacks to systems without and with security patterns we can make conclusions about the resistance to known categories of attacks [8] of the security patterns under consideration. The results are summarized in Table 1. Irrelevant entries to the specific security pattern are left blank.

	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				

Table 2. Resistance of the security patterns examined against STRIDE attacks.

4 CONCLUSIONS AND FUTURE WORK

In this paper we have estimated the resistance of specific security patterns against STRIDE [8] attacks. In order to achieve this we have built two systems, one without these security patterns and one using them and have studied these systems under two methodologies of evaluation: Web Application Penetration Testing and Static Analysis of code. Based on the results of the evaluation estimates of the resistance of each pattern to each category of STRIDE attacks.

Future work includes proposing new security patterns for the security flaws that our analysis showed that existing security patterns do not confront, building a mathematical model for the security of systems using security patterns, and development of a tool that based on the security patterns present in the design of a system can make rough estimates about the security of the system already from its design.

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

References

[1] **C. Anley**, Advanced SQL Injection in SQL Server Applications, NGSSoftware whitepaper, 2002
[2] **B. Blakley, C. Heath and Members of the Open Group Security Forum**, Security Design Patterns, Open Group Technical Guide, 2004
[3] **C.A. Berry, J. Carnell, M.B. Juric, M. M. Kunnumpurath, N. Nashi and S. Romanosky**, J2EE Design Patterns Applied, Wrox Press, 2002
[4] **Cgisecurity.com**, Cross Site Scripting Questions and Answers, http://www.cgisecurity.com/articles/xss-faq.html
[5] **S. Friedl**, SQL Injection attacks by example, http://www.unixwiz.net/techtips/sql-injection.html
[6] **E. Gamma, R. Helm, R. Johnson and J. Vlissides**, Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley, 1995
[7] **G. Hoglund and G. McGraw**, Exploiting Software, How to Break Code, Addison Wesley, 2004
[8] **M. Howard and D. LeBlanc**, Writing Secure Code, Microsoft Press, 2002
[9] **D. Hu**, Preventing Cross Site Scripting Vulnerability, SANS Institute whitepaper, 2004
[10] **JBoss Home Page**, http://www.jboss.com
[11] **A. Klein**, Divide and Conquer, HTTP Response Splitting, Web Cache Poisoning Attacks and Related Topics, Sanctum whitepaper, 2004
[12] **B. Livshits and M.S. Lam**, Finding Security Vulnerabilities in Java Applications with Static Analysis, In Proceedings of the 14^{th} USENIX Security Symposium, 2005
[13] **B. Livshits and M.S. Lam**, Finding Security Vulnerabilities in Java Applications with Static Analysis, Stanford University Technical Report, 2005
[14] **MySQL Home Page**, http://www.mysql.com
[15] **E. Roman, R.P. Sriganesh and G. Brose**, Mastering Enterprise JavaBeans, Third Edition, Wiley Publishing, 2005
[16] **B. Ross, C. Jackson, N. Miyake, D. Boneh and J.C Mitchell**, Stronger Password Authentication Using Browser Extenstions, In *Proceedings of the* 14^{th} *USENIX Security Symposium*, 2005
[17] **J. Scambray and M. Shema**, Hacking Exposed Web Applications, McGraw-Hill, 2002
[18] **K. Spett**, Cross Site Scripting, Are your Web Applications Vulnerable, SPI Labs whitepaper

[19] **SPI Labs**, SQL Injection, Are Your Web Applications Vulnerable?, SPI Labs whitepaper
[20] **C. Steel, R. Nagappan and R. Lai**, Core Security Patterns, Best Practices and Strategies for J2EE, Web Services and Identity Management, Prentice Hall, 2006
[21] **D. Spinellis**, Code Quality: The Open Source Perspective, Addison Wesley, 2006
[22] **J. Viega and G. McGraw**, Building Secure Software, How to Avoid Security Problems the Right Way, Addison Wesley, 2002
[23] **Watchfire Corporation**, http://www.watchfire.com
[24] **T. Wu**, A Real World Analysis of Kerberos Password Security, In *Proceedings of the 1999 Network and Distributed System Symposium*, 1999