Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 998–1003

Procedia Computer Science

www.elsevier.com/locate/procedia

International Symposium on Machine Learning and Big Data Analytics for Cybersecurity and Privacy (MLBDACP)
April 29 – May 2, 2019, Leuven, Belgium

Security Qualitative Metrics for Open Web Application Security Project Compliance

Ferda Özdemir Sönmez[a*]

[a] *Informatics Institute Middle East Technical University, Üniversiteler Mahallesi, Dumlupınar Bulvarı No:1 06800 Çankaya, Ankara, Turkey*

Abstract

The focus of this study is to find out repeatable features for large-scale enterprise web application production process related to based on OWASP security requirement list. As a result of a rigorous work including domain analysis for Java language and development frameworks and the examination of a large set of technical documents, 230 security qualitative metrics are discovered, under six categories. These security qualitative metrics are beneficial for security analysts as well as other parties such as designers, developers, and testers. The findings provide a developer/designer point of view and would help to make better decisions related to the environment set up, technology selection, and the architecture, design, and implementation details. As a result of this effort, the overall vulnerability level of the web applications would diminish significantly.

Keywords: OWASP; Enterprise Security; Security Qualitative Metric; Application Security; Web; Java

1. Introduction

Web applications have been the mainstream software choices for the enterprises. Enterprise web applications are highly integrated and with other systems. This makes them more sophisticated, more complex and increases the security requirements for these applications. Combination of the static web, dynamic web, mobile web, and web 2.0 provides an integrated communication environment for business and non-business users. There are many technology alternatives for web development. Using development frameworks and best practices in the overall architecture

* *Corresponding author. Tel.: +90-532-694-3503 ; fax: +90-532-694-3503 .*
E-mail address: ferdaozdemir@gmail.com

results with many advantages concerning information security such as using newest security implementations, easy integration with third parties, advanced communication standards, and central management of security. Combining the best practices with programmatically generated, standardized code increases these advantages and results with more robust and reliable applications. These applications also happen to be more secure. So far, various tools, architectures, run time environments, and many development platforms have emerged for the development of static and dynamic web applications. One of the most popular contemporary approaches is using Java programming language and popular Java development frameworks, namely Spring [1] and Hibernate [2] together with relational databases.

Qualitative metrics involve assessment through non-numerical reporting about a question or inquiry. In this work, the security-related features which are named "security qualitative metrics" by the author, related to large-scale enterprise web-based development are identified. This identification depends on domain analysis made by the examination of a real-world sample large-scale enterprise web-based Java application which relies on the selected architecture, strongly dependent on the use of Java frameworks, Spring, and Hibernate, and the examination of technical documents. The broadness of the security subjects commonly results with the examination of technology groups or different architectures independently. For example, Manadhata et al. [3] proposed a study measuring attack surfaces for SAP applications. Same authors suggested another study to measure enterprise software vulnerabilities [4]. Alshammari et al. [5] proposed security metrics for only object-oriented designs.

During the extraction of security qualitative metrics, a detailed evaluation of the sample large-scale application for its compliance to Open Web Application Security Project (OWASP) [6] is made. The results of this evaluation are combined with the features and scholarly suggestions extracted from technical documents, including OWASP cheat sheets [7] related to selected technologies, and the specification files related to the Java programming language [8] and selected development frameworks.

The OWASP Application Verification Standard Project (ASVS) [9] is taken as a basis during the definition of security qualitative metrics. OWASP ASVS is commonly used for web application security evaluation and the security requirements determination. It's broad coverage of security issues and its more frequent use were the reasons for selecting this standard over others. Goswami et al. [10] claimed that if the web project is compliant to OWASP, its attack surface would decrease significantly (by % 45). As a part of their work, they presented a set of security-related parameters. The detailed security qualitative metric list provided as a part of the current study is much more comprehensive compared to this list. There are also non-OWASP based web application security vulnerability expectations measurement systems. There are various attack surface calculation definitions in the literature. Theisen et al. [11] made a systematic survey for attack surface definitions for the web applications. They resulted with six group of themes for definitions. Several examples are Heumann et al. [12] which used a limited set of features to measure and compare the attack surface of several applications and Manadhata and Wing [13] which defined attack surface based on application entry, exit points, and channels. Attack surface calculation is the most common way to calculate the vulnerability level of web applications. Continuous compliance check is an alternative way to check the vulnerability level. However, compliance checks are not straightforward.

OWASP is an online group which produces standards, articles, tools, forum information related to web application security. This information is used by architects , developers, analysts, and the researchers. It provides a long list of security requirements; however, it does not provide an easy way to associate design features, environment properties, or technologies to these long list of security requirements. Thus, all kind of decision-makers for the web application development process who are familiar with software development, methodologies, languages, technologies, architectures, and the frameworks have difficulty to understand the effects of their decisions related to the project's overall compliance to the OWASP standard. In the author's opinion, if these decision-makers are provided with this information, they may make better decisions regarding security. This idea was the primary motivation for this study.

The rest of this paper is structured as follows. In Section 2, there is background, environment description, and the methodology. Section 3 contains the results for the study. Section 4 and Section 5 are devoted to the discussion and the conclusion sections.

2. Background, Environment Description, and the Methodology

During this work security perspective for a large-scale web application development project is examined. Due to privacy reasons no information that may cause conjecture of the project name, and purposes will be provided. Thus, business requirements and implementation details which are not necessary to mention for the purposes of this paper are out of the scope of this study. However, to describe the security-related decisions and their benefits for the compliance to OWASP standard an overview of the technical infrastructure of the project is provided in this section.

This sample enterprise application is developed by using the integration of contemporary Java technologies forming an advanced structure which facilitate rapid application development and also provide a robust framework to develop the business requirements. On top of this framework, a code generation application which has been modified based on the project's structure and technologies are utilized during the development. This code generation mechanism further speeded up the development project and provided fully standardized, clean code from front end to back end. Figure 1-a shows a simplified infrastructure schema and names of the technologies. Featuring properties are the encapsulation of the most contemporary technologies, standard libraries, use of a layered structure, centralization of controls, utility classes and a standardized code structure.

Making an OWASP based evaluation for the project is a contract requirement for this project. Thus, both manual inspection and tools based security vulnerability analysis are utilized in different phases of the project. Figure 1-b points out the security evaluation related tasks in the overall timeline of the project. The thick red line shows the duration when the manual code analysis for each OWASP item took place, and the green star shows the duration where the OWASP based automated security analysis tools are utilized.

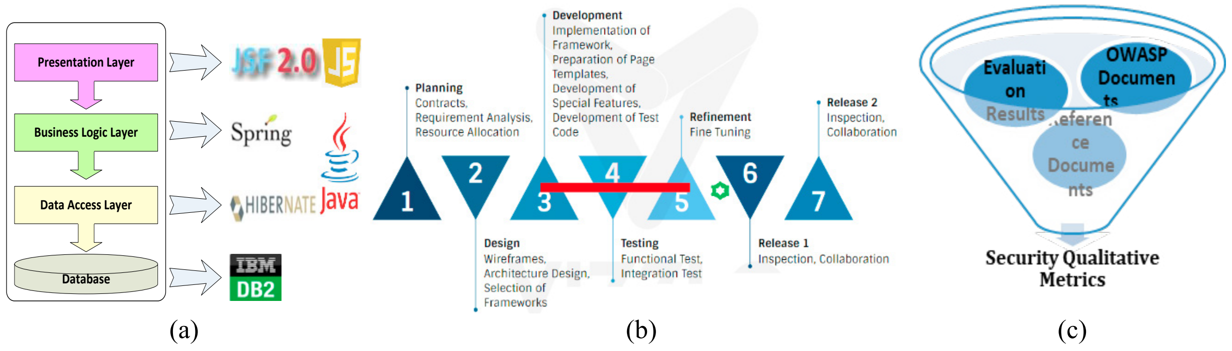
Presentation Layer
Business Logic Layer
Data Access Layer
Database
JSF 2.0
JS
Spring
HIBERNATE
Java
IBM DB2
(a)
Planning
Contracts, Requirement Analysis, Resource Allocation
Design
Wireframes, Architecture Design, Selection of Frameworks
Development
Implementation of Framework, Preparation of Page Templates, Development of Special Features, Development of Test Code
Testing
Functional Test, Integration Test
Refinement
Fine Tuning
Release 1
Inspection, Collaboration
Release 2
Inspection, Collaboration
1
2
3
4
5
6
7
(b)
Evaluation Results
OWASP Documents
Reference Documents
Security Qualitative Metrics
(c)

Fig. 1. (a) Technologies; (b) Timeline of security related tasks; (c) Definition of Security Qualitative Metrics

In order to conduct the study, a domain analysis is made. During the domain analysis, first, the code structure, including the code generator, developed part, centralized code, in-house, and standard libraries are examined. Once this analysis is finished and know-how gathered related to project structure, and implementation details, each OWASP requirement item is examined in-depth by using inspection and questioning and answering methods with the experts of the project including the senior developers, and the architects. At this step, the information provided by the OWASP web site is also frequently utilized. When shortcomings related to some OWASP requirements in the sample application are detected, suggestions from technical OWASP documents and other technical reference documents are made to fulfill all the OWASP requirement set. The proposed security qualitative metrics are formed by working on this consolidated information as shown in Figure 1-c.

When the evaluation results are gathered and examination of the technical documents finished, the overall results are prepared in report format. Following this, the resolutions from the evaluation results and other technical suggestions are grouped into categories for reuse. The textual descriptions for each finding are simplified and converted to qualitative metric phrases. At this step, some of the findings are subdivided to multiple expressions whenever necessary.

In the author's opinion, these classified findings would be beneficial from the initial phases of application development including analysis, design, implementation, and test. Having this categorized security qualitative metrics would allow knowing the security status and OWASP compliance results for the project, and would help to make better technology, environment, design, and implementation related decisions by various project stakeholders.

3. Results

The OWASP ASVS V.3. is a project which consists of a total 182 security requirements grouped under the following 19 topics: architecture design and threat modeling, authentication verification requirements, session management verification requirements, access control verification requirement malicious input handling verification requirements, output encoding/escaping, cryptography at rest, error handling and logging, data protection, communications, HTTP security configuration, security configuration verification requirements, malicious controls, internal security verification requirement, business logic, file and resources, mobile, web services, and configuration.

In this work, rather than going from the requirement to application features, a reverse approach is taken. Due to space limitations only a part of this list is provided in Table 1. The textual descriptions of the requirements can be found in the OWASP site. The requirement numbers associated with security qualitative metrics are in the form OWASP chapter number + "." +Requirement number.

The findings are grouped under six categories: A- Highlights of Application Architecture, Design, and Implementation, B- Highlights of Technologies, C- Highlights of Environment, D- Highlights of Code Generation, E- Highlights of Development Methodologies, and F- Highlights of Business Logic. The essence of each category is briefly described as follows: The first category corresponds to architecture, design, and implementation details. Technology-specific advantages and disadvantages are categorized in the second group. Environment properties including subjects related to enterprise infrastructure including security protection systems, network, system administration are grouped into the fourth category. Unique benefits of having a code generator providing a standardized code structure are arranged in the fifth category. Project independent (repeatable), business logic related findings are grouped under the six category.

Table 1-Security Qualitative Metrics

#	G	Description	OWASP	#	G	Description	OWASP	#	G	Description	OWASP
1	A	Use menu structure which suits business requirements groupings and hierarchies	1.4	78	A	Use HTTP header parameters and caching to prevent client side caching (Cache-Control, Pragma)	9.1, 9.4	155	B	Use of Java SecurityManager	13.1
2	A	Use of layered architecture	1.9	79	A	Make a list of sensitive data for application, e.g. monetary data	9.2	156	B	Use of mod-rewrite module in Apache Tomcat for custom rewriting rules	16.1
3	A	Using single-sign-on	2.1	80	A	Authorize a restricted group of users for application parts with sensitive data	9.2	157	B	Use of URLRewritingFilter to filter URL rewrites	16.1
19	A	Use of captcha to prevent brute force attacks	2.2	96	A	Define undesirable HTTP methods in web.xml	11.1	173	C	Disable Directory Browsing on web server	4.5
20	A	Use of centralized, encrypted authentication data	2.21	97	A	Define loggers in Controllers for undesirable HTTP methods	11.1	174	C	Do not store SVN/Git files on web server	4.5
34	A	Termination of session automatically after the timeout or after a period of inactivity	3.3, 3.4	111	A	Use of web.xml file to make filters for URL rewriting	16.1	188	C	Use antivirus protection in web server for external file input	16.3
45	A	Safe secondary access control for lower value or critical systems	4.15	122	A	Check incoming content type for Rest services	18.8	199	C	Give only DML (not DDL) access rights to app server database users	19.3
51	A	Use of resource files for all string variables	5.15	128	B	Use Spring Security for centralized security controls	1.7	205	C	Use Jar Signer as a part of automated deploy	19.7
52	A	Use of client-side validation together with server-side validation	5.18	129	B	Use of Spring MVC for the clear separation of data, view and controller components	1.9	206	C	Configure build automation tools to download third-party libraries from trusted servers	19.8

53	A	Use of Rest services having their own validations	5.19	130	B	Use Spring-MVC to help clearance of client code from secret codes and business logic.	1.1	207	D	Automatic code generation without business logic in client	1.1
54	A	Validate all types of inputs such as RSS, Cookie, and Batch files	5.19	131	B	Use bug-free technologies having reliable references	1.11	208	D	Use of standardized code to eliminate the risk of RFI	5.13
55	A	Use of servlet filters to clean unstructured data	5.21	132	B	Be skeptical to newly emerged technologies, libraries, third party tools etc.	1.11	209	E	Have a standardized front end to minimize the # of hidden fields, cookies, and ajax	9.7
56	A	Careful use of HTML editors and avoid unnecessary usage	5.22	133	B	Use Spring anti-CSRF tokens to protect from CSRF attacks	4.13	210	E	Use of parentheses in automatic code generation	13.1
58	A	Use of safe logout functions	5.26	135	B	Avoid using Java Native lib	5.1	212	E	Documentation of high-level architecture	1.3
59	A	Delete login information during logout	5.26	136	B	Use JSF or similar framework for central validation	5.6	213	E	Prepare application specific threat model including well-known threats	1.6
60	A	Secure cryptographic modules which fail safely	7.2	137	B	Use ORM (e.g. Hibernate) to prevent SQL injection	5.1	214	E	Manual inspect code before production to clean sensitive information log	8.7
61	A	The integrity of the ciphertext should be checked based on	7.2	138	B	Use of Spring validator	5.16	215	E	Decide data to be destroyed as a part of a retention policy	9.6
64	A	Use keyless hash functions	7.9, 7.11	141	B	Use of Java Regex for email, telephone or similar fields	5.2	218	E	Submit the URL to Strict Transport Security domains list maintained for each browser	10.2
65	A	Using physical keys	7.11	142	B	Use of InnerText instead of innerHTML in Javascript	5.24	219	E	Verify that OCSP Stapling is enabled through Digicert	10.15
67	A	Replace critical data with 00 when soft deleted	7.13	144	B	Use java.security.SecureRandom instead of java.util.random	7.6, 7.15	221	F	Check transactions related to monetary values	15.1
68	A	All keys and passwords should be replaced at installation	7.14	145	B	Use SHA-256/SHA-512, PBKDF2-HMAC rather Bcrypt against FIPS-140-2 or equivalent	7.7	222	F	Check transactions with date entries	15.1
69	A	Do not log sensitive data (session identifier, password, hash, or API token)	8.7	146	B	Error messages should not include sensitive data	8.1	223	F	Check transactions for sequential events	15.1
70	A	Prevent log injection by log designs with character encoding for unprintable characters and fixed file format	8.8	147	B	Central error handling mechanism/code	8.2	224	F	Check transactions related to external users	15.1
73	A	Filter external originated log entries before automatic processing of log files	8.9	150	B	Logs should include necessary data to form a timeline of events	8.4	227	F	Disruption of business should result with transaction interruption.	15.2
76	A	Keep log files on a different disk partitions with application files	8.12	153	B	Use of TLS for transfers with sensitive data	10.3	230	F	Use RBAC restrictions for access file and other important assets from mobile devices	17.2
77	A	Use of common clock among multiple web servers	8.13	154	B	Use of the latest TLS version compatible with the web server	10.16				

4. Discussion

The primary question to discuss is how to use this comprehensive information. The author recommends that the users of this list should first decide which parts of OWASP requirements apply to their specific situation. For example, if they do not develop mobile application, they can eliminate that part of the OWASP requirement set. If they do not have any cryptologic implementation, they may skip that section. Later, they should check how much their technologies overlap the suggested arrangement. If they have exactly the same technologies, they may benefit the provided findings the most. However, if some parts overlap, they may eliminate some of these proposed elements and continue with others. The resulting list can be used during decision making or during the evaluation of application security and the detection of the vulnerabilities. While this list can be taken as a group of directives which will guide decision makers, the percentage of obeying them will provide information about the security level and percentage of vulnerabilities covered regardless of application size or the number of application assets. This value can be calculated for different groups such as the percentage of obeying environment suggestions, the percentage of following implementation, design type of recommendations.

The metric descriptions are in general prepared in a directive or noun form. While obeying the directive or having the proposed feature would directly correspond to increased security, it will also expose strong or weak points of the application based on OWASP security requirements.

The provided list has three types of items. Some of the elements may cause direct elimination of risk for an OWASP requirement item, some others help reduce of risk, and a few of them are firmly related to the OWASP items which should be examined in-depth. Thus, even having all the elements for a web application does not mean that all OWASP requirements for a Java web application are covered % 100. Similarly, some OWASP requirements may not correspond to the web application's structure. For example, for a web application which does not rely on password-based authentication, a password related security requirement would be unnecessary. Thus, in some conditions even not including some elements may result with a secure application.

5. Conclusion

In this study, a list of web application development related elements which will ease the OWASP compliance analysis checks is presented. This list will facilitate the manual OWASP compliance checks. In author's opinion, if the manual OWASP compliance checks become more straightforward, then these may become alternative to attack surface calculations to determine the vulnerability levels.

The main limitation of this study was targeting a specific arrangement of technologies. It is already mentioned that broadness of the security issues causes selection of an arrangement of technologies for the studies. However, although, it was explicitly defined that the proposed list mostly compatible with the specified arrangement, the author paid much attention to provide application and technology independent information as much as possible and vendor or product specific offers are given as samples.

As a future work, this long list of security qualitative metrics might be elicited or grouped in different ways. An alternative categorization of these metrics may target user roles, such project manager, architect, or developer. Another categorization alternative may point out tasks related to various phases of the web application life cycle, such as the selection of project planning, and the preparation of the software development environment.

References

[1] R. Johnson, J. Hoeller, A. Arendsen, T. Risberg and C. Sampaleanu, Professional Java development with the Spring framework, John Wiley & Sons, 2009.

[2] E. Pugh and J. D. Gradecki, Professional Hibernate, Indianapolis, USA: Wiley, 2004.

[3] P. K. Manadhata, Y. Karabulut and J. M. Wing, "Measuring the Attack Surfaces of SAP Business Applications," in International Symposium on Software Reliability Engineering, Seattle, USA, 2008.

[4] P. K. Manadhata, Y. Karabulut and J. M. Wing, "Report: Measuring the Attack Surfaces of Enterprise Software," in International Symposium on Engineering Secure Software and Systems, Leuven, Belgium, 2009.

[5] B. Alshammari, C. Fidge and D. Corney, "Security Metrics for Object-Oriented Designs," in Proceedings of the 21st Australian Software Engineering Conference, Auckland, New Zealand, 2010.

[6] OWASP, "The OWASP Foundation," 5 11 2018. [Online]. Available: https://www.owasp.org/index.php/Main_Page.

[7] M. Woschek, OWASP Cheat Sheets, OWASP, 2015.

[8] H. Schildt, Java: The Complete Reference, Ninth Edition, New York City, USA: Mc Graw-Hill Education, 2014.

[9] OWASP, "Category:OWASP Application Security Verification Standard Project," 2019. [Online]. Available: https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project. [Accessed 26 1 2019].

[10] S. Goswami, N. R. Krishnan, Mukesh, S. Swarnkar and P. Mahajan, "Reducing Attack Surface of a Web Application by Open Web Application Security Project Compliance," Defence Science Journal, vol. 62, no. 5, pp. 324-330, 2012.

[11] C. Theisen, N. Munaiah, M. Al-Zyoud and J. C. Carver, "Attack Surface Definitions: A Systematic Literature Review," Information and Software Technology, vol. 104, pp. 94-103, 2018.

[12] T. Heumann, J. Keller and S. Turpe, "Quantifying the Attack Surface of a Web Application," in Konferenzband der 5. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik, Berlin, Germany, 2010.

[13] P. Manadhata and J. M. Wing, "An Attack Surface Metric," IEEE Transactions on Software Engineering, vol. 3, pp. 371-386, 2010.