Securing Web Applications: Techniques and Challenges

M. Vieira

Abstract— **Software security is nowadays a hot research topic, particularly in the Web domain. In fact, due to the impressive growth of the Internet and of Web applications, software security has become one vital concern in any information infrastructure. This paper discusses key techniques for security testing and assessment, providing the basis for understanding existing research challenges on developing and deploying secure Web applications.**

Keywords— **Security, Web applications, Vulnerabilities, Benchmarking, Secure Processes.**

I. INTRODUCTION

THE GOAL of security is to protect systems and data from intrusion. The risk of intrusion is related to the system vulnerabilities and the potential security attacks. The **system vulnerabilities** are an internal factor related to the set of security mechanisms available (or not available) in the system, the correct configuration of those mechanisms, and the hidden flaws on the system implementation. Many types of vulnerabilities are known and also taxonomies to classify them [1]. Vulnerability prevention consists on guarantying that the software has the minimum vulnerabilities possible (e.g. using security testing). On the other hand, vulnerability removal is the process of mitigating the vulnerabilities found in the system (e.g. by applying new security patches released by software vendors).

Security attacks are an external factor that mainly depends on the intentionality and capability of humans to maliciously break into the system tacking advantage of vulnerabilities. In fact, the success of a security attack depends on the vulnerabilities of the system and attacks are harmless in a system without vulnerabilities. On the other hand, vulnerabilities are harmless if the system is not subject of security attacks. The prevention against security attacks includes all the measures needed to minimize or eliminate the potential attacks against the system (by reducing the potential attack surface). Attack removal is related to the adoption of measures to stop attacks that have occurred before (e.g. using intrusion detection).

Secure Software behaves correctly in the presence of a malicious utilization (attack), even though software failures may also happen when the software is used correctly [2]. Thus, many times software development and testing concerns only with what happens when software fails and not with the intentions. This is where the difference between software safety and software security lies: in the presence of an intelligent adversary with the intention of damaging the system.

M. Vieira, Universityof Coimbra (UC), Coimbra, Portugal, mvieira@dei.uc.pt

In the last two decades, the World Wide Web radically changed the way people communicate and do business. The problem is that, as the importance of the assets stored and managed by web applications increases, so does the natural interest of malicious minds in exploiting this new streak. In fact, web applications are so widely exposed that any existing security vulnerability will most probably be uncovered and exploited by hackers. Hence, the security of web applications is a major concern and is receiving more and more attention from the research community. However, in spite of this growing awareness of security aspects at web application level, there is an increase in the number of reported attacks that exploit web application vulnerabilities [3], [4].

To prevent vulnerabilities developers must apply best coding practices, perform security reviews, execute penetration testing, use code vulnerability detectors, etc. Still, many times developers focus on the implementation of functionalities and on satisfying the costumer's requirements and disregard security aspects. Also, most developers are not security specialists and the common time-to-market constraints limit an in-depth search for vulnerabilities. Another problem is that, traditional security mechanisms like network firewalls, intrusion detection systems (IDS), and encryption, are not able to mitigate web application attacks because they are performed through ports that are used for regular web traffic [5] and even application layer firewalls can not protect the applications as that requires a deep understanding of the business context [6]. In this scenario, a large effort should be put on improving the state of the art in the security of software systems.

This paper discusses key concepts, techniques and tools for testing and assessing security in the context of Web applications and services. First, we discuss techniques and tools for detecting vulnerabilities, which have the greatest importance to help developers producing more secure code. Second, we introduce the concept of Vulnerability and Attack Injection, whose goal is to provide the means to introduce realistic vulnerabilities in applications code. This is extremely useful in different contexts, including: 1) for training security teams; 2) to evaluate security teams in a controlled environment; and 3) to estimate the total number of vulnerabilities still present in the code. Then we discuss security evaluation from the benchmarking (i.e. comparison) point-of-view, which allows assessing and comparing the security of systems and/or components, supporting informed decisions while designing, developing, and deploying complex software systems. Finally, we put security in the context of the **development lifecycle**, emphasizing the key security aspects that should be kept in mind when developing Web applications.

II. SECURITY TESTING

To identify security issues, developers must focus not only on testing the functionalities of the application but also on searching for dangerous security vulnerabilities that are present in the code and that can be maliciously exploited [2]. In this scenario, automated tools have a very important role on helping the developers to produce less vulnerable code.

Different techniques for the detection of vulnerabilities have been proposed in the past [1], but in practice these techniques can be divided in two main groups: **white-box** analysis, which consists of examining the code of the application without executing it (this can be done in one of two ways: manually during code inspections and reviews or automatically by using automated analysis tools); and **black-box** testing, which refers to the analysis of the program execution from an external point-of-view (in short, it consists of exercising the software and comparing the execution outcome with the expected result). Black-box testing is probably the most used technique for verification and validation of software.

In the context of security, both black-box testing and white-box analysis have limitations that are intrinsic to their characteristics. Black-box testing is based on the effective execution of the code and in practice vulnerability identification is only based on the analysis of the web application output. This way, the effectiveness of the process is always limited by the lack of visibility on the internal behavior of the application. On the other hand, white-box approaches like static analysis are normally based on the examination of the source code. The main problem here is that exhaustive source code analysis may be difficult and cannot find many security flaws due to the complexity of the code and the lack of a dynamic (runtime) view. Of course, black-box testing does not require access to the source code while static analysis does.

The effectiveness of automated vulnerability detection tools is frequently very low, thus using the wrong tool may lead to the deployment of applications with undetected vulnerabilities. The work presented in [7] shows the main findings of a practical study that compares the effectiveness of very well known and largely used penetration testing and static analysis tools in the detection of SQL Injection vulnerabilities in Web Services. Results show that the coverage of static code analysis tools (including FindBugs, Fortify 360, and IntelliJ IDEA) is typically much higher than of penetration testing tools (including HP WebInspect, IBM Rational AppScan, and Acunetix Web Vulnerability). False positives are a problem for both approaches, but have more impact in the case of static analysis. A key observation is that different tools implementing the same approach frequently report different vulnerabilities in the same code.

The challenge is that, although we frequently trust vulnerability detection tools, results highlight their limitations suggesting that it is necessary to improve the state of the art in vulnerability detection, for instance by combining different approaches. Also, it is important to define mechanisms to evaluate and compare different tools in order to select the tools that best fit each development scenario.

III. VULNERABILITY AND ATTACK INJECTION

Fault injection has become an attractive approach to validate specific fault handling mechanisms and to assess the impact of faults in actual systems, allowing the estimation of fault-tolerant system measures such as fault coverage and error latency [1]. In the past decades, research on fault injection has specially targeted the emulation of hardware faults, where a large number of works has shown that it is possible to emulate these faults in a quite realist way (e.g. [8], [9]). More recently the interest on the injection of software faults has increased, giving raise to several works on the emulation of this type of faults (e.g., [10], [11]). In practice, software fault injection deliberately introduces faults into the system in a way that emulates real software faults. A reference technique is G-SWFI (Generic Software Fault Injection Technique [10]), which supports the injection of realistic software faults (i.e. faults most likely present in a software) using educated code mutation. The faults injected are described in a library derived from an extensive field study aimed at identifying the types of bugs that can reasonably be expected to occur frequently in a software system.

The use of fault injection techniques to assess security is a particular case of software fault injection, focused on the software faults that represent security vulnerabilities or may cause the system to fail in avoiding a security problem. Security vulnerabilities are in fact a particular case of software faults, which require adapted injection approaches.

In [12] the vulnerabilities of six web applications were analyzed using field data based on a set of 655 security fixes. Results show that only a small subset of 12 generic software faults is responsible for all the security problems. In fact, there are considerable differences by comparing the distribution of the fault types related to security with studies of common software faults.

Neves et al. proposed a tool (AJECT) focused on discovering vulnerabilities on network servers, specifically on IMAP servers [13]. In their work the fault space is the binomial (attack, vulnerability) creating an intrusion that may cause an error and, possibly, a failure of the target system. To attack the target system they used predefined test classes of attacks and some sort of fuzzing.

A procedure inspired on the fault injection technique (that has been used for decades in the dependability area) targeting security vulnerabilities is proposed in [14]. In this work, the "security vulnerability" plus the "attack" represent the space of the "faults" that can be injected in a web application; and the "intrusion" is the "error". To emulate with accuracy real world web vulnerabilities this work relies on the results obtained in a field study on real security vulnerabilities, which were used to develop a novel Vulnerability Injection tool.

Conceptually, attack injection is based on the injection of realistic vulnerabilities that are automatically attacked, and finally the result of the attack is evaluated. As proposed in [15], a tool able to perform vulnerability and attack injection is a key instrument that can be used in several relevant scenarios, namely: building a realistic attack injector, train security teams, evaluate security teams, and estimate the total number of vulnerabilities still present in the code, among others.

The challenge is that current knowledge on vulnerability and attack models is quite limited, and additional studies are required to better understand how, where and when such faults should be injected (in a way that assures high representativeness). Also, existing work is focused on very specific types of vulnerabilities in the Web domain. Extending such approaches to additional domains is a relevant research challenge.

IV. SECURITY BENCHMARKING

Computer benchmarks are standard tools that allow evaluating and comparing different systems or components according to specific characteristics (e.g. performance, robustness, dependability, etc.) [16]. The work on performance benchmarking has started long ago. Ranging from simple benchmarks that target very specific hardware systems or components to very complex benchmarks focusing complex systems (e.g. database management systems, operating systems), performance benchmarks have contributed to improve successive generations of systems. Research on dependability benchmarking boosted in the beginning of this century [17]. Several works have been done by different groups and following different approaches (e.g. experimental, modeling, fault injection). Finally, work on security benchmarking is a new topic with many open questions.

Several security evaluation methods have been proposed in the past [18]–[21]. The Orange Book [20] and the Common Criteria for Information Technology Security Evaluation [19] define a set of generic rules that allow developers to specify the security attributes of their products and evaluators to verify if products actually meet their claims. Another example is the red team strategy [21], which consists of a group of experts trying to hack its own computer systems to evaluate security.

The work presented in [22] addresses the problem of determining, in a thorough and consistent way, the reliability and accuracy of anomaly detectors. This work addresses some key aspects that must be taken into consideration when benchmarking the performance of anomaly detection in the cyber-domain.

The set of security configuration benchmarks created by the Center for Internet Security (CIS) is a very interesting initiative [23]. CIS is a non-profit organization formed by several well-known academic, commercial, and governmental entities that has created a series of security configuration documents for several commercial and open source systems. These documents focus on the practical aspects of the configuration of these systems and state the concrete values each configuration option should have in order to enhance overall security of real installations. Although CIS refers to these documents as benchmarks they mainly reflect best practices and are not explicitly designed for systems assessment or comparison.

Vieira & Madeira proposed a practical way to characterize the security mechanisms in database systems [24]. In this approach database management systems (DBMS) are classified according to a set of security classes ranging from Class 0 to Class 5 (from the worst to the best). Systems are classified in a given class according to the security requirements satisfied. In [25] the authors analyze the security best practices behind the many configuration options available in several well-known DBMS. These security best practices are then generalized and used to define a set of configuration tests that can be used to compare different database installations. A benchmark that allows database administrators to assess and compare database configurations is presented in [26]. The benchmark provides a trust-based security metric, named minimum untrustworthiness, that expresses the minimum level of distrust the DBA should have in a given configuration regarding its ability to prevent attacks.

The use of trust-based metrics as an alternative to security measurement is discussed in [27]. This work also proposed a trustworthiness benchmark based on the systematic collection of evidences (collected using static analysis techniques) that can be used to select one among several web applications, from a security point-of-view.

Security benchmarking, and security assessment in general, is an open research problem. In fact, although there are several works in the literature, there is no "good enough" model for assessing and comparing the security of alternative systems and components. A key issue is that security is largely related with the "unknown" vulnerabilities and attacks, and comparing systems based on well defined attackloads may lead to conclusions that ultimately do not hold in the field (e.g. when a new vulnerability or attack type is discovered). Thus additional work is required to best understand the problem, propose generic frameworks and models for security comparison, studying the representativeness of attackloads, understand how new vulnerability and attack types can be considered, etc.

V. SECURITY IN THE SOFTWARE PROCESS

A software development process is composed of multiple phases [28]. To improve the situation in software security it is important not only to understand the existing approaches and tools but also to adequately integrate them in the development process, i.e. to use such approaches and tools in the points of the process where they can make the difference. Different authors divide the software process in different ways, but usually software development includes the following phases (which can be repeated in an iterative manner): initialization, design, implementation, testing, deployment and decommissioning.

The process starts with requirements gathering (including security requirements), followed by specification and design, implementation (coding), testing and deployment. Decommission takes place when the product is not useful/used anymore. Although code security concerns should be addressed during the entire software product development lifecycle, as highlighted by [29] especial focus should be put in three key phases [30]: implementation, testing, and deployment. The next points summarize the main challenges and put in the context of these three phases the concepts and techniques introduced before:

- **Implementation**: during coding we must use best practices that avoid the most critical vulnerabilities in the specific application domain. Examples of practices include input and output validation, the escaping of malicious characters, and the use of parameterized commands [1]. Vulnerability and attack injection techniques have in this phase a very important job in the evaluation of the best security testing tools to use. Also, for the success of this phase, it is essential to adequately train the development teams. For instance, experience shows that the main reason for the vulnerabilities existing in web application's code is related to training and education. First, there is a lack of courses/topics regarding secure design, secure coding, and security testing, in most computer science degrees [30]. Second, security is not usually among the developers' main skills as it is considered a boring and uninteresting topic (from the development point-of-view), and not as a way to develop new and exciting functionalities.
- **Testing**: as introduced before, there are many security testing techniques available for the identification of vulnerabilities during the testing phase [1]. To mitigate vulnerabilities, it is necessary to have well-trained teams read that adequately apply those techniques during the development of the application. The problem is that software quality assurance teams typically lack the knowledge required to effectively detect security problems. It is necessary to devise approaches to quickly and effectively train security assurance teams in the context of web applications development, by combining vulnerability injection with relevant guidance information about the most common security vulnerabilities. Also, benchmarking techniques should be applied to assess, compare, and select the most adequate security testing tools for each concrete scenario.
- **Deployment**: at runtime, it is possible to include in the environment different attack detection mechanisms, such as Intrusion Detection Systems (IDS) and Web Application Firewalls (WAF), among others. These mechanisms can operate at different levels and use different detection approaches. The main problems preventing their use are related to the performance overheads and to the false positives that disrupt the normal behavior of the system. In this phase, security benchmarking plays a fundamental role in helping to select the best alternatives (in terms of servers, security mechanisms, etc.) to use, according to specific security requirements. Also, vulnerability and attack injection techniques represent in this phase an efficient way to evaluate the effectiveness of attack detections mechanism to be installed.

VI. CONCLUSION

In this paper we introduced techniques for security testing and assessment in the context of web applications (some of them quite novel such as security benchmarking and vulnerability and attack injection). As an essential condition for deploying secure systems, we also discussed aspects related to the software development process. These are of extreme importance for software designers and developers and allow an effective assessment of the security attributes of the software components being designed/deployed.

The paper highlighted several research challenges in an attempt to motivate further research in these topics. The paper did not intend to provide a comprehensive survey, but to focus on key promising aspects in which research is need, but that can already be applied in the context of the software industry.

REFERENCES

[1] D. Stuttard and M. Pinto, *The web application hacker's handbook: discovering and exploiting security flaws*. Wiley Publishing, Inc., 2007.

[2] G. McGraw and B. Potter, "Software Security Testing," *IEEE Security and Privacy*, vol. 2, no. 5, pp. 81–85, 2004.

[3] S. Christey and R. A. Martin, "Vulnerability type distributions in CVE," *V1. 0*, vol. 10, p. 04, 2006.

[4] A. Stock, J. Williams, and D. Wichers, "OWASP Top 10," 2007.

[5] A. Singhal, T. Winograd, and K. Scarfone, "Guide to Secure Web Services: Recommendations of the National Institute of Standards and Technology," *Report, National Institute of Standards and Technology, US Department of Commerce*, pp. 800–95, 2007.

[6] OWASP Foundation, "OWASP Application Security FAQ Version 3," 2010. [Online]. Available: http://www.owasp.org/index.php/OWASP_Application_Security_FAQ. [Accessed: 09-Aug-2010].

[7] N. Antunes and M. Vieira, "Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in Web Services," in *15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009. PRDC '09*, Shanghai, China, 2009, pp. 301–306.

[8] J. Carreira, H. Madeira, and J. G. Silva, "Xception: A technique for the experimental evaluation of dependability in modern computers," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 125–136, 1998.

[9] M. Rodríguez, F. Salles, J.-C. Fabre, and J. Arlat, "MAFALDA: Microkernel Assessment by Fault Injection and Design Aid.," in *EDCC*, 1999, vol. 1667, pp. 143–160.

[10] J. A. Duraes and H. S. Madeira, "Emulation of Software Faults: A Field Data Study and a Practical Approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 849–867, 2006.

[11] J. Durães and H. Madeira, "Definition of Software Fault Emulation Operators: A Field Data Study.," in *DSN*, 2003, pp. 105–114.

[12] J. Fonseca and M. Vieira, "Mapping software faults with web security vulnerabilities," presented at the IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008., 2008, pp. 257–266.

[13] N. Neves, J. Antunes, M. Correia, P. Verissimo, and R. Neves, "Using Attack Injection to Discover New Vulnerabilities," in *International Conference on Dependable Systems and Networks, 2006. DSN 2006*, 2006, pp. 457–466.

[14] J. Fonseca, M. Vieira, and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," in *13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007)*, Melbourne, Australia, 2007, pp. 365–372.

[15] J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & attack injection for web applications," in *IEEE/IFIP International Conference on Dependable Systems & Networks, 2009. DSN '09*, 2009, pp. 93–102.

[16] J. Gray, *Benchmark Handbook: For Database and Transaction Processing Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992.

[17] K. Kanoun and L. Spainhower, *Dependability Benchmarking for Computer Systems*. Wiley-IEEE Computer Society Pr, 2008.

[18] Commission of the European Communities, *The IT Security Evaluation Manual (ITSEM)*. 1993.

[19] P. K. Infrastructure and T. P. Profile, "Common Criteria for Information Technology Security Evaluation," 2002.

[20] L. Qiu, Y. Zhang, F. Wang, M. Kyung, and H. R. Mahajan, "Trusted computer system evaluation criteria," in *National Computer Security Center*, 1985.

[21] Sandia National Laboratories, "Information Operations Red Team and Assessments^{TM}." [Online]. Available: http://www.sandia.gov/iorta/. [Accessed: 23-Sep-2012].

[22] R. A. Maxion and K. M. C. Tan, "Benchmarking anomaly-based detection systems," in *Proceedings International Conference on Dependable Systems and Networks, 2000. DSN 2000*, 2000, pp. 623 – 630.

[23] "Center for Internet Security." [Online]. Available: http://www.cisecurity.org/. [Accessed: 23-Sep-2012].

[24] M. Vieira and H. Madeira, "Towards a security benchmark for database management systems," in *International Conference on Dependable Systems and Networks, 2005. DSN 2005.*, Yokohama, Japan, 2005, pp. 592 – 601.

[25] A. A. Neto and M. Vieira, "Towards assessing the security of DBMS configurations," in *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008*, 2008, pp. 90 –95.

[26] A. A. Neto and M. Vieira, "A Trust-Based Benchmark for DBMS Configurations," in *15th IEEE Pacific Rim International Symposium on Dependable Computing, 2009. PRDC '09*, 2009, pp. 143 –150.

[27] A. A. Neto and M. Vieira, "Benchmarking Untrustworthiness," *International Journal of Dependable and Trustworthy Information Systems*, vol. 1, no. 2, pp. 32–54, 32 2010.

[28] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of software engineering*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2002.

[29] G. McGraw, *Software Security: Building Security In*. Addison-Wesley Professional, 2006.

[30] M. Howard and D. E. Leblanc, *Writing Secure Code*, 2nd ed. Redmond, Washington: Microsoft Press, 2002.

Marco Vieirais an assistant professor at the University of Coimbra, Portugal. His interests include dependability and security benchmarking, experimental dependability evaluation, fault injection, software development pro-cesses, and software quality assurance. Vieira received a PhD in computer engineering from the University of Coimbra. He is a member of the IEEE Computer Society.