# Software Security Comprehension

Bernhard J. Berger and Michaela Bunke

Technologie-Zentrum Informatik, Universität Bremen
Bibliothekstraße 1, 28359 Bremen, Germany

{berber|mbunke}@tzi.de

## Abstract

Software security is becoming more and more important with the increasing number of applications and platforms connected to the Internet, for example, enterprise applications, smartphones or the iPad. The growing importance makes it a progressively interesting field for developers, software designers, end users, and enterprises. Fixing security bugs belongs to the traditional field of software maintainace what makes it also necessary to apply software-comprehension techniques. In the following we present our ongoing research on software-security comprehension, a first step towards program comprehension tailored to the needs of security experts.

## 1 Introduction

Recently, an increasing number of vulnerabilities has been published and gets recognized by end users and programmers [11]. Due to the fact that security gains more and more attention during software development, software designers and maintenance programmers are progressively faced with implementing and applying security solutions to their software systems. The advent of security requirements a software system has to fulfil is often accompanied by a change of functional requirements, such as migration to a service-oriented architecture or software as a service, leading to a greater exposure to attacks.

The challenges experts are facing in this area are to find already existing security vulnerabilities and to understand the software-security aspects within a legacy software. In the past, mainly the first objective has been tackled which led to a range of academic and commercial static analysis tools aiming to detect potential security-related programming flaws such as buffer overflows, wrong API usage and a diverse number of injection vulnerabilities based on missing input validation [2].

All these tools have in common that their goal is to find security vulnerabilities based on erroneous coding patterns. The presentation of the results within these tools is consequently limited to several categorized lists of findings or aggregated dashboard-like views and execution traces leading to the identified vulnerabilities. They do not focus on conceptual errors including insufficient authorization checks, incorrect realization of access control or contradictory security checks. There are approaches to finding these conceptual errors with the help of model-checking techniques, but they do not check whether the implementation is in sync with the actual model.

The second challenge experts are facing, understanding software security, is mentioned by Chess and West in their book about secure programming with static analysis [3]. They state that tools like the Fujaba tool suite[13, 9], a general purpose program comprehension tool, are useful to make sense of large code bases. McGraw argues that it is not enough to use a general purpose program, but that software security tools have to be designed especially for security purposes ([8], page 125). Beyond general purpose program comprehension tools, there exist only a few approaches that use reverse engineering techniques to understand the security aspects of software. According to van Wyk and McGraw security experts are rarely development experts [14], therefore it is necessary to assist them during the process of program comprehension.

## 2 Objectives

With our current research we want to enhance the Bauhaus Tool Suite [12] to support analysis and visualizations in the security domain to bridge the aforementioned gap. At present, we are concentrating on two main points: architecture reconstruction and security-pattern detection.

Within the scope of the ASKS project[1] we try to understand the security aspects of Java enterprise applications. The first goal is to extract (semi-)automatically an architectural view, depicting the components within a software system. Due to the fact that we are focussing on a single application type, Java enterprise applications, we can take additional information such as deployment descriptors into account to create a software architecture that reflects the technical boundary conditions. With some supplementary data provided by a system developer or

[1]This work was supported by the German Federal Ministry of Education and Research (BMBF) under the grant 01IS10015 B. (ASKS project).

architect it is possible to reconstruct functional layers. In essence the information provided by the deployment descriptors and the system expert can help to extract a tree-tier architecture overlayed with layers defined by the domain expert.

The extracted architecture can be enriched afterwards with security-related aspects, such as the data-entry and data-exit points of those components. These points allow one to identify the attack surface, the collection of all resources an attacker can manipulate, of a component or of the whole application more easily. As a next step, we plan to add aggregated data flow information between entry and exit points to show explicitly the paths where data travels. By categorizing the data within the application according to the sensitivity that is accessed, we can detect paths where data enters a part of the application it should not enter. Later on it is possible to add additional information to the depiction, for instance the container-based authorization checks, that allow to enforce an access-control policy based on the groups a user belongs to.

The second research main point is the detection of security patterns. For the software maintenance process exist several reengineering tools to get a clearer view about the software structure and behaviour. However, only a few of those tools take the well-known design patterns [4] into account to support program comprehension [15]. Presently, none of them supports the detection of security patterns, but ensuring security is a significant task. The primary goal of security patterns is to harden software or infrastructures against common attacks and misuse [16, 6, 5]. They are also best practice and said to be proven for built-in security. Therefore, we plan to recognize security patterns so that security patterns can be preserved during the software maintenance process. Their highlighting allows one to well-directed implementations of new (security) features in software.

The vision of both presented approaches is that they can be combined and be used to support the security comprehension in the software lifecycle. For example by visualizing a security enhanced software architecture in combination with the enforced policies. This way a developer would be aware of the implemented security mechanisms.

## 3 Conclusion

The program comprehension way of looking at a problem can be used to support the security awareness of people who are involved in the software development lifecycle. Our early literature research on the topic of program-comprehension shows that only a few approaches investigate in security issues. We showed in our prior work that there are several open software security topics where program comprehension techniques can be applied to support the understanding of software security [10, 1]. Similar conclusions were gathered by Karppinen et al in their case study where they try to find backdoors in an application [7].

In our opinion there is a lot of research to be done to support the software-security community with a set of tailored analysis and program-comprehension techniques to enable them to use the benefits provided by program comprehension.

## References


[1] M. Bunke and K. Sohr. An architecture-centric approach to detecting security patterns in software. In *Proceedings 3rd ESSoS*, volume 6542 of *Lecture Notes in Computer Science*, pages 156–166. Springer, 2011.

[2] B. Chess and G. McGraw. Static analysis for security. *IEEE Security and Privacy*, 2:76–79, 2004.

[3] B. Chess and J. West. *Secure programming with static analysis*. Addison-Wesley Professional, 2007.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Object-Oriented Software*. Addison Wesley, 1994.

[5] M. Hafiz and R. E. Johnson. Evolution of the mta architecture: the impact of security. *Softw. Pract. Exper.*, 38(15):1569–1599, 2008.

[6] S. Haldikis, A. Chatzigeorigou, and G. Stephanides. A practical evaluation of security patterns. In *Proceedings of AIDC*, 2006.

[7] K. Karppinen, M. Lindvall, and L. Yonkwa. Detecting security vulnerabilities with software architecture analysis tools. In *Proceedings of IEEE ICSTW*, 2008.

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

[9] U. Nickel, J. Niere, and A. Zündorf. The fujaba environment. In *Proceedings of the 22nd international conference on Software engineering*, International Conference on Software Engineering 2000, pages 742–745, New York, NY, USA, 2000. ACM.

[10] K. Sohr and B. Berger. Idea: Towards architecture-centric security analysis of software. In *Proceedings of the 2nd ESSoS*. Springer, 2010.

[11] The H Security. Number of critical, but unpatched, vulnerabilities is rising. Online, 2010. `http://www.h-online.com/security/news/item/Number-of-critical-but-unpatched-vulnerabilities-is-rising-1067495.html`.

[12] Universität Bremen. Website, 2011. `http://www.informatik.uni-bremen.de/st/projektedetails.php?id=&projekt_id=100`.

[13] Universität Paderborn. Website, 2011. `http://www.fujaba.de/`.

[14] K. van Wyk and G. McGraw. Bridging the gap between software development and information security. *Security Privacy, IEEE*, 3(5):75 – 79, sept.-oct. 2005.

[15] M. VanHilst and E. B. Fernandez. Reverse engineering to detect security patterns in code. In *Proceedings of SPAQu*, 2007.

[16] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Proceedings of 4th PLOP*, 1997.