Studying the impact of security patterns on software architecture design through CWE weaknesses analysis and reverse engineering case study

PhD student name: Monica Buitrago[1], Ph.D. start date: 01/2021, Expected defense date: 01/2024, advisors:, Isabelle Borne[1], and Jérémy Buisson[2]

[1] Université de Bretagne Sud, IRISA, Vannes, France
monica-johana.buitrago-ramirez@univ-ubs.fr - isabelle.borne@univ-ubs.fr
[2] Écoles de Saint-Cyr Coëtquidan, IRISA, Guer, France
jeremy.buisson@irisa.fr

Abstract. Designing a secure software system is a challenging task in constant evolution, as is measuring the security level of the software architecture. To solve this, some authors have proposed to design the architecture using security patterns to mitigate vulnerabilities. On the one hand, there are different sets of metrics to evaluate software quality attributes, but there are no security metrics to evaluate the efficiency of the patterns or the security level of the system from its design. For this reason, we selected a list of known and widely used patterns, and we studied the relationships between patterns and the Common Weakness Enumeration (CWE) database. This study aims to mitigate vulnerabilities caused by weaknesses in the software. On the other hand, we have chosen a case study, a software developed by a third party (Bitwarden) which is representative of a secure and commercial software architecture. We reverse-engineered the application to extract all the data that make this system secure, such as the security patterns involved in its design. We also propose, with the obtained data, to find a way to quantitatively evaluate the security level of the software architecture.

Keywords: Software architecture · Security by design · Security patterns · Reverse engineering.

1 Introduction

With the current context of cybersecurity threats, engineers become better aware of the need that the systems they develop must take into account the security concern. In this context, there is a challenge: being able to decide, as soon as design-time, whether a system will be sufficiently secure, and adapt the design decisions accordingly.

To address this challenge, security patterns have been proposed, for example, by Fernandez et al. [18,11], as a catalog of security-related problems and solutions. Some studies show that the implementation of patterns is used to detect vulnerabilities, mitigate security threats, and increase the quality of software. The underpinning idea of security patterns is to capture expert knowledge in the form of documentation with a specific structure that contains proven solutions for recurring problems, here in the domain of security. Though, Yskout et al. [20] question the effectiveness of security patterns, as their empirical experimentation does not yield any clear conclusion.

In addition to security patterns, one can think that the higher the complexity and use of sensitive operations in a system, the higher the risk that this system may contain exploitable vulnerabilities [6,14]. So, metrics may provide some quantitative assessment on the chances that a system reaches

a desired security level, possibly as soon as the design phase if the metrics apply to design artifacts such as the architecture. Though, many previous works in this area focus on code metrics instead [4,8,14,17].

In this paper, we show our proposal to address these two research gaps, both the evaluation of the effectiveness of safety standards and the lack of specific metrics to measure the security of a software at the architecture level. In the following section we present works related to the use of security patterns and metrics in software architecture, as well as works that propose similar methodologies to ensure and guarantee software security from its design. In section 3, we present our progress in exploring the architectural CWE in relation with the existing security patterns. For this purpose, we selected the list of security patterns described in the books [18,11], as a tool to mitigate software weaknesses which in turn cause vulnerabilities. From the description of each pattern in the book, we studied which patterns could mitigate weaknesses mentioned in CWE's Architectural Concepts view [3]. That is, one or several patterns may mitigate one or several weaknesses. In section 4, we apply a reverse engineering methodology to a known application (Bitwarden), developed by a third party and with characteristics representative of a secure software architecture, as a case study. This methodology is composed of two stages: data extraction from the software architecture and data experimentation to measure the security level in the application. Finally, the section 5 presents the conclusions of the paper and points out future work.

2 Related work

Except complex or change-prone ones like decorator and template method, design patterns are known to improve correctness, performance and security quality attributes [10]. But, Feitosa et al. [10] studied only using GoF's general-purpose patterns and at the class level, so when the detailed design is available. Patterns can also be used to detect vulnerabilities [13,19] and mitigate security threats [5,16]. In the PhD project, we intend to consider the coarser-grained architecture, which comes sooner in the design process, and we specifically target security concerns.

Several metrics were proposed to measure various software attributes, such as complexity and variability [12,19,7,8,10,9]. Some metrics are specifically targeted at security [5,15]. Chondamrongkul et al. [5] proposed an analysis for the architecture of microservice-based software systems, to score the attack surface, defense in depth, least privilege, compartmentalization, man in the middle (MitM) and denial of service (DoS). To do so, they use an ontology reasoner to detect patterns for each of these scores. The approach is mostly based on the deployment view of the architecture.

Sejfia et al. [19] focused on vulnerability detection in software architectures, using a similarity metric to compare with vulnerability patterns by means of graph alignment and hierarchical agglomerative clustering. But, according to the reported results, neither methodology was successful, due to problems of computational cost and applicability in real-life scenarios. Besides, their objective is different, since they sought to represent a group of similar vulnerabilities by using a self-developed pattern.

3 Mitigating vulnerabilities through patterns

We selected Schumacher et al.'s security patterns [18] because, according to Google Scholar, they are widely cited. In parallel, CWE [3] enumerates 918 typical weaknesses that are common in software and hardware systems, of which 223 weaknesses concern architectural concepts.

Table 1: Results - patterns to mitigate the weaknesses from Authenticate Actors category.

CWE Category	# Weakness	Pattern	Frequency	Results
Authenticate Actors	28	Password Design and Use	11	Although [18] gives 21 patterns related to user *identification and authentication* (I&A), the 28 weaknesses can be mitigated using 3 patterns. Indeed, in the I&A category, CWE do not detail weaknesses related to Biometrics and Hardware Token mechanisms, such as Face Recognition, Finger Image, Magnetic Card and One-Time Password Token Smart Card, among others from [18].
		Automated I&A Design Alternatives	7	
		I&A Requirements	10	

From the description of each pattern, we investigated which ones can mitigate the weaknesses from CWE's Architectural Concepts view (CWE-ACV). The Table 1 shows an excerpt of this relation that we established between CWE and security patterns. The first column gives the CWE category, followed by the amount of weaknesses in that category according to the CWE database. The third column lists the patterns that can be used to mitigate the weaknesses of the category, as well as the amount of weaknesses mitigated by each individual pattern. The last column comments the CWE category, e.g., stating that 3 of 21 patterns of the I&A category and sufficient to mitigate all the weaknesses of the CWE's authenticate category.

The pattern-weakness relationship was established according to our research, knowledge and experience. However, categories with a higher number of weaknesses tend to be more complex to relate and mistakes could be made. To overcome this threat to validity, we rely on the CWE's research concepts view (CWE-RCV), which organizes the weaknesses in a hierarchy. With this additional information, we deduce whether a pattern can mitigate several weaknesses, i.e., a subtree in the hierarchy. The figure 1 shows the position of 6 weaknesses (in green - given by their CWE identifier) of the *audit* category in the CWE-RCV hierarchy. In the figure, 5 of them (identifiers 223, 778, 224, 532, and 779) belong to the same parent pillar 664 (improper control of a resource through its lifetime), which indicates they are related. So, these 5 weaknesses might be mitigated by the same security patterns. On the other hand, weakness 117 has a different parent pillar. So, its mitigating pattern may be another pattern.

4 Case Study: Bitwarden

To experiment our proposal, we study the Bitwarden case. This application is a password manager, that users can use to store and share passwords securely. We select this application for the following reasons: (1) The developers claim a high level of security by regular audits, research, certifications, among other reasons explained extensively on their website[1]. The application has millions of users, which reflects a good reputation and a high level of trust. (2) The application adopts the typical distributed multi-tiers architecture, with several front-ends (smartphone applications, desktop clients, web site, browser plugins), a back-end and a database. (3) It is an open source application hosted on GitHub, this allows us to explore and evaluate it.

[1] `https://bitwarden.com/` (visited on 19/04/2022).

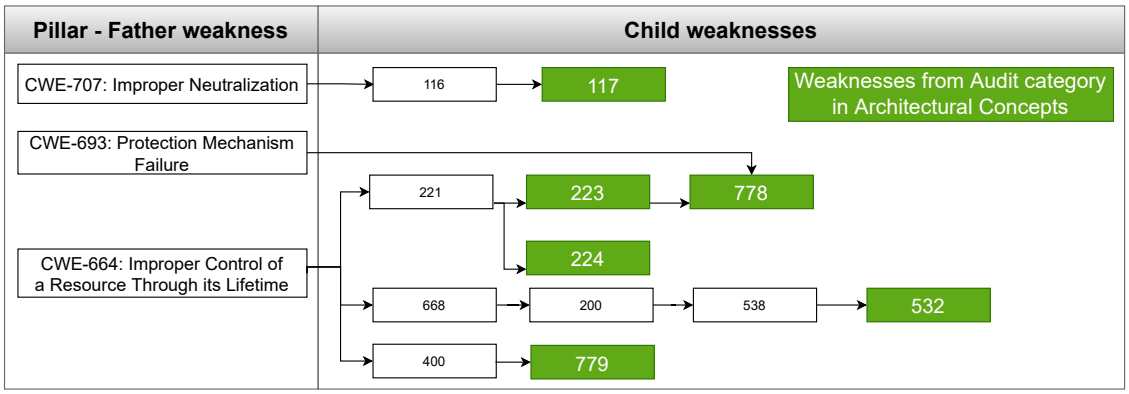

Fig. 1: Maping CWE's Architectural Concepts view into Research Concepts view - *audit* category

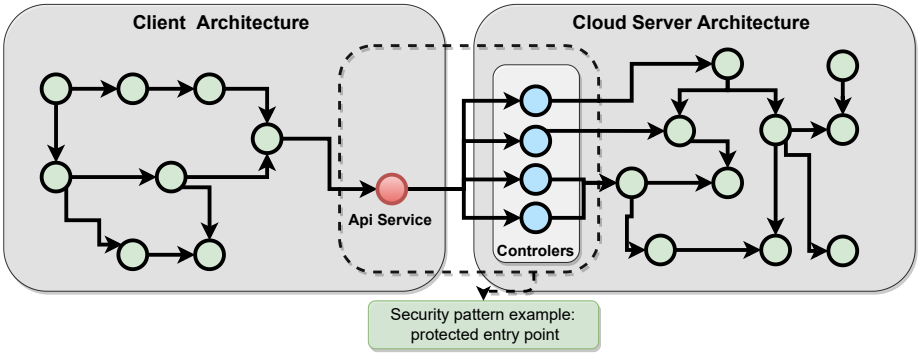

Fig. 2: Example of security pattern in a software architecture.

The Figure 2 shows an example of a software architecture (excerpt showing a front-end client and a back-end server) where a security pattern is found by modeling it, here the protected entry point pattern [11]: access to the server pass through well-identified entry points, ASP.Net Core controllers in the Bitwarden case, which are the typical locus, e.g., for policy enforcement points.

Figure 3 summarizes the process that we apply to the case study. As a first step, we consider only the web site and the back-end.

In the stage 1, we perform reverse engineering of Bitwarden, to recover its architecture from its source code. First, we extract the UML class diagram. The web site client code has 775 elements (classes and interfaces) and the server contains 1058 elements. Of these elements, we retain only those involved with the internal service-based structure and the client-server relationship (website 108 elements and server 197 elements). Noticeably, in the website, we drop all the elements related to the graphical user interface.

Both the website and the server are internally built as compositions of services. In the Bitwarden's class diagram, the pattern for service implementation and dependencies is simple and homogeneous. So, we abstract the structure of the website and of the server as a graph of ser-

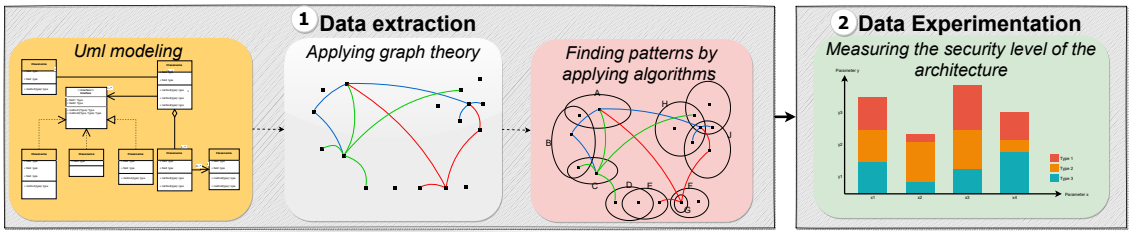

Fig. 3: Analysis methodology.

vices and dependencies. Then, we attempt to apply known algorithms, such as finding strongly connected components and clustering, in order to find composite structures and abstract even more the structure of the software.

In the stage 2, we plan to investigate a set of metrics to measure the level of security. As a first step, with the pattern of Figure 2 in mind, one of our ideas is to focus the metrics on the edges connecting composite structures.

5 Conclusion

Based on the two research gaps found in the literature review and the security problem in software architectures, we propose to use a list of established security patterns to mitigate vulnerabilities from the software architecture design and measure the effectiveness of their implementation using a quantitative evaluation system.

First, we study the relationship between security patterns and weaknesses in CWE, the results are shown in section 3. According to the description of the selected patterns and weaknesses, the patterns could mitigate the vulnerabilities caused by the weaknesses.

Moreover, we analyze the Bitwarden software architecture to serve as our case study, since its architecture is representative of many other real-world systems. The Bitwarden study is still in progress. We are still experimenting to find suitable composites. In parallel, we start the data experimentation stage, where we can observe some security-related patterns and we search for metrics that characterize their presence. The next step will be data experimentation, where we will develop a catalog of metrics to measure the security level of the software architecture.

In our future work, we also intend to study CAPEC [2] and ATT&CK [1] frameworks, which describe the attack techniques. Our long-term goal is to propose a metrics/pattern-based approach to assess the risks of attacks and weaknesses identified in the existing corpus.

References

1. ATT&CK®, `https://attack.mitre.org/`
2. CAPEC - Common Attack Pattern Enumeration and Classification (CAPEC™), `https://capec.mitre.org/`
3. CWE - Common Weakness Enumeration, `http://cwe.mitre.org/`
4. Alshammari, B., Fidge, C., Corney, D.: Security Metrics for Object-Oriented Designs. In: 2010 21st Australian Software Engineering Conference. pp. 55–64 (Apr 2010). https://doi.org/10.1109/ASWEC.2010.34

5. Chondamrongkul, N., Sun, J., Warren, I.: Automated Security Analysis for Microservice Architecture. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). pp. 79–82 (Mar 2020). https://doi.org/10.1109/ICSA-C50368.2020.00024
6. Chowdhury, I., Zulkernine, M.: Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities? In: Proceedings of the 2010 ACM Symposium on Applied Computing. pp. 1963–1969. SAC '10 (Mar 2010). https://doi.org/10.1145/1774088.1774504
7. Dalla Palma, S., Di Nucci, D., Palomba, F., Tamburri, D.A.: Toward a catalog of software quality metrics for infrastructure code **170**, 110726. https://doi.org/10.1016/j.jss.2020.110726
8. Du, X., Chen, B., Li, Y., Guo, J., Zhou, Y., Liu, Y., Jiang, Y.: Leopard: identifying vulnerable code for vulnerability assessment through program metrics. In: Proceedings of the 41st International Conference on Software Engineering. pp. 60–71. ICSE '19, IEEE Press (May 2019). https://doi.org/10.1109/ICSE.2019.00024
9. El-Sharkawy, S., Krafczyk, A., Schmid, K.: MetricHaven: More than 23,000 metrics for measuring quality attributes of software product lines. In: Proceedings of the 23rd International Systems and Software Product Line Conference - Volume B. pp. 25–28. SPLC '19. https://doi.org/10.1145/3307630.3342384
10. Feitosa, D., Ampatzoglou, A., Avgeriou, P., Chatzigeorgiou, A., Nakagawa, E.Y.: What can violations of good practices tell about the relationship between GoF patterns and run-time quality attributes? Information and Software Technology **105**, 1–16 (Jan 2019). https://doi.org/10.1016/j.infsof.2018.07.014
11. Fernandez-Buglioni, E.: Security Patterns in Practice: Designing Secure Architectures Using Software Patterns. Wiley (May 2013)
12. Iacob, M.E., Monteban, J., Sinderen, M.v., Hegeman, E., Bitaraf, K.: Measuring enterprise architecture complexity. In: 2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW). pp. 115–124. https://doi.org/10.1109/EDOCW.2018.00026
13. Jasser, S.: Enforcing Architectural Security Decisions. In: 2020 IEEE International Conference on Software Architecture (ICSA). pp. 35–45 (Mar 2020). https://doi.org/10.1109/ICSA47634.2020.00012
14. Morrison, P., Moye, D., Pandita, R., Williams, L.: Mapping the field of software life cycle security metrics. Information and Software Technology **102**, 146–159 (Oct 2018). https://doi.org/10.1016/j.infsof.2018.05.011
15. Márquez, G., Taramasco, C., Astudillo, H.: Defining security metrics to evaluate electronic health records systems: A case study in chile. In: 2020 IEEE International Conference on Software Architecture Companion (ICSA-C). pp. 173–180. https://doi.org/10.1109/ICSA-C50368.2020.00038
16. Pedraza-García, G., Noël, R., Matalonga, S., Astudillo, H., Fernandez, E.B.: Mitigating security threats using tactics and patterns: a controlled experiment. In: Proccedings of the 10th European Conference on Software Architecture Workshops. pp. 1–7. ECSAW '16 (Nov 2016). https://doi.org/10.1145/2993412.3007552
17. Saarela, M., Hosseinzadeh, S., Hyrynsalmi, S., Leppänen, V.: Measuring Software Security from the Design of Software. In: Proceedings of the 18th International Conference on Computer Systems and Technologies. pp. 179–186. CompSysTech'17 (Jun 2017). https://doi.org/10.1145/3134302.3134334
18. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. Wiley (Feb 2006)
19. Sejfia, A., Medvidović, N.: Strategies for Pattern-Based Detection of Architecturally-Relevant Software Vulnerabilities. In: 2020 IEEE International Conference on Software Architecture (ICSA). pp. 92–102 (Mar 2020). https://doi.org/10.1109/ICSA47634.2020.00017
20. Yskout, K., Scandariato, R., Joosen, W.: Do security patterns really help designers? In: Proceedings of the 37th International Conference on Software Engineering - Volume 1. pp. 292–302. ICSE '15 (May 2015). https://doi.org/10.1109/ICSE.2015.49