Maintenance of Secure Software: Suggesting the Optimal Secure Software Maintenance Methodology

Sean Carroll

Villanova University

"This paper or presentation is my own work. Any assistance I received in its preparation is acknowledged within the paper or presentation, in accordance with academic practice. If I used data, ideas, words, diagrams, pictures, or other information from any source, I have cited the sources fully and completely in footnotes and bibliography entries. This includes sources that I have quoted or paraphrased. Furthermore, I certify that this paper or presentation was prepared by me specifically for this class and has not been submitted, in whole or in part, to any other class in this University or elsewhere, or used for any purpose other than satisfying the requirements of this class, except that I am allowed to submit the paper or presentation to a professional publication, peer reviewed journal, or professional conference. In adding my name following the word 'Signature', I intend that this certification will have the same authority and authenticity as a document executed with my hand-written signature.
Signature Sean Carroll"

Abstract

This paper explores current secure maintenance design frameworks in an effort to present the most optimal secure software maintenance methodology for industry groups. The articles primarily focus on the issues encountered with such methodologies when they are tested in the commercial market with large-scale implementations of software. The ideal design of secure software should aim its maintenance methodology towards the goal of mitigating threats through the secure software development cycle. Maintenance of secure software is often not taken into consideration until functional goals in the design of software are achieved. Especially in the current development frameworks, the maintenance of secure software cannot be implemented until the later phases of the development lifecycle of software. Due to the design requirements and misunderstandings of developers, which combine to create chaos because the lack of knowledge and experience, current methodologies simply cannot reach ideal implementation. Maintenance of software must be clearly defined in order to separate the roles of maintenance and evolution in secure software design. Upon defining maintenance and its roles, maintenance goals can be addressed as to support the development of secure software along each phase of it of its development lifecycle.

Maintenance of Secure Software: Suggesting the Optimal Secure Software Maintenance Methodology

Current Secure Software Maintenance Methodologies are often addressed from a post-lifecycle perspective. When the development of software is complete and it is removed from implementation, its role in security is often addressed to discover optimal practices in the future. The struggle of software operation centers to gradually approach the issues of software maintenance is often one that comes from necessity, and the secure evolution and maintenance are often intertwined in practice. Seeking the ideal approach to secure software maintenance outlines a methodology that solves issues of implementation and integrates such methodology in a way that mitigates threats, but also ensures design and operational requirements are met.

Literature Review

The matter of optimal secure software maintenance to prevent vulnerabilities is one that few articles address specifically. It appears that because of the lack of implementation of secure practices in other areas of the secure software development cycle (SSDLC), industry has seen very few implementations that directly address this issue. Despite this difficulty, Thomazinho, L'Eràrio, and Fabri directly address the topic in their 2017 article, titled "A Case Study on the Strategy of Maintaining Commercial Software with a Large Number of Users." Thomazinho et al. lists software maintenance on the first page of their article as "an indispensable activity, without which existing systems would quickly become lagging and inefficient for organizations." The methodology in which this maintenance occurs is of great challenge in practice due to evolution of such software after it is delivered to a user within the scope of an organization (Thomazinho et al., 2017). Often times, the software is not developed by third-party groups, and the integration of security is one that is even more difficult when communication is limited to

clients, as it appears. Hanssen, Yamashita, Conradi, and Moonen discuss this issue in their 2009 article, titled, "Maintenance and Agile Development: Challenges, Opportunities and Future Directions." Hanssen et al. lists this phenomenon as software entropy, where maintenance becomes an issue in the evolution of software throughout its lifecycle due to the repeated changes that take effect in considerations for userability and other requirements. The entire agile development framework is in direct concern when discussing this topic of software entropy from a secure software maintenance point of view. Hanssen et al. assert that the practice of testing software after it has been built disregards the ultimate concerns for maintenance from the start. Because the development goals of initial design do not include maintenance as a primary concern, Hanssen et al. find this to be detrimental to deploying a secure software maintenance framework, as it requirements are known from the start, however only certain requirements are taken into consideration. Anwar, Idris, Ramzan, Shahid, and Rauf, mention the major challenges with this post-maintenance practice in their 2010 article, "Architecture Based Ripple Effect Analysis: A Software Quality Maintenance Perspective." Anwar et al. mentioned that the growing emphasis on software quality creates need for software quality maintenance as a result. The maintenance that occurs can create a ripple effect along the secure software development lifecycle if such maintenance is not addressed appropriately. In simple terms, there is a clear cost-benefit to maintaining and developing secure software in the beginning phase of its development lifecycle.

Discussion

Thomazinho et al.'s 2017 research paper limits its research to post-2009 articles that can be found on ACM, IEEE, and Springer online journal databases. The deceptive title of this article would leave one to deduce that only a case study is introduced, however less qualitative research

is presented than expected. The authors first note that software maintenance is preservation of pre-existing software, while evolution is unique in that it improves upon that software's functionality and overall quality. The systematic mapping in this article addressing a question of great interest to this literature review, "What are the best models, processes and/or practices to be adopted in software maintenance?" This is importance to note in Thomazinho et al.'s research because it clearly defines maintenance before an optimal secure software maintenance methodology is discussed. The authors answer to this question in their systematic approach is: a methodology that examines best practices in software design, examines common mistakes, and achieves low cost in maintenance, while selecting processes that provide a means to assist in communication between workgroups is one that is optimal. This provides the software maintenance team with an ongoing plan of action, creates efficiency in tasks, and ultimately provides a framework to optimize resources. The other papers' concerns of software entropy and the maintenance ripple effect are addressed under this proposed methodology.

Conclusion

Ultimately, the proposed methodology by Thomazinho et al. appears to take a culmination of current research into consideration in order to design a methodology that provides the best approaches that can be suggested. A framework that is optimal in use of resources and promotes long-term communication between workgroups as to what their priorities are, appears to be one that optimizes the way in which a maintenance team can operate. A more optimal methodology would be one that provides less-invasive integration, as in one that more easily integrates with current company-specific practices, rather than fundamentally changing such practices.

References

Anwar, S., Idris, F., Ramzan, M., Shahid, A. A., & Rauf, A. (2010). Architecture Based Ripple Effect Analysis: A Software Quality Maintenance Perspective. In *Information Science and Applications (ICISA), 2010 International Conference on* (pp. 1–8). IEEE.

Hanssen, G. K., Yamashita, A. F., Conradi, R., & Moonen, L. (2009). Maintenance and agile development: Challenges, opportunities and future directions. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on* (pp. 487–490). IEEE.

Thomazinho, H. C. S., L'Erário, A., & Fabri, J. A. (2017). A Case Study on the Strategy of Maintaining Commercial Software with a Large Number of Users. *Journal of Information Systems Engineering & Management*, *2*(4), 23.