The Use of Software Design Patterns to Teach Secure Software Design: An Integrated Approach

Johan Niekerk, Lynn Futcher

To cite this version:

Johan Niekerk, Lynn Futcher. The Use of Software Design Patterns to Teach Secure Software Design: An Integrated Approach. 9th IFIP World Conference on Information Security Education (WISE), May 2015, Hamburg, Germany. pp.75-83, 10.1007/978-3-319-18500-2_7 . hal-01334292

HAL Id: hal-01334292

https://hal.archives-ouvertes.fr/hal-01334292

Submitted on 20 Jun 2016

The use of software design patterns to teach secure software design: an integrated approach

Johan van Niekerk[1]and Lynn Futcher[2]

[1,2] Nelson Mandela Metropolitan University, Port Elizabeth, South Africa
{ Johan.VanNiekerk@nmmu.ac.za, Lynn.Futcher@nmmu.ac.za }

Abstract. During software development, security is often dealt with as an add-on. This means that security considerations are not necessarily seen as an integral part of the overall solution and might even be left out of a design. For many security problems, the approach towards secure development has recurring elements. Software design patterns are often used to address a commonly occurring problem through a "generic" approach towards this problem. The design pattern provides a conceptual model of a best-practices solution, which in turn is used by developers to create a concrete implementation for their specific problem. Most software design patterns do not include security best-practices as part of the generic solution towards the commonly occurring problem. This paper proposes an extension to the widely used MVC pattern that includes current security principles in order to teach secure software design in an integrated fashion.

Keywords: Information security education, secure software design, secure software development, software design patterns

1 Introduction

During software development, security is often dealt with as an add-on. This means that security considerations are not necessarily seen as an integral part of the overall solution and might even be left out of a design. Recently some efforts have been made to "shift some of the focus in security from finding bugs to identifying common design flaws" (http://cybersecurity.ieee.org/center-for-secure-design.html). However, efforts such as the IEEE centre for secure design is still relatively new and much work in this regard remains.

For many security problems, the approach towards secure development has recurring elements. Software *design patterns* are often used to address a commonly occurring problem through a "generic" approach towards this problem. The design pattern provides a conceptual model of a best-practices solution, which in turn is used by developers to create a concrete implementation for their specific problem.

The use of such design patterns has several major advantages. Firstly, the pattern provides a guideline towards best-practice. Secondly, the use of the pattern provides developers with a shared vocabulary that enables them to communicate complex design concepts easily and clearly. Due to these, and other benefits, design patterns are often taught in software design courses. Most software design patterns do not include **security principles** as part of the generic solution towards the commonly occurring problem. This paper proposes an extension to the widely used MVC pattern that includes current security principles in order to teach secure software design in an integrated fashion.

2 Teaching Secure Software Development

Khan and Mustafa [1] define secure software as: '*software that is able to resist most attacks, tolerate the majority of attacks it cannot resist, and recover quickly with a minimum of damage, from the very few attacks it cannot tolerate'*.

According to Burley and Bishop [2], there is an ever-increasing demand for software systems that are resilient, reliable and secure. They state that '*secure software development is a deep and tremendously important subject. Many problems arise from not focusing on the security aspects of software development*' [2].

Many software security vulnerabilities are not coding issues at all but design issues [3]. In order to meet the demands, opportunities and threats associated with software development, security needs to be integrated into the overall software development life cycle. However, the reality is that security is often perceived as a barrier to functionality, adding constraints and reducing flexibility. Software developers generally ignore the idea of security or consider it as an afterthought. This typically leads to software applications having many security flaws and weaknesses.

Microsoft authors Howard and LeBlanc [4], in support of secure software development, stress that software developers should avoid adding security as an afterthought for the following reasons:

- Adding any feature (including security) as an afterthought, is expensive;
- Adding security later may change the way features have been implemented. This, too, is expensive;
- Adding security later involves wrapping security around existing features, rather than designing features with security in mind; and
- Adding security as an afterthought may change the application interface, which may, in turn, break the code that has been used to rely on the current interface.

Taylor and Azadegan [5] support this notion, and state that: '*building secure systems requires incorporating security principles early and often throughout the software development life cycle*'. Information security should, therefore, be an integral part of the development process; and it should be taken into account at every stage of the software development life cycle.

In addition, software developers need to use improved practices that consistently produce secure software. Such practices should measurably reduce software specification, design and implementation defects; thereby, minimizing any potential risks. The development of secure software requires knowledge and techniques not commonly taught or practiced by most software developers [1].

In a report on the Summit on Education in Secure Software, Burley and Bishop [2] summarise some fundamental factors to ensure secure software development, namely:

- Understanding security, especially during design, requires a holistic approach;
- Programmers and non-programmers must be educated in the core principles and practices of secure software design;
- The principles of secure programming must be integrated into a curriculum designed to meet the cyber security challenges of the future; and
- Secure programming must be considered within the context of the full system design and deployment process.

According to Heyman, Yskout, Scandariato and Joosen [6], in the security discipline, a well-known principle calls for the use of standard, time tested solutions rather than inventing ad-hoc solutions from scratch. Various researchers propose that security patterns can potentially contribute significantly to the design and development of secure software, since they provide re-usable solutions to security problems, and incorporate expert knowledge. However, Yoshioka, Washizaki and Maruyama [7] state that: *'although various security patterns and techniques for using them have recently been proposed, it is still difficult to adapt them to each phase of the software development life cycle'*. Further research is therefore required to address the use of patterns within the software development life cycle in order to ensure that security concerns are integrated into the development process.

3 Teaching Advanced OO and Design Patterns

One of the benefits of the Object Orientated Programming (OOP) paradigm is that it allows software developers to reuse source code. Such reuse is supposed to bring many benefits, including increased productivity, improved code quality, and more design consistency [8]. In software development problems often recur, but not necessarily in the same context. Due to this difference in context, code reuse is not always practical since the different contexts of the problems might prevent the reuse of existing code without substantial modification.

In order to reuse a software solution, one would need a general solution that can be adapted to the specific problem's context. Such general solutions are not necessarily always possible in terms of reusable code, however it is often still possible to package domain-independent knowledge and expertise in a reusable way in the form of software design patterns [6].

A pattern can be described as '*a solution to a problem in a context*' [9]:

- The context is the situation in which the pattern applies. This should be a recurring situation;
- The problem refers to the goal you are trying to achieve in this context, but it also refers to any constraints that occur in the context [9];
- The problem should be a recurring problem [10];
- The solution provides a general design (core solution) that extracts the essence of the solution to resolve the problem for the given context and constraints [9,10].

Software designers rarely start a new design from first principles. Instead they rely on existing designs to inform the new solution. Design patterns provide a commonly used mechanism, or shared vocabulary to communicate such previous solutions to commonly occurring problems. It is important to note that a design pattern is not a finished design that can be transformed directly into code [11]. Instead, it provides a "template" for the solution that can be adapted to the problem's specific context [11]. Design patterns provide software designers with three main advantages:

- Firstly, the solution is known to be sound because it is time-tested;
- Secondly, benefits and drawbacks of a pattern are known in advance and they can be taken into account while sketching the solution;
- Thirdly, patterns establish a common vocabulary that can ease communication between different stakeholders [6, 10].

These advantages also make patterns very useful in the teaching of software design. Many complex system frameworks make extensive use of design patterns to provide basic underlying services within the framework. Knowledge of the underlying design patterns is often assumed, and used to aid in explanations regarding scalability, modularity, extensibility, etc. within these frameworks [11]. Such knowledge will, however, only aid in the discussions of complex frameworks if students '*understood the intent and implementations of each of the design patterns separately before combining them'* [11].

Design patterns can thus be seen as an important tool in the communication of knowledge regarding good software design. However, due to the need to present design patterns as a general template that is not specific to a particular context, most design patterns do not by default include any aspects related to secure software development. To a certain extent this exacerbates the problem of teaching secure software development since security should ideally not be dealt with as an optional add-on in a software design. Instead, systems should be designed as secure systems from the ground up. It would thus be beneficial for the teaching of secure software design, to have design patterns that incorporate basic secure design principles as an integral part of the pattern itself. Many design patterns could probably be adapted to include security concerns, however, this paper will only focus on one such pattern, the Model-View-Controller (MVC) pattern. The MVC pattern is widely used in modern software development and is especially useful in designs for a distributed n-tier architecture, which makes it an ideal pattern for the design of many online applications. As such this pattern was deemed appropriate for use in this paper.

4 Integrating Secure Design Principles into the Model-View-Controller (MVC) Pattern

The MVC pattern forms the basis for the very popular n-tier approach towards software design and is thus often taught in software design courses. The pattern is a compound pattern which uses other patterns to provide specific "services" to the design. The MVC separates the design and development of the user interface (View) from the underlying controlling logic for this view (Controller), which is further separated from the problem domain's underlying state, data, and application logic

(Model) [9]. The structure of the MVC pattern works to separate the responsibilities of components and is especially well suited to web applications [12].

The View shows the windows, buttons, and other controls to the user; the Controller interprets clicks and other commands; and the Model does the business logic and object retrieval – then relaying the changes to the View again [12]. The View can request state information from the Model, and the Controller can ask the View to update its display. This relationship is shown in Figure 1.

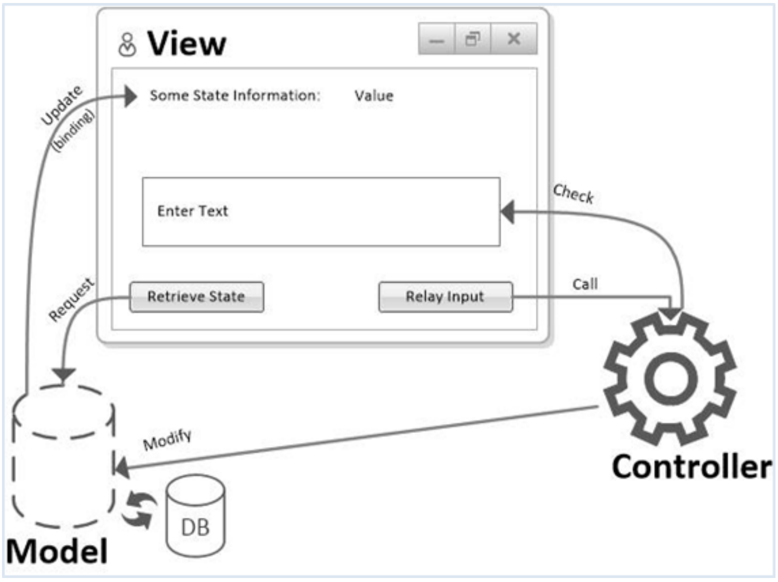

Figure 1: The Model-View-Controller [9]

Secure software development requires the designer, and developers, to consider the relevance of various secure design principles for the software's context of use. A comprehensive overview of such principles falls outside the scope of this paper. However, in previous work Colesky, Futcher and Van Niekerk [13] demonstrate how many of these principles can be integrated into the MVC pattern. The following discussion provides a brief overview of the proposed secure version of the MVC as shown in Figure 2.

Figure 2 provides a version of the MVC pattern with the following additional security principles incorporated:

1. Authentication, authorization, access control and trust services were added to the model between the controller and the model. The purpose of these services is to "guard all entrances";
2. A further layer of encipherment, which includes the hashing of passwords, encryption of sensitive data, and hiding of business logic through the preferred use of stored procedures were added to the model;
3. A notary function is added to log evidence of all transactions originating from the controller;

4. To combat input related risks a layer of mechanisms which include verification, the encoding, quoting, and escaping of characters, and validation of all inputs were added.

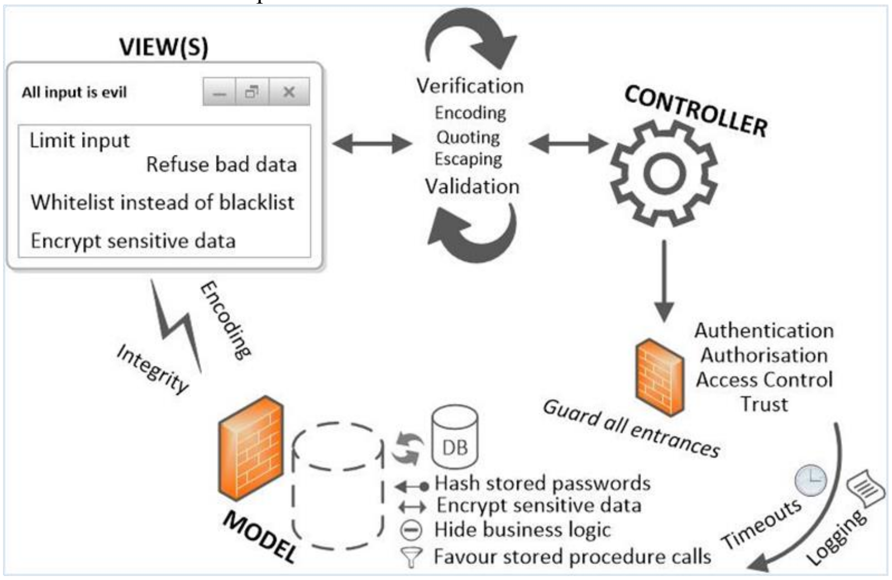

Figure 2: Security-conscious MVC [13]

The following section discusses the implications of using the security-conscious MVC for teaching secure software design.

5 Discussion

It is important to note that the proposed security-conscious MVC does not prescribe specific technologies for the secure services, but rather provides a specific design context where such services should be sensibly included.

Novice software development students often make mistakes such as placing the code for authentication mechanisms at view level. These mistakes are sometimes further exacerbated by having a username and password visible as unencrypted text within a web page. When developing a web application, many possible technologies could be used for access control and/or encipherment services. The intention of the security conscious MVC pattern is not to dictate a specific technology, but rather to emphasise that there **should** be an access control mechanism, and that this mechanism **should** exist between the Controller and the Model. Access control logic should thus not be exposed at the level of the View. Furthermore, by adding the encipherment layer to the pattern it provides a logical "prompt" for an educator to engage the class regarding what would be sensible to include under these services. For example, "*Should the authentication mechanism make use of encrypted communications*?" "*What would be an appropriate encryption technology to use for this context?*" "*Should data exposed by the model be encrypted*?"

One of the first considerations a designer is faced with when designing secure software is how to restrict, or control, access to the underlying data so that only authorised users can view or modify the data. From an access control point of view, the notions of authorisation, authentication, access rights, privileges and trust are of particular importance. Access to data should be given in accordance to the principle of *least privilege.* This principle dictates that an entity should be given access with as few rights as possible [14]; and it also requires that access be permitted for the shortest duration possible [4]. In the authors' experience, discussion of this principle is often done completely out of context and never practiced by software development students.

The use of the security conscious version of the MVC pattern provides a logical prompt for the inclusion of this topic during **every** discussion regarding an n-tier design. By including security considerations as a default into every web application (or similar) development project, students will get substantial exposure to the issues that should be considered and will, hopefully, also get substantial practice during laboratory exercises.

The structure of the presented security-conscious MVC pattern reflects both secure *design principles* and *best practices*. For example, in addition to stimulating discussion regarding the above principle of least privilege, this version of the MVC could also be used to introduce students to the principle of fail-safe defaults, which dictates that the default access granted to a resource should be *none*. Thus, unless access has been granted explicitly to a resource, access to that resource should be denied. In the presented model, this principle is implemented in conjunction with the principle of least privilege via a layered approach. Firstly, the practices of requiring authentication, authorization, and controlling access ensure that only trusted entities are given access to a specific resource. Secondly, because data is encrypted and business logic is hidden, all entities **have to** make use of the access control mechanisms in order to be able to see the underlying data in any form. Thus, should the programmer neglect to verify whether a specific entity should have access, the default behavior would be to present the underlying data in an encrypted format.

According to Martin et al. [15], the top four software vulnerabilities are SQL (Structured Query Language) injection, operating system command injection, buffer overflows and cross-site scripting, respectively. All of these vulnerabilities stem from various forms of malicious input. It is thus vital to include issues related to verification, the encoding, quoting, and escaping of characters, and validation of all inputs consistently during a software design curriculum. The use of this security conscious version of the pattern encourages active discussion regarding these important questions for every n-tier design. By including these topics, both in terms of underlying secure design principles and in terms of secure design best practices, as an integral part of the pattern, educators can ensure that these security services are not seen as an optional add-on, but rather as an essential part of the overall design.
'*Education is a very powerful tool in helping to write secure code. By understanding possible threats, programmers can save valuable time and create secure code while users can rest assured that their information is safer*' [16]. The authors believe that the integration of security principles into existing design patterns can be a powerful tool for educators to improve the teaching of secure software design.

5 Limitations and Future Work

The efficacy of this security conscious version of the MVC pattern as a tool to communicate security related concepts to students has not yet been tested. Furthermore, its perceived usefulness for software design educators has not been verified. In this approach, specific implementation details for such a secure MVC pattern was not included. A similar attempt to create a secure MVC pattern, that included more detailed implementation guidance, was presented by Delessy-gassant & Fernandez [17]. However, their approach focused on the inclusion of role based access control into the MVC and was thus considered less generic by the authors who, for the purposes of teaching, specifically aimed to include both security principles and best practices without being too specific regarding implementation. Future work should focus on validating the usefulness of this enhanced version of the pattern for the purpose of teaching secure software design. In addition, the integration of security principles into other software design patterns has not yet been examined.

6 Conclusion

Most aspects of information and cyber security are directly affected by the ability of software developers to produce secure applications. Vulnerabilities caused by poor coding or design practices are a major cause of security breaches. It has become essential for modern software development curricula to address security as an integrated part of the software development process. This paper presents an approach that includes secure software design principles into a commonly used software design pattern. This approach allows the consideration of security principles to form part of the overall design and thus not be relegated as an optional add-on. The authors believe that this approach could play a meaningful role in the teaching of secure software development. Society as a whole can no longer afford to treat security as an afterthought.

References

1. Khan, R. A. & Mustafa, K. (2008). Secured Requirement Specification Framework. American Journal of Applied Sciences, 1622-1629.

2. Burley, D. & Bishop, M. (2011). Summit on Education in Secure Software: Final Report. National Science Foundation.

3. Howard, M. (n.d.). Lessons Learned from Five Years of Building More Secure Software. Retrieved January 21, 2015, from MSDN Magazine: https://msdn.microsoft.com/en-us/magazine/cc163310.aspx#S1

4. Howard, M. & LeBlanc, D. (2003). Writing secure code: Practical strategies and techniques for secure application coding in a networked world. Microsoft Press.

5. Taylor, B. & Azadegan, S. (2006). Threading Secure Coding Principles and Risk Analysis into the Undergraduate Computer Science and Information Systems Curriculum. Information Security Curriculum Development Conference (InfoSecCD) (pp. 24-29). Kennesaw: ACM.

6. Heyman, T., Yskout, K., Scandariato, R., & Joosen, W. (2007). An Analysis of the Security Patterns Landscape. 29th International Conference on Software Engineering Workshops.

7. Yoshioka, N., Washizaki, H. & Maruyama, K. (2008). A survey on security patterns. Progress in Informatics. Special Issue: The future of software engineering for security and privacy (5), 35-47.

8. Barzilay, O., & Urquhart, C. (2014). Understanding Reuse of Software Examples: A Case Study of Prejudice in a Community of Practice. Information and Software Technology, 56(12), 1613–1628.

9. Freeman, E., Freeman, E., Sierra, K., & Bates, B. (2004). Head First Design Patterns.

10. Dooley, J. (2011). Software Development and Professional Practice. Apress.

11. Pieterse, V., & Marshall, L. (2010). What is a Design Pattern? SACLA, 1–25. Retrieved from http://web.up.ac.za/ecis/SACLA2010PR/ SACLA2010/Papers/ SACLA030.pdf

12. Syromiatnikov, A. & Weyns, D. A Journey through the Land of Model-View-Design Patterns, WICSA, 2014, 2014 IEEE/IFIP Conference on Software Architecture (WICSA), 2014 IEEE/IFIP Conference on Software Architecture (WICSA) 2014, pp. 21-30

13. Colesky, M., Futcher, L., & Van Niekerk, J. (2013). Design patterns for secure software development: demonstrating security through the MVC pattern. 15th Annual Conference on WWW Applications. 10-13 September 2013. Cape Town

14. Meier, J., Mackman, A., Vasireddy, S., Dunner, M., Escamilla, R., & Murukan, A. (2003). Improving Web Application Security: Threats and Countermeasures (p. 919). Microsoft Press. Retrieved from http://books.google.com/books?hl=en&lr=&id=Spti0mHhlsUC&oi=fnd&pg=PP2&dq=Improving+Web+Application+Security+Threats+and+Countermeasures&ots=KEfBrKEhQM&sig=mnrDoHKZH93NkQWktonnAw9greE

15. Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). 2011 CWE / SANS Top 25 Most Dangerous Software Errors. Retrieved from http://cwe.mitre.org/top25/archive/2011/2011_cwe_sans_top25.pdf

16. Yu, H., Jones, N., Bullock, G., & Yuan, X. Y. (2011). Teaching secure software engineering: Writing secure code. 2011 7th Central and Eastern European Software Engineering Conference, CEE-SECR 2011, 1–5.

17. Delessy-gassant, N., & Fernandez, E. B. (2012). The Secure MVC pattern. In 1st LACCEI International Symposium on Software Architecture and Patterns (pp. 1–6). Panama City, Panama.