A set of courses for teaching secure software development

Eduardo B. Fernandez, Shihong Huang, and Maria M. Larrondo-Petrie
Dept. of Computer Science and Eng.
Florida Atlantic University, Boca Raton, FL 33431

Introduction

A good percentage of the software deployed in industrial/commercial applications is of poor quality, it is unnecessarily complex, and contains numerous flaws that can be exploited by attackers. Every day the press reports of attacks to web sites or databases around the world, resulting in millions of dollars in direct or indirect losses. This situation does not appear to improve. There are several reasons for this situation, including the pressure to bring products to the market quickly, the complexity of modern software, the lack of knowledge about security of most developers, and others. Until recently the only vendors' response to problems of security was to provide patches to fix the latest vulnerability found. However, patches are clearly not the best solution: it is hard for system administrators to keep up with the latest patches and the patch itself may open new possibilities for attack. There are two basic approaches to improve application security: 1) examine final production code and look for possible problems, e.g., buffer overflow conditions [How03] or 2) plan for security from the beginning. We believe that the solution lies in developing secure software from the beginning, applying security principles along the whole lifecycle. As indicated, a good part of the problem is that developers are not, in general, acquainted with security development methods. We see the use of patterns as a fundamental way, even for developers with little experience, to implicitly apply security principles.

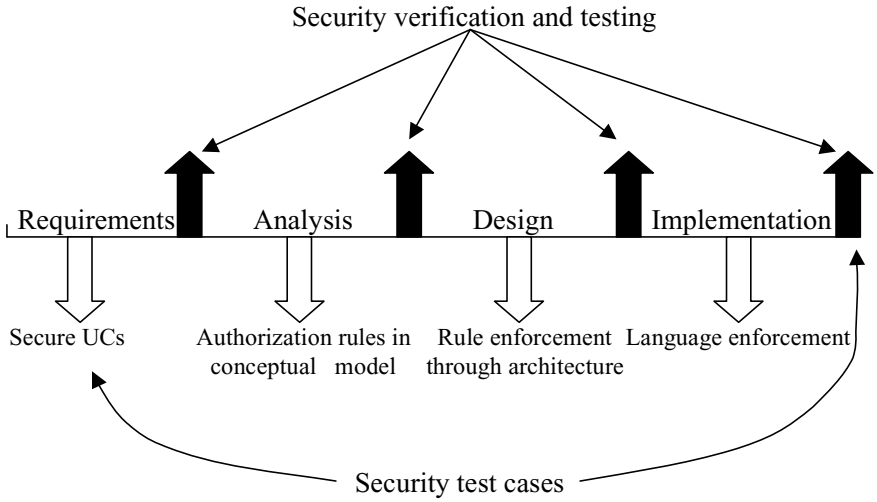
Patterns are one of the most important developments in software. A pattern solves a specific problem in a given context and can be tailored to fit different situations. A pattern embodies the knowledge and experience of software developers and can be reused in new applications. Analysis patterns can be used to build conceptual models [Fer00], design patterns can be used to improve the software design [Gam94], and security patterns can be used to build secure systems [Sch06]. Patterns embody good design principles and by using them, the designer is implicitly applying these principles. By learning and applying them, students learn good design methods [Lar05]. Because of their use of abstraction, patterns are valuable in understanding complex systems. We currently teach our security courses using patterns and we have found them especially useful for explaining security concepts and for teaching good systems design [Fer05a, Fer05b]. Applying such a methodology requires catalogs of security patterns. We have developed patterns for authorization models [Fer01], for operating systems security [Fer02], for authentication [Fer03], for firewalls [Del04], for web services standards [Del05], and for other security mechanisms. Some of these have been collected in a book [Sch06].

Florida Atlantic University offers degrees in Computer Science and Computer Engineering. Our security and software engineering courses already stress the need to address safety and security concerns at every level of the organization, in every phase of development, and in every system component and level of implementation. Specifically, we teach graduate and undergraduate versions of an introductory security course that presents an overview of the main topics of data and network security [Fer05a]. We just introduced a graduate course on the security of distributed systems. In all these courses, we have intended from the beginning to present a conceptual, design-oriented approach, explaining the reasons behind the many existing security mechanisms. Security encompasses all the system architectural levels and requires a unifying conceptual approach. Otherwise it becomes a collection of techniques and mechanisms to solve disjoint problems. We also have the support of the Dept. of Mathematics that offers courses on

cryptography. However, we believe we have to go beyond these courses to educate students who can also build secure software. Our intention is to complement the existing courses with courses that show ways to develop secure software. We are starting with graduate courses but in the future (as we did with the introductory security course) we may offer undergraduate versions.

As a guideline for our security courses we have developed a methodology for secure systems design. A basic idea of our methodology [Fer04, Fer06a] is that security principles must be applied at every development stage and that each stage can be tested for compliance with those principles. At the Requirements Stage we analyze the possible attacks to the system and we define policies to stop or mitigate them [Fer06c]. From the Analysis Stage we generate authorization rules that apply to the conceptual model. From the Design Stage we enforce rules through the architecture, including interfaces and distribution artifacts [Fer05c]. In the Implementation, Deployment and Maintenance Stages, language enforcement of the countermeasures and rules is required. Security verification and testing occurs at every stage of development.

A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with security principles. As indicated, another basic idea is the use of patterns to guide security at each stage. Figure 1 shows a secure software lifecycle, indicating where security can be applied (white arrows) and where we can audit for compliance with security principles and policies (dark arrows).



The courses about security in general use patterns to illustrate different mechanisms. They have been described in detail in [Fer05a] and [Fer05b]. To these courses we intend to add two new courses focusing specifically on software security aspects. The idea is to follow this methodology to teach the different sections of the course.

The new course on secure software development is intended to show different ways and approaches to build secure software. Although we follow and emphasize our methodology we will show also other approaches, including methods based on code analysis. Work on Aspect-oriented programming [Vie01] appears promising to handle code security and will be discussed. We will contrast our approach to the methodology used by Microsoft [How03].

We start by defining a context for a system in the style of Bishop [Bis04]. For example, a financial institution is affected by government regulations, and because it handles money it can be the target of a variety of attacks. The course gives strong value to security requirements, an aspect somewhat neglected in security textbooks.

We justify the use of specific security measures by the need to defend against some attacks, An attacker has an objective or goal that he wants to accomplish, e.g., steal the identity of a customer, transfer money to his own account, etc. To accomplish his purposes, he must interact with the system, trying to subvert one or more actions in a use case (he might do this indirectly). Low level actions such as attacking a system through a buffer overflow attack, are just ways to accomplish these goals but not goals in themselves. Looking at use cases is consistent with the idea that security must be defined at the highest system levels, a basic principle for secure systems [Fer06b].

Once we understand the possible attacks, we can define policies to stop them. These policies are used in turn to guide the selection and implementation of security mechanisms; for example where we should use authentication and the type of authentication required. If the attacks indicate that we require authorization we can then find the specific authorization rules that are needed. In an earlier paper we proposed a way to find all the rights needed by the actors of a set of use cases in an application [Fer97]. The idea is that all the use cases of an application define all the possible interactions of actors with the application. We need to provide these actors with rights to perform their functions. If we define appropriate rights, attacks can be prevented or mitigated. It is also in these interactions where attackers could try to misuse the system and the list of attacks gives us guidance of where to be careful.

Figure 2 shows the activity diagram for the use case “Open account” in a financial institution. Potentially each action (activity) is susceptible to attack, although not necessarily through the computer system. For this use case we could have the following attacks:

A1.The customer is an impostor and opens an account in the name of another person
A2.The customer provides false information and opens an spurious account
A3.The manager is an impostor and collects data illegally
A4.The manager collects customer information to use illegally
A5.The manager creates a spurious account with the customer’s information
A6.The manager creates a spurious authorization card to access the account
A7.An attacker tries to prevent the customers to access their accounts.
A8.An attacker tries to move money from an account to her own account

After the requirements we look at the analysis stage. Analysis patterns, and in particular semantic analysis patterns, can be used to build the conceptual model in a more reliable and efficient way [Fer00]. Security patterns describe security models or mechanisms. We can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases. In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. Then we only need to additionally specify the rights for those parts not covered by patterns. We can start defining mechanisms (countermeasures) to prevent attacks.

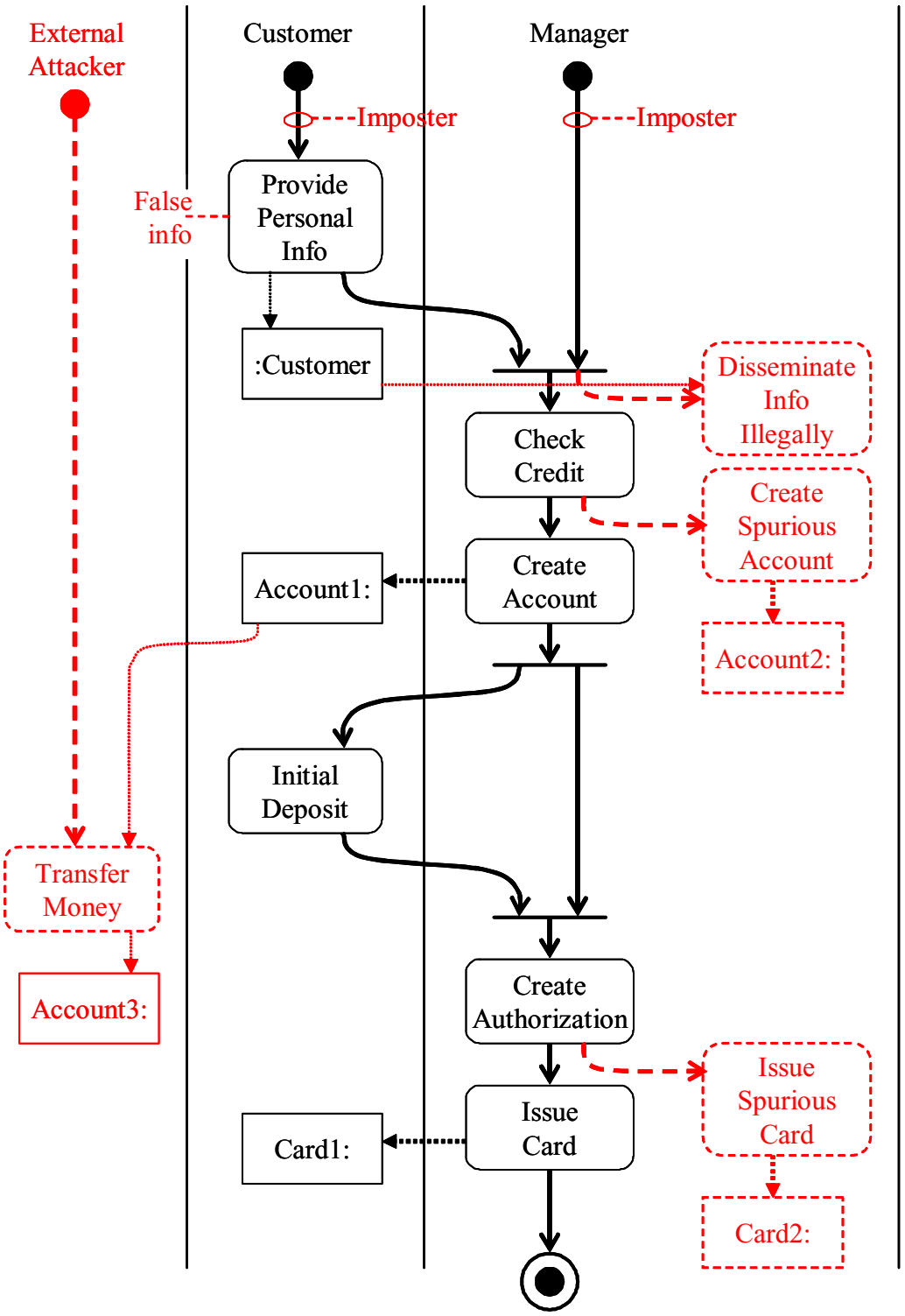


Figure 2. Activity diagram for use case “Open account”.

We proceed then to the design stage. When we have the possible attacks to a system, design mechanisms are selected to stop these attacks [Fer06a]. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage. Secure interfaces enforce authorizations when users interact with the system. Components can be secured by using authorization rules for Java or .NET components. Distribution provides another dimension where security restrictions can be applied. Deployment diagrams can define secure configurations to be used by security administrators. A multilayer architecture is needed to enforce the security constraints defined at the application level. In each level we use patterns to represent appropriate security mechanisms. Security constraints must be mapped between levels. Figure 3 shows the use of the MVC pattern [Bus96] to admit patients in a controlled way, e.g., only the Administartive Clerk role can perform this task.

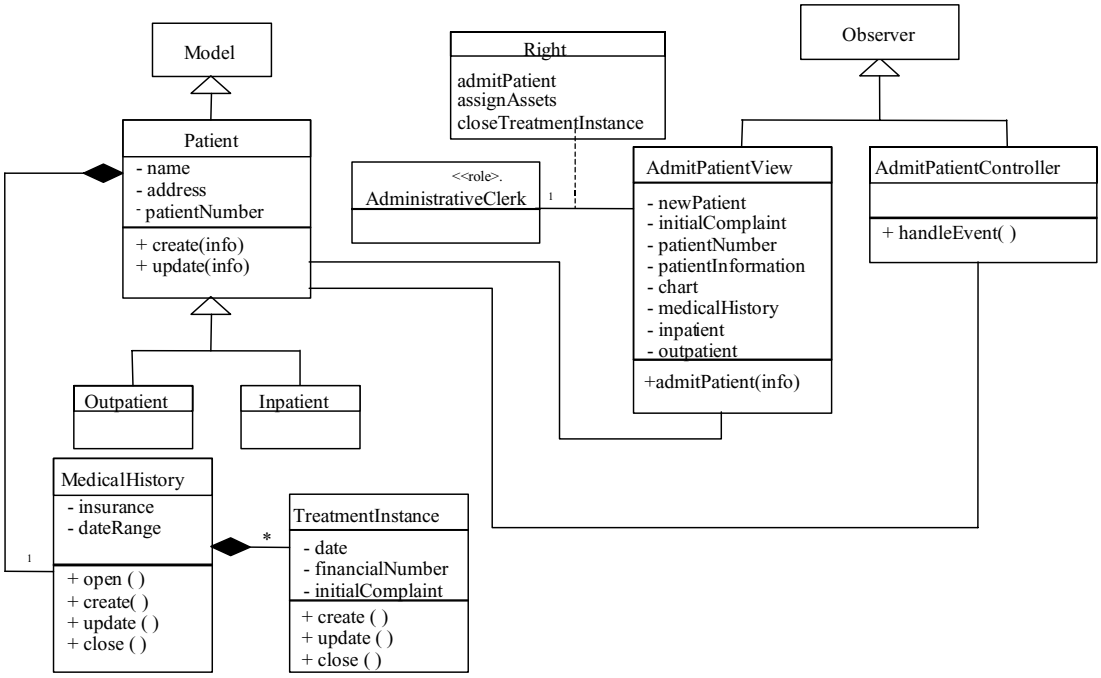


Figure 3. MVC pattern used to control access to patient records.

The other course is Secure Software Evolution. This will include topics on software audit of legacy systems, identification of antipatterns, and reengineering legacy systems to increase security. After students graduate, many of them will work on maintaining and evolving legacy systems. Probably these legacy systems have been developed in an ad hoc manner and without proper consideration of security. This course intends to make them aware of the problem and some solutions.

Conclusions

We believe the proposed courses will be as well received as our other security courses. The use of patterns is a unique aspect of all our offerings and appears to be a good approach to teaching security. Following the steps of the lifecycle will make the course more realistic. We teach a course on object-oriented design where we follow this approach and it works well because the students can see how requirements become finally code. In this case they should end up with secure code.

Acknowledgements

This work was supported through a Federal Earmark grant from DISA, administrated by Pragmatics, Inc. The reviewers provided valuable suggestions that improved this paper.

References

[Bis04] M. Bishop, "Teaching context in information security", *Procs. of the 6th Workshop on Education in Computer Security*, July 2004, 29-36.

[Bus96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, Volume 1. J. Wiley, 1996.

[Del04] N. Delessy-Gassant, E. B. Fernandez, S. Rajput, and M. M. Larrondo-Petrie, "Patterns for application firewalls", *Procs. of the Pattern Languages of Programming Conference(PLoP* 2004), http://hillside.net/patterns

[Del05] N. Delessy-Gassant and E. B.Fernandez, "Patterns for the eXtensible Access Control Markup Language", *Procs. of the Pattern Languages of Programs Conference* (PLoP 2005). http://hillside.net/patterns

[Fer81] E. B. Fernandez, R. C. Summers, C. Wood, *Database Security and Integrity*, Addison-Wesley, Reading, Massachusetts, Systems Programming Series, February 1981, 320 pp., Japanese translation, 1982.

[Fer00] E.B. Fernandez and X. Yuan, "Semantic analysis patterns", *Procs. of 19th Int. Conf. on Conceptual Modeling, ER2000*, 183-195.

[Fer01] E. B. Fernandez and R. Pan, "A Pattern Language for security models", *Procs, of the Pattern Languages of Programming Conference (PLoP 2001),* http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions

[Fer02] E. B. Fernandez, "Patterns for operating systems access control", *Proc. of the Pattern Languages of Programming Conference (PLoP 2002),* http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

[Fer03] E.B.Fernandez, and R. Warrier, "Remote Authenticator/Authorizer," *Proc. of the Pattern Languages of Programming Conference (PLoP 2003),* http://hillside.net/patterns/

[Fer04] E.B. Fernandez, "A methodology for secure software design", *Procs. 2004 Intl. Conference on Software Engineering Research and Practice (SERP'04),* Las Vegas, NV, June 21-24, 2004.

[Fer05a] E.B.Fernandez and M. M. Larrondo-Petrie, "Using UML and security patterns to teach secure systems design", *Procs. of the ASEE 2005 Annual Conference.*

[Fer05b] E.B.Fernandez and M. M. Larrondo-Petrie, "Teaching a course on data and network security using UML and patterns", *Procs. of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2005).*

[Fer05c] E.B. Fernandez, T. Sorgente, and M.M. Larrondo-Petrie,."A UML-based methodology for secure systems: The design stage". *Proceedings of the Third International Workshop on Security in Information Systems (WOSIS-2005),* Miami, May 24-25, 2005.

[Fer06a] E. B. Fernandez, M. M. Larrondo-Petrie, T. Sorgente, and M. VanHilst. "A methodology to develop secure systems using patterns", chapter in "*Integrating security and software engineering: Advances and future vision",* H. Mouratidis and P. Giorgini (Eds.), 2006.

[Fer06b] E. B. Fernandez, E. Gudes, and M. Olivier, *Secure Software Systems*, Addison-Wesley 2006 (to appear).

[Fer06c] E.B.Fernandez, M. VanHilst, M.M.Larrondo-Petrie, and S. Huang, "Defining security requirements through misuse actions" , sent for publication.

[Gam94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Boston, Mass., 1994.

[How03] M. Howard and D. LeBlanc, *Writing secure code*, (2nd Ed.), Microsoft Press, 2003.

[Lar05] C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3nd edition. Prentice-Hall, 2005.

[Sch06] M. Schumacher, E.B. Fernandez, D. Hybertson, F. Buschmann, and P. Sommerlad, *Security Patterns: Integrating security and systems engineering*, J. Wiley & Sons, 2006.

[Vie01] J. Viega, J.T. Bloch, and P. Chandra, "Applying Aspect-Oriented Programming to security", *Cutter Journal*, vol. 14, No 2, February 2001, 31-39.