Third LACCEI International Latin American and Caribbean Conference for Engineering and Technology (LACCET'2005)
"Advances in Engineering and Technology: A Global Perspective", *8-10 June 2005, Cartagena de Indias, COLOMBIA*

A Secure Systems Development Methodology as a Guide for Teaching Security

Eduardo B. Fernandez, PhD
Florida Atlantic University, 777 Glades Road SE-408, Boca Raton, FL 33431-0991 USA
ed@cse.fau.edu

María M. Larrondo Petrie, PhD
Florida Atlantic University, 777 Glades Road SE-308, Boca Raton, FL 33431-0991 USA
petrie@fau.edu

Abstract
Our introductory course on data and network security presents an overview of the main topics of security and has a conceptual and design emphasis. The Unified Modeling Language (UML) is the accepted standard for software development and it is a visual language very appropriate for the description of system architecture. Software patterns are well established for software analysis and design as a way to improve reusability and reliability. We have adopted an approach that combines UML and patterns to present models and mechanisms for security. We are also using this approach in a forthcoming security textbook. We have proposed a methodology to develop secure systems and we use it as a basis for all the chapters of this course. We describe here our course indicating its use of running examples incorporating security patterns.

Keywords
Patterns, security education, software engineering, UML

1. Introduction

Software systems must be built using sound principles and methodologies to achieve good quality and avoid security problems. Students need to learn how to design systems in a systematic and conceptual way. This requires a conceptual and unified understanding of how different mechanisms and subsystems work together to provide security. Most practical systems are quite complex, often containing or interacting with off-the-shelf components. Therefore, it is also necessary to be able to analyze existing systems in order to extend them or to combine them with new systems.

We teach a graduate (Fernandez, 2005a) and an undergraduate version (Fernandez, 2005b) of an introductory security course that presents an overview of the main topics of data and network security. We have intended from the beginning to present a conceptual, design-oriented course, explaining the reasons behind the many existing security mechanisms. Security encompasses all the system architectural levels and requires a unifying conceptual approach or it becomes a collection of techniques and

mechanisms to solve disjoint problems. Without a conceptual approach every new system is a surprise, instead of being another manifestation or embodiment of known principles and approaches.

The Unified Modeling Language (Rumbaugh, Jacobson and Booch, 1999) (UML) is the accepted standard for software development and it is a visual language very appropriate for the description of system architecture. Because of its graphical nature, it is intuitive and allows a convenient description of structural aspects. It can also be combined with the Object Constraint Language (OCL) (Warmer and Kleppe, 2003) or some other formal language, e.g. Z, to make some aspects of the model more precise. Software patterns (Buschmann, Meunier, Rohnert, Sommerlad and Stal, 1996) (Gamma, Helm, Johnson and Vlissides, 1994) (Larman, 2005) are well established as a way to improve reusability and reliability of software analysis and design. A specific variety of patterns, security patterns, have been proposed as a way to build secure systems. Numerous security patterns have been developed and methodologies on how to use them for secure software design are appearing (Fernandez, 2004).

We have adopted an approach that uses UML combined with software patterns to introduce examples of secure mechanisms that can be used as building blocks when designing secure systems. This approach is also used in a textbook in preparation by the first author (Fernandez, Gudes and Olivier, 2005). We have tried this approach in several offerings of these courses, including both academic and industrial institutions.

The paper is organized as follows: Section 2 indicates why object-oriented design and patterns are convenient to describe and build secure systems, while Section 3 shows details of the courses. Section 4 describes an example of the use of security patterns for teaching about security models. We end with some conclusions.

2. Object-oriented design and patterns

UML models are quite intuitive and can conveniently describe structural properties. However, they are less precise than formal methods. We can therefore integrate formal and informal techniques, describing our models using UML notation enhanced with constraints expressed in OCL. Adding formal constraints improves precision and reduces ambiguity in the model. It also gives students a gentle introduction to more formal models.

The development of object-oriented systems starts with the definition of use cases that describe the interactions with the system. These use cases define the functional specifications of the system and are used to guide all stages of development. A use case diagram, part of the UML standard, describes all the use cases for a particular application and the actors involved in them.

Object oriented models include two types of models: A static model, normally a class diagram, which describes the data/information aspects of the system, and a set of dynamic models including state, sequence, collaboration, and activity diagrams. State, sequence, collaboration, and activity diagrams provide additional aspects, such as collaboration between objects, and can describe workflows. Formal or informal constraints can refine and make all these models more precise. A significant advantage of object-oriented models is that they can be easily converted into software. The dynamic aspects of such a model can be implemented as operations in classes and the data parts of a class can be mapped to relational databases (Larman, 2005). These models are also convenient to represent the many restrictions and documents required by standards such as HIPPA, the Health Insurance Portability and Accountability Act (HIPPA, 1996), and the Sarbanes-Oxley Act (Sarbanes and Oxley, 2002); for example, documents for

medical visits can correspond directly to model classes. This is an important aspect because many regulations for enterprises require security restrictions on their documents.

One of the most important developments in software is the concept of *pattern*. A pattern solves a specific model 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 (Fernandez and Yuan, 2000), design patterns can be used to improve software design (Gamma, Helm, Johnson and Vlissides, 1995), and security patterns can be used to build secure systems (Schumaker, Fernandez, Hybertson and Buschmann, 2005). Patterns embody good design principles and by using them, the designer is implicitly applying these principles, thus improving the quality of her designs. By learning and applying them, students learn good design methods. Because of their use of abstraction, patterns are valuable to understand and compare complex systems. This is particularly useful in a rapidly-changing field where new approaches and products are constantly appearing. Catalogs of patterns are appearing to help designers in building all kinds of complex systems.

3. The course

As indicated earlier, the course is an introduction to data and network security. *Security* is the protection against:

- Illegal (unauthorized) data disclosure (*confidentiality*).
- Illegal data modification (*integrity*). Unauthorized modification of data may result in inconsistencies or erroneous data. Data destruction may bring all kinds of losses.
- Denial of service (attacks to *availability*)—Users or other systems may prevent the legitimate users from using the system.
- Lack of accountability *(non-repudiation*)—Users should be responsible for their actions and should not deny what they have done.

The definition of security above describes security in terms of defending against some types of attacks. The generic types of defenses (also known as *countermeasures*) include:

- Identification and Authentication —Ways to prove that a user or system is the one he/it claims to be. The result of authentication may be a set of *credentials*, which prove identity and may describe some attributes of the authenticated entity. This could involve *biometrics.*
- Authorization and Access control —Authorization defines permitted access to resources depending on the accessor (user, executing process), the resource being accessed, and the intended use of the resource. Access control is the mechanism used to enforce authorization. It includes confidentiality and data integrity.
- Audit—Implies keeping a log of actions that may be relevant for security, for monitoring, and for further analysis, such as forensics and documenting compliance with standards.
- Cryptography— It can be used for hiding the contents of data (secrecy), for authentication, or for other types of defenses.
- Intrusion detection—Alerts the system when an intruder is trying to attack the system.

The way software is developed (software process) and the specific methodology used for it, are very important to produce secure systems. Security must be a concern from the start of development.

Security principles should be applied at every development stage. We have developed a secure software development cycle we consider necessary to build secure systems (Fernandez, 2004). Figure 1 shows a secure software lifecycle, indicating where security can be applied and where we can audit for compliance with security policies.

Security is a multilayer problem – one cannot secure just one architectural layer. We consider the system structure as a set of hierarchical layers (a pattern in itself (Buschmann, Meunier, Rohnert, Sommerlad and Stal, 1996)) and we study the security properties of each layer as well as of the whole system. We consider specifically the hardware layer, the operating system layer (including communications), the database system layer, and the application layer. The lower layers are needed to enforce the application model constraints, which we define using the UML use cases. Patterns are used at all levels to facilitate mappings between levels. The course includes chapters on the security aspects of all these layers as well as on general aspects, its outline is shown in the Appendix. For each layer, we show its effect on security and possible defense mechanisms that can be implemented at that level (usually described through patterns). We try to correlate the layer to its lower layer, in particular which mechanisms of the lower layer are required to support those in the higher layer. We describe below what happens at each stage. The course sequence roughly follows the lifecycle sequence.

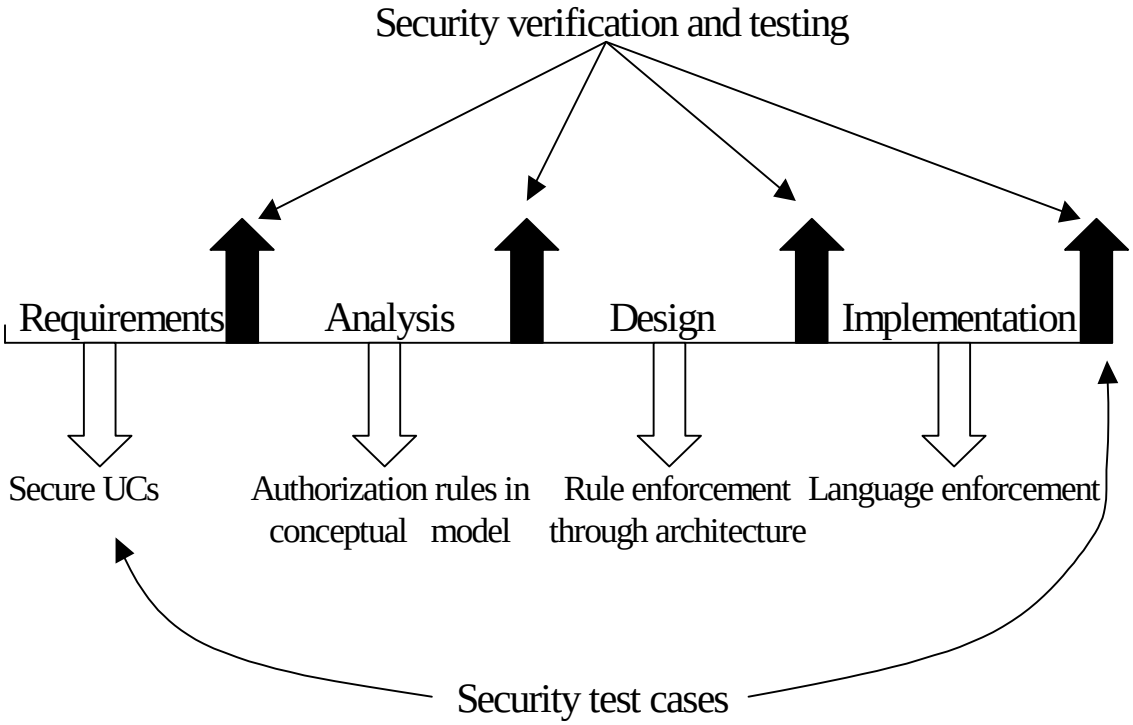

Figure 1: Secure software lifecycle

Requirements stage: *Use cases* for the system are defined at this stage, where a use case describes an interaction with the system. Attacks are then related to use cases. The first chapter of the course introduces several running examples. We model their use cases and relate attacks to the actions in each use case.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more convenient and efficient way (Fernandez and Yuan, 2000). Once an accepted conceptual model exists we can select a security model according to the type of attacks we consider important. We have developed patterns for all standard models (Fernandez and Pan, 2001) and instances of these patterns can be combined with the classes of the conceptual model we need to protect. We can also define access rights for the participating actors according to a least privilege policy (Fernandez and Hawkins, 1997).

Design stage: User interfaces should correspond to use cases and the rights defined for the actors can be enforced through them (Fernandez, Sorgente and Larrondo-Petrie, 2005). Distributed structures, involving brokers and web services, can also receive authorizations to enforce the security restrictions at the lower levels. Finally, components can be secured by using rules defined according to the authorization needed for each component. Deployment diagrams can define secure configurations, to be used by security administrators.

Implementation stage: This stage requires reflecting in the code the security constraints defined for the application. Because these constraints are expressed as classes and associations, they can be implemented as the rest of the application. We may also need to select specific security packages, e.g., a firewall product, a cryptographic package. These packages enforce the security constraints defined in the application.

At the end of each stage we can perform audits to verify that the security policies are being followed. We can now verify that our mechanisms take care of all the types of attacks defined in the requirements stage. In order to select the appropriate mechanisms we need to look at how the attacks for each use case are actually realized through the architectural layers of the system.

Applying such a methodology requires catalogs of security patterns (Schumacher, Fernandez, Hybertson, and Buschmann, 2005). We have developed patterns for authorization models (Fernandez and Pan, 2001), for operating systems security (Fernandez, 2002), for authentication (Fernandez and Warrier, 2003), for firewalls (Delessy-Gassant, Fernandez, Rajput and Larrondo-Petrie, 2004), and for other security mechanisms.

The undergraduate version of the course emphasizes the use and design of secure systems, while the graduate version puts emphasis on design and research aspects. In the undergraduate version the project is fixed for all students (individual or groups of up to four), while in the graduate version students select topics from a list and can even propose their own topics.

4. Running examples

To illustrate the steps of the methodology we use a set of running examples. They have been chosen to include a variety of applications and they are changed each term. Some of them are:

- A financial institution, described in this section.
- A medical records system, for use of medical personnel and patients during hospital treatment.
- An automated home, where many of the functions of a family home are controlled by computers.
- A remote voting system, where voters can vote locally or through the Internet.

As mentioned earlier, one of the first steps in system design should be the analysis of the possible attacks to the specific system and their consequences when successful. This analysis can be used to define the countermeasures we need and will also be useful later to evaluate the system security. This stage is called *vulnerability assessment* and several methodologies and tools are starting to appear to make it more precise. We have proposed analyzing the effect of attacks on each action in each use case. The idea is that attackers have some action in mind, e.g., capturing the list of the customers of the institution. Below we apply this idea to a specific example.

Consider a financial institution with use cases described by Figure 2. These describe a financial system for a company that provides investment services to its customers. Customers hold accounts and send orders to the company for buying or selling commodities (stocks, bonds, real estate, and art). Each

customer account is in charge of a custodian (a broker), who carries out the orders of the customers. Customers send orders by email or by phone. Brokers advise their customers about investments. A government auditor visits periodically to check for application of laws and regulations.

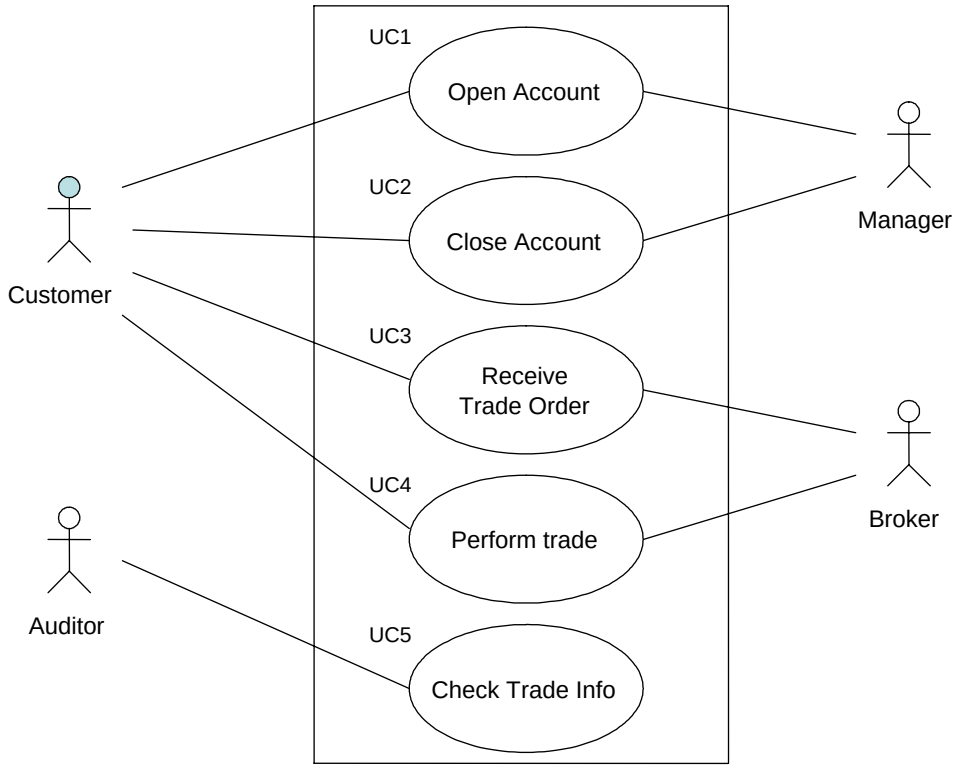

Figure 2. Use cases for a financial institution

We can now relate attacks to use cases. These include:

- Use Cases 1 and 2: Customer is an impostor. Possible confidentiality and integrity violations.
  Manager impersonation. Can capture customer information. Confidentiality attack.
  Repeated requests. Denial of service.
- Use Cases 3 and 4. Broker impersonation. Possible confidentiality violation.
  Customer can deny giving a trade order. Repudiation.
  Customer impersonation. Confidentiality or integrity attacks.
  Stockbroker may embezzle his customers' money. Integrity violation.
- Use Case 5. Impersonation of auditor. Confidentiality violation.

It is important to look for attacks first at this level and not in the lower levels. There are many possible attacks in a complex system but we need to concentrate on those that are a significant threat for our specific application. We can now define the rights needed by each actor to participate in its use cases. For example, a Manager must have the right to open and close accounts but not the right to withdraw money from the account (this would prevent embezzlement by the Manager). The following stages (not described for lack of space), would lead to architectures to enforce the defined rights (Fernandez, Sorgente, Larrondo-Petrie, 2005).

5. Conclusions and Future Directions

All students taking the course are acquainted with object concepts in their courses on data structures and introduction to programming and most of them also take a course on object-oriented design during their last years of undergraduate studies, which gives them the proper background to understand basic UML models. Their reaction to this approach has been very positive because they see the course as a way to learn, not only security, but also to reinforce their knowledge of object-oriented software design. We are also using this approach in a forthcoming security textbook (Fernandez, Gudes and Oliver, 2005). We believe this approach provides an appropriate level of precision while preserving an engineering approach for security concepts.

Acknowledgements

This work was supported through a Federal Earmark grant from DISA, administrated by Pragmatics, Inc. Comments from our students have been very valuable in improving the course.

References

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M., (1996). *Pattern-Oriented Software Architecture, Vol. 1, A System of Patterns*. John Wiley & Sons Ltd., New York, NY.

Delessy-Gassant, N., Fernandez, E. B., Rajput, S., and Larrondo-Petrie, M. M., (2004). "Patterns for application firewalls", *Proceedings. of the 11th Conference on Pattern Languages of Programs, PLOP 2004*, Allerton Park at Monticello, Illinois, USA, September 8-12, 2004. http://hillside.net/plop/2004/papers/ndelessygassant0/PLoP2004_ndelessygassant0_0.doc

Fernandez, E. B. and Hawkins, J. C. (1997). "Determining Role Rights from Use Cases". *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, Fairfax, Virginia, USA, November 6-7, 1997, 121-125. http://doi.acm.org/10.1145/266741.266767

Fernandez, E.B., and Yuan, X. (2000). "Semantic analysis patterns", *Proceedings of 19th International Conference on Conceptual Modeling, ER2000*, Salt Lake City, UT, USA, October 9-12, 2000, Editors: A. H. F. Laender, S. W. Liddle, V. C. Storey, Lecture Notes of Computer Science LNCS Vol. 1920, Springer-Verlag, 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf

Fernandez, E. B., and Pan, R. (2001). "A Pattern Language for security models", *Proceedings of the 8th Conference on Pattern Languages of Programs, PLoP 2001*, Allerton Park at Monticello, Illinois, USA, September 11-15, 2001. http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ebfernandezandrpan0/PLoP2001_ebfernandezandrpan0_1.pdf

Fernandez, E. B. "Patterns for operating systems access control". *Proceedings of the 9th Conference on Pattern Languages of Programs, PLoP 2002,* Allerton Park at Monticello, Illinois, USA, September 8-12, 2002. http://jerry.cs.uiuc.edu/~plop/plop2002/final/OSSecPatt7.doc

Fernandez, E. B. and Warrier, R. "Remote Authenticator/Authorizer", *Proceedings of the 10th Conference on Pattern Languages of Programs, PLoP 2003,* Allerton Park at Monticello, Illinois, USA, September 8-12, 2003. http://jerry.cs.uiuc.edu/%7Eplop/plop2003/Papers/Fernandez-remote-authenticator.pdf

Fernandez, E. B. (2004). "A methodology for secure software design". *Proceedings of the International Conference on Software Engineering Research Practice, SERP'04,* Las Vegas, Nevada, USA, June 21-24, 2004, Vol. 1. Editors: H. R. Arabnia and H. Reza, CSREA Press, 130-136.

Fernandez, E. B. (2005a). "CIS 6370 Computer Data Security Syllabus", Florida Atlantic University. http://polaris.cse.fau.edu/~ed/CIS%206370Outline.pdf

Fernandez, E. B. (2005b). "CEN 4540 Introduction to Data and Network Security Syllabus", Florida Atlantic University. http://polaris.cse.fau.edu/~ed/UndergradSecWithObjs.pdf

Fernandez, E. B., Gudes, E., and Olivier, M. (2005). *Secure Software Systems.* Addison-Wesley, Reading, MA (to appear).

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

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

HIPAA (1996), "Public Law 104-191: Health Insurance Portability and Accountability Act of 1996", August 21, 1996, 104th U.S. Congress, Washington D.C. http://www.hipaa.org/

Larman, C. (2005). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. 3rd edition. Prentice-Hall, Upper Saddle River, NJ.

Rumbaugh, J., Jacobson, I., and Booch, G. (1999) *The Unified Modeling Language Reference Manual*. Addison-Wesley, Boston, MA.

Schumacher, M., Fernandez, E.B., Hybertson, D. and Buschmann, F. (Eds.) (2005). *Security Patterns.* John Wiley & Sons Ltd, New York, NY (to appear).

Sarbanes, P. S., Oxley, M. (2002) "Sarbanes-Oxley Act of 2002", January 23 2002, 117th U. S. Congress, Washington D.C. http://news.findlaw.com/hdocs/docs/gwbush/sarbanesoxley072302.pdf

Warmer, J. and Kleppe, A. (2003). *The Object Constraint Language*, 2nd edition, Addison-Wesley, Reading, MA.

Biographic Information

Eduardo B. FERNÁNDEZ (http://polaris.cse.fau.edu/~ed), is a professor in the Department of Computer Science and Engineering at Florida Atlantic University, and the leader of the Secure Systems Research Group (http://www.cse.fau.edu/~security). He has published numerous papers and three books on different aspects of security, object-oriented analysis and design, and fault-tolerant systems. He holds a Ph.D. degree from UCLA. His industrial experience includes 8 years with IBM.

María M. LARRONDO PETRIE: Dr. Petrie is Associate Dean of Engineering and Professor of Computer Science & Engineering at Florida Atlantic University (http://www.cse.fau.edu/~maria), and a member of the Secure Systems Research Group at FAU. She is Vice President of Research of the Latin American and Caribbean Consortium of Engineering Institutions, serves on the American Society of Engineering Education's Minority Division Board, was on the ACM SIGGRAPH Education Board and was President of Upsilon Pi Epsilon Honor Society for the Computing Sciences. She has published numerous papers in the areas of security, analysis & design of complex systems, software engineering, computer graphics and engineering education.