Using UML and security patterns to teach secure systems design

Eduardo B. Fernandez, and María M. Larrondo Petrie

Department of Computer Science and Engineering, Florida Atlantic University, USA

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. There is a graduate and an undergraduate version of this course. A security course should encompass all the system architectural levels and provide a unifying conceptual approach or it becomes a collection of techniques and mechanisms to solve disjoint problems. For several reasons, formal methods are not appropriate for this purpose. 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. The students' 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.

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 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[1] and an undergraduate version[2] 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.

Formal methods are not appropriate for this purpose because the students may not have the appropriate mathematical background, and formal models may not exist for all the components of the system. Formal methods are not convenient to describe the structural properties of systems, a necessity in security analysis. The Unified Modeling Language[3] (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[4] (OCL) or some other formal language, e.g. Z, to make some aspects of the model more precise. Software patterns[5, 6, 7] 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 a few methodologies[8] on how to use them for secure software design are appearing.

Security courses have been classified as being scholarly oriented or training oriented [9]. Ours is more of a scholarly approach in that we try to emphasize principles and conceptual unification of topics. On the other hand, we have a clear engineering orientation, emphasizing applications, so this approach should also be attractive for training-oriented courses. The textbook used should match the type of course. Existing textbooks for security courses take one of two approaches:

- A purely descriptive approach, giving mostly examples and little theory[10].
- A highly formal approach, discussing mostly topics for which there are well-developed formal models[11].

The purely descriptive approach is not sufficient because of its lack of conceptual structure. While easy to follow, it does not provide more than a set of special cases, related only through their general objectives. This kind of textbook is more appropriate for training-oriented courses. The highly formal approach is hard for the students to follow and its emphasis on topics of existing models may neglect important topics and obscure very significant aspects of a system. We have adopted an intermediate approach: use a graphical, semi-formal notation, UML, combined with software patterns. This approach is also used in a textbook in preparation by the first author[12]. We have tried this approach in several offerings of these courses, including both academic and industrial institutions. Patterns have been used to teach software engineering[13] but we do not believe they have been used to teach security.

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

A variety of formal languages and approaches has been used to develop secure systems[11, 14]. Our experience with formalizing complex access control models has shown that the resulting expressions are not intuitive, require mathematical sophistication, and it is difficult to describe structural properties of the system[15], a basic requirement for any software architecture. On the other hand, UML models are quite intuitive and can conveniently describe structural properties. However, they are less precise than formal methods. We can therefore take a middle ground, integrating 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[7]. These models are also convenient to represent the many restrictions and documents required by standards such as HIPAA[16] and Sarbanes Oxley[17]; 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[18], design patterns can be used to improve software design[6], and security patterns can be used to build secure systems [19]. 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. Because of their use of abstraction, patterns are valuable to understand 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.

When building secure systems it is important to understand what attacks are possible and their effects on the system. We have introduced a general methodology for building secure systems[8] and we use this methodology to guide the course. A few running examples are used at different stages to illustrate how they result in secure systems. We start by defining a context for a system in the style of Bishop[20]. For example, a financial institution is affected by government regulations, such as Sarbanes Oxley[17], and because it handles money it can be the target of a variety of attacks. Then we consider what are the specific effects of attacks on the confidentiality and integrity of the system. For example, in a financial system an impostor opening an account for a customer could get access to his private information (a confidentiality violation). Use cases are also a reference for defining the roles of the actors interacting with the system[21]. The rights for these roles can be obtained from the use cases, based on the activities that these roles need to perform (application of a least privilege policy).

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 [5]) and we study the security properties of each layer as well as of the whole system. Figure 1 shows a typical layer structure and possible attacks. 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. The access rights can be described as authorization patterns in the conceptual application model (see Section 4), and they are enforced through the lower levels. 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.

Applying such a methodology requires catalogs of security patterns[18]. We have developed patterns for authorization models[22], for operating systems security[23], for authentication[24], for firewalls[25], and for other security mechanisms. Some of these are being collected in a book [19].

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. Using security patterns

An example illustrates the use of patterns to explain security models. We use three patterns from one of our papers[22] to introduce access control models. The most basic access control model is the access matrix, which describes rules (Authorization rules) that specify who is entitled to access what and in what way[10, 11]. Figure 2 shows a basic pattern that describes authorization rules in an access matrix model. The Subject indicates an active entity (e.g. a user or process) that is authorized to access a Protection Object (e.g., a data item, or an I/O device) in a specific way, defined by a Right (e.g., read, write, or object-oriented method). The Right indicates the type of access allowed and can include a predicate or condition constraining access, and a copy flag indicating whether the right can be copied (delegated) to other users. This basic pattern is then combined with other concepts to describe more complex models.

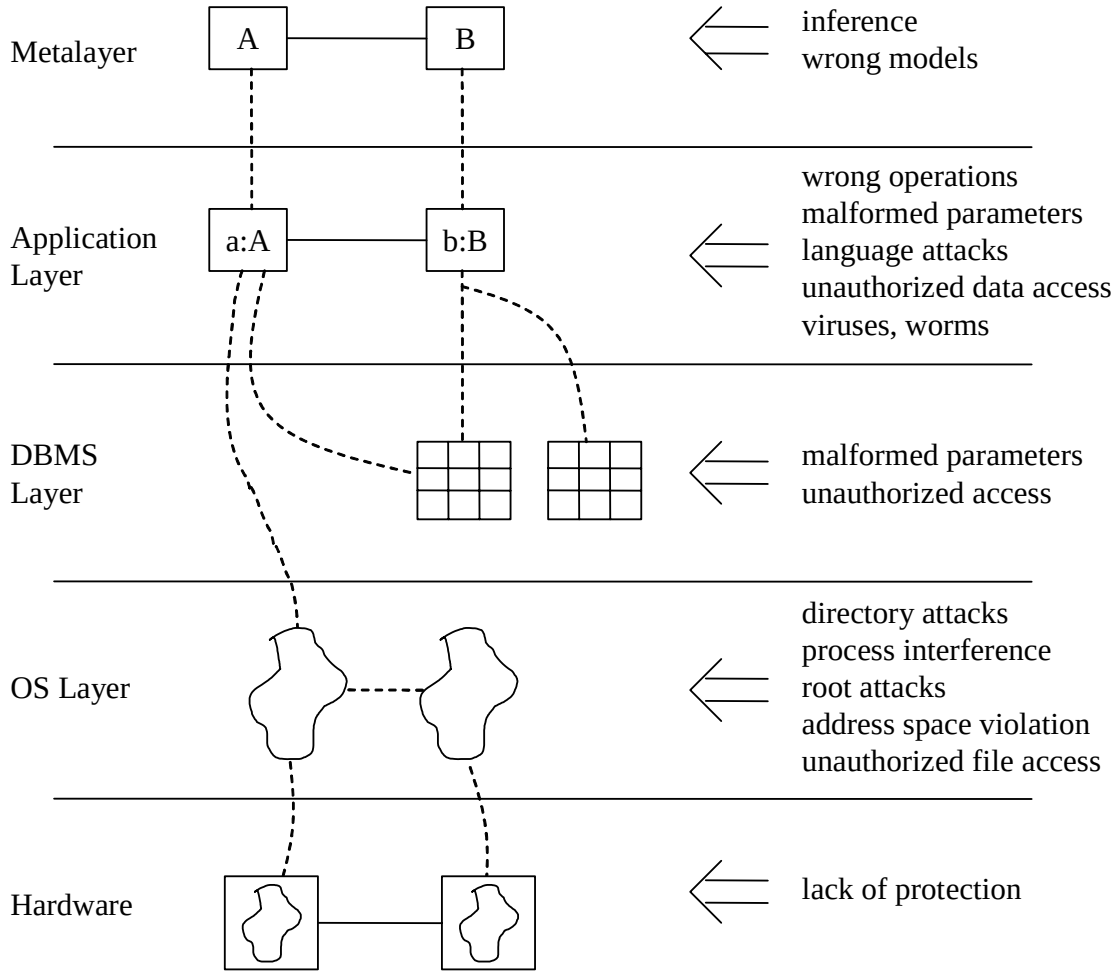

Figure 1. Typical layers of a system.

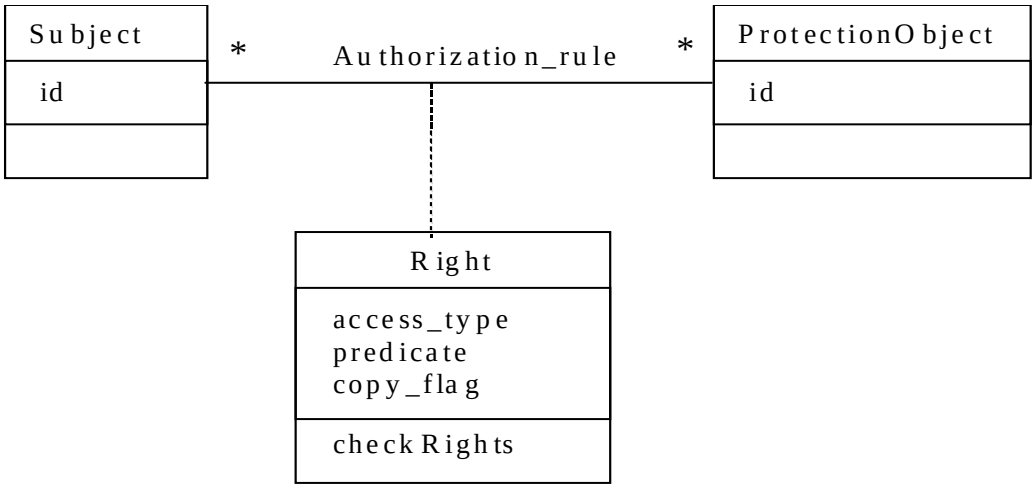

Figure 2. Class diagram for the Authorization pattern

The most common model used in current systems is Role-Based Access Control (RBAC), where rights are assigned to roles that correspond to job functions or tasks. This model can be explained as an extension of the previous model. Figure 3 shows a pattern for RBAC where the Authorization pattern is used to indicate the access rights of a role instead of a general subject. The user is now a member of one or more roles. The copy flag does not appear in this model because roles normally are not allowed to copy (delegate) rights to other roles.

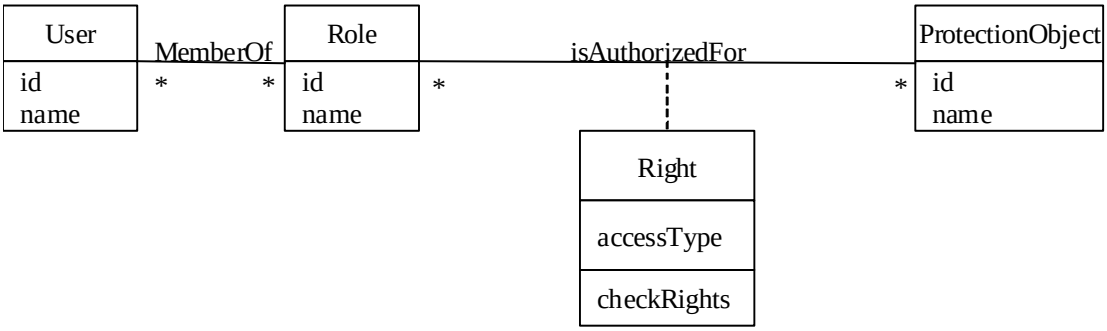

Figure 3. Class diagram for the Role-Based Access Control pattern

Finally, Figure 4 adds the concept of Session as a context for using a role. The subset constraint indicates that the roles used in a session are a subset of all the roles for which the user is authorized. This model also adds role hierarchies described by a Composite pattern[6], and separates administrative roles from operational roles. Administrators are given special rights such as ‘add users to roles’, and ‘add rights to roles’. Users can form groups to decrease the number of role registrations.

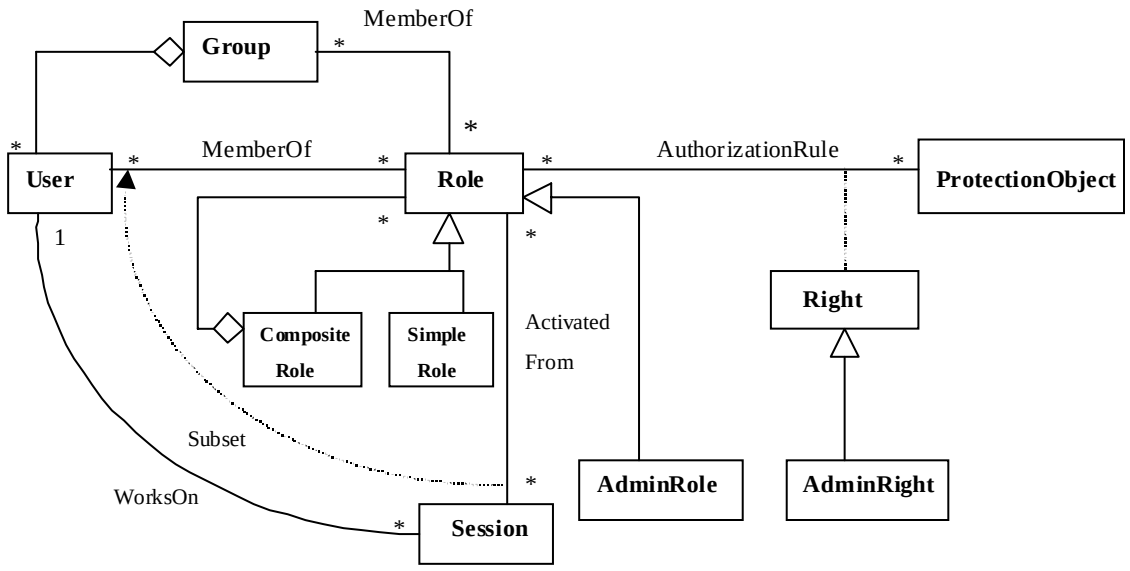

Figure 4. Pattern for Role-Based access Control with sessions and role hierarchy

This sequence of models makes clear for example, that RBAC is an extension of the Access Matrix model where collecting users into roles can decrease the number of subjects to be considered and thus simplifies the task of the security administrator. It also shows that RBAC can be what is called a 'mandatory' model[26], where subjects cannot copy their rights to other subjects and are further restricted in their actions, as opposed to a 'discretionary' model[26]. As indicated earlier, a clear advantage of UML models is that they can be easily converted to software. For example, students can build Java or C++ simulations of these models converting directly the classes in the models into classes in the programming language.

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[12]. We believe this approach provides an appropriate level of precision while preserving an engineering approach for security concepts. Future directions include adding a Business Flow level and incorporating security in existing models that are used to specify business processes and flow.

Acknowledgements

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

References

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

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

[3] Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modeling Language Reference Manual*. Boston, MA: Addison-Wesley, 1999.

[4] Warmer, J. and Kleppe, A. *The Object Constraint Language* (2nd Ed.). Reading, MA: Addison-Wesley, 2003.

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

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

[7] Larman, C. *Applying UML and Pattern (3rd ed.).* Upper Saddle River, NJ: Prentice Hall PTR, 2004.

[8] Fernandez, E. B. "A methodology for secure software design". *Proceedings of the 2004 International. Symposium on Web Services and Applications (ISWS'04),* Las Vegas, NV, June 21-24, 2004.

[9] Bishop, M. "Computer security education: Training, scholarship, and research". *IEEE Computer*, Vol. 35, No. 4, April 2002, 30-32.

[10] Anderson, R. *Security Engineering*. New York, NY: John Wiley & Sons Ltd, 2000.

[11] Bishop, M. *Computer Security: Art and science*. Reading, MA: Addison-Wesley, 2003.

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

[13] Kendall, E.A. "Utilizing patterns and pattern languages in education". *Annals of Software Eng*ineering, Vol. 6, Issue 1-4. Red Bank, NJ: J. C. Baltzer AG, Science Pub, (April 1999), 281-294.

[14] Landwehr, C. E. "Formal Models for Computer Security". *ACM Computer Surveys,* Vol. 13, No. 3, September 1981), New York, NY: ACM Press, 1981, 247-278.

[15] Fernandez, E. B., France, R. B., and Wei, D. "A formal specification of an authorization model for object-oriented databases". *Proceedings of the 9th IFIP WG11.3 Conference on Database Security*, Rensselaerville, NY , August 13-16, 1995.

[16] HIPAA. http://www.hipaa.org/

[17] OpenPages, "Sarbanes Oxley Express, SOX 404: An Internal Controls Documentation Module", July 2003, http://www.openpages.com/solutions/openbooks/404.asp

[18] Fernandez, E.B., and Yuan, X. "Semantic analysis patterns", *Proceedings of 19th International Conference on Conceptual Modeling, ER2000*, 2000, 183-195. Also available from: http://www.cse.fau.edu/~ed/SAPpaper2.pdf

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

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

[21] Fernandez, E. B. and Hawkins, J. C. "Determining Role Rights from Use Cases". *Proceedings of the 2nd ACM Workshop on Role-Based Access Control*, November 1997, 121-125. http://www.cse.fau.edu/~ed/RBAC.pdf

[22] Fernandez, E. B., and Pan, R. "A Pattern Language for security models". *Proceedings of PLoP 2001*, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions

[23] Fernandez, E. B. "Patterns for operating systems access control". *Proceedings of PLoP 2002*, 2002. http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

[24] Fernandez, E. B. and Warrier, R. "Remote Authenticator/Authorizer", *Proceedings of PLoP 2003*, 2003. http://hillside.net/patterns/

[25] Delessy-Gassant, N., Fernandez, E. B., Rajput, S., and Larrondo-Petrie, M. M. "Patterns for application firewalls", *Proceedings. of the Pattern Languages of Programs Conference*, 2004, http://hillside.net/patterns

[26] Pfleeger, C. P. *Security in computing*, 3rd. Edition, Prentice-Hall, 2003.

Appendix: Outline of the graduate course on data and network security.

1. **Introduction:** motivation, definitions, attacks, defenses, the Internet, a methodology for designing secure systems, resources. Viruses, worms, denial of service, attackers.

2. **Security Policies and Models:** Institution policies, System security policies (closed vs. open systems, ownership vs. administration, centralized vs. decentralized,…). Security models: access-matrix, multilevel models (Bell-Lapadula and Biba models). Models for information flow. Clark-Wilson, Role-based models.
3. **Cryptography.** Classical ciphers. Symmetric ciphers (DES and its variants, AES, Asymmetric encryption (Public key systems, Digital signatures and hashing functions) Encryption protocols.
4. **Security in Hardware and Operating systems:** System architecture, Process and resource protection (modes, rings), Memory protection, Address space structure, File protection. User and system authentication. Commercial operating systems (Unix, Windows, Linux), Weaknesses in commercial operating systems. How an operating system is attacked. Secure (trusted) operating systems (Virtual Vault, Pitbull, Trusted Solaris).
5. **Program and application security:** Malicious software: Trojan horses, Viruses, and worms, the buffer overflow problem. Protection in Java, client side (Sandbox model and the security manager), Components (J2EE and JAAS, .NET). Copyright protection.
6. **Security in database systems:** . Basic architecture and concepts of database management systems (DBMSs), security in Relational and SQL-based databases (Ingres, Oracle). Roles and group models in DBMS and SQL-99. Security in object-oriented databases. The inference problem, security in statistical databases.
7. **Network Security:** Networks, the OSI and Internet layers. Network attacks (Port scanning, SYN flooding, DDoS). Firewalls: packet filters and proxy-based firewalls. Secure layers: HTTPS, SSL, IPSec, SOAP security. Secure applications: secure shell, secure mail systems: PGP, SMIME. Virtual private networks. Wireless-device security. Intrusion detection (IDS).

8. **Internet and distributed system security:** Web security, Servers and their protection, authorization and authentication, cookies. HTTP server and application server security (WebSphere, Sun ONE, IIS, Apache). XML security: transmission and storage security. Web services (UDDI, WSDL, ebXML). Web privacy. Distributed systems: CORBA, >NET Remoting. Portals and LDAP. Security administration
9. **Developing secure software:** The development of secure systems: traditional and modern approaches. The software engineering cycle. The importance of development methods. Formal and semiformal design methods, using patterns and UML. Formal verification and proving correctness. Evaluating security

Biographic Information

EDUARDO B. FERNANDEZ (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.

MARIA M. LARRONDO PETRIE: Dr. Petrie is Associate Dean of Engineering and Professor of Computer Science & Engineering at Florida Atlantic University, and a member of the Secure Systems Research Group at FAU. She serves on the ASEE Minority Division Board, is Vice President of Research of the Latin American and Caribbean Consortium of Engineering Institutions, was on the ACM SIGGRAPH Education Board and was President of Upsilon Pi Epsilon Honor Society for the Computing Sciences.