A methodology for secure software design

Eduardo B. Fernandez
Dept. of Computer Science and Eng.
Florida Atlantic University
Boca Raton, FL 33431
ed@cse.fau.edu

1. Introduction

A good percentage of the software deployed in industrial/commercial applications is of poor quality and contains numerous flaws that can be exploited by attackers. There are many reasons for this and there is no doubt that we have a serious problem, 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. The products of some companies are famous for their lack of security. Until recently the only vendors' response was to provide patches to fix the latest vulnerability found or to blame the users for their lack of caution. However, patches are clearly not a 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 this situation: 1) Examine final production code and look for possible problems, e.g., buffer overflow conditions [13]. 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 life cycle. Part of the problem is that developers are not, in general, acquainted with security methods. We see the use of patterns as a fundamental way to implicitly apply security principles even by people who have little experience.

We present here a methodology to build secure software. This approach makes use of basic principles of security and object-oriented development. We consider object-oriented design, the Unified Modeling Language (UML), and patterns [1,12] as essential in the creation of well-designed software. The UML is a notation for visually modeling object-oriented systems and is now one of the most the accepted standards for software development. Another important development in software is the concept of *pattern*. A pattern embodies the knowledge and experience of software developers and can be reused in new applications. A pattern solves a specific problem in a given context and can be tailored to fit different situations. Analysis patterns can be used to build conceptual models [5] and security patterns can be used to build secure systems [8]. Patterns can embody the classical good security principles [18].

The next section discusses our approach to build secure systems, while Section 3 considers each aspect in some detail. The last section shows some conclusions.

2. The development of secure software

As indicated earlier, the way software is developed (software process) and the specific methodology used are very important to produce secure systems. First of all, the development methodology is important. Both [16] and [17], emphasize the value of information hiding and encapsulation. This indicates that the object-oriented approach is to be preferred over a procedural approach. As we shall see, there are other advantages of the object-oriented methodology that are important for security.

The main idea in the proposed methodology is that security principles should be applied at every development stage and that each stage can be tested for compliance with those principles. We sketch first a secure software development cycle that we consider necessary to build secure systems and then we discuss each stage in detail. Figure 1 shows a secure software lifecycle, indicating where security can be applied and where we can audit for compliance with security policies:

Requirements stage: Use cases define the required interactions with the system. From the use cases we can determine the needed rights for each actor and thus apply a need-to-know policy [3]. Since actors may correspond to roles, this can be interpreted as a Role-Based Access Control (RBAC) model. Note that the set of all use cases defines all the uses of the system and from all the use cases we can determine all the rights for each role. We can then consider possible attacks in the context of these use cases.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way [5]. We can build a conceptual model where repeated applications of the Authorization pattern (see below) 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. This makes the job of defining rights even easier.

Design stage: User interfaces should correspond to use cases. Interfaces can be secured applying again the Authorization pattern. Secure interfaces enforce authorizations when users interact with the system. Finally, components can be secured by using JAAS rules defined according to the authorization rules for Java components or using .NET authorizations for .NET components. Deployment diagrams can define secure configurations to be used by security administrators. A multilayer architecture is needed now to enforce the security constraints defined at the application level [4]. In each level we use patterns to represent appropriate security mechanisms.

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

At the end of each stage we can perform audits to verify that the institution policies are being followed. If necessary, the security constraints can be made more precise by using (Object Constraint Language (OCL) instead of textual constraints. Patterns for security models define the highest level. At each lower level we apply the model patterns to specific mechanisms that enforce these models. In this way we can define patterns for file systems, for web documents, for J2EE components, etc. We can also evaluate of a new or existing system using patterns. Patterns help understand the security structure of each component to allow their composition and define secure interfaces. If a system doesn't contain an appropriate pattern then it cannot support the corresponding secure model or mechanism.

We can combine different types of patterns to get different functionality and quality. For example, [14] combines RBAC, filter, and connection patterns (The last two are patterns for distributed systems). We analyze these stages in detail in the next section.

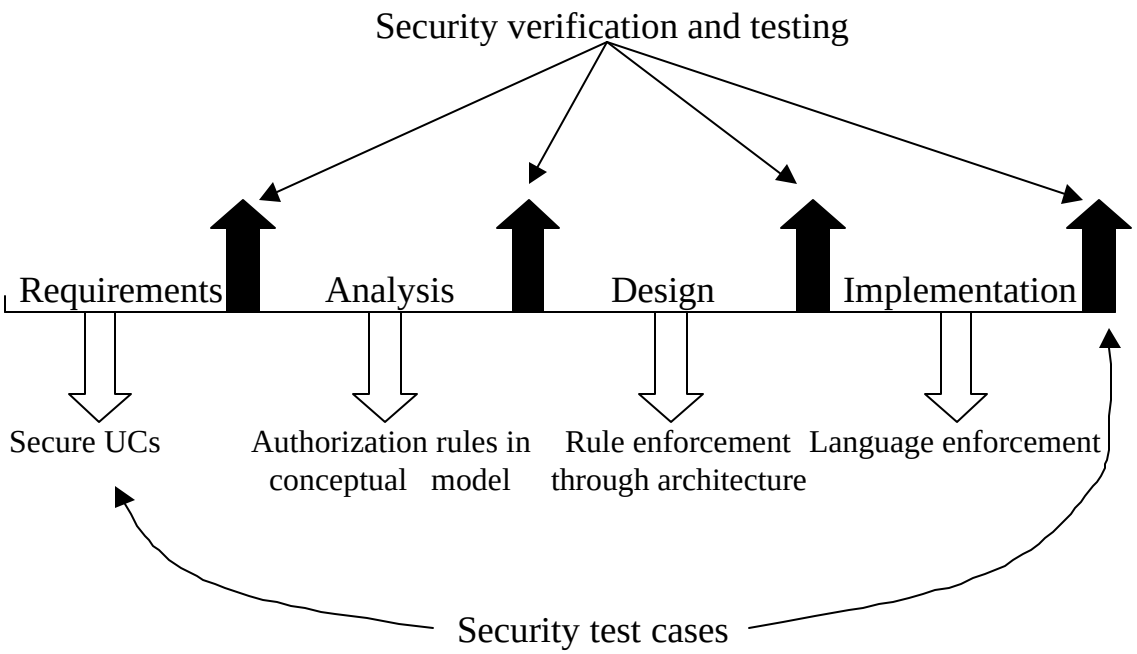

Figure 1. Secure software lifecycle

3. Details of some stages

3.1 Architectural levels

Experience has shown that a good way to build dependable systems is to structure them into a set of hierarchical layers. Layers of abstraction enforce downward-only functional dependencies. Unit A is said to *depend* on B whenever an action of B, or a change to B, or total unavailability of B, can affect A [15].

Some important issues for hierarchies are: what functions to include in each layer and how much security we need in each layer to accomplish an intended level of security for the whole system. Not all layers need be equally secure, the lower levels are more critical and need stronger protection. Security and other non-functional requirements affect all the architectural levels of a system. The Layers architectural pattern [1] is therefore a good starting point to apply these requirements. Using layers we can define patterns at all levels that together implement a secure or reliable system. The main idea of the Layers pattern is the decomposition of a system into hierarchical layers of abstraction, where the higher levels use the services of the lower levels. We have discussed earlier, why all these levels must be coordinated to assure security [4] and how the definition of non-functional specifications should be done at a specific level [2].

The conceptual enterprise models, both static and dynamic, are defined at the application level. It is here where the security (and other type) policies of the institution should be applied. At this level the semantics of the application are well understood and roles can be used to apply the need-to-know policy; i.e., we can define the needed rights according to the functions of each role [3]. Other non-functional aspects are also specified here, e.g., the required degree of reliability. The lower levels enforce the restrictions defined at the higher levels. Each level has its own security mechanism and should participate in enforcing the security constraints. For example, a DBMS

enforces the authorizations in the application by restricting access to database items; this restriction is propagated down to control access to the files where this data resides.

3.2 Use cases and possible attacks

As we indicated earlier since use cases define all the interactions with the system we can find from them the rights needed by these roles to perform their work (need to know). Figure 2 shows the use cases for a voting system that allows voting in the precinct, in a precinct that is not your own precinct, and through the Internet. A voter has the right to register and to vote, the precinct officer keeps list of registered voters and tallies the votes.

We can then relate possible attacks to use cases.. For example, a possible attack against voter registration corresponds to an impostor trying to get registered with false identity or attributes. An attack against remote voting would be an attempt to send an invalid vote or to intercept a vote with the purpose of changing it.

Relating attacks to use cases provides a systematic and relatively complete list of possible attacks. Each attack can be analyzed to see how it can be accomplished in the specific environment. The list can then be used to guide the design and to select security products. It can also be used to evaluate the final design by analyzing if the system defenses can stop all these attacks.

3.3 Authorized applications

We use the access matrix and RBAC as reference models. Multilevel models are also possible but when used at the application level they are too rigid; however, they are useful at lower levels. When we apply the access matrix model, the next step is to define patterns that represent authorization rules or policies, as shown in Figure 3 [6]. This model describes an entry of the access matrix, (s, o, t, p, f), where s is a subject, o is a protection object, t an access type, p a predicate constraining the application of the rule , and f a copy flag, indicating if the right can be transferred [17]. The classes of Figure 3 are abstract classes and specific authorization models are defined by concrete classes. Conceptual or domain models of systems can be built using analysis patterns [5] and we have developed a collection of these patterns for aspects such as inventories, order processing, and others. These security patterns can be applied to analysis patterns to define semantic subsystems that combine the advantages of patterns with the advantages of high-level authorization definition. In this case, the user of the pattern would have a structure to define the specific rights his application requires. For example, in [8] we showed an analysis pattern for a secure inventory system. In that model, the auditor is authorized to check for discrepancies in stock, while the stock keeper is authorized to correct or adjust these discrepancies. Similarly, the stock manager can add new stockrooms, etc. The specific rights for each role come from use cases and are derived as in [3]. Each use case has a set of actors who interact with the system. If actors are given rights according to their functions in the use cases of the system, we are implementing a need-to-know policy. Starting from patterns at the application level we need to define patterns for the lower levels. For example, we have developed patterns for operating systems [7, 9], firewalls [10], and other security mechanisms. For systems that use web services we have developed security patterns for application firewalls and assertion coordination [11].

3.4 Secure system architecture

Figure 4 shows some of the secure mechanisms that can help stop the attacks defined through the use cases. For example, remote users would require certificates for authentication, the voting machine hardware and software needs to be certified for security, the precints are connected through a VPN, etc.

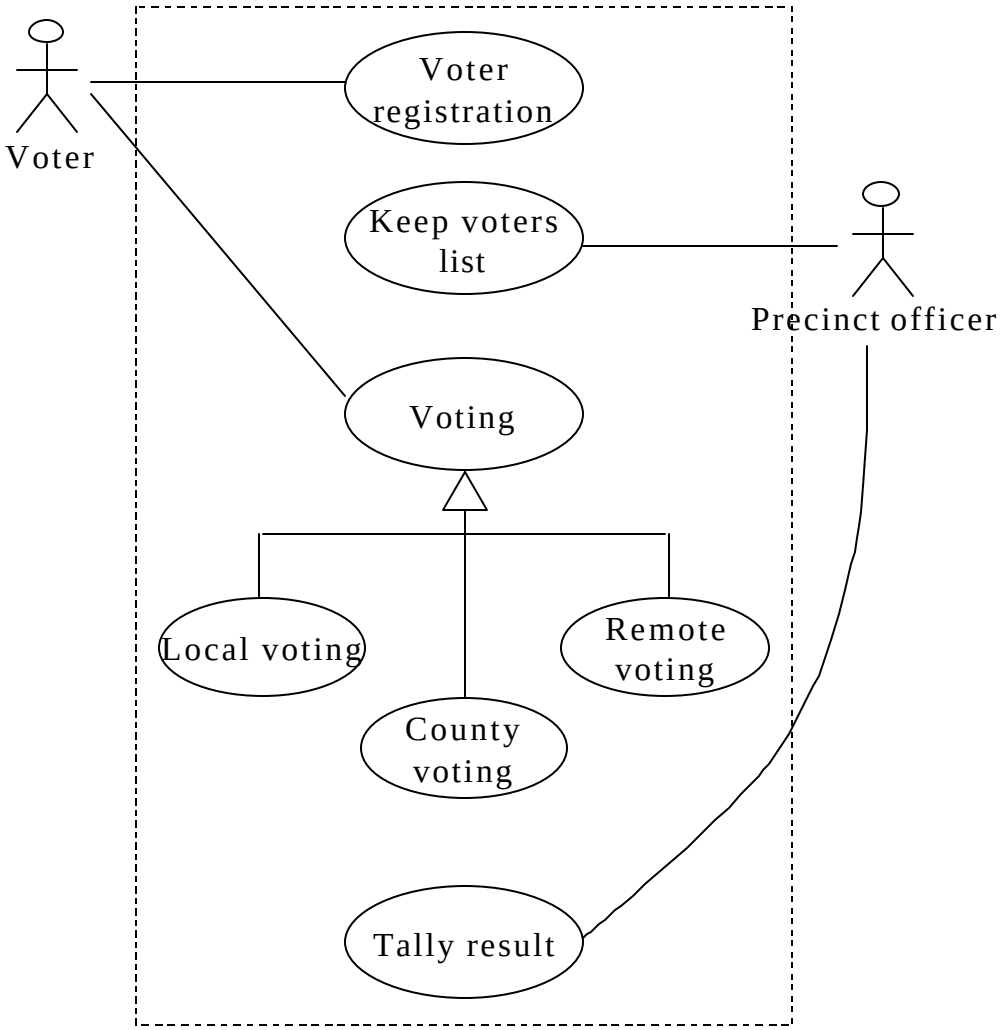

Figure 2. Use cases for a voting system.

4. Conclusions

The combination of multilayer architectures with patterns provides a framework to develop a systematic and reusable approach to building systems that satisfy specific non-functional requirements. Security patterns embody good design principles and by using them, the designer is implicitly applying these principles. Work is needed to add more patterns in each level and to collect and unify these patterns. We are working now in a catalog of security patterns [19]. We also need to define guidelines to apply the methodology more precisely at each level. Finally, we need to evaluate this methodology in a real environment; for now we are applying it to specific examples, such as distributed medical records, Internet voting, and distributed financial institutions.

5. References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal., *Pattern-oriented software architecture*, Wiley 1996.

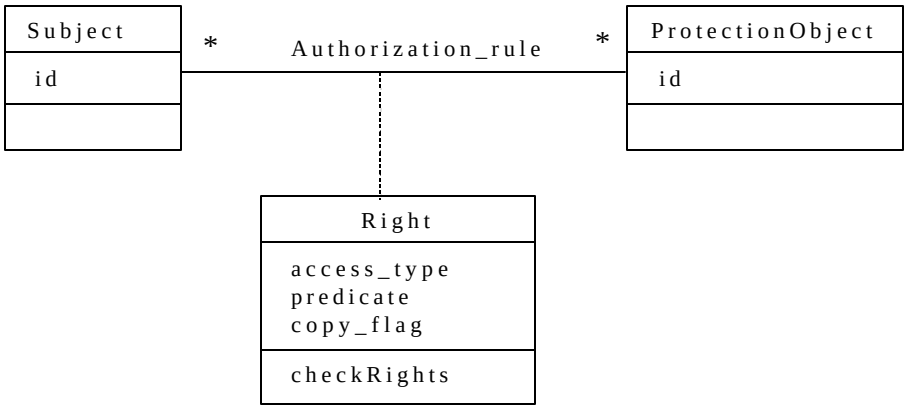

Figure 3. A pattern for authorization rules.

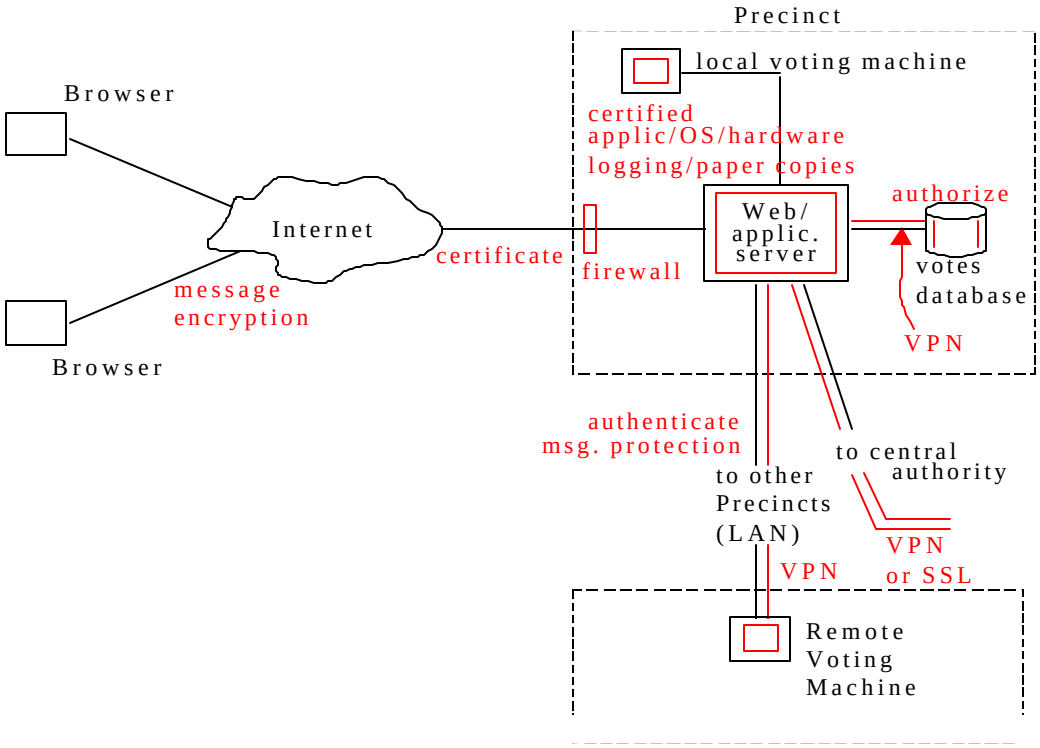

Figure 4. A secure system architecture

[2] E. B. Fernandez and R. B. France, ``Formal specification of real-time dependable systems '', *Proc. of First IEEE Int. Conf. on Eng. of Complex Comp. Systems*, Fort Lauderdale, FL, November 6-10, 1995, 342-348.

[3] E.B.Fernandez and J.C.Hawkins, "Determining role rights from use cases", *Procs. 2nd ACM Workshop on Role-Based Access Control*, November 1997, 121-125.

http://www.cse.fau.edu/~ed/RBAC.pdf

[4] E.B.Fernandez, "Coordination of security levels for Internet architectures", *Procs. 10th Intl. Workshop on Database and Expert Systems Applications*, 1999, 837-841. http://www.cse.fau.edu/~ed/Coordinationsecurity4.pdf

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

[6] E B. Fernandez and R.Y. Pan, "A pattern language for security models", *Procs. of PLoP 2001*, http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/accepted-papers.html

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

[8] E.B.Fernandez, "Layers and non-functional patterns", *Procs of ChiliPLoP, 2003.* Phoenix, March 10-15, 2003. http://hillside.net/chiliplop/2003/

[9] E.B.Fernandez and J.C.Sinibaldi, "More patterns for operating systems access control", *Procs. EuroPLoP 2003*, http://hillside.net/europlop

[10] E. B. Fernandez, M. M. Larrondo-Petrie, N. Seliya, N. Delessy, and A. Herzberg, "A Pattern Language for Firewalls", *Procs. of PLoP 2003.*

[11] E.B. Fernandez, "Two patterns for web services security", to appear in Procs. of the International Symposium on Web Services and Applications, Las Vegas, NV, June 2004.

[12] E. Gamma, R. Helm,R. Johnson, and J. Vlissides, *Design patterns –Elements of reusable object-oriented software*, Addison-Wesley 1995.

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

[14] V. Hays, M. Loutrel, and E.B.Fernandez, "The Object Filter and Access Control framework", *Procs. Pattern Languages of Programs (PLoP2000) Conference*, http://jerry.cs.uiuc.edu/~plop/plop2k

[15] P.G.Neumann, "On hierarchical design of computer systems for critical applications", *IEEE Trans. on Software Eng.*, vol. SE-12, No 9, September 1986, 905-920.

[16] P.G.Neumann, "The role of software engineering", *Comm. of the ACM*, Vol. 36, No 5, May 1993, 114.

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

[18] J.H.Saltzer and M.D.Schroeder, "The protection of information in computer systems", *Procs. of the IEEE*, Vol. 63, No 9, 1975, 1278-1308. A web version is in: http://web.mit.edu/Saltzer/www/publications/protection/index.html

[19] M. Schumacher, E.B.Fernandez, D. Hybertson, and F. Buschmann, *Security Patterns*, to be published by J. Wiley & Sons, 2004.