An Extended Reference Monitor for Security and Safety

Eduardo B. Fernandez[1], Michael VanHilst[1], David laRed Martinez[2] and Sergio Mujica[3]

[1] Department of Computer Science and Engineering, Florida Atlantic University
777 Glades Road
Boca Raton, Florida 33431 USA
[2] Departamento de Informática, Facultad de Ciencias Exactas y Naturales y Agrimensura - Universidad Nacional del Nordeste
Avenida Libertad 5460
3400 Corrientes, Argentina
[3] Facultad de Ingenieria y Ciencias, Universidad Adolfo Ibanez
Diagonal Las Torres 2640
Peñalolén - Santiago, Chile
{ed, mike}@cse.fau.edu, lrmdavid@exa.unne.edu.ar, sergio.mujica@udp.cl

Abstract. We present here a unified way to handle the combined enforcement of safety and security through the use of patterns, in particular through the concept of Reference Monitor. A Reference Monitor is an abstract mechanism that applies authorization rules to decide access. We first define an Extended Reference Monitor with reified decisions and then apply the idea to a dual monitor that can handle security and safety.

Keywords: Reference monitor, security, authorization, safety.

1 Introduction

Although it is a non-functional aspect, security is a fundamental objective of any system where there are assets that may be targets of attacks because of their economic value or potential for disruptive impact [1]. Security is the protection against intentional attacks and until recently most studies of security have focused on the protection of information assets based on their monetary and business value. Recent terrorist threats have brought interest to the protection of physical assets as well. Correspondingly, the studies of threats to infrastructure have focused on safety. Safety is the freedom from unacceptable risks, including threats to human lives, the environment, or to costly facilities. Safety is often expressed with assertions on how to avoid unsafe conditions, e.g. an elevator must not open its doors when moving. These assertions are usually related to specific states of a control system. Safety is often confused with reliability or with availability. Some perceive safety as providing "zero defect" or "failure free" software, others consider it a matter of providing uninterrupted service. While reliability and availability are clearly important, they are not sufficient to provide safety. It is also not enough to apply the same security measures used in IT systems to critical systems. While superficially there are many commonalities, delving deeper one finds many differences [2]. These differences imply the need for a specialized use of mechanisms and development methodologies. However, there are commonalities and it is important to identify them to produce systems that can enforce more than one type of requirement and to save development work. This paper is an attempt in that direction.

Hazards are states or conditions of the system, that when combined with some state of the environment, lead to mishaps. A mishap is the occurrence of an unwanted event that leads to human or environment harm. Leveson categorizes hazards into three groups [3]: The system is not available. The system generates an incorrect output. The system misses a hard deadline. Security attacks can deny service, illegally modify system state, or introduce artificial delays; that is, security attacks can directly affect safety through all three hazard categories. On the other hand, an unsafe condition can cause security problems. All of these commonalities point to the need to deal with security and safety in a unified way. We need to produce systems where security and safety are developed together. Our long-term objective is the development of a complete methodology for critical systems but we are initially concentrating on some specific aspects. Safety requires reliability and methods to improve reliability are also of interest to us. Because of the changing nature of these systems, we also consider flexibility as an added requirement. Security and safety concerns have been mostly developed separately, although a few works try to combine them, e.g. [4,5,6].

The conceptual control models of safety-critical systems, both static and dynamic, are defined at the application level. It is here also where the security policies of the institution should be applied. At this level the semantics of the application are well understood. We can also apply here institutional policies and policies that correspond to regulations. Other non-functional aspects are also specified here, e.g., the required degree of reliability and safety assertions. The lower levels enforce the restrictions defined at the higher levels. Each level has its own security mechanism and should participate in enforcing the constraints [7]. 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 [8].

An important development in software is the concept of *pattern*. A pattern is a solution to a recurrent problem in a given context [9]. A pattern embodies the knowledge and experience of software developers and can be reused in new applications. Well-thought patterns also implicitly apply good design principles. The typical solution provided by a pattern comes in the form of a class diagram complemented with some sequence diagrams and possibly activity or state diagrams. This level of detail and precision allows designers to use them as guidelines and to users to understand the use of the mechanism they represent.

We present here a unified way to handle the combined enforcement of safety and security through the use of patterns, in particular through the concept of Reference Monitor. A Reference Monitor is an abstract mechanism that applies authorization rules to decide access [10]. Concrete realizations of the reference Monitor may exist at each architectural level. In order to define this Reference Monitor precisely we provide UML class and sequence diagrams to describe its structure and behavior.

Section 2 describes some background, while Section 3 presents our extensions to the idea of Reference Monitor. We apply those extensions in Section 4 to define a Reference Monitor that can also handle safety. We then discus related work and we finish with some conclusions and ideas for future work.

2 Threats and Hazards

In previous work we introduced an approach for finding security requirements based on misuse activities (actions) [11].This method starts from the activity diagram of a use case (or a sequence of use cases). Each activity is analyzed to see how it could be subverted to produce a misuse of information. This analysis results in a set of threats. We then consider which policies can stop or mitigate these threats. We recently extended that approach to consider the type of misuse (confidentiality, integrity,...) that can happen in each activity, the role of the attacker, and the context for the threat. This extended analysis results in a finer and more systematic way to find threats and we can identify now more threats [12]. We can apply this approach to control systems to identify security threats and safety hazards as well as the effect of errors.

Mishaps happen because a dangerous situation was not considered in the requirements, or because there was a failure in some unit of the system, or because there was an intentional action taken against vulnerable elements of the system. Using our systematic approach to find threats we can also find hazards. Hazards must be prevented in the design of the software for the system. In other words, safety is dependent on the correctness, reliability, and security of the system. Conversely, safety features or restrictions (e.g. emergency access) or reliability failures can lead to security breaches. However, high reliability and security do not guarantee safety.

Once misuse actions have been uncovered, appropriate security policies must be selected to stop and/or mitigate them. This activity could be aided by some comprehensive security policy list. However, this selection should result in a minimum set of mechanisms instead of mechanisms piled up because they might be useful. We discuss these issues in [11] and [12]. Through examples, we have found that with a good catalog of policies we can stop or mitigate most threats or hazards.

Policies lead to specific countermeasures. For example, authorization requirements can be enforced through the abstract concept of Reference Monitor. A Reference Monitor intercepts requests for resources (protection objects) and decides if the request is valid according to the authorization rules.

3 The Extended Reference Monitor

We proposed a pattern for a Reference Monitor in [13]. We present here a pattern for an Extended Reference Monitor that allows a more precise definition of the decision made by this mechanism. Specifically, we modify the Reference Monitor pattern to include Decision as a separate class, i.e. the decision is reified and has attributes. This addition allows manipulation of of decisions as well as defining conditions or other additional restrictions. We present only some sections of the complete pattern, following the template used in [9].

Intent: Enforce authorizations when a subject requests a protection object and provide the subject with a decision which can have conditions.

Context: A multiprocessing environment where subjects request protection objects to perform their functions and access resources maybe bsed on conditions.

Problem: Authorization can be defined according to different models, e.g. access matrix or Role-Based Access Control. If we don't enforce the defined authorizations it is the same as not having them, subjects can perform all type of illegal actions. Any user could read any file for example. How can we control the subjects' actions?

Decisions can be sometimes more complex than a Boolean response, e.g. when a user wants to access some database items, she may be authorized to access a subset of the data she requests. In this case, the decision is not Boolean (yes/no) but only some items are released. Also, many times there is a need to store decisions for possible reuse.

Intent: Enforce authorizations when a subject requests a protection object and provide the subject with a decision which can have conditions.

Context: A multiprocessing environment where subjects request protection objects to perform their functions and access resources maybe bsed on conditions.

Problem: Authorization can be defined according to different models, e.g. access matrix or Role-Based Access Control. If we don't enforce the defined authorizations it is the same as not having them, subjects can perform all type of illegal actions. Any user could read any file for example. How can we control the subjects' actions?

Decisions can be sometimes more complex than a Boolean response, e.g. when a user wants to access some database items, she may be authorized to access a subset of the data she requests. In this case, the decision is not Boolean (yes/no) but only some items are released. Also, many times there is a need to store decisions for possible reuse.

Solution: Define an abstract process that intercepts all requests for resources, checks them for compliance with authorizations, makes decisions based on these authorization rules and stores those decisions including their attributes.

Fig. 1 shows a class diagram showing a Reference Monitor with a reified decision. In this figure Set_of_Authorization_Rules denotes a collection of Authorization rules organized in a convenient way and corresponding to some security model. The Actual Subject (Client) can be a process which makes a Request for a Protection Object with the intention of applying to it some access type. The Reference Monitor makes a Decision according to the authorization rules. Examples of Concrete Reference Monitors are file permission systems, memory managers, or other resource controllers. Fig. 2 shows a sequence diagram showing how checking is performed.

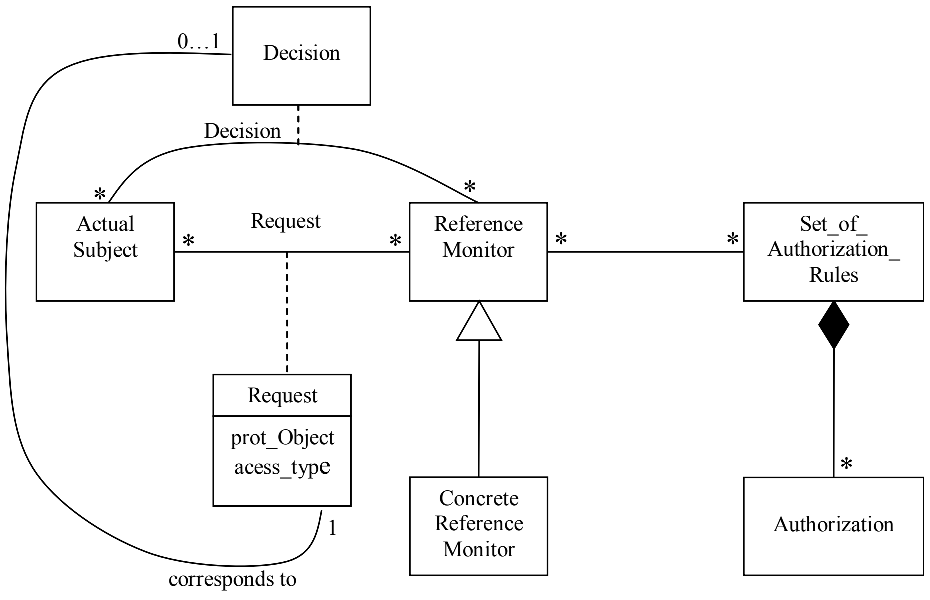

Fig. 1. Class diagram for the modified Reference Monitor.

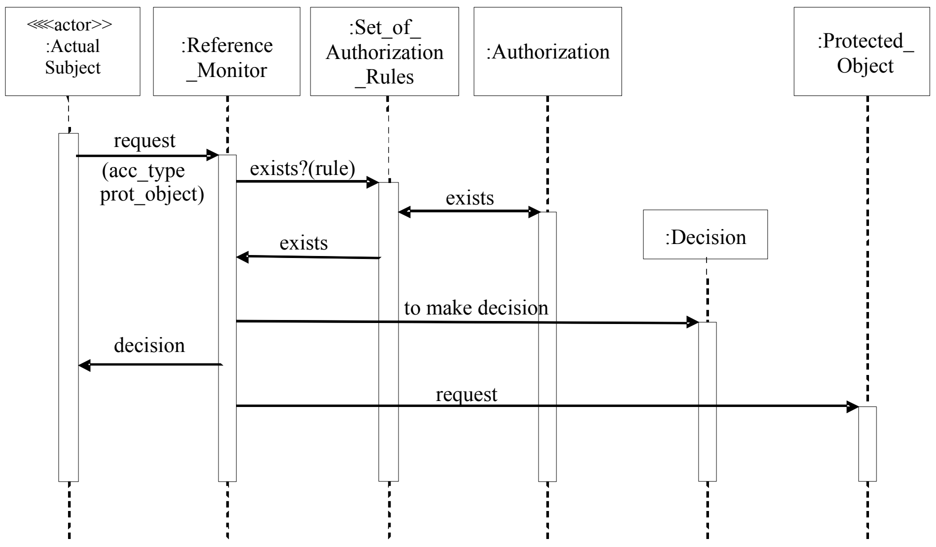

Fig. 2. Sequence diagram for enforcing security of requests.

4 Safety Reference Monitor

We see security conditions and safety assertions as similar (from the point of view of their enforcement). To enforce the constraints of the combined security/safety model we are proposing an Extended Reference Monitor which intercepts and verifies client requests for authorization and safety compliance. A UML class model for an Extended Reference Monitor is shown in Fig. 3. The Client sends requests to access an Object in a specific way (access type). The extended Reference Monitor separates the requests into security and/or safety requests. The Authorization Monitor has a set of authorization rules and enforces security constraints such as: "Only an authorized individual or role can change a valve setting". The safety Monitor enforces safety assertions: An operator may be authorized to send some command to a nuclear reactor but the conditions (state) are such that this command would produce an unsafe state, so the Safety Monitor rejects this command.

While the Authorization Monitor needs to control any types of accesses in the requests, the Safety Monitor only needs to consider requests that may change the state of the system. Both authorization rules and safety assertions need access to the state of the system in order to decide the validity of requests. We can classify different types of access constraints:

- Modal, based on current or temporal factors:
  - the current operating mode
  - access only if situation critical or non-critical
  - access only for off-line (test) operation
  - access only when resource hungry logging is active
- Schedule, based on schedule or process phase
  - only during quiet periods
  - only at scheduled time in process sequence
  - only when needed
- Discretionary, depending on the current load
  - access only when not busy relative to time constraints
  - access only when solar power is available
- Validated, involving active analysis
  - a semantic or pragmatic constraint analyzes the consequences of a request in the current context and performs checks against acceptable ranges.
- Preemptable, access could be cut off based on other control needs:
  - monitor plays a continuous and active role in fulfilling safety needs
  - access suspended while system performs schedule dependent task
  - access times out

These, and other constraints, can be modeled as conditions on the safety assertions and can be conveniently described in UML extended with OCL [14].

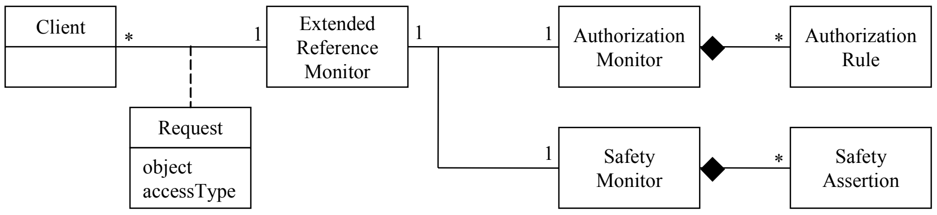

Fig. 3. Class diagram of the Extended Reference Monitor

5 Related Work

A systematic approach to safety-critical systems is proposed in [15]. However, they did not consider data aspects, they worked mostly with dynamic aspects described by statecharts. They don't consider security either. M. Tichy has studied pattern-based synthesis of fault-tolerant embedded systems [16]. His objective was to automate the application of fault tolerance into embedded systems. For this purpose, he developed a formal model based on Petri nets. In order to define the points where to apply fault tolerance mechanisms he considers the possible faults of the system. This last objective is similar to ours, but we analyze safety and security threats instead of reliability threats. L. Grunske focused on validation of safety properties: If a software architecture does not satisfy its requirements, it is modified by adding patterns that improve its safety properties [17]. His approach is based on transformational patterns that effectively perform model refactoring [18]. Y. Choi [19] converts use cases scenarios into activity diagrams (as we do in [20]), which in turn become Finite State Machines modeled using RSML-e, a formal language that has a model checker where hazard conditions can be identified.

Nonfunctional aspects such as security or safety have a strong effect on the resulting system architecture. A variety of approaches for software architecture design have been proposed [21]. However, all of them have relatively little to say about non-functional aspects, they mostly provide generic recommendations, with little detail on how to apply them. We follow mostly an approach based on RUP [22], but we will consider some of the other approaches if they have specific aspects that appear useful.

None of these studies explicitly considers the use of security patterns or provides a general methodology for secure or safe design. None of these approaches attempts to combine security and safety either. However, the work of Choi is very relevant and we are adapting some of her results to evaluate security threats. Closer in objectives are attempts for secure software development.

A general methodology for developing security-critical software which in particular can be used to develop secure DBMSs has been proposed in [23]. It makes use of an extension of the Unified Modeling Language (UML), UMLSec, to include security-relevant information. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLsec models against security requirements and has been used in a variety of industrial projects [24]. This model is clearly applicable to support our work.

Mouratidis and his group use a special methodology, Tropos, to model security. Their work started modeling requirements [25]. They have also looked at other stages; for example how to test security along the lifecycle. Instead of UML they use special diagrams and they do not use patterns. They have recently added models using UMLSec [26].

Only the last two of these approaches provide a relatively complete development methodology, the others consider only specific aspects. None of them considers safety or reliability aspects. We believe that there is a need for a complete development approach with a strong conceptual basis that combines security and safety. Our work is a step in that direction.

6 Conclusion

Many critical systems are data-driven systems. An air traffic control system that must keep track of many aircraft positions, a patient drug-delivery system where drug dosages and times are based on stored data, a building hosting a nuclear reactor where the movement of people must be restricted, are examples of this type of systems. Security is especially critical in these systems because an illegal modification of the related data could affect the safety of many people. Our Extended Reference Monitor can be the basis of an access control model that combines the specification and enforcement of security and safety for both information and physical structures [27]. We are implementing a preliminary version now. A byproduct of this work has been a new course on critical infrastructure systems [28].

A particularly interesting implementation approach is the use of components in a service-oriented architecture developed using an MDE approach [29], where the Extended Reference Monitor can be implemented using XACML. An important result of a systematic lifecycle for safety-oriented designs is a procedure for certification [30]. We intend to analyze how patterns can help for this purpose. Another aspect of future work is the integration of other security patterns [31] and fault-tolerance patterns [32]. The project started at Florida Atlantic University in the USA but it now includes participation from Argentina and Chile.

References

1. McMillan, R.: Critical Infrastructure Often Under Cyberattack. IDG News Service, November 11, 2008
2. Naedele, M.: Addressing IT Security for Critical Control Systems. In: 40th Hawaii International Conference on System Sciences, p. 115. IEEE, New York (2007).
3. N. Leveson, N., Safeware: System Safety and Computers. Addison-Wesley, Reading (1995)
4. Lautieri, S., Cooper, D., Jackson, D.: SafSec: Commonalities Between Safety and Security Assurance, In: Redmill, F., Anderson, T. (eds.) Constituents of Modern Systems Thinking:

Proceedings of the Thirteenth Safety Critical Systems Symposium, pp. 65--78. Springer, London (2005)

5. Ligatti, J., Bauer, L., Walker, D.: Edit Automata: Enforcement Mechanisms for Run-time Security Policies. International Journal of Information Security, 4(1-2) 2--16. Springer (2005)
6. Littlewood, B., Strigini,L.: Redundancy and Diversity in Security. In: 9th European Symposium On Research in Computer Security, pp. 42--428. Springer (2004).
7. Fernandez, E.B., France, R.B.: Formal Specification of Real-Time Dependable Systems. In: 1st IEEE International Conference on Engineering of Complex Computer Systems, pp. 342--348. IEEE (1995)
8. Fernandez, E.B.: Coordination of Security Levels for Internet Architectures. In: 10th International Workshop on Database and Expert Systems Applications, pp. 837--841. IEEE (1999)
9. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture Volume 1: A System of Patterns. Wiley, West Sussex (1996).
10. Gollmann, D.: Computer Security, 2nd Edition. Wiley, West Sussex (2006).
11. Fernandez, E.B., VanHilst, M., Larrondo-Petrie, M.M., Huang, S,: Defining Security Requirements through Misuse Actions, In: Ochoa, S.F., Roman, G.-C. (eds.) Advanced Software Engineering: Expanding the Frontiers of Software Technology, pp. 123--137. Springer (2006)
12. Braz, F.A., Fernandez, E.B., VanHilst, M.: Eliciting Security Requirements Through Misuse Activities. In: 19th International Workshop on Database and Expert Systems Applications, pp. 328--333. IEEE (2008)
13. Fernandez, E.B.: Patterns for Operating Systems Access Control. In: 9th Pattern Languages of Programs Conference. Hillside Group (2002)
14. Warmer J.B., Kleppe, A.G.: The Object Constraint Language: Getting Your Models Ready for MDA, 2nd Edition, Addison-Wesley, Boston (2003)
15. Leveson, N.G., Heimdahl, M.P.E., Hildreth, H., Reese, J.D.: Requirements Specification for Process-Control Systems, IEEE Trans. on Soft. Eng., 20(9), 684--707, (1994)
16. Tichy, M.: Pattern-Based Synthesis of Fault-Tolerant Embedded Systems. In: Doctoral Symposium of the Fourteenth ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 1--18. ACM, New York (2006)
17. Grunske, L.: Transformational Patterns for the Improvement of Safety Properties in Architectural Specification. In: 2nd Nordic Conference on Pattern Languages of Programs. Hillside Group (2003)
18. Fowler, M.: Refactoring: Improving the Design of Existing Code, Addison-Wesley, Upper Saddle River (1999)
19. Choi, Y.: Early Safety Analysis: from Use Cases to - Component-based Software Development. Journal of Object Technology. 6(8), 185--203 (2007)
20. Fernandez, E.B., Larrondo-Petrie, M.M., Sorgente, T., VanHilst, M.: A Methodology to Develop Secure Systems Using Patterns, Chapter 5. In: Mouratidis, H., Giorgini, P. (eds.) Integrating Security and Software Engineering: Advances and Future Vision, pp. 107-126. IDEA Group, Hershey (2006)
21. Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P.: A General Model of Software Architecture Design Derived from Five Industrial Approaches. Journal of Systems and Software, 80(1), pp. 106--126 (2007)
22. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley, Upper Saddle River (1999)
23. Jurjens, J.: Secure Systems Development with UML. Springer, Berlin (2004).
24. Best, B., Jurjens, J., Nuseibeh, B.: Model-based Security Engineering of Distributed Information Systems Using UMLsec. In: 29th International Conference on Software Engineering (ICSE 2007), pp. 581--590. ACM, New York (2007)

25. Mouratidis, H., Giorgini, P., Manson, G.: Integrating Security and Systems Engineering: Towards the Modeling of Secure Information Systems. In: 15th Conference on Advanced Information Systems 2003. LNCS, vol. 2681, pp. 63--78. Springer, Heidelberg (2003)
26. Mouratidis, H., Jürjens, J., Fox, J.: Towards a Comprehensive Framework for Secure Systems Development. In: 18th Conference on Advanced Information Systems 2006. LNCS, vol. 4001, pp. 48--62. Springer, Heidelberg (2006)
27. Fernandez, E.B., Ballesteros, J., Desouza-Doucet, A.C., Larrondo-Petrie, M.M.: Security Patterns for Physical Access Control Systems. In: S. Barker, S., Ahn, G.J. (eds.), Data and Applications Security XXI, LNCS, vol. 4602, pp. 259--274, Springer (2007)
28. Fernandez E.B., Larrondo-Petrie, M.M.: A Course on Security for Critical Infrastructure Systems. In: 6th Latin American and Caribbean Conference for Engineering and Technology, (2008)
29. Delessy N., Fernandez, E.B.: A Pattern-Driven Security Process for SOA Applications. In: 3rd International Conference on Availability, Reliability, and Security, pp. 416--421. IEEE (2008)
30. Heimdahl, M.: Safety and Software Intensive Systems: Challenges Old and New. In: International Conference on Software Engineering 2007: Future of Software Engineering, pp. 137--152. ACM Press (2007).
31. Fernandez, E.B.: Security Patterns and a Methodology to Apply Them. In: Spanoudakis, G., Maña, A. (eds.) Security and Dependability for Ambient Intelligence. Springer (2009)
32. Buckley, I., Fernandez, E.B.: A Survey of Fault Tolerance Patterns. Department of Computer Science and Engineering, Florida Atlantic University (2008)