Incorporating Database Systems into a Secure Software Development Methodology

Conference Paper · October 2008

DOI: 10.1109/DEXA.2008.100 · Source: IEEE Xplore

4 authors, including:

Eduardo B. Fernández
Florida Atlantic University
319 PUBLICATIONS **3,752** CITATIONS

SEE PROFILE

Nobukazu Yoshioka
National Institute of Informatics
144 PUBLICATIONS **825** CITATIONS

SEE PROFILE

Hironori Washizaki
Waseda University
280 PUBLICATIONS **1,346** CITATIONS

SEE PROFILE

Incorporating database systems into a secure software development methodology

Eduardo B. Fernandez[1], Jan Jurjens[2], Nobukazu Yoshioka[3], and Hironori Washizaki[4]
[1]*Dept. of Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA*
[2]*Computing Department, The Open University, Milton Keynes, MK7 8LA GB*
[3]*GRACE Center, National Institute of Informatics, 2-1-2*
Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
[4]*Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo, 169-8555, Japan*
ed@cse.fau.edu, j.jurjens@open.ac.uk, nobukazu@nii.ac.jp, washizaki@waseda.jp

Abstract

We have proposed in the past three separate methodologies for secure software development. We have found that they have many common and complementary aspects and we proposed a combination of them that appears as a good approach to secure software development. The combined methodology applies security at all stages, considers the architectural levels of the system, applies security policies through the use of patterns, and formalizes some portions of the design. We have studied in some detail how to elicit and describe security requirements, how to reflect these requirements in the conceptual model, how to estimate some performance aspects, how to formalize some aspects such as communication protocols, and how to map the conceptual requirements into design artifacts. A design aspect which we have not studied is the incorporation of databases as part of the secure architecture. The database system is a fundamental aspect for security because it stores the persistent information, which constitutes most of the information assets of the institution. We present here some ideas on how to make sure that the database system has the same level of security than the rest of the secure application.

1. Introduction

We have proposed in the past three separate methodologies [6, 10, 17]. We have found that they have many common and complementary aspects and we proposed a combination of them that appears as a good approach to secure software development [7]. We will refer to the combined approach as FJY. The combined methodology applies security at all stages, considers the architectural levels of the system, applies security policies through the use of patterns, and formalizes some portions of the design. We have studied in some detail how to elicit and describe security requirements, how to reflect these requirements in the conceptual model, how to estimate some performance aspects how to formalize some aspects such as communication protocols, and how to map the conceptual requirements into design artifacts. A design aspect which we have not studied is the incorporation of databases as part of the secure architecture. Databases usually require a variety of related functions packed together a Database Management System (DBMS). The database system is a fundamental aspect for security because it stores the persistent information, which constitutes most of the information assets of the institution. We present here some ideas on how to make sure that the database system has the same level of security than the rest of the secure application. There are two possible ways to look at the problem of developing software for secure databases: building a general (application-independent) secure DBMS and building a database system which is part of a secure application. We think that this discussion is of interest even if we do not have any specific methodology in mind and we are just trying to build secure databases.

In the first approach, the DBMS is just a complex software application in itself and a general secure software methodology can be applied without or with little change. Object-oriented applications typically start from a set of use cases, which define the user interactions with the system under development. In this particular case, use cases would define the typical

functions of a DBMS, e.g. search, query, update, administration, and security, and these functions would be included in the requirements. The DBMS would follow an appropriate model, Role-Based Access Control (RBAC) appears as the most flexible, selectable in the analysis stage, which defines security constraints for the functions defined by the use cases. In some cases, it may be possible to support more than one security model. This approach would result in a secure DBMS where security would be a general nonfunctional requirement. The approach results in a general-purpose secure DBMS, where all applications will have the same level of security.

Another problem occurs when a designer needs to build a specific user application (or type of application) that includes a DBMS as part of its architecture, e.g. a financial system (most applications require a database but the degree of security needed may vary). This approach is discussed in [3, 5, 9]. In this case, the DBMS is tailored to the level of security desired for the specific type of application. Typically, this approach considers how to define and enforce a set of application-specific access control rules that follow some security model and how to reflect them in the schema and other parts of the DBMS. Most of these studies emphasize the security of the database schema or maybe the complete database system without much concern for the rest of the application. A methodology such as FJY can also be applied here, the DBMS being one of the architectural levels of the system, although these methodologies have little to say about the contents of the specific rules that are needed in the schema (only their safe storage but not their consistency or security).

Section 2 considers how our methodology would handle the first case, while Section 3 discusses the latter case. Section 4 presents a discussion of issues, and Section 5 draws some conclusions.

2. General secure database systems

In this case, as indicated earlier, the DBMS itself is a complex application requiring a general good level of security. We discuss here how our methodology would handle this case. Because of the generality of the resultant DBMS it may be difficult to prove formally general security properties. An early approach in this direction was based on adding security services to a general-purpose DBMS, e.g. INGRES [15] or System/R [1]. In fact, many commercial systems use this approach. Some of them build them as part of their design, others add them later as external services.

We look now at the stages of FJY, considering how would we accommodate the special characteristics of database systems.

Domain analysis stage: The domain here is everything related to databases, including data languages, authorization, concurrency control, etc. A domain model would include all these concepts. The supported model and the need for specialized database features should be determined at this point. The approach (general DBMS or application-oriented system) should also de defined at this stage.

Requirements stage: Use cases define the required interactions with the system. Each activity within a use case is analyzed to see which threats are possible. Activity diagrams indicate created objects and are a good way to determine which data should be protected and which data is persistent and is to be stored in the DBMS. The objects in the activity diagram could use stereotypes such as <<persistent>> to indicate persistence. Any requirements for degree of desired security should be expressed as part of the use cases. The degree of security comes from analysis of the institution assets. The threats lead to policies that can stop or mitigate them. The use cases are the standard functions of databases, e.g. query, search, define authorization rules, etc.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. The policies identified from the requirements as well as global (institution or regulatory) policies, can now be expressed as security patterns based on some model, e.g. access matrix or RBAC [14]. One can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases. In fact, analysis patterns can be built with predefined authorizations according to the needs of the roles in their use cases. Patterns for authentication, logging, and secure channels are also specified at this level [8]. Persistence indications are carried over this stage. One can verify at this stage that the secure conceptual model includes patterns that can handle the identified threats.

Design stage: When we have found the needed security patterns and added their representation in the conceptual model, we can select mechanisms that correspond to their concrete software realizations [4].

A specific security model, e.g. RBAC, is now implemented in terms of software artifacts. User interfaces correspond to use cases and may be used to enforce the authorizations defined in the analysis stage when users interact with the system. Components can be secured by using authorization rules for Java or .NET components. Security restrictions can be applied in the distribution architecture. Deployment diagrams can define secure configurations to be used by security administrators. System behavior fragments can be used to consider also performance aspects [16]. A multilayer architecture is needed to enforce the security constraints defined at the application layer. In each level we use patterns to represent appropriate security mechanisms. Classes defined as persistent should be mapped to DBMS structures, typically relations (tables). These relations would also store the metadata, including the authorization system of the DBMS. The general architecture of the DBMS could use views [1] or query modification [15] for access enforcement. Views are part of the external schema and usually stored as relations.

Implementation stage: This stage requires reflecting in the code the security patterns defined in the design stage. Because these patterns are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented programming languages. In this stage one can also select specific security packages or Commercial Off-the-Shelf (COTS) components, e.g., a firewall product or a cryptographic package. Some of the patterns identified earlier in the cycle can be replaced by COTS components. Performance aspects become now important and may require design iterations; for example, functions such as search and query must be efficient.

An important aspect for the complete design is assurance. One can formally verify each pattern used in a design but this does not in general verify their combination in a complete system. One can, however still argue that since one has used a careful and systematic methodology with verified and tested patterns, the design should provide a good level of security. Since security patterns embody good design principles they should increase the security of the system. Validation can be done in a qualitative way: The set of patterns can be shown to be able to stop or mitigate the identified threats. Specific design units can be formally validated using UMLsec [2, 10, 11] or a similar approach.

3. Building specific applications incorporating DBMSs

In this case we are building a known application. Since this approach tailors the degree of security of the DBMS to the application, one can add the required level of security using formal proofs when necessary, which are now easier because we have to deal only with a specific application. Specialized operating systems and hardware are also possible and may be needed to reach the required level of security. The methods described in the last section still apply here, except that additional requirements must be considered. Databases work through transactions and a concurrency control system serializes transactions to prevent inconsistencies; high-security multilevel databases require that the concurrency control system preserves security and we need to avoid covert channels. Below we repeat some of the steps of the methodology and see how these requirements affect them.

The persistent aspects of the conceptual model are typically mapped into relational databases. The design of the database architecture could be done using the approach of Section 2 and according to the requirements from the uses cases for the degree of security needed and the security model adopted in the analysis stage. A tradeoff is using an existing DBMS as a COTS component.

Domain analysis stage: A domain model is defined for the application. Legacy systems are identified and their security implications analyzed. Domain and regulatory constraints are identified and used as institutional policies that apply to any application.

Requirements stage: As before, use cases define the required interactions with the system for the chosen application. Each activity within a use case is analyzed to see which threats are possible. Again, activity diagrams indicate created objects and are a good way to determine which data should be persistent and stored in the DBMS. The policies that can stop the threats can be used to define which security patterns are needed.

Analysis stage: There is no change in this stage with respect to the previous case. However, the selected model must correspond to the type of application; for example, multilevel models have not been successful for medical applications. In fact, most commercial applications use RBAC because of its flexibility.

Design stage: The same issues apply as in the previous case. The only difference is in the DBMS, which we can build using the approach of Section 2 or select from some COTS DBMS. In both cases the secure conceptual model defines the expected security functions of the resultant DBMS. The classes which we indicated as persistent in the activity diagram and in the conceptual model must be mapped to relations. An application class may be mapped to more than one relation. This implies the need to map also the authorization rules. The security schemas that implement a specific security model are designed based on the semantics of the application and mapped from the application to the corresponding relations. At this stage UMLsec is useful to prove design properties. UMLsec is supported by extensive automated tool-support for performing a security analysis of its models against the security requirements [11], and has been used in a variety of industrial projects [2]. Since we know the specific subjects and objects involved in a given application, formal models can verify that the semantics of the models corresponds to the requirements.

Implementation stage: The same considerations discussed in Section 3 for this stage apply. For example, [5] uses Oracle 9i Label Security. For high security applications, testing and validation are more important. The threats identified in the requirements stage must be shown to be handled by the design. The platform used is also very important in reaching the desired level of security.

4. Related work

[9] presents a detailed analysis on how to build secure databases, including database design and software engineering aspects. [5] presents a methodology that consists of four stages: requirements gathering; database analysis; multilevel relational logical design; and specific logical design. Here, the first three stages define activities to analyze and design a secure database. The last stage consists of activities that adapt the general secure data model to one of the most popular secure database management systems: Oracle*9i* Label Security.

These approaches are database-centered in that they do not consider the applications that will use the database. We are concerned with the development of secure databases as part of building complex applications that require persistent storage. We need to include access control not only for the database elements but also for the non-persistent units and secure all the units of the system, e.g. distribution subsystems.

5. Conclusions

We have considered two approaches to building secure databases with our methodology. In the first approach we used the methodology to build the DBMS itself. Such a project becomes an example of using the proposed methodology and results in a DBMS with a good level of security. The second case requires to consider explicitly the characteristics of using a DBMS as part of an application and can reach higher levels of security because of its specialization; however, its use is more restricted.

The first approach most likely would be used by a vendor of DBMS products. When developing a new product or a research prototype, the systematic pattern-based approach would make the software more extensible and more secure with respect to a system designed in more conventional ways. In the second approach, the designers would probably use a COTS DBMS, although in this case security of the persistent data will depend on the security of that component. Ad hoc designs are possible but more expensive.

Both approaches need to be explored in more detail, although the second approach has already been explored to some extent; we have shown here only the general ideas behind them. Few patterns for database security exist and some would be of interest for future work: database adapters, view and query modification architectures for security, and mapping patterns.

Acknowledgement

This paper is based on some aspects of [12]. The comments of the referees were useful to improve this paper.

References

[1] M. M. Astrahan et al., “System R: Relational Approach to Database Management.”, ACM Trans. on Database Syst., vol. 1, No 2, 1976, 97-137

[2] B. Best, J. Jurjens and B. Nuseibeh, Model-based Security Engineering of Distributed Information Systems using UMLsec, 29th International Conference

on Software Engineering (ICSE 2007), ACM, 2007, pages 581-590

[3] S. Castano, M. Fugini, G. Martella, and P. Samarati, Database security, Addison-Wesley, 1994.

[4] E.B.Fernandez, T. Sorgente, and M.M.Larrondo-Petrie, "A UML-based methodology for secure systems: The design stage", Procs. of the Third International Workshop on Security in Information Systems (WOSIS-2005), Miami, May 24-25, 2005, 207-216.

[5] E. Fernández-Medina and M. Piattini, "Designing secure databases". Information and Software Technology, vol. 47, No 7, 2005, 463-477.

[6] E. B. Fernandez, M.M. Larrondo-Petrie, T. Sorgente, and M. VanHilst, "A methodology to develop secure systems using patterns", Chapter 5 in "Integrating security and software engineering: Advances and future vision", H. Mouratidis and P. Giorgini (Eds.), IDEA Press, 2006, 107-126.

[7] E.B.Fernandez, N. Yoshioka, H. Washizaki, and J. Jurjens, "Using security patterns to build secure systems", position paper in the 1st Int. Workshop on Software Patterns and Quality (SPAQu'07), Nagoya, Japan, December 3, 2007, collocated with the 14th Asia-Pacific Software Engineering Conference (APSEC).

[8] E.B.Fernandez and X.Y. Yuan, " Securing analysis patterns", Procs. of the 45th ACM Southeast Conference (ACMSE 2007), March 23-24, 2007, Winston-Salem, North Carolina.

[9] M. Fugini, "Secure database development methodologies", in Database Security: Status and Prospects, C. E. Landwehr (Ed.), Elsevier Science Publishers B.V. (North- Holland),1988, 103-129.

[10] J. Jurjens, Secure systems development with UML, Springer-Verlag, 2004.

[11] J. Jurjens, "Sound Methods and Effective Tools for Model-based Security Engineering with UML", 27th International Conference on Software Engineering (ICSE 2005), ACM, 2005, pages 322--331.

[12] J. Jurjens and E.B.Fernandez, "Secure database development", to appear in the Encyclopedia of Database Systems, Ling Liu and Tamer Oszu (eds.), Springer Verlag, Berlin, Germany.

[13] R. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman, "Role-Based Access Control Models", IEEE Computer, Vol.29, No.2, IEEE Press, 1996, 38-47

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

[15] M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES", ACM Trans. on Database Syst., vol. 3, 1976, 189-222.

[16] N. Yoshioka, S. Honiden, and A. Finkelstein, "Security patterns: A method for constructing secure and efficient inter-company coordination systems", Procs. of the 8th Int. IEEE Enterprise Distributed Object Computing Conference, 2004.

[17] N. Yoshioka, "A development method based on security patterns", Presentation, NII, Tokyo, Japan, 03/29/06.