Misuse Case-Based Design and Analysis of Secure Software Architecture

Joshua J. Pauli

College of Business and Information Systems
Dakota State University
Madison, SD 57042, USA
josh.pauli@dsu.edu

Dianxiang Xu

Department of Computer Science
North Dakota State University
Fargo, ND 58105, USA
dianxiang.xu@ndsu.edu

Abstract

This paper presents an approach to the architectural design and analysis of secure software systems based on the system requirements elicited in the form of use cases and misuse cases. We identify architectural components and their connections and analyze whether or not a candidate architecture can address security concerns. This provides a smooth transition from requirements specification to high-level design for engineering secure software systems, and greatly improves the traceability of security concerns, which allows a system developer to know what requirement an architectural component references back to. We demonstrate our approach through a case study on a security-intensive hospital information system.

Keywords: Software architecture, security, use case, misuse case

1. Introduction

Software is at the root of the widespread computer and information security problems [1]. Security issues in software systems are normally handled in an ad hoc, for instance 'penetrate-and-patch', manner. This makes it very difficult to validate the security features of a software system. In addition to the barriers raised by the impossibility of exhaustive testing of all possible failures and the unfortunate introduction of language (e.g. C)-specific security holes in system implementation [1, 2], an ad hoc approach that lacks rigorous requirements analysis and software design offers little support for validation and quality assurance of software artifact. Due to the rapidly increasing severity of software security threats, it is imperative that security concerns be addressed in the early stages of software development lifecycle.

Misuse cases, i.e. negative scenarios or use cases with hostile intent, have been proposed recently as a new avenue to elicit non-functional requirements, particularly security requirements [3, 4, 5, 6, 7]. Use cases have proven helpful for the elicitation of, communication about, and documentation of functional requirements [8, 9]. The integral development of use and misuse cases provides a systematic way for the elicitation of both functional and non-functional requirements [5]. A critical challenge to this approach is how misuse case based security requirements specification can further facilitate the design and implementation of software systems where security is a major concern. To our knowledge, no work has been done to meet this challenge.

As a key step to bridge the gap, this paper focuses on the software architecture perspective, which describes the components and their connections in the overall structure of a software system [10]. Specifically, we investigate how security requirements in the form of misuse cases can be used to guide architectural design and analysis. We demonstrate the approach through a case study on a security-intensive hospital information system (HIS). According to the security requirements elicited as misuse cases, we evaluate whether or not proposed candidate architectures are able to address required security features. The approach not only offers a smooth transition from requirements elicitation to high-level architecture design, but also greatly improves the traceability of security concerns, which allows a system developer to know what requirement an architectural component references back to [11]. Moreover, since the security requirements of a software system are already taken into consideration in the architecture design, the architecture specification is a valuable resource for detailed design, and can be applied as a basis for testing and validating the system implementation in terms of desired security features.

The rest of this paper is organized as follows. Section 2 gives an overview of our approach. Section 3 introduces the case study on a hospital information system (HIS) and discusses the misuse cases for HIS. Section 4 presents the analysis of candidate architectures for HIS. Section 5 reviews related work. Section 6 concludes the paper.

2. Overview of the Approach

Our approach places security as a primary system goal as opposed to an afterthought or add-on. It consists of the following steps:

1) Identifying users & behaviors

We first identify the users of the proposed system and the tasks that these users perform during "normal" usage of the system. The use of requirements gathering techniques such as interviewing, task description, and role playing can aid in gaining an idea as to what tasks each user performs. Then, we identify the users that complete any tasks that have an association with security.

2) Identifying use cases & misuse cases

Use/misuse case diagrams are used to depict use and misuse cases. Misuse cases are threats to the normal use cases. We also add additional use cases that provide a means to mitigate the threats.

3) Proposing and evaluating architectures

Candidate components of an architecture are obtained from the tasks and use/misuse case diagrams. We consolidate overlapping tasks into single task that covers all actions of the tasks. We also identify each task as either direct (no changes needed to the components involved) or indirect (changes needed to the components involved). If a component is indirect, list the components affected and the changes that need to be made. Then, we use a fish-eye diagram to show the components that need changes highlighted by a larger size and/or a different color. This is convenient to system designers down the road. Repeat these steps for each architectural proposal.

4) Selecting the best fitting architecture

To select the best fitting architecture, we take all the positive aspects from each proposal and adjust the components and their interactions into a comprehendible model. We create the architectural drawing to illustrate the updated components and interactions. An important factor to include in this step is to supply the supporting rational for the changes that were made.

Our approach is different from traditional architecture design and analysis due to the misuse cases. From both use cases and misuse cases, we define the services that the system must provide and prevent the ways that the system might be misused. Security goals of the system therefore weigh heavily.

3. Misuse Cases in HIS

The HIS provides services to every hospital employee that may or may not come in contact with actual patients. The possible users of the system are secretaries, nurses, doctors, pharmacists, IT personnel, business office personnel, and other administrative positions. Each employee must have permissions set for the databases and data warehouse that control what information each employee is allowed to access. Also, each employee will be assigned an access level by his or her superior that restricts the areas of HIS that he or she can enter.

Each HIS user has a specific set of tasks. For example, a doctor is primarily responsible for 1) diagnosing the patient's ailment at the time of the appointment; 2) prescribing treatment (prescriptions, therapy, etc…) to the patient; 3) entering the details of the appointment into the HIS once the appointment is complete to document the appointment as a whole; 4) entering pharmacy orders to be transmitted to the pharmacy; and 5) setting the access levels for the nurses and secretaries to restrict areas inside HIS. A typical visit from the patient's perspective would include many (direct or indirect) interactions with many different users (secretary, nurse, doctor, pharmacist, etc.).

We have investigated the misuse cases that involve behaviors of those users who have a very clear-cut set of behaviors. Modeling these behaviors makes the creation of misuse cases easier to understand, and at the same time, ensure a great deal of coverage to the security goals of the system. The users of the system that have a definite relationship to the system security include secretary, nurse, doctor, pharmacist, etc. They have the most interaction with not only the patients, but the system as a whole. Example behaviors or interactions closely related to the HIS security are: 1) Secretary entering patient information; 2) Nurse entering preliminary appointment information; 3) Doctor entering appointment findings; 4) Doctor transmitting pharmacy orders to the pharmacy; 5) Pharmacist receiving pharmacy order, etc.

To elicit system requirements, we extend the process of use case modeling due to the introduction of misuse cases. For the interactions mentioned above, we have identified corresponding misuse cases. As an example, Fig. 1 shows the doctor's use/misuse cases. Each white oval represents a use case, either a genuine use of the system or mitigation efforts. The black ovals represent misuse cases attempting to inflect harm onto the HIS. The 'Crook' and darkened 'Doctor' are actors completing misuse cases. This means that harm may be done to the system if they are allowed. The 'Security' actor would be completing mitigating use cases in an effort to block any harm that may be directed at the HIS. Due to the limited space, we will not elaborate on the textual description of these use and misuse cases.

4. Architectural Design and Analysis of HIS

In general, a variety of different architecture designs that meet the same system requirements may exist. From

the security perspective, this section discusses the evaluation of candidate software architectures according to misuse cases. For simplicity, we only use two candidates for the HIS. Based on the analysis, a recommendation as to what the final architecture should include is also given.

We create candidate architectures with the following guidelines in mind: 1) The architecture should include all the main actors from the use/misuse cases. For the HIS, this would naturally include 'Doctor', 'Patient', 'Pharmacist', etc. 2) The architecture should include the use/misuse cases that the actors complete. Not only the "normal" use cases such as 'Enter Appointment Findings' and 'Enter Pharmacy Orders', but also the mitigating use cases such as 'User Recognition' and 'Log Access Attempts' for example.

4.1 Analysis of Candidate Architecture #1

Fig. 2 shows the candidate architecture #1. Table 1 depicts the relationship between use/misuse cases and all the components that appear in the architecture. While some of these individual components will be present on any architecture, they will have different configurations from one candidate to another.

For this case study, a component is a group of related functionality or data that collectively would make-up the system. The connections between components are simply shown with arrows leaving and entering their respective components. They are meant to show what components interact with each other and the degree of interaction (single-headed arrow vs. double-headed arrow). Although the architectures are specified in an informal manner, they do serve the purpose of illustrating the groupings of functionality that the system will undoubtedly need. Our ongoing work with this approach includes rigorous specification of architectural components and connectors.

Next we evaluate the candidate architecture to see how well this architecture "fits" the misuse and use cases. It is not difficult to see that there are numerous system tasks that could be considered as over-lapping from one user to another. Table 2 shows some tasks that different users complete. To group system tasks, we determine whether each use case or misuse case is direct or indirect. A direct use case means that a normal execution of the system will allow the use case or misuse case to be completed successfully. An indirect use case means that some change needs to be made to the architecture in order for the use case or misuse case to be completed successfully. Because there is no previous architecture to review to check how the components actually interact with each other, the direct/indirect analysis that follows is somewhat of an estimation of how a system may behave under similar circumstances. Each system task is numbered, given a description (this is where any grouping of use cases will occur), labeled as direct or indirect, and any changes that will be needed are noted.

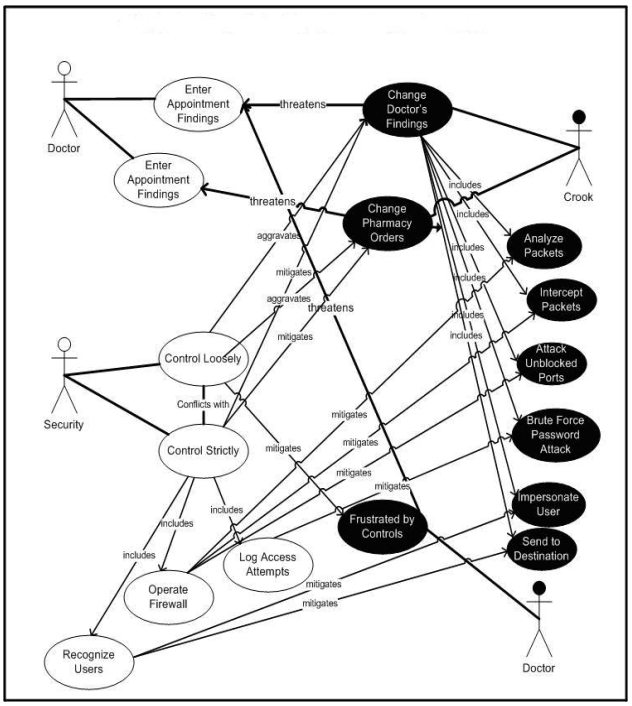

Figure 1. Doctor use & misuse case model in HIS

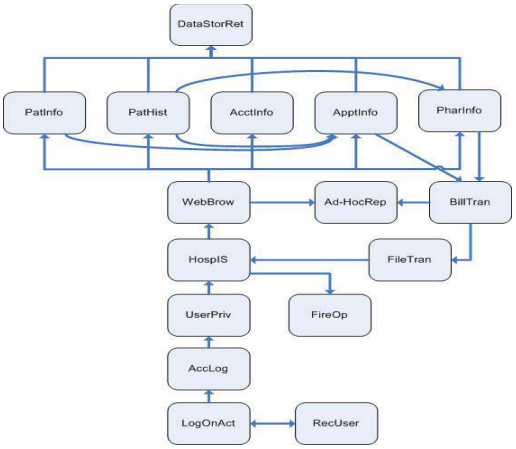

Figure 2. Candidate architecture #1 for HIS

One visualization technique that has been proven to be very useful to highlight component interactions is a fish-eye view diagram in which each component is created with a size proportional to the number of interactions that it has. Fig. 3 shows the fish-eye view diagram for candidate architecture #2. It calls attention to the most architecturally significant features of the hospital system as it currently exists. It can be used as a guide for designers and developers in their allocation of time and

effort during the project. The development team can always revisit any part of this informal analysis for reference of clarifications dealing with the HIS.

Table 1. Components vs. misuse/use cases for candidate architecture #1

COMPONENT	ABBREVIATION	Breach Authentication	Breach Security Level	Change Findings	Change Information	Change Orders	Log On	Enter Information	Transmit Orders	Transmit Bill	Receive Orders	Receive Payment
Patient History	PatHist			X		X		X			X	
Patient Information	PatInfo				X	X		X			X	X
Account Information	AcctInfo				X			X	X	X	X	X
Appointment Information	ApptInfo			X	X	X		X		X		X
Pharmacy Information	PharInfo			X		X		X	X	X	X	
Bill Transmitting	BillTran				X	X		X	X	X	X	X
Web Browsing	WebBrow			X	X	X		X	X	X		
File Transferring	FileTran			X	X	X		X	X	X		
Ad-Hoc Reporting	Ad-HocRep								X		X	X
Data Storing / Retrieving	DataStorRet	X		X	X	X		X	X	X	X	X
Access Attempt Logging	AccLog	X					X					
Firewall Operating	FireOp			X	X	X		X	X	X		
Recognizing Users	RecUser	X	X				X				X	
Log-on Activities	LogOnAct	X					X					
User Privileges	UserPriv		X			X	X		X			

Table 2. User tasks in HIS

User	*Tasks*
Pharmacist	11. Receive Pharmacy Orders from Dr 12. Retrieve Patient Info and History 13. Transmit Pharmacy Bill to Insurance
Business Office	14. Receive Pharmacy Bill 15. Receive Doctor Bill …

Table 3. Indirect use cases for architecture #1

Description	*Indirect?*	*Needed Changes*
Log access attempts	Indirect	Changes to 'LogOnAct', 'AccLog' to ensure that the access attempts are logged.
Recognize users	Indirect	Changes to 'LogOnAct' and 'RecUser' to ensure the proper measures are in place.
Port to Another OS	Indirect	Main changes to 'HopsIS', 'WebBrow', 'FireOp', and 'FileTran' for OS migration.

4.2 Analysis of Candidate Architecture #2

Let us consider candidate architecture #2. This allows for comparisons between the two candidates, highlighting the positives and negatives of each. From the list of fundamental services identified, a more detailed list is created that includes all the components that may appear in the architecture from each use case. While the system tasks stay the same for both architectural proposals, the system tasks that are direct or indirect may be different, as will any component that may need to be changed.

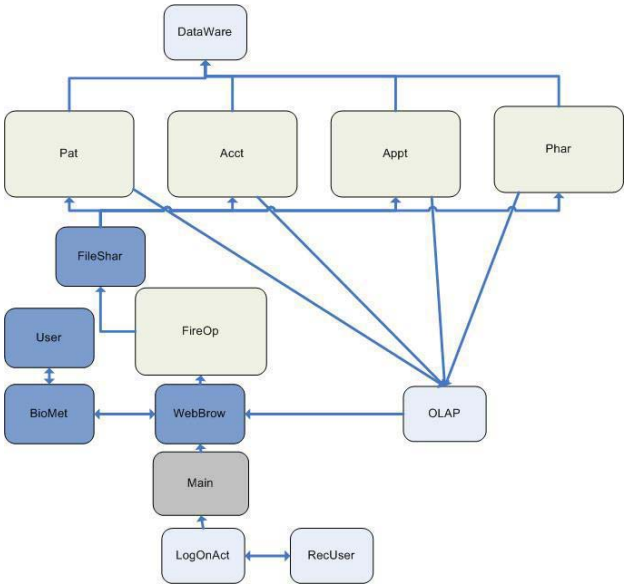

Figure 3. Fish-eye diagram of architecture #2

Due to the limited space, we only show the corresponding fish-eye diagram (Fig.3). Just as the case for the first proposal, the problematic misuse and use cases that have been classified as indirect would now need to be prioritized before making the necessary changes that would complete the architecture.

4.3 Recommending the Architecture

Using the above idea, we may evaluate and compare a number of candidate architectures and then recommend one with the best fit. In reality, most systems' architecture ends up being a combination of the positive points illustrated by each architectural proposal. The goal of selecting the architecture with the best fit is to extract those strengths and form a final architecture that makes the most of these strengths. Also, conserving the overall functionality of the system must also be a high priority.

The use cases and misuse cases for the architecture will remain the same regardless of the final decision. What will change are the actual components that make up the architecture and how these components are arranged and how they interact amongst themselves. Table 4 shows a recommended component identification checklist that lists the components and the high-level use cases and misuse cases that they will interact with. The architectural

diagram to show how these components are arranged and how they may interact is unveiled in Fig.4.

The reasons for these changes to the architectural proposal were driven by the strengths of each independent proposal. The recommended architecture includes characteristics of each of the previous proposals in a way that makes the overall architecture more efficient and secure.

Table 4. Components vs use/misuse cases for the recommended architecture

		HIGH-LEVEL MISUSE & USE CASES											
COMPONENT	ABBREVIATION	Breach Authentication	Breach Security Level	Change Findings	Change Information	Change Orders	Log On	Enter Information	Transmit Orders	Transmit Bill	Receive Orders	Receive Payment	
Patient History	PatHist			X		X		X			X		
Patient Information	PatInfo				X	X		X			X	X	
Account Information	AcctInfo				X			X	X	X	X	X	
Appointment Information	ApptInfo			X	X	X		X		X		X	
Pharmacy Information	PharInfo			X		X		X	X	X	X		
Bill Transmitting	BillTran				X	X		X	X	X	X	X	
Web Browsing	WebBrow			X	X	X		X	X	X			
User	User	X	X			X	X	X	X	X	X	X	
Biometric Measures	BioMet	X	X				X				X		
File Sharing	FileShar			X	X	X		X	X	X	X	X	
Online Analytical Processing	OLAP								X	X	X	X	
Data Warehousing	DataWare	X		X	X	X		X	X	X	X	X	
Main Program	Main	X	X	X	X	X	X	X	X	X	X	X	
Access Attempt Logging	AccLog	X					X						
Firewall Operating	FireOp			X	X	X		X	X	X			
Log-on Activities	LogOnAct	X					X						
User Privileges	UserPriv		X			X	X		X				

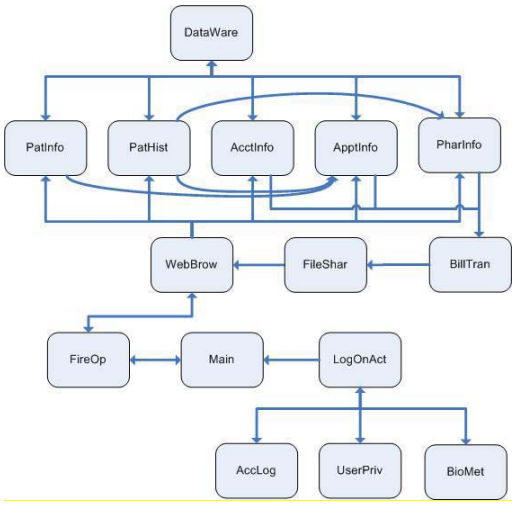

Figure 4. Recommended architecture for HIS

For example, it is the opinion of this case study that using 'Biometric Measures' (from candidate #2) is superior to using 'Recognize Users' (from candidate #1), because it identifies the exact manner in which this architecture will provide that level of security. Another example is to use 'OLAP' (from #2) instead of 'Ad-hoc Reporting' (from #1). Again, 'OLAP' is more definite than simply saying that the HIS will have 'Ad-hoc Reporting' capabilities. Another notable inclusion on the recommended architecture is that information pertaining to patients, accounts, appointments, and pharmacies were retained instead of grouping all these components into more generic versions (candidate #2).

5. Related Work

Our approach is related to the work on misuse cases, architectural analysis, and security engineering. Due to space limits, we only review the closest related work.

To meet the need of modeling system behavior that should be avoided, Sindre and Opdahl proposed misuse cases as the inverse of use cases [3]. A misuse case can be defined as a completed sequence of actions which results in loss for the organization or some specific stakeholder. Also they have proposed a general template for misuse case description [4]. Alexander discussed a variety of applications of misuse cases beyond security requirements elicitation, such as eliciting general "-ility" requirements, exceptions, and test cases [5]. Misuse cases are also useful for the trade-off analysis, the goal of which is to enable stakeholders to make an informal and correctly-based judgment in a possibly-complex situation [6]. Employing a use/misuse case representation may make such a judgment more likely if it helps people to visualize the structure of the situation accurately, and in a way that emphasizes the essential points of conflict that create the need for a trade-off. McDermontt and Fox introduced a similar notion, called abuse cases, for security requirements analysis [12]. A complete abuse case defines an interaction between an actor and the system that results in harm to a resource associated with one of the actors, one of the stakeholders, or the system itself. McDermott further applied the abuse case based approach to the construction of an assurance argument as a collection of abuse case refutations [13]. Obviously, the above work has focused on the elicitation of security requirements; it has nothing to do with the transition from requirements and architectural design and analysis.

Determining the extent to which a proposed software system meets desired quality criteria is desirable for a decent software development process. Kazman et al proposed a scenario-based analysis of software architecture [14]. Scenarios are used to express the particular instances of each quality attribute important to the customer of a system. The architecture under consideration was analyzed with respect to how well or how easily it satisfies the constraints imposed by each scenario. The approach has several steps: 1) describing candidate architecture; 2) developing scenarios; 3)

performing scenario evaluations; 4) revealing scenario interaction; and 5) performing overall evaluation. Kantorowitz et al. designed a framework for use case-oriented software architecture, which enables a "direct" manual translation of sufficiently detailed natural language use case specifications into code [15]. The produced software centers around use case components that implement the different use cases of the application. While we were motivated by the work along this line, our focus is on dealing with security concerns at the level of misuse case-based software architecture.

In addition, fault-trees, or threat trees, have been used to model security threats and evaluate security risks [2]. An application is composed of threat targets that could have vulnerabilities. A threat tree describes the decision-making process an attacker would go through compromise the component. The idea underlying this specific representation of threats is essentially similar to misuse cases. But it is more at the level of detailed design than at the architectural design level.

6. Conclusions

We have presented the approach to the analysis of secure software architecture according to the requirements in the form of use/misuse cases. The findings of the architectural analysis can be used in detailed design of the system and in validation of the system implementation. Also, the architectural analysis can be re-visited at anytime to get a better understanding of the underlying architecture or to clear up any confusion amongst system developers. Dealing with security issues at the software architecture level can make a system more resistant to vulnerabilities.

Architectural design is often a heuristic process even though the requirements specification is available. The analysis in our current approach is done in an informal way per se. For the future work, it is worth investigating a rigorous way to formalize the process. For example, both use/misuse cases and software architecture might be described in formal specification languages so that critical desirable properties (security in particular) of architecture designs can be verified against the requirements.

Acknowledgments

This work was supported in part by the NSF under grant EPS-0132289.

References

1. Viega, J. and McGraw, Gary. *Building Secure Software: How to Avoid Security Problems in the Right Way*. Addison Wesley, 2002.
2. Howard, M. and LeBlanc, D. *Writing Secure Code*. Microsoft Press. 2nd Edition, 2003.
3. Sindre, G. and Opdahl, A.L. Eliciting security requirements by misuse cases. In *Proc. of TOOLS Pacific 2000*, pp. 120-131, 2000.
4. Sindre, G. and Opdahl, A.L. Templates for misuse case description. In *Proc. of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001)*, June 2001.
5. Alexander, I. Misuse Cases: Use Cases with Hostile Intent. *IEEE Software*, 58-66 (January/February 2003).
6. Alexander, I. Initial Industrial Experience of Misuse Cases. *Proc. of IEEE Joint International Requirements Engineering Conference*, pp. 61-68, 2002.
7. Firesmith, D.: Security Use Cases. Journal of Object Technology, Vol. 2, No. 3, 53-64. (May-June 2003).
8. Bittner, K. and Spence, I. *Use Case Modeling*, *Object Technology Series,* Addison Wesley, 2003.
9. Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1994.
10. Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*, 2nd Edition, Addison Wesley, 2003.
11. Pressman, R. S., *Software Engineering: A Practitioner's Approach*, 6th Edition, McGraw Hill, 2004.
12. McDermott, J. and Fox, C. Using abuse case models for security requirements analysis. In *Proc. of the 15th Annual Computer Security Application Conference*, pp. 55-66, 1999.
13. McDermott, J. Abuse-case-based assurance arguments. In *Proc. of the 17th Computer Security Applications Conference (ACSAC'O1).* New Orleans LA USA: Dec 2001, pp. 366-374.
14. Kazman, R., Abowd, G., Bass, L., and Clements, P. Scenario-based analysis of software architecture. *IEEE Software*. pp. 47-55, November 1996.
15. Kantorowitz, E., Lyakas, A., and Myasqobsky. Use case-oriented software architecture. *CMC03*, 2003.