THREAT-DRIVEN ARCHITECTURAL DESIGN OF SECURE INFORMATION SYSTEMS

Joshua Pauli
College of Business and Information Systems, Dakota State University, Madison, SD 57042, USA

Dianxiang Xu
Department of Computer Science, North Dakota State University, Fargo, ND58105, USA

Keywords: Software architecture, Security, Threat model, Use case, Misuse case, UML.

Abstract: To deal with software security issues in the early stages of system development, this paper presents a threat-driven approach to the architectural design and analysis of secure information systems. In this approach, we model security threats to systems with misuse cases and mitigation requirements with mitigation use cases at the requirements analysis phase. Then we drive system architecture design (including the identification of architectural components and their connections) by use cases, misuse cases, and mitigation use cases. According to the misuse case-based threat model, we analyze whether or not a candidate architecture is resistant to the identified security threats and what constraints must be imposed on the choices of system implementation. This provides a smooth transition from requirements specification to high-level design and greatly improves the traceability of security concerns in high assurance information systems. We demonstrate our approach through a case study on a security-intensive payroll information system.

1 INTRODUCTION

Software security has been critical to information assurance. Software security issues in information systems are traditionally handled in an ad hoc or afterthought (e.g. 'penetrate-and-patch'), manner. This offers little support for validation of desired security properties due to the lack of a rigorous process for security requirements analysis and secure software design. In particular, design-level vulnerabilities are a major source of security risks in code. For example, design-level problems accounted for around 50% of the security problems uncovered during the Microsoft's "security push" in 2002 (Hoglund, 2004).

To address software risks at the design level, the threat modeling approach centers around determining and ranking the threats to the system based on the decomposition of application or an available architecture design, followed by choosing techniques for mitigating and responding to the threats. Since security threats, modeled by attack trees, often involve much detail of implementation techniques, it is not clear how the threat model can be traced back to the application-specific security requirements. In fact, the current threat modeling approach does not provide any explicit way for the elicitation and analysis of security requirements.

Misuse cases, i.e. negative scenarios or use cases with hostile intent, appear to be a new avenue to elicit security requirements (Alexander, 2002, 2003; Sindre, 2001a, 2001b; McDermott, 1999, 2001; Firesmith, 2003). Use case modeling is a proven method for the elicitation of, communication about, and documentation of functional requirements (Jacobson, 1994; Bittner, 2003). The integral development of use cases and misuse cases provides a systematic way for the elicitation of various system requirements, both functional and non-functional (Alexander, 2003). A critical issue 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.

This paper presents an approach to bridging the gap between misuse case based security requirements and high-level architecture design. On one hand, we treat identification of security threats as part of requirements elicitation and model them with misuse cases. UML sequence diagrams (UML 2.0) are exploited to describe the decision-making process an attacker would go through to compromise or misuse the system. On the other hand, we drive architecture design by dealing with the identified security threats in the process of application decomposition (as apposed to determining and mitigating security threats after the decomposition in the threat modeling approach). According to the security threats modeled by misuse cases, we evaluate whether or not a proposed candidate architecture is able to resistant to the security threats and what constraints should be imposed on the choices of implementation techniques in order to mitigate the threats. The treatment of security threats in the earlier phases of system development can reduce overall development cost due to the absence of a variety of vulnerabilities. We also keep track of the mapping between the use/misuse cases and the architectural components. This makes it easier to locate and fix security defects in later phases of development. Moreover, since the software security requirements are already taken into consideration in the architecture design, the architecture specification is an invaluable resource for detailed design, implementation, and validation.

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 payroll information system (PIS) and discusses the misuse cases in PIS. Section 4 presents the design and analysis of architectures for PIS. Section 5 reviews related work. Section 6 concludes the paper.

2 THREAT-DRIVEN ARCHITECTURAL DESIGN

Our approach places security as a primary system goal as opposed to an afterthought or an add-on. It starts by the elicitation of system requirements in terms of use cases, misuse cases, and mitigation use cases. The overall picture of the system requirements is captured by use/misuse case diagrams. Use cases as in (Jacobson, 1994) represent the tasks that legitimate users perform during “normal” usage of the system. Misuse cases depict possible security threats that threaten the normal use cases of the system. These are the threats that an attacker may pose to the system to violate security properties, such as confidentiality, privacy, and availability. Mitigation use cases indicate the means for mitigating corresponding misuse cases. Different from existing work on misuse cases (Alexander, 2002, 2003; Sindre, 2001a, 2001b; McDermott, 1999, 2001), we also model the decision-making process an attacker would go through to compromise or misuse the system. UML 2.0 sequence diagrams are used as a modeling tool for this purpose. The modeling of attack processes is critical to evaluating whether and to what extent a software architecture is resistant to security threats.

Once the requirements are available, we decompose the tasks into a list of fundamental services that the system must produce. These services are the basis for identifying components of which a candidate software architecture will be made. It is necessary to check for any overlapping system tasks, though. If any are found, consolidate the like tasks into one task that covers all the actions of the consolidated tasks. Once a complete list of unique system tasks are in place, identify each system task as either *direct* (no changes needed to the components involved) or *indirect* (changes needed to the components involved). If a component is indirect, then list the components affected and the changes that need to be made. (Often times these will be abstract in nature.) A candidate architecture is formed by including the identified components and configuring the connections among the components. To support traceable design and analysis, our approach uses a table to keep track of the mapping between the use/misuse cases and the components, as will be shown in Section 4. Given a use case, for example, it is easy to know which components realize the use case.

A candidate architecture is then evaluated according to the security requirements. Specifically, for each threat (i.e. misuse case), we check to see if it can be prevented in the candidate architecture, and if not, what constraints must be imposed on the selection of implementation techniques to mitigate the threat. Based on the evaluation of different candidate architectures, we can take all the positive aspects from each candidate and modify them to make the components fit together into a comprehensible model. An important factor to include in this step is to supply the supporting rational for the changes that were made.

The requirements elicitation and architecture design in the approach is part of an iterative development process. In addition, the security threats modeled by the misuse cases can be rated in terms of risks, as in the threat modeling approach (Hoglund, 2004). When limited resources (e.g. budget and time) are available, more attention can be paid to mitigating the threats with higher risks. This is beyond the scope of this paper.

3 MISUSE CASES IN PIS

This section introduces a payroll information system that could be put in place at any company. We first describe the basic functionality of PIS and then discuss the use and misuse cases.

The PIS provides payroll services to every employee that works for, and is paid by, the company. The application server, database server, and web server are responsible for applications, database transactions, and web access, respectively. The possible users of the system are administrators, users, web developers, and auditors. 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 PIS that he or she can enter. These two measures are an effort to control the new information that each employee can enter and to control the stored information that each employee can retrieve.

Each employee has different tasks or use cases that are completed during normal usage of the system. For example, the tasks of a payroll administrator include managing employee information, completing administrative tasks, generating reports, etc. For each task, we identify if it is associated with security issues. In addition to normal use cases, we also identify misuse cases and mitigation use cases. Misuse cases are use cases with hostile intent or threats to the normal use cases. They essentially reflect various ways of violating desirable security properties, including privacy, confidentiality, availability, etc. Mitigation use cases are the means for mitigating the corresponding threats in order to achieve the security properties.

We have identified the use cases, misuse cases, and mitigation use cases for the PIS. Fig. 1 shows the overall use/misuse case diagram. Each white oval represents a use case; either a genuine use of the system or a mitigation effort. The black ovals represent misuse cases that are attempting to inflect harm onto the PIS. They indicate various ways that misusers would plan to harm the system and the data within it. A use case connected to a misuse case with a dashed arrow is a mitigation use case that mitigates the corresponding threat. For the sake of clarity, the 'mitigates' relations are not labeled in the diagram.

We exploit UML sequence diagrams to model the behaviors of use/misuse cases. In particular, the sequence diagram of a misuse case describes the decision making process an attacker or misusers may go through to compromise or misuse the system. As an example, Fig. 2 shows the sequence diagram for the "Spoof Computer ID" misuse case. Each rectangle across the top of the diagram represents a

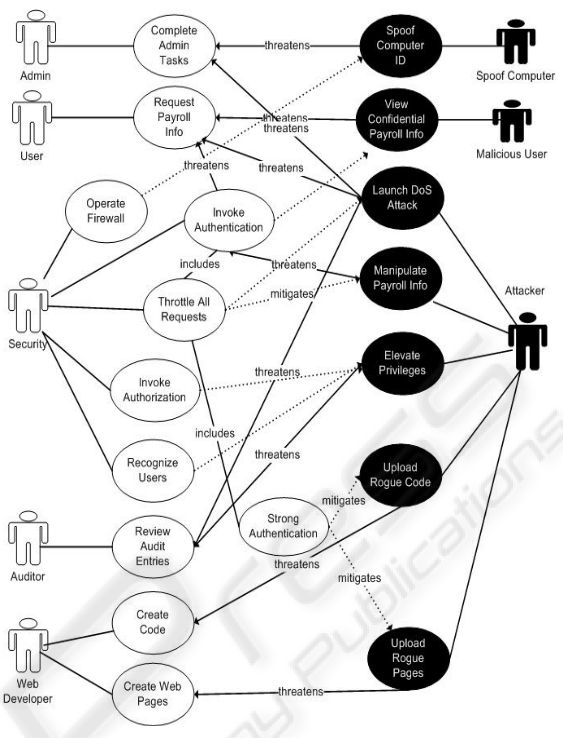


Figure 1: The use/misuse case diagram for PIS

business role that is related to the misuse case and the arrows represent the sequence of actions that take place between the roles during the execution of the misuse case. 'Spoof Computer' is the computer

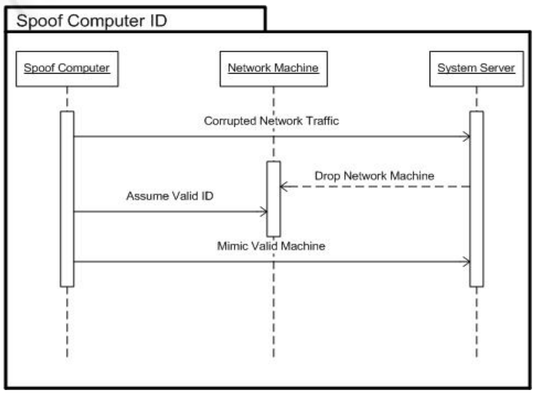


Figure 2: The sequence diagram for the misuse case Spoof Computer ID

that is portraying a valid network computer ('Network Machine'), controlled by the misusers, which is sending out corrupted network traffic in hopes that the 'System Server' will deem it necessary to simply drop the valid network computer from the network to eliminate that traffic. Once the 'Network Machine' has been dropped by the

'System Server', the 'Spoof Computer' assumes the identity of the dropped 'Network Machine'.

Each use/misuse case also has a textual description as in (Sindre, 2001b). Due to the limited space, we will not elaborate on this.

4 ARCHITECTURE DESIGN AND ANALYSIS OF PIS

Software design is in general a heuristic process. Given the system requirements (use cases, misuse cases, and mitigation use cases), there exist a variety of possible architecture designs. For secure software design, it is important to keep in mind not only the fundamental services that the system must include, but also those user actions that have a heavy security influence. For the PIS case study, we have evaluated three candidate architectures. Due to the limited space, this section focuses on the design and evaluation of one candidate architecture and then describes the recommended architecture, together with its associated security mechanisms and constraints.

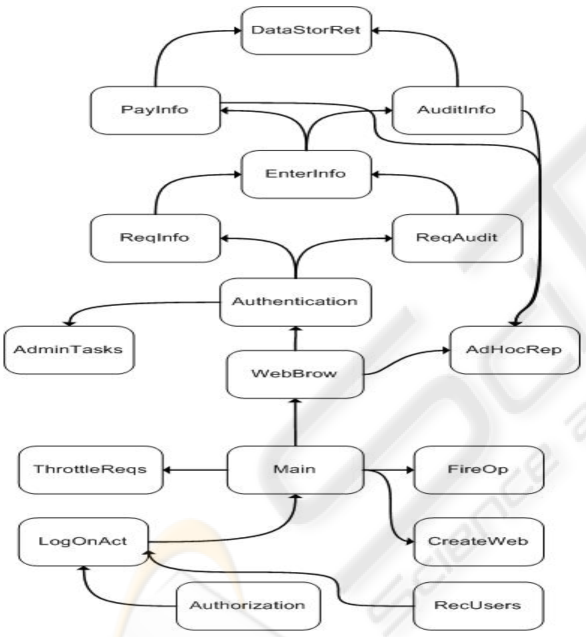
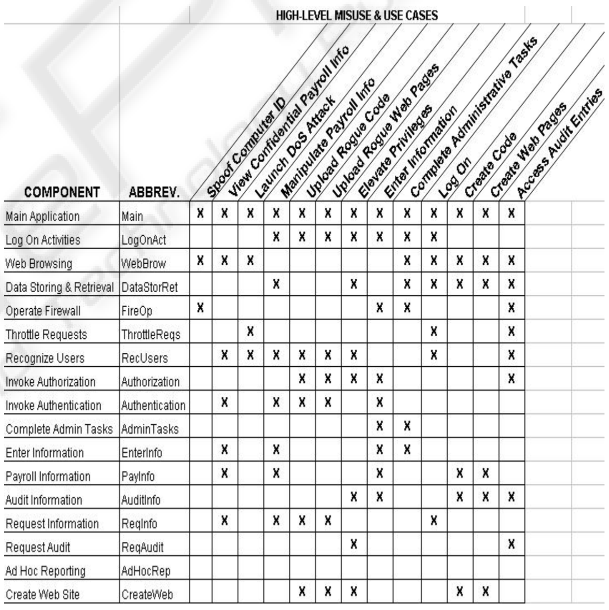


Figure 3: The candidate architecture #1 for PIS

In general, we create candidate architectures with the following requirements in mind. 1) The architecture should include all the main actors from the use cases and misuse cases. For the PIS, this would naturally include 'Administrator', 'User', 'Web Developer', 'Auditor', etc. 2) The architecture should include the use cases that the actors complete. Not only the "normal" use cases such as 'Complete Administrative Tasks' and 'Create Web Pages', but also the mitigating use cases such as 'Invoke Authentication' and 'Recognize Users' for example.

According to use/misuse cases, a candidate architecture is formed by including the functional components and configuring the connections among them. Fig. 3 shows the candidate architecture #1. Any actual piece of hardware that had a definite purpose in the system was classified as a component such as the firewall and data storage. Also, a group of related functionality was also classified as a component such as Log-On Activities, Creating the Web Site, and Administrative Tasks. Finally, the core functionality of the system that needed to be explored in more detail was also classified as separate components such as Enter Information, View Information, Request Audit, and Request Information. While this component identification process is informal in nature, it is sufficient to start the process of proposing possible architectures that meet the overall requirements of the PIS system.



		HIGH-LEVEL MISUSE & USE CASES												
COMPONENT	ABBREV.	Spoof Computer ID	View Confidential Payroll Info	Launch DoS Attack	Manipulate Payroll Info	Upload Rogue Code	Upload Rogue Web Pages	Elevate Privileges	Enter Information	Complete Administrative Tasks	Log On	Create Code	Create Web Pages	Access Audit Entries
Main Application	Main	X	X	X	X	X	X	X	X	X	X	X	X	X
Log On Activities	LogOnAct				X	X	X	X	X	X	X			
Web Browsing	WebBrow	X	X	X						X	X	X	X	X
Data Storing & Retrieval	DataStorRet				X			X		X	X	X	X	X
Operate Firewall	FireOp	X							X	X				X
Throttle Requests	ThrottleReqs			X							X			X
Recognize Users	RecUsers		X	X	X	X	X	X			X			X
Invoke Authorization	Authorization					X	X	X	X					X
Invoke Authentication	Authentication		X		X	X	X		X					
Complete Admin Tasks	AdminTasks								X	X				
Enter Information	EnterInfo		X		X				X	X				
Payroll Information	PayInfo		X		X				X			X	X	
Audit Information	AuditInfo							X	X			X	X	X
Request Information	ReqInfo		X		X	X	X				X			
Request Audit	ReqAudit							X						X
Ad Hoc Reporting	AdHocRep													
Create Web Site	CreateWeb					X	X	X				X	X	

Figure 4: Components vs use/misuse cases for architecture #1

Fig. 4 depicts the relationship between use/misuse cases and all the components that appear in architecture #1. Each component is shown with an "X" marking what use/misuse case that the component would be part of.

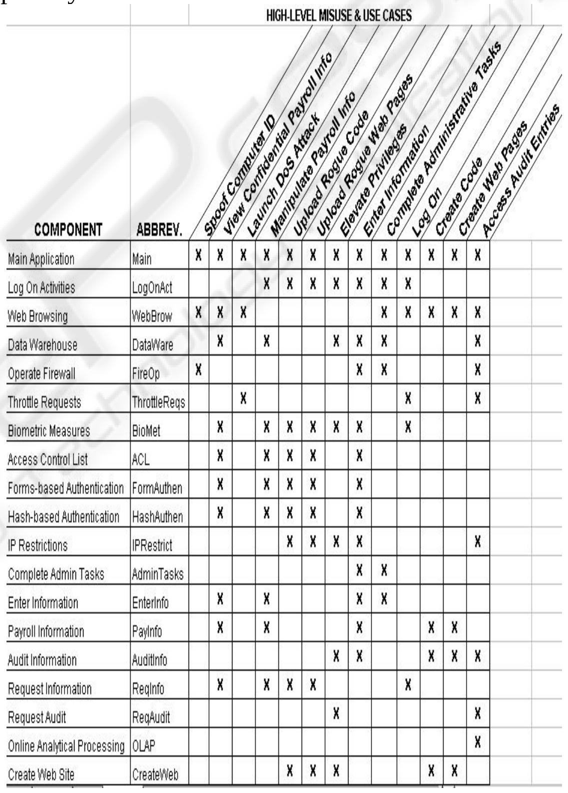
To evaluate the candidate architecture, we check to see how well it "fits" the misuse and use cases. Before further analysis can be done, we need to do some grouping of system tasks - a determination of whether each use case or misuse case is direct or indirect. A direct use case means that a normal 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. From the listing of changes that are needed to each component, we can tell that certain components obviously need more changes that others. Table 1 shows part of the tasks and needed changes.

Table 1: Tasks and needed changes

Task	Direct/ Indirect	Needed Changes
Invoke Authorization	Indirect	Changes to 'LogOnAct', and 'RecUsers' to ensure that the authorization measures will not interfere with logging onto the system and will work with the recognition of users.
Throttle System Requests	Indirect	Changes to 'Main' to ensure that the throttling of system requests will not interfere with normal usage of the system.
Recognize Users	Indirect	Changes to 'LogOnAct', 'Authorization', 'RecUser' to ensure that the proper measures are in place for user recognition. i.e. passwords,biometrics, etc)

Using the above idea, we may evaluate and compare a number of candidate architectures. For space reasons, we won't elaborate on other candidate architectures for the PIS. Here, we briefly compare architecture #1 with another candidate architecture (i.e. architecture #2) we have evaluated. Architecture #2 was much more generic in numerous components. For example, where #1 had individual components of 'ReqInfo', 'ReqAudit', 'EnterInfo', 'PayInfo', and 'AuditInfo', architecture #2 simply had the components of 'PayInfo' and 'AuditInfo'. The two latter components would have included all the functionality of the previous five components. Because of the vast amount of functionality involved in the use cases dealing with all audit and payroll information, it was determined that the final architecture should include individual components as stated in the first proposed architecture. Inversely, architecture #2 proposed the components of 'HashAuthen', 'FormAuthen', and 'ACL' to replace the 'Authentication' component in the first proposal. These are a few examples of the differences between the two proposed architectures for PIS. These differences can be fully seen in the final architectural recommendation.

Based on the evaluation and comparison of different candidate architectures, we can recommend one with the best fit. Obviously, this is a process that would include many different levels of stakeholders of the system. 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 should be to extract those strengths and form a final architecture that makes the most of these strengths. At the same time, conserving the overall functionality of the system must also be a very high priority.



		HIGH-LEVEL MISUSE & USE CASES												
COMPONENT	ABBREV.	Spoof Computer ID	View Confidential Payroll Info	Launch DoS Attack	Manipulate Payroll Info	Upload Rogue Code	Upload Rogue Web Pages	Elevate Privileges	Enter Information	Complete Administrative Tasks	Log On	Create Code	Create Web Pages	Access Audit Entries
Main Application	Main	X	X	X	X	X	X	X	X	X	X	X	X	X
Log On Activities	LogOnAct				X	X	X	X	X	X	X			
Web Browsing	WebBrow	X	X	X						X	X	X	X	X
Data Warehouse	DataWare		X		X			X	X	X				X
Operate Firewall	FireOp	X							X	X				X
Throttle Requests	ThrottleReqs			X							X			X
Biometric Measures	BioMet		X		X	X	X	X	X		X			
Access Control List	ACL		X		X	X	X		X					
Forms-based Authentication	FormAuthen		X		X	X	X		X					
Hash-based Authentication	HashAuthen		X		X	X	X		X					
IP Restrictions	IPRestrict					X	X	X	X					X
Complete Admin Tasks	AdminTasks								X	X				
Enter Information	EnterInfo		X		X				X	X				
Payroll Information	PayInfo		X		X				X			X	X	
Audit Information	AuditInfo							X	X			X	X	X
Request Information	ReqInfo		X		X	X	X				X			
Request Audit	ReqAudit							X						X
Online Analytical Processing	OLAP													X
Create Web Site	CreateWeb					X	X	X				X	X	

Figure 5: Component checklist of the recommended architecture for PIS

Fig. 5 shows a recommended component identification checklist that lists the components and the 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. 6, which is a UML 2.0 class (component) diagram with ports. The ports represent how the components interact and how the functionality within each component is utilized during these interactions.

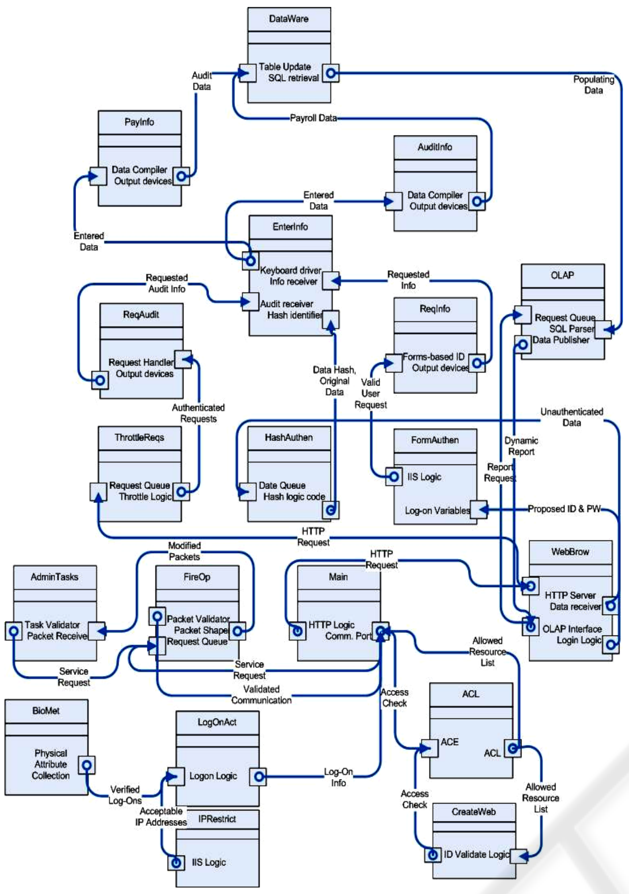


Figure 6: Recommended architecture for PIS

The recommended architecture includes characteristics of each of the previous proposals in a way that makes the overall architecture much stronger and more efficient. 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 candidate #2) instead of 'Ad-hoc Reporting' (from candidate #1). Again, 'OLAP' is more definite than simply letting the PIS have 'Ad-hoc Reporting' capabilities. Another notable inclusion is that information pertaining to payroll information, audit information, and requesting that information were retained instead of grouping all these components into more generic versions. This made it easier to illustrate the interaction between these components in a manner that would be understandable. Other additions include form-based and hash-based authentication, and access control lists (ACL) for authorization measure.

The recommended architecture can well protect the system from the identified possible misuses. Table 2 shows some components that are threatened by the misuse cases as well as what components mitigate the threats.

Further detail is also needed for each mitigating component to truly know what security mechanisms are in place to protect the system. Fig. 6 shows the recommended component interaction architecture with a more detailed explanation of the security mechanisms that are present in order to mitigate the misuse cases. Table 3 shows a detailed explanation of the security mechanisms that are in place for each mitigating component in Fig. 6.

Table 2: Misuse cases, threatened components and mitigating components

Misuse Case	Threatened Components	Mitigating Components
Spoof ID	Main	FireOp, ACL
	LogOnAct	IPRestrict, BioMet
	Admin Tasks	FireOp
View Payroll Info	LogOnAct	IPRestrict, BioMet
	PayInfo	ThrottleReqs, HashAuthen, FormAuthen
	AuditInfo	ThrottleReqs, HashAuthen, FormAuthen
	DataWare	ThrottleReqs, HashAuthen, FormAuthen
Launch DoS Attack	ReqAudit ReqInfo WebBrow	ThrottleReqs, FormAuthen, FireOp

Table 3: Security mechanisms for mitigating components

Component	Security Mechanisms
BioMet	Biometrics use physical attributes of an individual to initiate three approaches of protection. Physical access to resources (logging in) Entitlement to resources (privileges) Recording of forensic information (to use at a later date)
IPRetrict	IP Restrictions are a feature of Internet Information Services (IIS). Any part of a website can be limited so that only certain IP addresses, subnets, and DNS names can access it. Once a user logs into the system, they are only allowed to access certain parts of the site.
ACL	Access control entities (ACEs) contain what a user can do within the system (read, write, create) The access control list (ACL) is referred to once a user has begun the authentication process. An access check is performed against the ACL when a user requests a system resource.
FireOp	A firewall controls communication (flow of packets) to and from a group of networked machines. Based on rules, the firewall determines what is allowed to reach the machines. Firewalls can also modify packets that pass through the network to disguise the address of the machines behind the firewall.
Throttle Reqs	Throttling requests is a measure that aims to simply reduce the number of requests made to the system. A small number of anonymous requests are allowed. A large number of authenticated requests are allowed.
Hash Authen	The main goal of hash-based authentication is confidentiality. This is realized by passing data through a cryptographic function called a hash. This process yields a relatively small value that uniquely identifies the original data. The hash tells nothing about the data; it simply uniquely identifies it. Tampering is combated by comparing the hash attached to the data with a newly computed hash of the same data. If a match occurs, the original data has not been tampered with.
Form Authen	Forms-based authentication is generally an application-specific implementation. This process takes places over a secure SSL/ TLS connection. Login information for each user is stored in a database or XML configuration file.

5 RELATED WORK

Threat modeling (Hoglund, 2004) is a sound approach to addressing software risks at the design level. Based on the decomposition of application, it evaluates the threats and risks to a system and chooses techniques to mitigate the threats. Security threats are modeled by attack trees, which describe the decision-making process attackers would go through to compromise the system. To make the current threat modeling a rigorous engineering process for engineering secure information systems, the missing link is an explicit way for security requirements elicitation and a smooth, traceable transition from security requirements to system design. From this perspective, our work is obviously different from the threat modeling approach. Our approach deals with threat identification and mitigation during the requirements phase rather than the design phase. The security requirements are addressed throughout the process of architecture design and analysis.

Sindre and Opdahl proposed misuse cases as the inverse of use cases to model system behavior that should be avoided (Sindre, 2001a). A misuse case can be defined as a completed sequence of actions which results in loss for the organization or some specific stakeholder. They identified several relations between ordinary use cases and misuse cases, such as includes, extends, prevents and detects. Also they have proposed a general template for misuse case description (Sindre, 2001b). Alexander discussed a variety of applications of misuse cases beyond security requirements elicitation, such as eliciting general "-ility" requirements, exceptions, and test cases (Alexander, 2003). 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 (Alexander, 2002). 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 (McDermontt, 1999) 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. A DAG structure, similar to attack tree in penetration testing, was used to describe abuse cases. McDermott further applied the abuse case based approach to the construction of an assurance argument as a collection of abuse case refutations (McDermontt, 2001). 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 (Kazman, 1996). 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 consisted of 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 (Kantorowitz, 2003). Using this framework, 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.

6 CONCLUSIONS

We have presented the threat-driven approach to the design of secure software architecture, where threats are modeled by 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 in the earlier phases of software development lifecycle can make a system more resistant to vulnerabilities.

Designing software architecture is often a heuristic process even if the requirements specification is available. To make our approach rigorous, we are investigating the formalization of use cases, misuse cases, mitigation use cases, and architectural design. This will allow us to gain high confidence in the system by disproving the existence of identified threats in the architectural design. Another aspect of future work is enhancing the approach with the capability of tradeoff analysis for conflicting functional and security requirements that often exist in real-world information systems. As an integral part of the architecture design process, the tradeoff analysis is of importance for recommending a software architecture for detailed design.

ACKNOWLEDGMENTS

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

REFERENCES

Alexander, I. 2002. Initial industrial experience of misuse cases. In Proc. of IEEE Joint International Requirements Engineering Conference, pp. 61-68.

Alexander, I. 2003. Misuse cases: Use cases with hostile intent. IEEE Software, pp. 58-66 (January/February 2003).

Bittner, K. and Spence, I. 2003. Use case modeling, Object Technology Series, Addison-Wesley, 2003.

Firesmith, D. 2003. Security use cases. Journal of Object Technology, Vol. 2, No. 3, 53-64. (May-June 2003).

Hoglund, G. and McGraw, G. 2004. Exploiting software: How to break code. Addison-Wesley.

Howard, M. and LeBlanc, D. 2003. Writing secure code. Microsoft Press. 2nd edition,

Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G. 1994. Object-oriented software engineering: A use case driven approach. Addison-Wesley, 1994.

Kantorowitz, E., Lyakas, A., and Myasqobsky, A. 2003. Use case-oriented software architecture. CMC03.

Kazman, R., Abowd, G., Bass, L., and Clements, P. 1996. Scenario-based analysis of software architecture. IEEE Software. pp.47-55, November 1996.

McDermott, J. 2001. Abuse-case-based assurance arguments. In Proc. of the 17th Computer Security Applications Conference (ACSAC'O1). New Orleans LA USA, pp. 366-374.

McDermott, J. and Fox, C. 1999. Using abuse case models for security requirements analysis. In Proc. of the 15th Annual Computer Security Application Conference, pp. 55-66.

Sindre, G. and Opdahl, 2001a. A.L. Eliciting security requirements by misuse cases. In Proc. of TOOLS Pacific 2000, pp. 120-131.

Sindre, G. and Opdahl, A.L. 2001b. Templates for misuse case description. In Proc. of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ’2001).

Swiderski, F. and Snyder, W. 2004. Threat modeling. Microsoft Press.

UML 2.0. http://www.uml.org/

Viega, J. and M., Gary. 2002. Building secure software: How to avoid security problems in the right way. Addison Wesley, 2002.