https://doi.org/10.48009/1_iis_2006_150-154

SECURE SOFTWARE DEVELOPMENT USING USE CASES AND MISUSE CASES

Meledath Damodaran, University of Houston-Victoria, damodaranm@uhv.edu

ABSTRACT

There is a need to inculcate in students the idea of secure system development. This paper investigates the application of use cases to the identification of security threats and security requirements; these can then be incorporated into the software design and implementation and used as a basis for testing. The method is easy to teach and easy to use. It provides a highly organized way of thinking about security early in the software life cycle. It can be a tool to inculcate secure software development among students.

Keywords: Use Cases, Misuse Cases, Security Use Cases, Secure Software Development

INTRODUCTION

University curricula traditionally do not emphasize security in system or software development. Courses such as OOP&D, SA&D, and software engineering mainly focus on building software that do something. Yet, security and reliability, and defensive design in general, do not happen by accident. Even software vendors emphasize to their clients how feature-rich the product is, with user-friendliness and convenience thrown in as bonus. Security, reliability and robustness are special considerations and often clubbed together under "non-functional" features. Software developers and architects don't typically think of security as a top priority in the midst of their busy development activities. In order to meet the security problem, businesses end up hiring the best security professionals they can find, well after acquiring or building the systems, as if security is something that can be "bolted on" or "patched in" after the fact. Even academics fail in this area because far too many of their "products" (the graduates) don't think like a hacker while they build systems. Some aren't convinced that security "adds value" to their end products. The result is that the systems they end up building typically have many vulnerabilities. A theoretical understanding of security concepts alone does not result in a change of behavior during system development.

But secure software development instruction need not be such a formidable task. We believe that there is a way to weave in elements of secure and defensive systems development in several classes throughout the academic curriculum. We will suggest one method of doing that in this article.

The outline of the rest of this paper is as follows. After briefly explaining the concept of misuse cases, we outline a method for building a misuse-case based threat and security requirements model. Next, we discuss the potential benefits of this method and also point out a few of its drawbacks. The results of a classroom experience introducing this method to the information systems students in one of our classes is presented in the next to the last section. We close with some conclusions.

USE CASES AND MISUSE CASES

Use Cases

Use cases [4] are very effective in capturing users' and other stakeholders' functional requirements. A use case represents one or more **actors** interacting with the system in order to get a job done. The use case model of the functional requirements of the system consists of a well-organized collection of use cases representing the details of system interaction from the perspectives of the representatives of the different user groups. Each use case typically represents how one user would be interacting with the system. The use case model consists of a use case diagram and the description of each use case using a template. The use case description typically includes a normal scenario (or flow) and one or more alternate scenarios (or flows.) Each scenario is in the form of a sequence of events between the actor and the system.

Use cases are very popular in requirements gathering and specification. It is an important part of modern development methodologies such as Rational Unified Process (RUP.) Their benefits have been well-documented. They are used in software development as well as system engineering/development life cycles [3].

Misuse Cases

Misuse cases [1, 7, 2, 5] represent threats: the multitudinous ways in which an attacker interacts with the system to thwart, break into, damage, abuse, or misuse the system. A misuse case is a use case from the point of view of an actor hostile to the system under design [7]. The goal of a misuse case is not system functionality per se, but a threat posed by a hostile actor to the system functionality represented by the use cases. Secondarily, misuse cases also represent user errors and omissions, accidental or careless. An example will illustrate their use [7]:

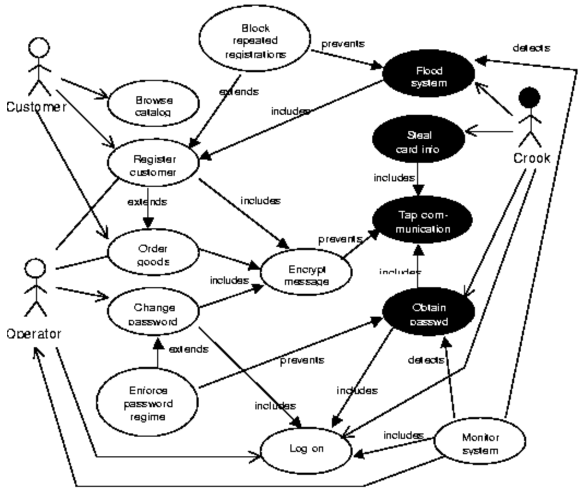

Figure 1. Example of a Use and Misuse Cases

The left column of ovals denote use cases, and they represent the functional features of the system. The black ovals denote misuse cases and represent security hazards. The middle column and bottom left and bottom right ovals denote “security use cases”; these are generated to thwart the threats that the misuse cases represent. We will discuss security use cases in more detail below.

Sindre and Opdahl [7, 8] give detailed examples of how the scenarios in which such 'negative' agents attempt to defeat the system under design can be elicited as misuse cases.

A METHOD FOR BUILDING A SECURITY REQUIREMENTS MODEL

Determination of security requirements traditionally starts with, and is based on, an analysis of the assets to be protected, followed by a risk modeling and risk analysis exercise. An overall security plan guides these activities. Various other methods are also utilized for identifying security requirements. The method suggested in this paper should not be thought of as replacing any of these methods, but instead complementing them. Yet, regarding the traditional emphasis in asset-based risk analysis, it should be noted that although it tends to do a very good job of identifying the threats against the assets, a dimension it omits is the services and features that need to be protected. Misuse case based threat identification directly addresses this omission. Hence, they fill in where the traditional methods are weak.

We suggest a method for building threats and security requirements from use cases and misuse cases:

1. First identify **actors** (representing user classes) and build a comprehensive set of **use cases** as usual.
2. For each use case, brainstorm and identify how 'negative' agents would attempt to defeat its purpose or thwart some of the steps in the use case description; this leads to the major **misuse cases**. During the brainstorm sessions the focus should be to identify as many ways an attacker could cause harm in the service provided by the use case in focus; details of such attacks may be determined later. Each of these modes of attacks becomes a candidate misuse case. The goal is to identify security threats against each of the functions, areas, processes, data, and transactions involved in the use case from different potential risks such as unauthorized access from within and without, denial of service attacks, privacy violations, confidentiality and integrity violations, and malicious hacking attacks. In addition to modes of attacks, the process should also try to uncover possible user mistakes and the system responses to them. Often these mistakes could cause serious issues in the functioning or security of the system. By identifying all inappropriate actions that could be taken, we would capture all actions of abnormal system use—by genuine users in terms of accidental or careless mistakes and by attackers trying to break or harm the system function.
3. Show the **relationships** between each use case and the corresponding misuse cases in a diagram such as Figure 1. Use of words such as “threatens” and “steals” would be found useful to show these relationships as portrayed in Fig. 1.
4. After the misuse cases have been constructed, identify **security use cases** to counter or thwart the intended purpose of each misuse case. A simple example is one in which we would construct a new security use case called “Encrypt the Message” to thwart the “Tap Communication” misuse case (see Figure 1.) Note that we called these new use cases “security use cases,” as they do not represent functional requirements of the system per se (no user or

stakeholder probably ever asked for encryption, for example.)

5. Continue steps (2) through (4) for each major use case until one is satisfied that (a) all reasonable threats to the basic functionality and services of the system (as represented by the use case model) are identified and represented as misuse cases and (b) each of these threats has been thwarted by one or more newly introduced "security use cases."

Microsoft's new threat modeling method gives several useful guidelines for identifying threats along use cases [6].

A short example, provided by my student Chandramohan Muniraman, will illustrate the method. Consider the system function of providing access to users of the system. Suppose that users access a login form, enter their account name and password, and request access to the system. The system verifies their credentials, authenticates them, and provides them access by way of a form with options for performing further actions. So this essentially is the **use case** for this function—specifying a business or system requirement.

The **misuse cases** in this case are what an attacker would be doing with this function of the system. They may try to (1) gain unauthorized access to the system by password guessing, (2) intercept the communication messages and find out the account details, (3) flood the system with access requests and cause denial of service attack. These become the misuse cases for this normal system function.

The next step is to prevent these misuses of the system by identifying security features, in the form of **security use cases**, to protect the system from the above threats. These may include the following: (1) System should lock the account after a few unsuccessful attempts and provide some easy and safe means of resetting the password for the actual account user; (2) Encrypt messages during transit; (3) A stateful inspection of some kind should detect repeated requests from a source system or user and prevent them or hold off actual connection until further responses could be verified from the requestor.

DISCUSSION

The discussion will mainly focus on the major benefits and shortcomings of the suggested method as the basis for identifying threats and more generally as the basis for secure software development. As far as benefits are concerned, we see many. First and foremost, the use case method is one of the most popular methods for eliciting requirements. It is a central part of the Unified Process and UML, for example. Since use cases are developed anyway as part of the system development artifacts, it makes sense to use them as a means of discovering a major portion of the security requirements. As mentioned before, other traditional and important approaches such as asset-and-risk based methods should continue to be used. Microsoft's newly released threat model [6] describes several such complementing approaches.

A claim may be made that information about most, if not all, security-related requirements is hidden in the use cases, as use cases capture all user-based system functionality. If that is so, a complete and comprehensive use case document should give us an excellent starting point to derive most of, if not all, the security requirements.

Use cases can be used as a basis for much of the testing. Similarly, security oriented requirements may be used as the basis of security-oriented testing. It should be noted that security oriented testing such as risk-based testing and extensive penetration testing is often omitted due to cost and schedule constraints in many development projects.

We believe that it is easy to teach this method to a novice developer or a student— those without much background in security or experience in development. For example, it should not be too difficult for a novice to ask the following for each use case: What are the different ways to (a) abuse this? (b) make this use case not work? or (c) have something go wrong in the flow of events in the normal or alternate flows that constitute the use case description?

The approach, though simple, provides us with a methodical handling, modeling, and specification of functional and adversarial usage, rather than the haphazard approach that is common place in many security threat identification exercises.

Threat modeling is gaining momentum. For example, Microsoft offers free threat modeling tools and resources, and a new portal "Threats and Countermeasures" has just been launched. There is an elevated level of concern for this activity in all industries and university programs in computing. But threat modeling is often thought of as an abstract and difficult concept. Techniques such as the use-case based threat modeling described here, along with traditional asset-and-risk based methods make it possible for anyone to do a secure system design reasonably well.

As already mentioned, university computing and information systems programs have a difficult time adequately teaching security as a topic in an already overcrowded curriculum. There are far too many topics to be covered in programming and system development classes, and even courses such as operating systems and database find it difficult to adequately cover computer and information security. Yet, covering misuse cases right after doing use cases in a software engineering or SA&D class is a practical and natural way to introduce security and inculcate security-thinking in students. Inclusion of security requirements in the SRS document may be made a mandatory requirement in student projects. These efforts take very little additional class time, and the benefits in the students' becoming security-conscious are certainly worth that extra time. The topic may be covered in a beginning OOP&D class as well, with the result that from the very beginning students have a "security" frame of mind.

There are a few potential problems or limitations with this method for eliciting security requirements. The first is that one should not rely on only this method, but use other conventional methods of eliciting security requirements also. Examples include data oriented methods, identifying threats along data flows, Threat/Attack Trees [6], identifying threats and attacks unique to the application domain (such as web application,) and the commonly used asset-and-risk based methods already mentioned.

Another possible criticism could be that secure software development is much more than constructing a set of misuse cases along with a set of security requirements to counter the threats that they model and incorporating them in the design, implementation and testing of the system and its artifacts. Secure software development and secure programming involves understanding and managing software-induced security risks, language-based flaws and pitfalls, and subjecting all security related artifacts to thorough objective risk analyses and testing [5.] Secure software development involves knowledge of coding errors and how to avoid them, formal risk analysis, penetration testing, and code reviews, among other topics. The method outlined in this paper does very little in these important areas.

Yet another shortcoming of the method is that there are no standards, guidelines or generally accepted practices on what constitutes a good, quality set of misuse cases, or use cases, for that matter. People develop vastly different types of use cases, some high level and some with lots of detail, for example. Use/misuse cases are not formal techniques. The subject is very new, and there is not a vast body of literature documenting successful applications of this method in real world settings.

A CLASS EXPERIMENT

"Information System Security" is a beginning level graduate class in the Master's program in Computer Information Systems at University of Houston-Victoria. The author taught this course in the spring of 2006 as a fully online class. "Secure software development" is the last topic listed in the class schedule. This topic is not covered in the textbook. Most students were not familiar with use cases. Even though software engineering project management and SA&D are core courses, they are not prerequisites for this class.

An assignment was given about two-thirds into the semester with the objective of introducing them to (1) some of the material on secure software development and (2) a hands-on experience in developing security requirements using the method outlined in this paper. The assignment required them to complete 3 steps: (1) Go through a PowerPoint presentation on use-case based requirements gathering. The presentation included the example of the development of the use case model of a customer's shopping experience on Ebay. (2) Read the original version of this paper. (3) For the Ebay example introduced in the use case presentation, develop a comprehensive set of misuse cases; security use cases (in response to the misuse cases;) a diagram showing the use/misuse/security use cases, relationships among them, and the actors involved; and a description of each of the misuse cases and security use cases using a template.

Space limitations do not allow us to go into the details, but the results met this instructor's expectations. Of the twelve students left in the class when this assignment was given, five mastered the method described in this paper, applied it to the Ebay problem, and developed a comprehensive set of security requirements as per the instructions. Two students showed their understanding of the material but for some reason gave only one misuse case and one security use case each (instead of a comprehensive set of these, as was asked.) Two of the students showed partial understanding of what they had read and were not able to apply it at all. One, a normally bright student, did not understand what was required to be delivered, and did not seek help from me, probably due to his very late start!

Finally, three students either did not attempt the assignment at all, or did not make enough progress on it to submit anything.

I feel that this was a somewhat unorthodox assignment for these students. There was a perception that much was expected of them in this assignment: studying use case based development and security requirements using misuse cases and security use cases. In hind sight, this may have caused a few students to probably feel intimidated at the outset and give up. Nevertheless, three students appeared to really like the assignment, and a majority of them did master the technique to a degree they could apply it. Lastly, the results may have been different if the class were a face-to-face offering and not an online one.

CONCLUSION

Misuse cases appear to be a very effective tool for identifying the security threats corresponding to the functional requirements of the system as represented in the use case model. They form the basis for constructing a set of “security use cases” that counter each of the threats. The step-by-step method described here is easy to teach and appears easy to use. It provides a highly organized way of thinking about system and software security. Until the implementation of use/misuse cases, security for many of the students and practitioners may have been more or less an abstract concept. With misuse cases, they have a practical technique to identify and include in the design a set of security threats and a set of remedies for these security threats. The experience of using such a method can help inculcate secure software development in students. For them and for their future employers, security is bound to remain a matter of serious concern.

For widespread industry-wide applicability of this method for security requirements elicitation and specification, the method would need to be incorporated into existing CASE tools. Also, there should be standards and guidelines for their use. When software developers and IT managers start to realize the benefits and cost-effectiveness of this relatively simple method, or methods such as this, we expect the frequency of their use to increase.

ACKNOWLEDGEMENT

My thanks are due to my student, Mr. Chandramohan Muniraman, for permission to include his example in this paper.

REFERENCES

1. Alexander, I.F. (2003). Misuse Cases: Use Cases with Hostile Intent, *IEEE Software*, January, 58-66
2. Firesmith, D. (May-June 2003). Security Use Cases, *Journal of Object Technology*, *2*(3), 53-64. Available: http://www.jot.fm/issues/issue_2003_05/column6
3. Hruschka, P. & Rupp, C. (2001). Echt Zeit' für Use-Cases (German), *OBJEKTSpektrum*, 4, 63-70.
4. Jacobson et al (1992). *Object Oriented Software Engineering – A Use Case Driven Approach.* Boston, MA: Addison-Wesley.
5. McGraw, G. (2006). *Software Security: Building Security In*, Boston, MA: Pearson Education
6. Meier, J.D., Mackman, A., & Wastell, B. (2005). How To Create a Threat Model for a Web Application at Design Time, Microsoft Corp. Technical Report http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/tmwahowto.asp
7. Sindre, G., & Opdahl, A.L. (2000). Eliciting Security Requirements by Misuse Cases, *Proceedings of TOOLS Pacific*, 20-23 November, 120-131
8. Sindre, G., & Opdahl, A.L. (2001). Templates for Misuse case Description, *Proceedings of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality*, Interlaken, Switzerland, 4-5 June