https://doi.org/10.48009/2_iis_2007_193-199

A PRACTICAL APPROACH TO INCLUDE SECURITY IN SOFTWARE DEVELOPMENT

Chandramohan Muniraman, University of Houston-Victoria, *chandram@houston.rr.com*
Meledath Damodaran, University of Houston-Victoria, *damodaranm@uhv.edu*

ABSTRACT

This paper explores the question of, and suggests some specific methods for, building security into an application software system, starting from the requirements gathering phase and continuing on with each of the subsequent phases of the software development life cycle: requirement analysis and specification, design, development, testing, and implementation.

Keywords: Application Security, Software Development Life Cycle (SDLC), Designing and Building a Secure Application, Security Threats.

INTRODUCTION

In the early phases of developing a new application software system only the functional requirements and user constraints are generally addressed. Security issues are typically considered only at the end of design phase or sometimes in the implementation phase. Even here only a few measures to assure minimal protection from common, well-known security threats are incorporated. Here too, the security features of the operating system, data bases, and the operating environment of the users are often relied upon and considered as enough protection in most cases.

Much later, security threats might be discovered or actually faced, and expensive additional protections and measures might be added. These would be relatively expensive, because of the design constraints and the limited options available at this stage. Also, these protections may not seamlessly and effectively integrate into the system, and maintenance of these systems would become difficult after several such upgrades.

But application software systems should handle many of the basic security functions independently and not delegate it to the operating environment or the users. Security needs have to be addressed both in terms of protecting the system and data from unauthorized access, and with a view to providing and enforcing business controls as defined in the organization's security policy. The enterprise and OS level security should be viewed as complementary to application security. It is good to not rely on any one of these as the sole level of security for the system.

The purpose of this paper is to describe some practical ideas and approaches to consider during the SDLC of a software application in order to weave in security naturally as a part of the process. Specific methods to build security into an application software system are described, starting from the requirements gathering phase and continuing on with each of the subsequent phases of software development life cycle: requirement analysis and specification, design, development, testing, and implementation. Various researchers have studied different *aspects* of secure application development, but with the exception of the excellent but relatively voluminous treatises, books by McGraw and Viega [7,11], we have not found a comprehensive yet concise description on how to weave in security throughout the SDLC of a software application, a guide to the software engineer or systems analyst. Our purpose is to provide such a practical and effective guide. Some of the ideas that we propose are common sense, some may be generally known practices, some may have been considered or brought out by others. Except for the cited parts, we have not tried to gather together others' ideas. An exception is the section on threat modeling; this is a review and very little of it is our own; we did pull together relevant work from others for this section, as indicated.

REQUIREMENT GATHERING

If we want to develop a secure system, security will have to be built into it right from the beginning, the requirement gathering phase [9]. Software requirements should include security requirements at a high level in various functions, processes, components, transactions and data in the system. In doing so, the security policy of the company should be considered, and its requirements adhered to.

For gathering security requirements, some of the same techniques and tools that are used for gathering general business requirements may be employed, and from the same people: business users, management, and other stake holder groups. In addition, people from other infrastructure IT groups, such as system administrators and database administrators, enterprise security team, internal audit, and legal departments could be consulted, for gathering additional requirements, risks or needs for building security into the application.

Many different methods could be employed for capturing security requirements needed to secure each of the business functions, areas, processes, and transactions from different potential risks -- such as unauthorized access from within and outside the network, denial of service attacks, and malicious hacking attacks. The specific methods and techniques to be used depend on the particular system, constraints, and available resources. Risk analysis is an important tool for this purpose, but due to space limitations and because of the fact that it is well-known, we will not discuss this here. Here we discuss two methods that could be used to identify the security requirements of a business or enterprise application:

- Use and abuse cases
- Threat modeling

Use and Abuse Cases

Use cases [6] define functional requirements of a specific module of the system. The use case model represents the normal usage of the system function. But an attacker will not be using the system in this normal sense; attackers will try to misuse or abuse the normal system function in order to break the system. So thinking how an attacker could abuse or circumvent a normal system function provides us with a set of risks or threats for this function, represented as abuse cases [1,10,5,7,4]. These abuse cases are then used to develop a set of security features, each designed to protect the system from the corresponding abuse. This set of security features will become the security requirements, represented as security use cases. This process can be used to also uncover the user mistakes and the system responses to them. Often these mistakes could cause serious issues in the functioning or security of the system. By defining all adverse actions that could be taken by a user, we would capture all actions of abnormal system use, whether it be by genuine users making accidental mistakes or attackers trying to break the system function.

For example, one of the normal functions of the system could be providing access to users of the system. This could be in the form a login form, wherein users will access the form, enter their account name and password, and request access to the system. The system will verify their credentials, authenticate them and provide them access by way of a form with options for performing further actions. So this becomes the use case for this function, specifying the business requirement.

The abuse cases are what an attacker could do with respect to this system function. For example, they may try to:

- Gain unauthorized access to the system by password guessing
- Intercept the communication messages and find out the account details
- Flood the system with access requests and cause denial of service attack

These are then the abuse cases corresponding to one normal system function. The next step is to prevent these abuses from successfully launched, by implementing security features to counter them. For our example, the following are some possible security requirements:

- System should lock the account after a few unsuccessful attempts; and should provide easy, safe means of resetting password for the actual account user
- Cryptographic authentication systems should be used, which encrypt messages
- Need a way to identify repeated request from a source system or user and prevent them; or could hold off actual connection until further responses could be verified from the requestor

In a complex system with many functional modules and different entry points to each of them, the large set of security requirements so discovered need to be integrated and organized, avoiding duplicate or conflicting requirements, which would be a complicated task.

Fig. 1 illustrates use cases, abuse cases, and security use cases derived from the abuse cases, for the example of an online bidding/buying system. The use cases correspond to the normal system usage described in terms of users' actions and interactions with the system. The abuse cases represent the various actions that could be taken by users hostile to the system, with the intent to intrude or hack into the system, capture sensitive data in the communication packets, or in some other way compromise or harm the system. Then the security use cases are derived from these, so that these hostile actions in abuse cases could not be used to attack the system.

<<insert Figure 1 here>>

Fig. 2 uses a use case template to describe a security use case derived from an abuse case for the online bidding/buying system.

<<insert Figure 2 here>>

Threat Modeling

Threat modeling is another technique that can be used to identify, explain and record all threats from every possible source which could attack the system [3]. This will focus the attention on the serious threats that threaten the environment and the system, rather than looking for all general possible ways or all weaknesses in the system. After defining the threat model of the application system, means of mitigating them could be designed.

Threat models have been studied and various approaches to it have been proposed, see [3, 2, 8] for example. Microsoft [8] provides a threat modeling tool to create application threat model documents, with data entry points, assets, data flow diagrams, threats, threat trees and vulnerabilities in a tree view. In addition, the article gives several useful guidelines for identifying threats along use cases.

To create a threat model for an application its intended functions and architecture must be known, which could be obtained from some of the following design documents: use cases and usage scenarios, data flows, data schemas, deployment diagrams [8].

The steps in defining a threat model for an application system are: (1) Identify Security Objectives, (2) Create the application overview, (3) Decompose the application, (4) Identify threats, and (5) Identify vulnerabilities.

Space limitations require us to refer the reader to [8] for details. Here it will suffice to show Figure 3 below. This shows a threat model for identity theft. In this, the boxes with blue lines might be eliminated during the risk evaluation process.

<<insert Figure 3 here>>

REQUIREMENT ANALYSIS AND SPECIFICATION

In the analysis phase, all the specified security requirements are evaluated to see if these are really required for the security of the proposed application. Sometimes a few features provided by the operating system or the users' operating environment may be sufficient, especially if the security risks or issues are non-critical. Also the budget outlays and allocations might need to be taken into consideration along with the criticality of the risks and the value of the application protected, while considering security solutions, to justify the cost. Maintenance efforts and cost may also need to be taken into consideration.

Database, front end forms, server, middleware and network should all be included in the analysis from the view of providing complete and adequate security to the application and its users. An application vulnerability could be exploited to gain access to the server or the network hosting it and then eventually gain control of it, to delete files, access critical data or impact other applications.

For an application to perform its normal functions it may need a root or super user level access to the operating system, since it may need to connect to server and perform certain operations like accessing system files and writing log or output files. This part of the system needs to be very secure for these very reasons.

If the application system uses databases, access to database should be protected and tables should have audit trail columns. All appropriate database security features should be considered and incorporated and utilized in the database design, as appropriate. Access to the database should be controlled by database access control also, such as assigning roles and granting them individual accesses. Other middleware server issues such as session hijacking should also be considered and protected by encryption technologies.

DESIGN

In this phase all the requirements are carefully considered while designing a solution based on the cost, available technologies, company standards and security policies, and compatibility with OS and other existing systems. The inherent vulnerabilities and features of the proposed application system, platform, operating system, development tools and hardware are considered along with the specification requirements to design security into the application. The following are some issues that need to be addressed while building security into the system being designed:

- **Role based data access levels and access controls.** Database tables should be striped by a logical or physical grouping of the company's organizations, departments, business groups or countries. This will facilitate providing access to data for their respective groups easy from the front end user interface.

- **Architectural design.** The complete design of the system should be broken down into individual modules, functions and processes. This may be in terms of the following groupings: (1) Business, such as the parts of the system for different business units, (2) Logical, such as in common components of the system, (3) Physical, such as for similar actions of the units of the system, (4) Technological, such as utilizing similar technology for system components. The inherent features, vulnerabilities and issues in the components of the application system such as software languages, tools, operating system and their implementation options chosen, must be addressed here, and analyzed to design the optimal solution for satisfactory security performance.

- **Functional design.** Conflicting requirements, exceptions of the general security policies and special requirements, such as providing access to a third party system interface for a specific business process or reading data from an external system interface, must be addressed at this stage and all special security requirements for these should be designed.

- **Security Policy for the Application.** Based on the threat model for the application system, an acceptable use policy must be developed for the application system and its various features and functions [2]. All practices, procedures and system administration procedures must be specified. This must agree with the general policies and procedures of the company. If there are additional special requirements these must be added to the policies.

DEVELOPMENT

Development phase may include creation of databases, programs, forms for user interface, and reports. Each development area should build security requirements designed for that function as per the requirement specification. Different technologies, language features, tool or other third party solutions may need to be incorporated into the system. The following are some security issues to be addressed while the system is being developed:

- If, due to some constraints in the development tools, languages or OS, it is not be possible to develop the security features exactly as designed, alternatives should be considered to ensure that the original requirement specifications are met.
- Programmers and developers need to be educated about the security implications of the system and the security design so that they are fully security conscious, and handle common, easily avoidable errors such as buffer overflows by testing their components with appropriate "odd" input data [7].
- Features designed for security operations, such as logs and audit trails, will actually be developed in this phase, and secure locations and methods for these have to be utilized.
- Integration of various components and their security features will be a challenging task, especially if there are components developed by third parties or from the market – such as cryptographic systems and identity management systems

TESTING

The security requirement specifications should be used as the basis for unit testing, integration testing, and system tests. All phases of testing should all include separate testing guidelines for testing the security aspects. Security audit of the individual system may also be performed, specifically from outside the network. The platform and OS vulnerabilities, and security strength of database implementation may need to be tested. Comprehensive application risk analysis and specific testing of the security implementation should also be performed.

Security testing tools may be used to reduce the manual efforts, but different tools may need to be utilized for testing security threats in different areas, because threats differ with environments and a single tool would not be able to help analyze all the different types of threats.

Some additional points important to consider in the testing phase include:

- **Code review.** Peer code review will be useful in eliminating many overlooked potential issues [7]. But in case of systems with a lot of source code programs to review, this could be a very tedious job and could become mechanical. A commercial source code analysis tool could prove useful for this analysis. Some of these products are from vendors such as Secure Software, Fortify, and Coverity [7]. Integrated Development Environment (IDE) tools might also have integrated source code analysis tools built into them.

- **Turning off or removing unnecessary options and features.** The security review should also reveal all the configurations, options and features that are not necessary for the designed system, and which could pose potential security risks and should be removed or turned off. Some of these could be changes to server and other system settings that could make them vulnerable, and other code based features that could be accessing other systems or providing access to them, in an insecure manner.

- **Integration of third party solutions**. Security of all third party products, tools and solutions should confirm to the needs and standards of the system developed. These solutions would be connected or integrated into the system by interfaces, APIs, remote program calls or web services. These integration points could be the weak links, which if compromised could pose a threat to the entire system and its host. In these cases security should not be compromised for development efforts and time, cost or ease of access.

IMPLEMENTATION

During implementation of the application system, the implementation decisions should be based on the design specifications, and the intended functionality and requirements. These include setting correct options, enabling intended features, and configuring the systems. After the implementation, individual configurations and units should be tested and evaluated. Special consideration should be given to critical security systems and integrations with other systems such as: cryptographic systems, identity management systems, database connections, and interfaces and integrations with other systems.

Security Audit Testing

After implementation in the production environment actual security testing should be performed [9]. This could be internal, external or both. The initial audit could serve as a baseline for further improvements or maintenance. Afterwards, periodic security audit and evaluation should be performed based on the security policies and procedures.

Change Management

Configuration and source code management systems would go a long way in controlling, and managing modifications, updates and upgrades to the system, in terms of changes to configurations, programs, forms or reports. Controls should be enforced for testing the changes in a development environment and end user testing in a user acceptance environment, before moving them into production environment.

SUMMARY

Building security into a system should start from the requirements gathering, specification and design phases of the software development life cycle, and continue into development, testing and implementation. This would ensure that security is carefully considered and designed into the system and not just added in a haphazard or piecemeal fashion. Security requirements may be derived from functional requirements by using the technique of abuse cases and security use cases. This provides a straightforward approach to look at each function, analyze its normal usage requirements, determine the potential risks in terms of system abuse, and finally provide safeguards against these risks in the form of security requirements. Threat modeling is another technique to discover security requirements.

Security and security features design should consider the inherent vulnerabilities in the development language, tool or operating system. Database accesses and interfaces to and from other systems should be designed based on the environment, and the risks involved. Different parts of the system processes and transactions should be analyzed, and secured based on the sensitivity of the data by implementing encryption, authentication and identity management systems. Comprehensive application risk analysis and specific testing of the security implementation should be performed. During implementation the configurations options should be implemented based on the original design and intent. After implementation periodic comprehensive security audit must be performed. The result of all this is an application system in which security is not an after-thought but an essential feature.

REFERENCES

1. Alexander, I.F. (2003). Misuse Cases: Use Cases with Hostile Intent, *IEEE Software*, Jan., 58-66
2. Apvrille, Axelle and Pourzandi, Makan. (2005). "Secure Software Development by Example," IEEE Security & Privacy, vol. 3, no. 4, July/August, 2005, pp. 10-17. http://www.computer.org/portal/site/security/menuitem.6f7b2414551cb84651286b108bcd45f3/index.jsp?&pName=security_level1_article&TheCat=1015&path=security/v3n4&file=apvrille.xml&. As obtained on March 15, 2007.

3. Burns, Steven F. (2005). Threat Modeling: A Process to Ensure Application Security. GIAC Security Essentials Certification (GSEC) Practical Assignment. http://www.sans.org/rr/whitepapers/securecode/1646.php . As obtained on March 15, 2007.
4. Damodaran, Meledath. (2006) Secure Software Development Using Use Cases and Abuse cases. *Issues in Information Systems*, vol. 7, no. 1, 150-154. http://www.iacis.org/iis/2006_iis/PDFs/Damodaran.pdf. As obtained on March 15, 2007.
5. Firesmith, D (2003). Security Use Cases, *J. of Object Technology*, vol. 2, no. 3, May-June, 53-64. www.jot.fm/issues/issue_2003_05/column6. As obtained on May 29, 2007.
6. Jacobson et al (1992). *Object Oriented Software Engineering – A Use Case Driven Approach.* Boston, MA: Addison-Wesley.
7. McGraw, Gary. (2006) *Software Security-Building Security In.* Addison-Wesley.
8. Meier,J.D., Mackman, Alex and Wastell, Blaine. (2005). How to Create a Threat Model for a Web Application at Design Time, MSDN, Microsoft Corporation. http://msdn2.microsoft.com/en-us/library/ms978516.aspx. As obtained on March 15, 2007.
9. NIST. (2004). National Institute of Standards and Technology, Computer Security Division, Information Technology Laboratory. *Security Considerations in the Information System Development Life Cycle*.
10. Sindre, G., & Opdahl, A.L. (2000). Eliciting Security Requirements by Misuse Cases, *Proceedings of TOOLS Pacific*, 20-23 November, 120-131
11. Viega, John and McGraw, Gary. (2001) *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley.

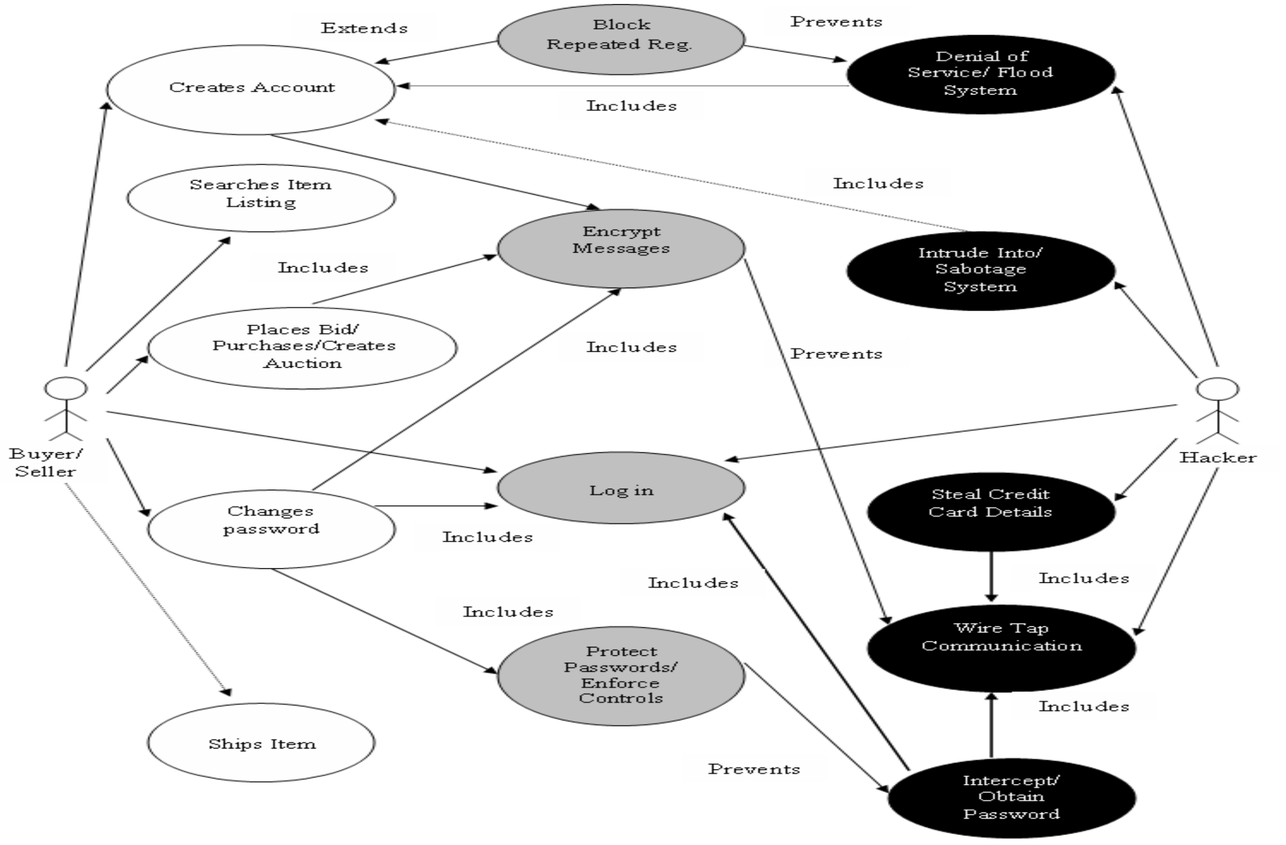

Figure 1. Shows use cases at left, misuse cases at right and security use cases in the middle

Use Case ID:	UC-1		
Use Case Name:	Denial of Service/Flood System		
Created By:	CM	Last Updated By:	
Date Created:	04/08/06	Date Last Updated:	

Actor:	Hacker/Intruder
Description:	A hacker uses the internet to connect to the website and register with different names and thereby tries to flood the system and prevent it from servicing the requests of genuine buyers and sellers.
Preconditions:	The website should be available and accepting connections from the internet. Other than this no other conditions are required.
Postconditions:	1. The system creates an account and stores the details in its database, and displays the success or failure of the operation. 2. This utilizes a lot of system resources, such as memory, processing power and database I/O. 3. This results in lesser resources are available for additional genuine buyers and sellers tying to create accounts in the system
Priority:	High
Frequency of Use:	Potentially high, could happen anytime
Normal Course of Events:	Actor actions — System responses 1. Hacker connects to website; — 2. Home page displayed 3. Access registration form; enters minimum required info; — 4. Some validation of user And inputs; 5. Submits user creation request; — 6. Processes, creates user and shows results.
Alternative Courses:	1. May access other pages/info; 2. System will show these info
Exceptions:	1. May enter invalid/duplicate input info; 2. System may fail to create user;
Includes:	Create account
Special Requirements:	Normal system functions are sufficient
Assumptions:	Hacker has intention to harm the system, resources and potential to cause these actions
Notes and Issues:	This is actually an abuse case

Figure 2. Description of a Security Use Case Using a Use Case Template

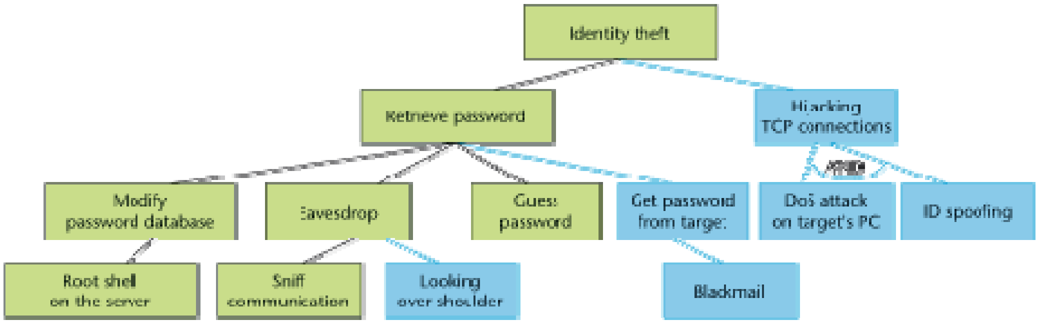

Figure 3. Threat model for identity theft [1]