Master Thesis
Software Engineering
Thesis no: MSE-2007-12
March 2007

Secure Software Development

- Identification of Security Activities and Their Integration in Software Development Lifecycle

Syed Rizwan Ahmed

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Syed Rizwan Ahmed
E-mail: rizwanahmed24@yahoo.com

University advisor(s):
Dr. Bengt Carlsson
Assistant Professor
Department of Systems and Software Engineering

School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : + 46 457 271 25

ABSTRACT

Today's software is more vulnerable to attacks due to increase in complexity, connectivity and extensibility. Securing software is usually considered as a post development activity and not much importance is given to it during the development of software. However the amount of loss that organizations have incurred over the years due to security flaws in software has invited researchers to find out better ways of securing software. In the light of research done by many researchers, this thesis presents how software can be secured by considering security in different phases of software development life cycle. A number of security activities have been identified that are needed to build secure software and it is shown that how these security activities are related with the software development activities of the software development lifecycle.

Keywords: Software, Security, Software Development Life Cycle, Building secure software.

TABLE OF CONTENTS

1 INTRODUCTION

Software industry has seen some phenomenal growth over the last decade and this growth is continuing with rapid pace. Over the years the software industry has excelled in every field. It is hard to think of any field that does not involve the use of software. Whether it is business, sports, media, defense, or exploratory missions, the need of software support can not be turned down. Software is directly or indirectly involved in storing, controlling, communicating, presenting, computing and even generating information. Use of software has revolutionized the world in every aspect.

Today software control large financial systems, communication systems, databases and even deadly missile programs. Misuse of software in financial sector can lead to heavy economic loss; misuse of software in communication sector can lead to communication sabotage, misuse of software in databases can lead to critical data theft, and worst of all could be the misuse of missile controlling system that could endanger human life. The roles and functionalities, the software performs, and their nature makes them vulnerable to attacks. Software is growing in terms of size, complexity, extensibility, and connectivity [1]. Windows 3.1 contained 3 millions line of code while Windows XP contains 40 million lines of code [1]. Java and .NET platforms are extensible in nature, and are designed to accept mobile code update and extensions [1]. Use of LAN, WAN and internet has stretched connectivity into new dimensions. The attacker can now exploit the extensible property of the system by sitting at a distance. With the increase of involvement of software in almost every field, the amount of control going into the hands of software, changing nature of software, and the changing environment surrounding them, one should not be surprised to see the increasing demand for security in software.

A secure system does what it is supposed to do and restrains from doing anything that it is not supposed to do. The main aspects of security are Confidentiality, Integrity and Availability [2]. These three aspects also are referred as CIA. Confidentiality is all about maintaining privacy, i.e. prevention of unauthorized disclosure of information. Integrity is about ensuring the accuracy and completeness of information, i.e. prevention of unauthorized modification of information. Availability is about ensuring the availability of information to authorized hands. Any software that enlists these three main aspects described above can be considered as secure software.

Organizations usually consider security as a post development activity. Security is not given consideration during the pre-development and development phases. Organizations do not realize that "*Software security is an emergent property of a complete system, not a feature* [1]". After the completion of software development, organizations try to incorporate security as a patch [1]. Furthermore organizations spend lot of money in purchasing good firewalls and antivirus programs and consider that this outer shield is enough for making software secure. However their current approach is not working and organizations continue to incur heavily losses due to exploitation of security flaws.

From the above discussion it can be seen that securing software in post-development manner is not good enough and there is a need for finding out better ways and means for securing software.

1.1 Purpose Statement

The purpose of this thesis is to explore how security can be incorporated in software and how secure software can be developed.

1.2 Intended Audience

Software security is relatively a new area and is attracting both industry and researchers. The industry is interested because it wants to develop secure software to avoid heavy losses it incur as a result of insecure software. Researchers are interested because this area has not yet matured and there is plenty of room available for improvement and innovation.

This thesis targets both academia and industry. In academia this thesis can be used to understand software security and to understand how security activities are related with the software development activities in the software development lifecycle. In the industry, software development organizations can follow the proposed security development activities in secure software development lifecycle for building secure software.

1.3 Aims and Objectives

Based on the purpose statement the aim of the thesis is to find out how security can be incorporated in software. For this purpose following objects are required to be met.

- Identify the need of software security
- Understand security in the context of software and understand how software security differs from other forms of security, for example, information security, application security etc.
- Investigate current practices of organizations towards software security
- Investigate if there are problems in current approach of organizations towards software security. If yes then what are those problems?
- If there are problems in current approach of organizations towards software security, then find out how organizations can develop secure software.

1.4 Research Questions

Based on the objectives, the following research questions have been formulated:

1. How do organizations usually perceive software security?
2. Are there problems in the way organizations perceive software security? If yes, then what are those problems and how organizations can overcome those problems?
3. What should organizations do in order to develop secure software?

1.5 Research Methodology

A detailed and comprehensive literature survey was carried out to achieve a set of objectives. The literature survey was based upon articles from authentic sources like IEEE and ACM, proceedings of conferences, books and authentic web sources.

1.6 Structure of Thesis

This section describes the structure and the contents of the thesis.

Chapter 2 i.e. ‘Software Security’ explains the term ‘software security’ and differentiates it with other closely related forms of security. It explains the need of software security and discusses some of the popular security flaws. This chapter basically sets the background for later chapters.

Chapter 3, i.e. ‘Security Improvement Activities’ proposes activities that need to be performed to build secure software. This chapter mainly focuses on literature survey. Proposed activities are represented in graphical form to achieve better understanding of the reader.

Chapter 4 i.e. ‘Security Activities in Software Development Life Cycle’ explains the activities identified in chapter 3 in relation with the software development activities of the software development lifecycle. This chapter uses diagrams to achieve better understanding of the reader. At the end of the chapter, all of the security activities are represented in graphical form in one complete picture.

Chapter 5 i.e. ‘Discussion’ focuses on answering research question and explains the contribution of this thesis in the knowledge domain. Conclusion is presented in chapter 6 while future work is presented in chapter 7.

2 SOFTWARE SECURITY

This chapter presents the concept of software security stated in the literature. This chapter will serve as context building for upcoming discussion about improving security in the next chapters.

2.1 The Notion of Software Security

Here is a look on some of the definitions of software security:

- "*Software Security is the ability of the software to resist, tolerate, and recover from events that intentionally threaten its dependability*" [5]
- *"Software Security is about building secure software: designing software to be secure, making sure that software is secure, and educating software developers, architects, and users about how to build secure things"* [4]
- *"The idea of engineering software that continues to function correctly under malicious attack"*[6]
- *"The process of designing, building, and testing software for security"* [4]
- *"Defends against software exploit by building software to be secure in the first place, mostly be getting the design right (which is hard) and avoiding common mistakes (which is easy)"* [14]
- *"Software Security is system-wide issue that takes into account both security mechanisms (such as access control) and design for security (such as robust design that make software attacks difficult)"[4]*

There is not even one standard definition of software security. Different people have defined security in different terms. The important thing to notice in the above definitions is that most of them talk about 'building secure software', instead of 'securing software'. Building secure software actually means to design and implement secure software while securing software aims to first build the software and then secure it. The concept of software security will be explained in more detail later in this chapter.

2.2 Need for Software Security

Software security has been a neglected area for quite sometime. This does not mean that the issue was never raised before, but it was underestimated, misunderstood and was not practiced the way it should have been. Security books began publishing in the year 1999, where researchers and practitioners started talking about how security can be built into the system [3]. The whole idea was to incorporate security into the development process. Security is often considered as an add-on feature that can be incorporated once the development cycle is complete. In most organizations, security is considered as the domain and the responsibility of the infrastructure people who maintain antivirus programs, firewalls and intrusion detection systems [4]. Infrastructure people are not system developers, designers or architects. Given the role of security individuals, they try their best in setting up the best firewalls, antivirus programs, etc.

System developers, designers, architecture and requirement analysts are unaware of the concept of software security and give little or no consideration during development process. This negligence results in large number of security problems within the software. According to CERT Coordination Centre (CERT/CC) of SEI, about 90% of the reported security incidents are as a result of exploits in design and development flaws [3] and the majority of these flaws are caused as a result of bad coding styles for example buffer overflows. Malicious attackers exploit these flaws even in the presence of standard security techniques. If the firewalls, antivirus programs and intrusion detectors had been able to secure software then software security would not have attracted the amount of attention that it has now.

Since the beginning, the software industry has been focusing on building fast, cheaper and quality software. Satisfying client's demand efficiently as well as effectively through cost and time, has been the industry's first priority. Different development life cycle methodologies like 'Extreme Programming (XP)' have evolved over the period of time. Many process improvement activities like 'Capability Maturity Model (CMM)' have also emerged but the focus has still remained on functional requirements. Security was not considered during development phases. Even if it was considered, it was treated as a separate activity. The isolation of security aspects from the development lifecycle resulted in an unsecured, vulnerable software making it easy to exploit by malicious attackers.

Protecting a network full of carelessly built software (from security point of view) is very difficult. Even with the use of best traffic filters, firewalls, antivirus, the attackers find their way in. An alternate and better approach is to have software that provides no room for attack. This does not rule out the need of good antivirus programs, firewalls and intrusion detectors. Both need to be improved with time but security in software requires more emphasis since the current practices are quite far behind in this regard.

In addition to the above mentioned factors, there are other factors that fuel the need for software security. These factors include complexity, extensibility and connectivity [1]. Software is growing bigger in terms of size and complexity as they keep on performing bigger and bigger tasks. Windows XP contains 40 million lines of code as compared to Win 3.1 3 million lines of code [1]. Growing size and complexity offers room for more design and code level flaws. Extensibility offers cheaper and a faster way of providing updates to software. Sun Micro System's Java and Microsoft's .Net platform are designed to dynamically accept code and update [1] making them more vulnerable to malicious code. Third factor is connectivity. There has been extra ordinary growth in terms of connectivity during the last decade. Evolution of LAN, WAN and internet has revolutionized the connectivity among computers. It is hard to imagine a standalone computer. This allows an attacker to attack by sitting at a distance. All these factors go in favor of the attacker and make software more and more vulnerable to attack.

With this evolving vulnerable nature of software, ineffectiveness of existing techniques such as firewall and antivirus programs, unawareness of the concept of security among developers, designers and architectures, and negligence of security in development methodologies, process improvement techniques suggests that there is a great need for coming up with a better way of incorporating security in software. This can only be achieved by building security into the development process, i.e. considering the security aspect in each and every phase of the software development life cycle and creating awareness of security among developers, designers, architects, and all other stakeholders involved in the development process.

2.3 Software Security Vs Application Security

Application security can be defined as:

"*Application security is about protecting software and the system that software runs in a post-facto way, after development is complete*" [4]

There is a considerable difference between the idea of software security and application security. As mentioned earlier, software security is about building/engineering secure software by planting security into the development lifecycle. It involves incorporating security into every phase of the development life cycle and educating stakeholders about the notion of security. On the other hand application security also means securing software after the development is completed, i.e. the securing software in a post facto way [4]. In addition it also follows approaches such as penetrate and patch, and input filtering [4]. Application security tends to secure software from known and exploited vulnerabilities. It does nothing towards building secure software. It is hard to secure defected software than defect-free software, for example, software having buffer overflow vulnerabilities can be protected by input filtering but a better technique would be to have software without buffer overflow vulnerabilities. Therefore in comparison with application security, it can be said that software security is a better solution than application security towards securing software.

2.4 Software Security Vs Information Security

Information Security can be defined as:

"*The concepts, techniques, technical measures, and administrative measures used to protect information assets from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use*" [7]

To understand the difference between software security and information security, it is important to understand the difference between inherent nature of information and software [5]. Information is something that is passive in nature. Information can be harmful if stolen, deleted or modified. Software on the other hand is active in nature. It can be harmful both actively and passively. Software can be stolen for misuse and can be exploited during execution. Securing information requires avoiding unauthorized access to the system where information is stored. Information security has nothing to do with the way information was created while software security depends a lot on how the software is created. In short information security is about protecting information from unauthorized access while software security is about building secure software.

2.5 Qualities of a Secure Software

Secure software can be defined as the "*software that is resistant to intentional attack as well as unintentional failures, defects and accidents*" [5]. To achieve resistance to attacks, failures, defects and accidents, the software should be correct, robust and reliable [7]. These three qualities if not achieved, surely decrease the security of the software.

Correctness is the quality that specifies how well the software conforms to its specification [7]. Failure to satisfy functional requirements may lead to security vulnerabilities but that is not the only cause of failure. Often the specifications are incomplete and lead to security vulnerabilities. For example the most common

vulnerability i.e. buffer overflow can occur if the specification does not specify the maximum limit of input data or how overflows will be handled [7]. Therefore to achieve correctness, the specifications must be complete and software should satisfy them completely.

Robustness is the percentage of time that a product can continue to function in the event of unusual conditions [7]. It means that software should not stop working or should not behave inappropriately if given an unusual input and it should not crash as well. Software should have the capability to reset or move to any stable state. This can be only achieved if consideration is given during the design and development phases.

"Reliability of the product is measured by the percentage of operating time, which the product performs requested operations correctly" [7]. There may be instances in which the software does not perform correctly and fails to a certain input. Reliability can also be the measure of vulnerability. More reliability means less vulnerability and vice versa. Like robustness and correctness, consideration should also be given to reliability during the design phases of software development.

2.6 Security Flaws

Software security flaws are the defects and vulnerabilities present inside the software that can be exploited to gain control over the system. There could be a number of reasons for having security flaws inside the software, e.g. lack of awareness of security among the development team, failure to identify security requirements during the requirements gathering phase, neglecting security in the design phase, choice of unsafe programming language, incomplete functional specifications, etc. There are many types of security flaws. The few common ones are Buffer overflows, Misplaced trust, Race condition, and Poor random number generator [9].

2.6.1 Buffer Overflow

"*Buffer overflows are proof that the computer science, or software programming, community still does not have and understanding (or, more importantly, firm knowledge) of how to design, create, and implement secure code*" [10]. Buffer overflow occurs when more data is written into the buffer than its capacity. Most of the security attacks are done by exploiting this vulnerability. It has been one of the most common security vulnerability in last ten years [10]. A study by Wagner showed that approximately 50% of all large exploits are buffer overflows [9]. They are popular because they are easy to exploit and provide exactly what the attacker is looking for [11]. They usually occur because of careless programming and use of unsafe programming languages like C and C++. C and C++ do not inherently check the amount of the data written into the buffer and if the programmer fails to apply the bound check, then this can result to the buffer overflow vulnerability being exploited.

Buffer overflows can be avoided by specifying details in specifications regarding input limits and by educating programmers. Since they make up 50% of the total vulnerability, a large number of security attacks can be avoided if programmers make software free from buffer overflow vulnerability.

2.6.2 Race Condition

"*A race condition is a programming fault which produces unpredictable program state and behavior due to un-synchronized concurrent executions"* [12]. It is widely found in security vulnerability. It only occurs in shared memory parallel programs where access to shared memory is not synchronized [13]. Due to a very basic nature of

this kind of problem, it is growing with time as computing is moving towards parallel processes, multithreading and distributed environment [1, 9]. It is hard to trace and difficult to avoid.

Most of the race conditions are not security vulnerabilities [9]. It is often thought that by reducing the time between the events, we can control race conditions. This is not true, as computers are becoming faster and faster, the race condition can occur even in unimaginable small seconds of time frame. To avoid race conditions, the developer should be aware of their nature and should try to remove the time between verifying a resource and using it [9].

2.6.3 Poor Random Number Generator

Random number generators: As the name suggests, are used to generate random numbers. A true random number generator is the one that generates numbers, which are impossible to guess based on the past knowledge. Developers often use non true random number generators and this becomes really critical when these are used in security critical programs like cryptography. If random numbers are predicable then attackers can exploit them to break the security code. Random number generator should not be used without having prior knowledge of their quality. Poor random number generators make the program more vulnerable.

2.6.4 Misplaced Trust

Like buffer overflows and poor random number generators, misplaced trusts also fall into the category of carelessness, and lack of awareness and knowledge. Software is becoming increasingly interactive. Each time software interacts with other pieces of code, question should be asked, whether to trust it or not. Therefore to be on safe side, trust should be avoided as much as possible. Developers often consider in-house data to be safe and do not perform input validation [9] which can result into security vulnerabilities.

2.7 Towards building Secure Software

So far we have learnt that software security is about building secure software. Security can not be achieved by securing software only with the help of firewalls, anti-viruses etc. If we want to secure software then we need to build secure software. The next chapter identifies and explains activities that are carried out for building secure software.

3 SECURITY IMPROVEMENT ACTIVITIES

Have you ever tried to find out the root cause of unsecured software? Who could be responsible for unsecured software? Requirements analysts do their best to identify functional and non functional requirements that meet the customer's demand. System designers do their best to come up with most robust design. Developers do their best to develop application in the most efficient manner. Testers do their best to find out defects from the software. From requirement gathering to software development, from software testing to maintenance, the whole project team puts their best effort into the Software Development Lifecycle to ensure quality software but even then the software is produced with a number of security flaws. With so much effort going into the software and with everyone doing their job with the best of their efforts, it makes you wonder that from where do these security flaws come from? To answer this question we need to ask ourselves, have we talked about security so far? Is security considered at any stage of the software development? I am afraid the answer is, no. It is not, or in other words, security is completely ignored in a way that there exists no such thing as security. Developers only develop what is specified in requirements. If security requirements were there in first place, then the steps following the requirements gathering phase which include design, development and testing would have taken care of security and if security was incorporated in every phase of the software development life cycle, then software developed through it would have been more secure.

This chapter focuses on the activities that should be performed to incorporate security in the software development life cycle. Security activities are divided into different phases of the life cycle starting from the Requirements Phase but there are a few activities that need to be performed before the actual project can start of. Therefore this chapter begins with Pre-requirements phase.

3.1 Pre-requirements Phase

Security activities performed in this phase set the foundation for all the activities starting from the requirements phase right up to the maintenance phase. Pre-requirements phase focuses on training resources and obtaining the interest of top management. The following activities are performed during this phase:

3.1.1 Security Training

Knowing what to do is not enough if you do not know how to do it. Many organizations are unaware of the importance of security and those who are aware of security are unaware of how to improve it. There is a great need to create awareness of security importance. Educating project team is not enough. Everyone from top management to customer support, developers to testers, all need to be educated. The whole organization should be aware of the importance of security. If the top management is not aware of the security, then they would not be willing to spend time and money on it. If business people are not aware of it then they would not be able to satisfy customer concerns. The education may differ from role to role but it is a requirement for everyone.

There is no one particular way of educating that fits best to all organizations. Every organization has to adopt the way that best suits to it. Security training should be divided into two parts i.e. general training and specific training.

3.1.1.1 General Training

General training is for the whole organization. Its purpose is to create awareness of security in an organization's environment. Importance of security, its advantages, disadvantages if neglected, new concepts, measures, methodologies, etc. are discussed on abstract level. These general trainings should be scheduled periodically; the interval between these trainings will vary from organization to organization e.g. if new people have joined, then the interval can be minimized. If there is no new entry and existing employees are already aware of security, then there is no need to conduct training. General training will prove to be successful if everyone in an organization starts thinking about security in their plans, scheduling and acts (if needed).

3.1.1.2 Specific Training

Contrary to general training which is for the whole organization, specific training is limited for the project team only. Specific training deals with individual projects. Each project brings in some security threats and flaws depending upon its nature. The goal of specific training is to identify these flaws; threats associated with them, and identify security measures that should be taken to secure software from these threats. A good way to achieve this goal is to provide knowledge of previous attacks on related software. The project team should have understanding of attacks, tools, techniques, and areas of interests for attackers [16]. Specific training should target people in different roles accordingly. For example, requirement analysts are educated for identifying security requirements. System designers are educated so that they can develop the design keeping in mind the security threats to that particular project. System developers are educated so that they write secure code not only that fulfills the security requirements but also according to the system design. Developers are also educated to follow development guidelines in order to avoid common security flaws such as buffer overflow, misplaces trust, poor random number generators, etc. Software testers are educated so that they have the knowledge of security risks, threats and flaws, and test software accordingly.

General and specific trainings develop a security culture into the organization with everyone taking security seriously. This security culture will open up doors for accepting other security improvement activities. Few organizations make separate security groups and assign them the responsibility of security [15]. This is not a good solution since security is everyone's duty not just a few individuals. Also it has been seen that few organizations make network security team responsible for software security which is even a bigger mistake [15]. Security trainings are successful if each individual is aware of security importance and knows what role he or she has to play towards the development of secure software.

3.1.2 Develop Risk Management Framework

Consider a scenario where you are near to software release and a critical security risk is identified. You are fully aware of what damage this bug can cause and you want to delay the release until the bug is fixed. How do you convince the top management? [15] Alternately you may require extra resources to fix the bug. Convincing top management requires facts and figures. If top management or any other decision maker is unaware of the impact that can be caused by a critical security flaw, then most probably they will not accept your justification. Therefore it is better to have risk management framework incorporated in the beginning, so that top management and business owners can be educated about the risks and their possible impact. Risk management framework also serves as a reminder of security importance over a period of time.

3.1.3 Think Deep

Security is an emergent property of a software system, not just a collection of security features [4, 15]. Organizations often consider security as a sum of some security features that can be incorporated [4]. They should think deep. Instead of treating security as a feature that can be painted onto software, organizations should think of ways to resist attacks [15] and insert this resistance deep into the software and methodology, used to develop it.

3.2 Requirements Phase

Actual software development life cycle starts form requirements phase. During requirements phase, goal of the project team from security perspective is to identify security requirements applicable on project and articulate them in a clear and understandable form. Following activities are performed in requirements phase.

3.2.1 Identify Security Requirements

Requirements describe what the system should do. Requirements are the bases of all the phases in software development. Success of any software project typically depends upon the requirements. If requirements are ambiguous, incorrect and incomplete, then the whole project is at risk and if they are clear, correct and complete, then the whole project can be a success (ignoring other constraints). Requirements come from various sources for example customers, competitors, economic and political environments etc. Requirements are usually identified by specialists known as requirements analysts but anybody from the project team can come up with requirements.

Security requirements describe the security aspect of the project. They are identified along with the identification of the system's functional and non functional requirements. Security objectives for the project are identified first and then security requirements are identified to fulfill the security objectives. Just like functional and non-functional requirements, security requirements are also identified at the start of the project and at the start of each new version. Security requirements may change on later stages. Change prone security requirements plus other security requirements should also be considered for placing into configuration management just like other functional and non-functional requirements. During the requirement phase, the project team considers how security will be integrated into the development process [21]. Project team tries to come up with security requirements to maximize security with minimum effect on the overall plans and schedules [21]. Most of the security requirements are identified during the requirements gathering phase, and some come from threat models, industry standards, and certification processes later on [21].

3.2.2 Develop Misuse Cases

During requirement gathering, the focus is on 'what' and not on 'how'. Once functional, non-functional and security requirements are identified, gathered, analyzed, agreed upon and, documented, the next step is move towards 'how' on abstract level without going into the design and implementation details. For requirement elicitation, use cases have proven to be quite useful [20]. Use cases are useful because they provide an overview of the overall functionality with the help of simple diagrams. They explain functions that a user can perform on the system with the help of actual steps but they are also good at eliciting functional requirements simultaneously. Non functional requirements (for example, Login time for user should not be more than 5 seconds) and security requirements (for example, any unauthorized access to system

must be blocked) are not well explained with the help of use cases. Security requirements are often described as, what the system should not do, while functional requirements describe what the system should do. Use cases are designed to explain functional requirements therefore they are good at specifying what the system should do rather than what system should not do [18]. To overcome this limitation of use cases, Guttom Sindre and Andreas L. Opdahl developed misuse cases.

"*A Misuse Case is the inverse of a use case i.e. A function that a system should not allow*" [18]. Misuse cases are similar to use cases but specify what a system should not do. Instead of actor(s), misuse cases define 'Mis-actor'. "*A Mis-actor is the inverse of an actor, i.e. an actor that one does not want the system to support, an actor who initiates misuse cases*" [18]. Misuse cases are very useful in eliciting security requirements. They can identify different types of attackers and differentiate between them. They also identify a system's state before and after the attack. The idea of placing misuse cases in the Use case diagram instead of a separate diagram provides several advantages. With a common diagram, it is easy to identify relation between use cases and misuse cases. It can be seen that how a normal actor can act as mis-actor. There are attacks which do not require any extra functionality and can be carried out by using the system's allowed functionalities e.g. flooding. For preventing misuse cases to happen there might be requirements for new use cases. In short misuse cases are very useful in eliciting security requirements and in understanding possible attacks, actors, relation with allowed functionalities, estimating amount of loss, and determining system's behavior before and after the attacks.

3.2.3 Prepare Requirements Document

After identification of requirements (functional, non-functional and security) and eliciting them with use cases and misuse cases, a complete requirement document is prepared. This document serves as the guide through out the development process and provides a base line for functional, non-functional and security requirements. This document represents security goals and its perspective in software development. The reason for placing security and other requirements into the same document is to promote the culture of treating security requirements with the same importance as other functional and non-functional requirements. Once the security documents are finalized inside the requirements document, then all the following activities will base on this one complete document and will serve as a first big step towards making security a part of the software development life cycle.

Security activities in requirements phase are shown in the figure below:

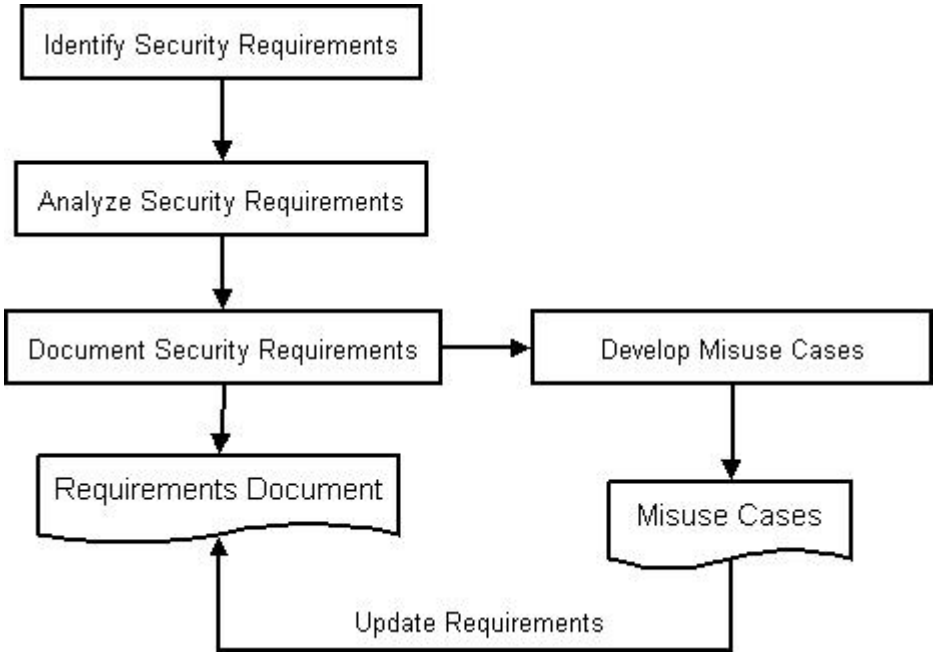

Figure1. Security Activities in Requirements Phase

3.3 Design Phase

Design phase builds upon artifacts developed during the requirements phase. Requirements artifacts describe 'what' to build with minimal level of details. The design phase focuses on 'how' and describes how the features identified during the requirements phase, will be implemented. In the design phase, the project team elaborates on software features identified in the use case document. Software architecture, software components, behavior of different components, interaction between the components, use of programming language, selection of algorithms, data structure, etc. are also identified. Design phase acts as a bridge between the requirement and implementation phases therefore it has to be complete and concise to serve as a starting point for software implementation.

The design phase holds a key importance for security purposes. Large numbers of security problems have the design phase as their origin. According to Howard and LeBlanc, 50% of security bugs are found during the design phase [25]. As the design artifacts mature they are analyzed to identify the impact on software from the security point of view. From the security's perspective, the following activities are performed during the design phase:

3.3.1 Build Security Architecture

Software architecture lays the foundation for software design activities. Software architecture contains the structure of the system comprising of software elements, external visible properties of those elements, and relationships among them [23]. Security architecture defines software architecture from the security point of view. It identifies important components whose correct functionality is important for security [21] along with their properties and interaction between them. In addition to that, security architecture defines design techniques, use of strong typed language, application of least privilege, and minimization of attack surface [21].

Security architecture provides an overall picture of security design [21]. It does not contain a detailed design or any development details. Security architecture does not

remain the same though out the life cycle. Identification of new threats [24], changes in priorities of threats, change in software architecture etc cause security architecture to be revisited regularly and updated accordingly.

3.3.2 Identify Interaction Points

Almost all software applications interact with other software applications in one way or another. Software applications exist in an environment where they exchange data and perform functions with other software applications. Different components of software also interact with each other. It will be hard to imagine of any software that functions alone. Software architecture and design show interaction of components within the software and with other software. From the security point of view, these interactions are very important because all of them are not safe. To identify interaction points, information about the integration point must be available. This includes data flows, users, and supporting systems [21]. These points are identified and their nature is analyzed. Questions like when, how, why these interaction take place and between which components, whom to trust, whom not, are answered.

3.3.3 Identify Assets and their Access Points

Identification of assets that software manages and their access points is a very important step towards secure software. Assets may include data, information, piece of code etc. These assets may be stored in the software on run time in logical structures or may reside in physical storage like magnetic disks. Identification of these assets, when and how they are stored and accessed is important for identifying security threats associated to these assets.

3.3.4 Minimize Software Attack Surface

Since no software can be 100% secure, therefore virtually every software can be successfully attacked. Minimizing or reducing software attack surface is a technique used to minimize the attacks possible on the software system. Howard and Lipner has defined attack surface area as *"The attack surface of an app is the union of code, interfaces, services, protocols, and practices available to all users, with a strong focus on what is accessible to unauthenticated users"*. [34]

During the design phase project team identifies the set of feature that should be available to user by default. The project team tries to come up with the minimum number of features with minimal privileges that are good enough for most of the users. There may be the cases where surface area has to be increased to provide more features and more privileges but then the project team needs to identify those cases and take pro-active actions to ensure security with those cases. The design phase should specify what features the software will have with default settings, in what circumstances the software will have more feature and for which users. The design phase should also specify how software design deals with these circumstances.

3.3.5 Draw Threat Models

Building upon misuse cases developed during the Requirement phase, threat models are developed to identify threat origins, their execution criteria, conditions required for them to occur, and the impact they can cause on the system [22]. In security development, threat models serve as a guideline during all the stages of the software development life cycle. They help in developing an understanding of security issues related to a specific project; they help in attracting attention of stakeholders towards security importance; they help in doing early planning towards possible future security mishaps etc. Threat models keep on updating during software development.

There could be many reasons for it for example; identification of new threats, change in understanding of previously documented threats, addition or deletion of software functions, changes in external environment etc.

To create a threat model, first identify areas that could be of attacker's interest. The areas themselves speak a lot about types of threats that software can undergo. With the identification of areas of an attacker's interest, there are other things that need to be identified. These include: functionalities the software system will perform and how it will perform, technologies that will be used to build the software [17], operating environment on which the software will function, and attack patterns on similar software systems. After gathering this information threats are identified and are documented. Each threat is documented with its description, target, risk, defensive technique and risk management strategy [17]. Finally the documented information is used to prioritize threats based on the impact they can cause.

Security activities during the design phase are shown in the figure below:

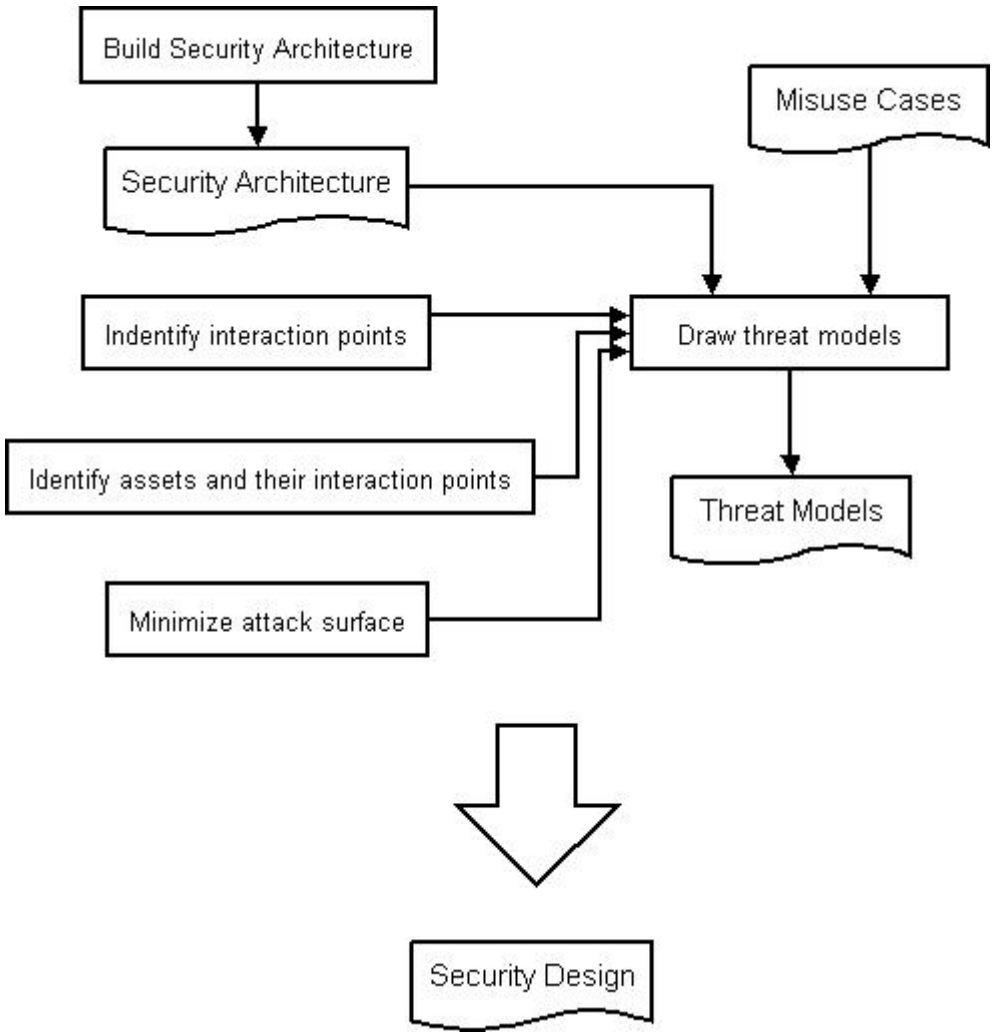

Figure2. Security Activities in Design Phase

3.4 Implementation Phase

Most of the security activities described so far are design centric. Design centric activities focus on architectural flaws built into software design but overlook

implementation level flaws that a software developer might introduce into the software during coding [28]. Like requirements and design phases, securing an implementation phase is important too. It brings its own flaws that are only born with code and are impossible to detect in earlier phases.

Implementation phase is the last phase where security flaws can enter into the system. If secured, then the chances of flaws entering into the software are minimized. Here it is important to understand the difference between design level flaws and implementation level flaws. Design level flaws prevent the application from operating securely no matter how secure the code is [29]. On the other hand implementation level flaws are those flaws which the programmer introduces into the application due to poor coding, lack of knowledge of security or use of un-safe methods etc.

Implementation phase starts with secure code writing followed by static code analysis with the help of automated tools and manual code reviews.

3.4.1 Write Secure Code

To avoid security flaws in the implementation phase, the first and the most important thing the development team can do is to write secure code. Now the question is how to write secure code? The development team must be educated about implementation level security flaws. Development team should understand the nature of implementation level security flaws, how they occur and what impact they can have on the software. Specific training provided at the start of the project should have covered it, if not, then educate the development team before they write any insecure code.

At the time of coding, the project team has passed through the requirements and design phases and has acquired considerable knowledge of security. Artifacts developed during early phases especially during the design phase, guide developers during the implementation phase. Developers make use of threat models to identify critical threats and take extra care while coding corresponding functionalities.

There are security flaws that are born during coding and therefore can not be detected or removed during the design phase. These flaws can be avoided if security guidelines are followed during code writing. These guidelines contain information about implementation level security flaws (e.g. buffer overflows, race conditions, input validation etc), how they occur, their impact on software, and how these flaws can be avoided. These guidelines are revised occasionally for incorporation of new security flaws and methods to avoid them. The development team should have this guideline in front of them during coding.

3.4.2 Perform Static Analysis of Code

Till now we have been trying to avoid flaws. From requirements to actual coding, emphasis was on prevention. From coding onwards emphasis will be on detection because now the flaws which we have been trying to prevent may exist in the system. The first step towards detection is to perform static analysis of code.

Static analysis of code is performed with the help of automated tools. This is a fast, cheap and effective way to detect common security flaws in code. There are flaws in the code that automated tools detect very efficiently. These flaws include buffer overflows, un-initialized variables, integer overflow etc. It is important to note that the use of security tools will not make the software secure, it only helps making the software secure [26].

3.4.3 Perform Code Review

Code reviews are performed to detect security flaws in the code that are left after static code analysis. Automated tools used for static analysis have limited power. They can only find flaws for which they are designed for. To supplement automated tools, experienced developers perform manual code reviews. This is an expensive and time consuming process. Going through tens and thousands of lines of code is a difficult task. Code reviewers should know, what they are looking for and should prioritize their work [27]. Design documents and threat models help reviewers in prioritizing their work.

Code reviews should also focus on flaws that static analysis has identified because it is not necessary that all particular types of flaws have been removed after static analysis. Code review can be done in three steps. First step is to rerun the static analysis tool [27] and look for errors and warnings generated. Some warnings could actually be errors or hiding errors [27]. This practice helps in identifying areas of code that need special attention. Second step is to look for common vulnerabilities [27]. Most notable vulnerabilities are arithmetic issues, buffer overflows, cryptographic issues and SQL injection [27]. Finding these vulnerabilities can remove large number of possible security flaws. The third step is to thoroughly analyze critical areas [27].

Security activities during the implementation phase are shown in the figure below:

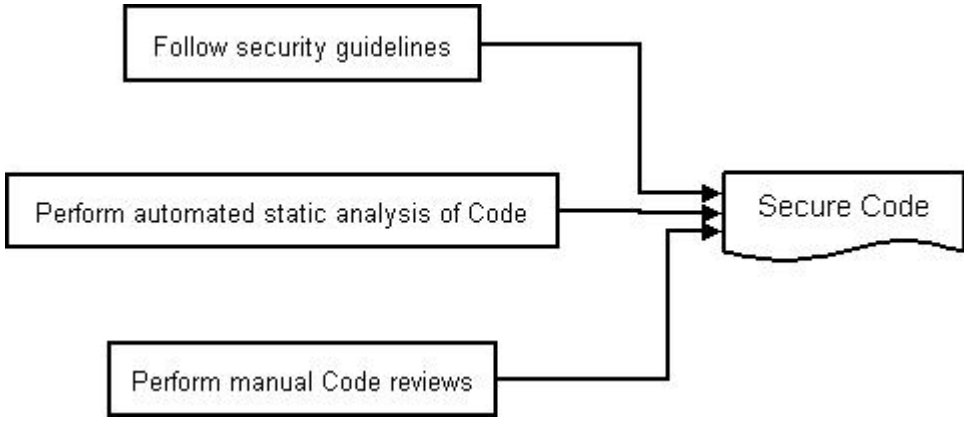

Figure3. Security Activities in Implementation Phase

3.5 Testing Phase

Until now the focus has been on making secure software secure by incorporating security into the software development activities and by preventing security flaws chipping into the software. When the software enters into the testing phase, it is functionally complete [21]. Now we cannot make software secure by preventing flaws entering into the system but what we can do is to identify flaws that already exist in the software.

Security testing is considered as a difficult job than the usual functional testing. It does not quite follow the rules applied to usual software validation [30]. It is different from usual software testing. Usual software testing is usually based upon normal user behavior. A tester tries to put himself inside a normal user's shoe but an attacker is not a normal user. Understanding the mindset of an attacker is totally different from that of normal user. Security testing demands testers to have expertise in security concepts, security flaws and attack methods.

Security testing like normal testing needs test planning, test bed preparation and actual testing. The goal of security testing is simple but quite demanding, which aims to find the maximum possible security flaws.

3.5.1 Test Planning

In security testing, test planning can be considered as the most important phase. During test planning, testers make use of misuse cases and threat models to identify possible attacks, how they can be carried out and their impact. Testers then prepare test scripts for testing. These scripts are designed to attack software successfully.

During test planning, testing schedule is created. Priorities are assigned to test cases. Execution pattern of test cases is identified. Testing depth is identified for different areas. High risk or security critical areas may need more detailed testing than other areas. Tool selection is done. Bug severities are revised according to security testing. In short security test planning is not only about designing test cases but is about organizing the whole security testing activity efficiently.

3.5.2 Test Bed Preparation

Just like test planning, test bed preparation has its importance in security testing. Test bed preparation involves setting up hardware, software and network environment for the software to test. In security testing this becomes crucial. Along with setting up the usual test environment, the tester should think of an environment which may support attacks. This requires good knowledge of software attacks and software hardware environments.

Depending upon the software and hardware environments it will work upon, different test beds should be prepared. These test beds represent the environment that an attacker may create for an attack.

3.5.3 Testing

Security testing is difficult compared to the usual functional testing. In security testing, the tester needs to think like an attacker [31]. Unlike functional tests, the output of security tests may be unanticipated and may require further analysis [31]. For example: function A should perform a specific functionality, but it is possible that while performing that specific functionality, function ‘A’ is also performing some other hidden functionality. If the tester does not thoroughly analyze the result, then he may overlook that hidden functionality performed by function ‘A’ [32].

Security testing must involve two approaches [31] that are: Performing functional testing of security mechanisms, and, Performing risk based testing to simulate attacker’s behavior. Functional testing of security mechanisms make use of use cases while risk based testing make use of misuse cases. Risk based testing requires understanding of the attacker’s mindset.

Testing activity can be supported by the use of automated tools. Many automated tools are available in the market but the problem is that most of them are for functional testing, but there are few new testing tools for security testing being introduced in the market. The use of automated tools can cut down manual testing efforts and testing time, hence providing more time for the testing team to focus on risky areas.

Security activities during the testing phase are shown in the figure below:

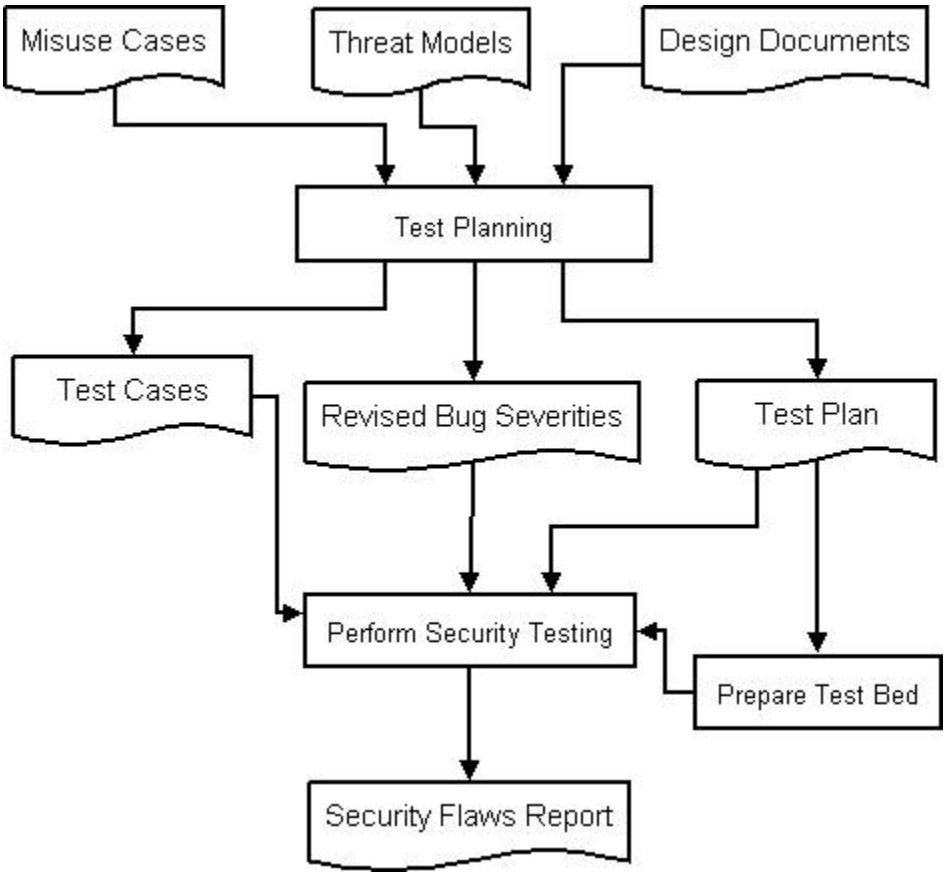

Figure4. Security Activities in Testing Phase

3.6 Release and Deployment Phase

When the software reaches the Release phase, it is assumed that most of the security flaws have been identified in preceding phases and the software is stable and is ready for deployment. But before deployment, the software is subjected to security review and audit. The purpose of this security review and audit is to identify any remaining security flaws and gain confidence over the software.

3.6.1 Security Review

After testing phase, the software is subjected to security review. The purpose of this review is to find out any remaining security flaws. This does not mean that a whole security testing cycle will be run. Security review has to be well planned so that it does not take much time and resources. During the review, over all state of the software is analyzed. Security critical areas undergo detail review and previously identified bugs are rechecked. Security Review can be considered as the last activity where software is analyzed for security. After security review the next person to identify a security flaw could be an attacker.

3.6.2 Security Audit

After security review the software is subjected to security audit to obtain final words on software security. Audit phase is nothing more than a question answer session. Based on the checklist, the security audit seeks to identify whether security practices have been followed during the software development. Statistics are obtained

and drawn in presentable form for the management. Top management look at results of the security audit and get an over all picture of the software from the security point of view. After the security review and audit, decision is made for the software release.

Security activities during the release and deployment phase are shown in the figure below:

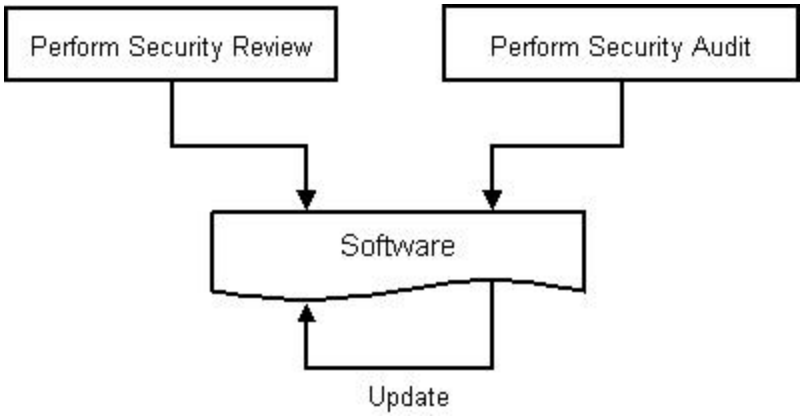

Figure5. Security Activities in Release and Deployment Phase

3.6.3 Deployment

Security deployment is nothing more than deploying software with default settings and configurations. These settings are identified during the design phase. The goal of these settings is to reduce possible attacks by minimizing the attack surface.

3.7 Maintenance Phase

After deployment the software goes into the hands of the end user. The software is not 100% secure and it can never be [21]. There might be some identified security flaws present that were not fixed during software development due to time constraints or any other reason. These flaws are looked again, prioritized and fixed. Besides fixing of known flaws, the software is monitored for new threats. New threats keep on coming and efforts are required to protect the software against them. Patches are often released to secure software from new threats and identified flaws. Maintenance is an ongoing process and does not end until the software is completely out of use or taken over by a new software.

Security activities during the maintenance phase are shown in the figure below:

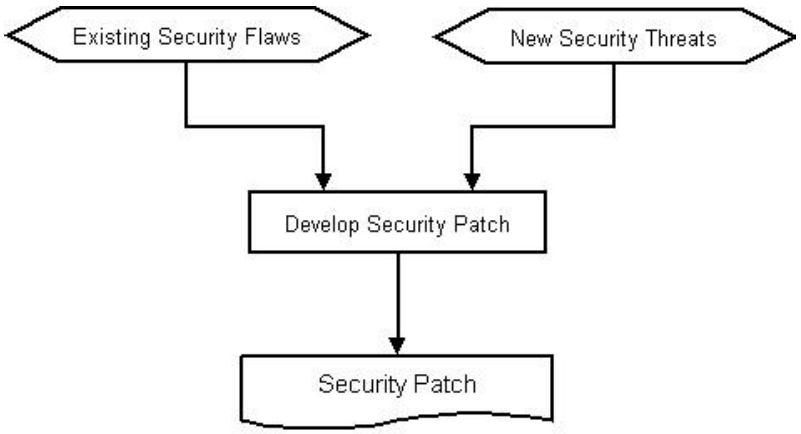

Figure6. Security Activities in Maintenance Phase

4 SECURITY ACTIVITIES IN SOFTWARE DEVELOPMENT LIFECYCLE

So far we have discussed the idea of software security, its need, and the activities needed to be performed in order to improve security in software or in other words activities required to build secure software. Different activities have been identified in different phases of the software development lifecycle. Sequence of these activities and relation among each other is shown in the form of diagrams but we have not related these security improvement activities with the usual activities of Software Development Lifecycle (SDLC). This chapter explains how security activities identified in chapter 3 go along with the usual activities in different phases of the software development life cycle.

4.1 Requirement Phase

Software development lifecycle starts with requirements phase. The goal of the requirements phase is to identify software requirements that fulfill the client's demands and represent them in a way that is both clear and understandable. Along with the identification of software requirements, efforts are made to identify security requirements. Security requirements come from different sources and at different times. Identification of security requirements is more difficult than the identification of functional requirements because they are not prominent like functional requirements. For example the client may give the functional requirements and then the security requirements are needed to be dug out of functional requirements. Anyhow over here we are not going to go into the details of identification of security requirements. The important thing here is that the security requirements are identified at the same time when functional requirements are identified. Some of the security requirements are identified later from threat modeling done during the design phase.

After identification, security requirements undergo analysis along with other software requirements. After analysis, requirements are documented requirements document. The requirements document contains functional, non functional and security requirements. Using this document, the project team makes use cases. Use cases are an important technique to understand high level system flows. In parallel misuse cases are also made along with use cases. Misuse cases help in understanding the system from the security point of view. It is quite likely that new requirements are identified, existing requirements are changed, and some requirements are deleted during the making of use cases and misuse cases. This requirements' change calls for updating of the Requirements document.

Security activities during the requirements phase make sure that security requirements are given the same importance as are the software functional and non functional requirements. Security requirements go through analysis and elicitation just like software functional and non functional requirements. Security activities during the requirements phase serve three purposes. First, security requirements are identified and elicited. Second, with security requirements in hand, the project team understands and realizes the importance of security. Last but not the least, with security requirements in hand, budget, resources, and time for security activities in the upcoming phases can be better estimated.

Figure 7 shows the security activities during the requirements phase:

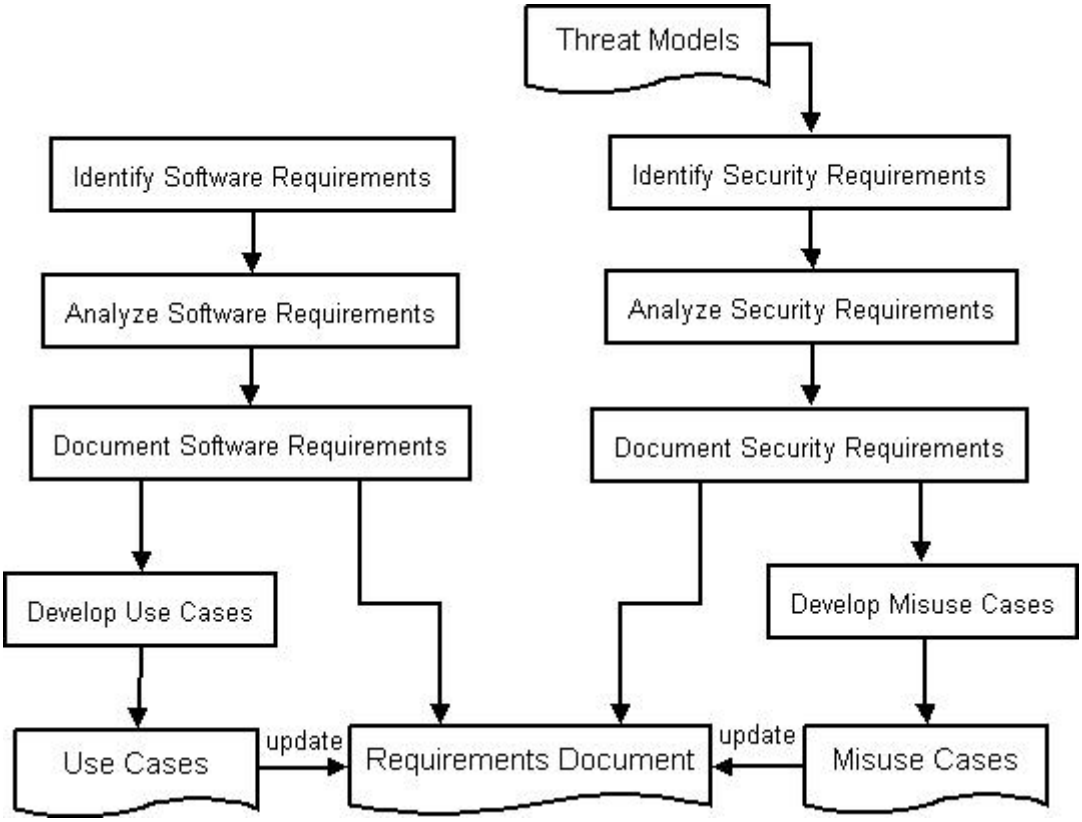

Figure7. Security and Software Development Activities in Requirement Phase

4.2 Design Phase

The design phase begins with the making of software architecture. Software architecture identifies software key components and their interaction. Security architecture is developed from software architecture. Security architecture defines software's key components from the security point of view, thus giving us a bird eye view of security. Security architecture is then described in detail with the help of software design documents. First interaction points are identified between the components. This helps in understanding the work flow of software and develops an in-depth knowledge of security issues that may arise. Interaction points between security critical components and normal components are of more importance because these interaction points can provide access to critical areas. The next step towards security design is to identify assets and their access points. The access points of these assets could also be from interaction points identified earlier. Understanding access paths to assets and securing them is very important. The fourth step towards security design is to minimize the attack surface. This step does not require much effort and is an efficient way of narrowing down the attacker's operating space. Software design is analyzed in detail and efforts are done to identify how exposed surface areas can be reduced for maximum number of users without affecting their requirements by cutting down functionalities available to them.

So far the project team has focused on the identification of attacker's interest, possible access points, security critical areas, etc, during the design phase. The next step is to identify the threats that are applicable to software. All of the security information that is gathered in design phase so far goes into threat modeling. Threat models can be considered as an important milestone on the path towards secure software because the project team now knows that they are securing the software and

from what as compared to their naïve understanding of securing the software before the introduction of the threat model.

Security design activities provide complete information on how the software can be attacked, what can be attacked, which areas are attack prone, what kind of threats are applicable etc. In the light of this information, security design is continuously updated for security incorporation. Security activities during the design phase can be best understood in the figure below:

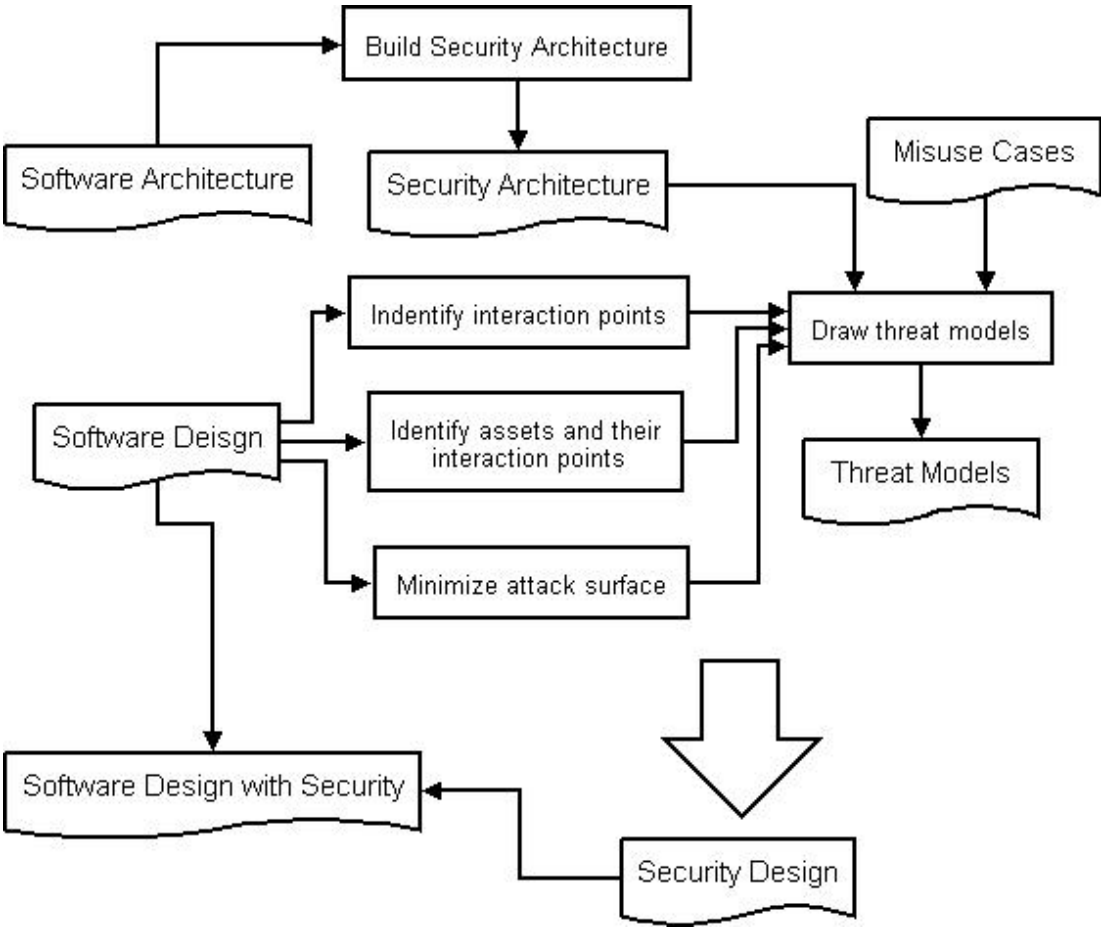

Figure8. Security and Software Development Activities in Design Phase

4.3 Implementation Phase

From the security perspective, the implementation phase plays a dual role. First role is to avoid security flaws entering into the software and second is to detect flaws that are present in the software. The first role is performed by writing secure code. Writing secure code is extremely important because all the efforts done so far will go in vain if not done. Secure code is written by the following software design (in which security design is incorporated during the design phase), considering threats identified in threat models and by following security guidelines. All these three factors are important for writing secure code and missing anyone might result in security flaws. The second role i.e. to detect security flaws begins with static analysis. Code written by the development team is analyzed with the help of automated tools. These automated tools look for common security vulnerabilities. After automated static analysis, manual reviews are performed. A manual review is an efficient way of finding security flaws in a software code. Manual reviews mark the end of security

activities in an implementation phase. After this the software is fully functional and is ready to go testing phase.

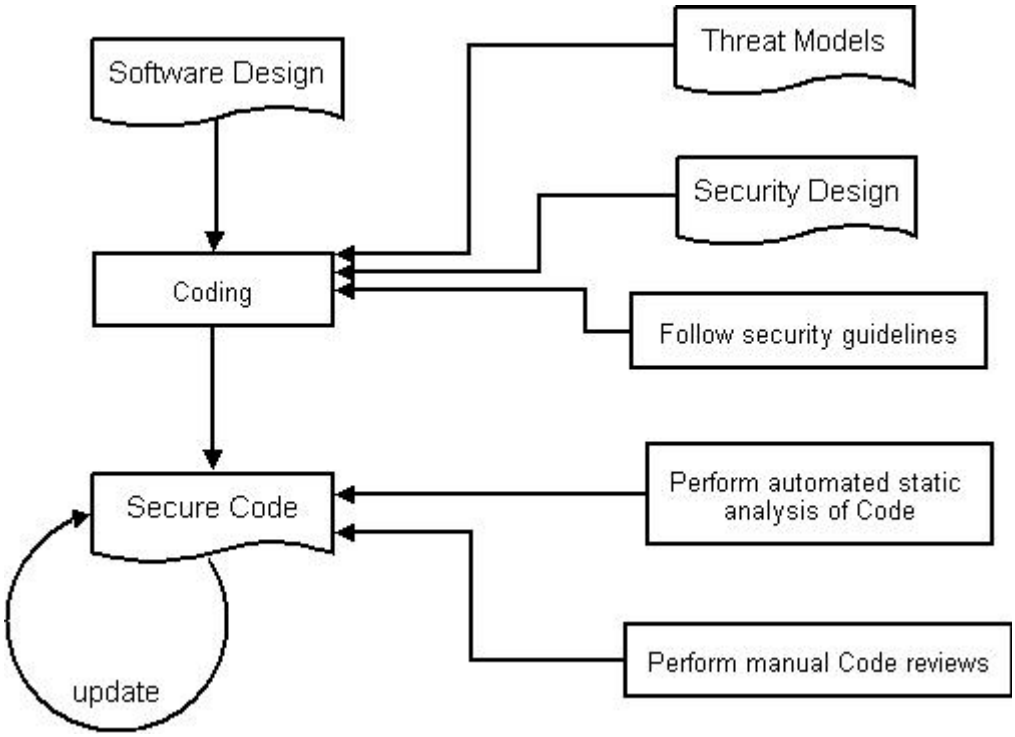

Figure9. Security and Software Development Activities in Implementation Phase

4.4 Testing Phase

The testing phase starts with test planning. Testing team starts test planning during the implementation phase because test planning does not require the availability of software code. While the developers are writing code, testing team prepares for the testing activity. Using design documents, threat models and misuse cases, the testing teams create security test cases. Security test cases are designed to attack software successfully. During test planning any change in the software design must be communicated to the testing team otherwise the testing team will end up making test cases that might not be applicable. Based on software nature and applicable threats, bug severities are revised because a minor functional bug in software code may require few minutes to fix but can have a big impact from the security point of view. These revised severities are discussed with the development team. After test-planning, a test bed is prepared. Test bed most likely contains several software and hardware environments depending upon the nature of testing.

After implementation, the software is shipped to the testing team. Testing team has already prepared a test bed and done test-planning; therefore it directly starts with testing. Testing is mainly carried out by running test cases developed during test-planning. During testing, the development team and the testing team work in collaboration. The testing team identifies security flaws, reports them to the development team, and the development team fixes them into the code. The testing phase completes when all the test cases are run, and regression testing of all critical areas has taken place.

Security activities performed during the testing phase are shown in the figure below:

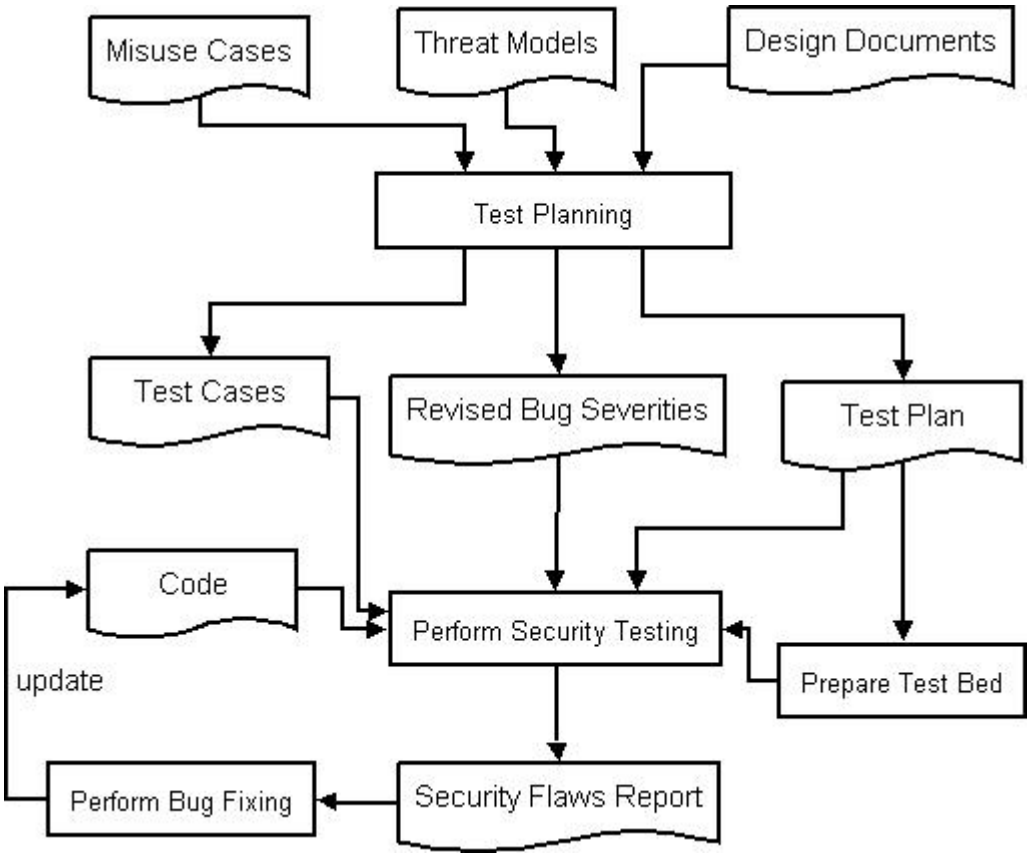

Figure10. Security and Software Development Activities in Testing Phase

4.5 Release and Deployment Phase

Release and deployment are two separate phases and occur at different points of time but since there is not much work left to be done considering security activities, therefore I consider them as single phase. Before the software is released, a security review is conducted. The goal of the review is to identify the remaining security flaws. A review report is generated at the end of the review. Development team fixes the code against security flaws identified in the review report. After review, a security audit is conducted. The whole project team is involved in the audit. Software may undergo changes if issues are identified during audit. An audit report is generated at the end of audit. Based on the audit report, the management makes a decision for the software release. After the release the software is ready for deployment.

Security activities performed during release and deployment phase are shown in the figure below:

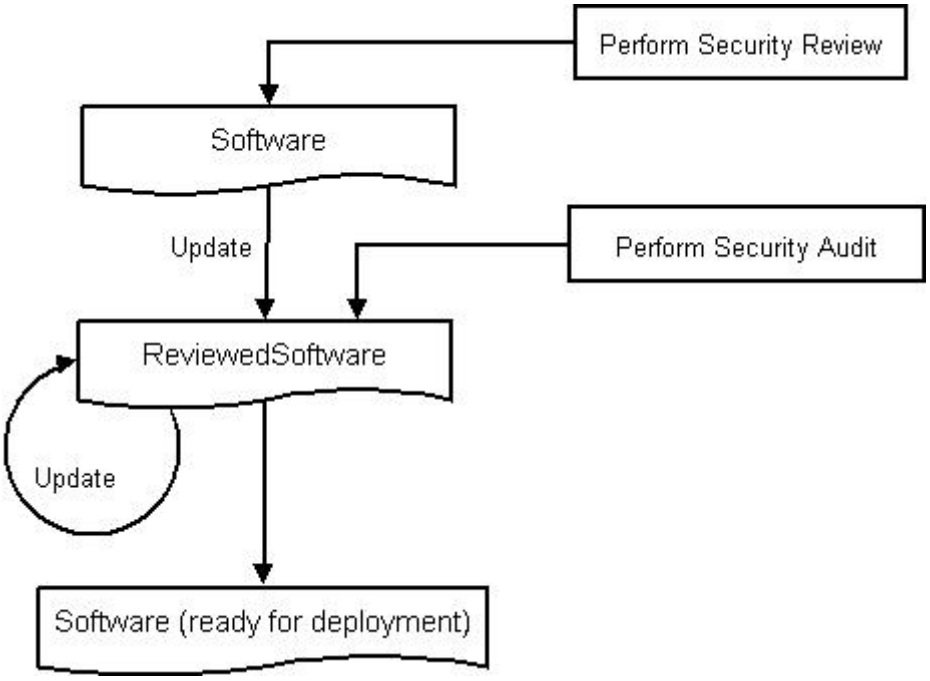

Figure11. Security and Software Development Activities in Release and Deployment Phase

4.6 Maintenance Phase

After the release and deployment, the software goes under commercial use. It is possible that some known security flaws have been left during software development and the software is released with some non critical security flaws. Later at some point in time, decision is made to fix those flaws. Therefore some coding is done to remove those security flaws in the form of a patch. The path is then applied to the software and after testing, the patch is released. The same procedure is repeated for newly identified threats.

Security activities performed during maintenance phase are shown in the figure below:

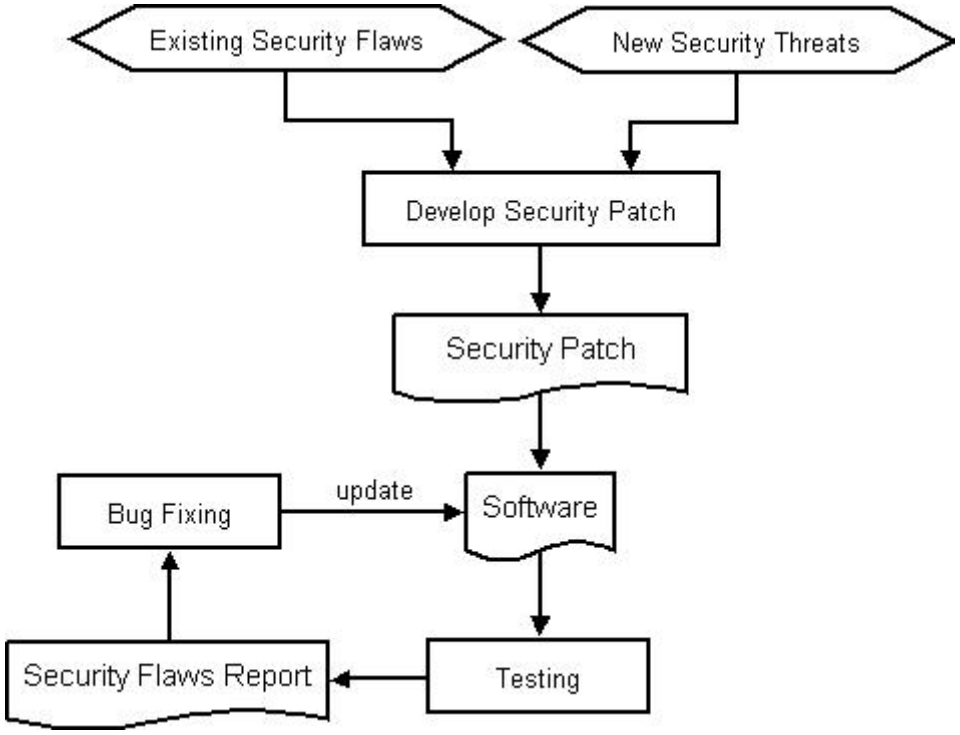

Figure 12. Security and Software Development Activities in Maintenance Phase

4.7 Complete Picture

In this chapter we have seen how security activities are related with the software development activities in the software development lifecycle. Figure 13 shows all of the software development phases and how they are related with each other.

It is true that to build secure software, security should be considered in all the phases of the software development lifecycle. But it is not necessary to give security the same importance in all the phases. Software nature, Use of programming language, operating environment, etc, affect the level of importance that should be given to security in different phases of software development. For example, if the software is developed in C++, then from security perspective, design and implementation phases should be given more importance because C++ brings in some security flaws (e.g. buffer overflows) that other languages like Java do not. If the software is to be integrated in larger system with other software application then design phase (access rights, surface area minimization etc) and testing phase (integration testing) hold more value. Organization's priorities also have their own effect, for example, if time is more important than security, then organization may choose to give security less importance during development phases and then focus on maintenance phase. This way the organization may gain money due to timely delivery of software but may lose money due to costly bug fixing during maintenance. The bottom line of all this discussion is that the security is important, but how much important and at what cost, this is up to the organization to decide.

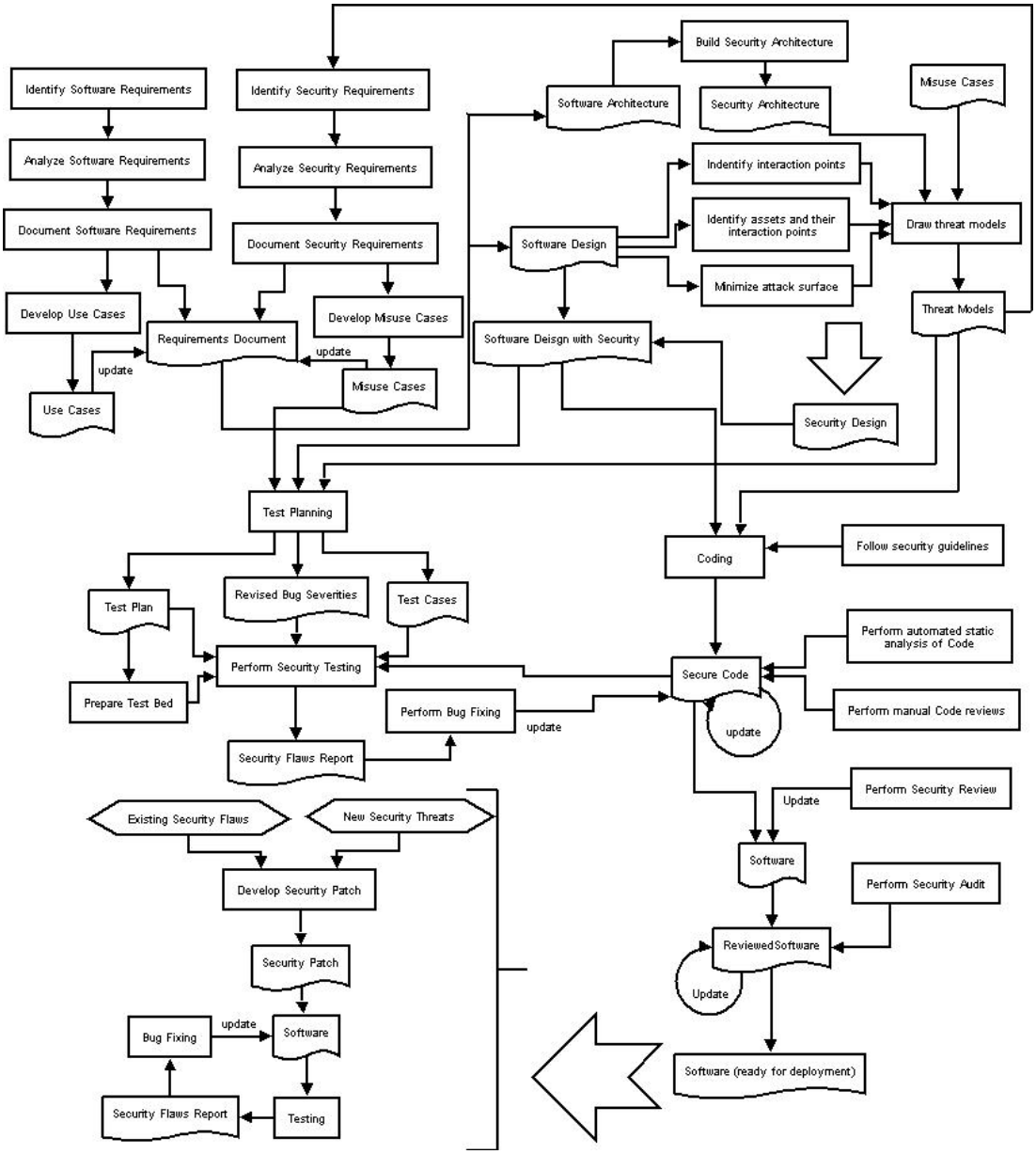

Figure 13. Security Activities in Software Development Life Cycle

5 DISCUSSION

This thesis has presented security activities that should be performed to build secure software, and has shown how the security activities are related with usual activities in different phases of software development. The security activities identified in this thesis can be divided into phases just like the software development phases. The phases then would be Security Requirements, Security Design, Security Implementation, Security Testing, Security Release and Deployment, and Security Maintenance. How security activities are performed and how the software goes through different security phases depends upon the development methodology being followed.

Many development methodologies exist today. These methodologies can be divided into two main groups, sequential and incremental. Sequential development separates software development into distinct phases, such that one phase should not be started before the completion of preceding phase [33]. While in incremental development, software is developed in increments. Each increment goes through all phases of software development. Since security activities are highly related with software development activities and mostly dependant upon artifacts produced by them, therefore they are usually carried in the same way as software development activities. This means that if software development is sequential and divided into distinct phases then security phases would also be sequential and distinct. If software development is incremental then security phases would also be performed in increments. Figure 14 shows security phases in sequential and incremental development.

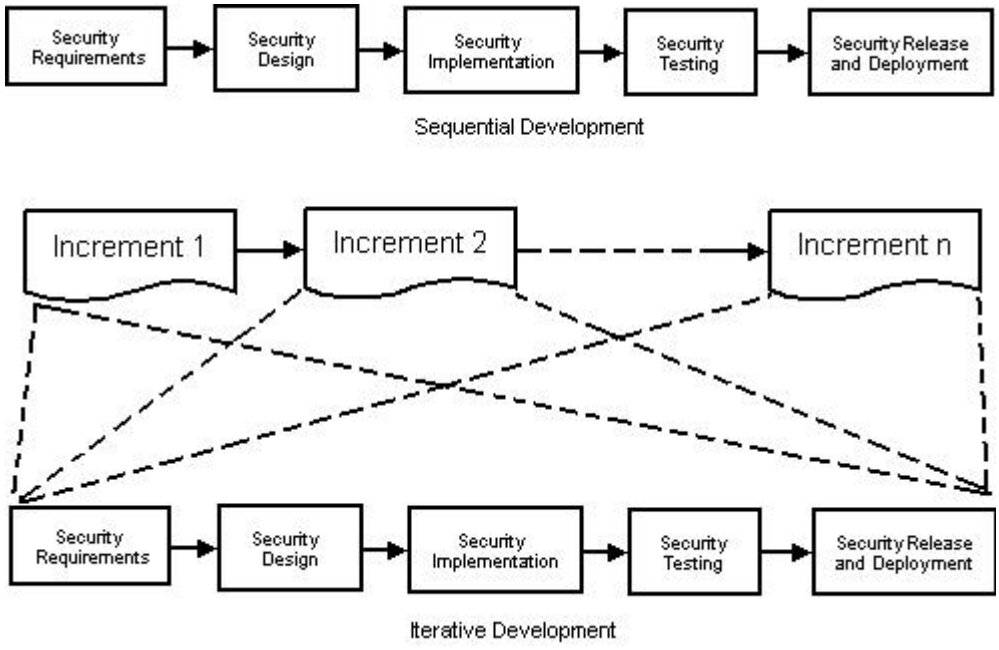

Figure 14. Security Phases in Sequential and Iterative Development

In incremental development the increments can be developed sequentially or in parallel. But irrespective of whether the increments were build sequentially or in parallel, integration has to be performed. One problem that incremental development brings in from security perspective is that the complete picture of software is not available during development of increments. As each increment is developed separately therefore security issues that might have been identified by looking at

complete picture of software may remain unidentified during development of increments. To overcome this problem, integration testing for security has to be performed in detail.

The above discussion only presents one issue that needs to be tackled when incremental development methodology is followed. Each development methodology has its own pros and cons e.g. Agile methodology works on the principle of minimal documentation with very short development cycles, resulting in minimal input to security requirements and design phases. But on the other hand agile offers high probability of finding security bugs (as compared to waterfall) before full scale commercial use (due to early client response), and also offers cheap bug fixing due to short development cycles. I am not going into the discussion of these issues since this is not in the scope of this thesis. The purpose of this thesis was to explore how security can be incorporated in software. We have learnt that to incorporate security in software we have to build security into the software by following security activities mentioned in chapter 3. Exploring what issues are faced while performing security activities in different development methodologies and how these issues could be tackled would be a separate study. But one point which is important to note here is that software development and security activities may vary in the way they are carried out in different development methodologies but overall the activities remains the same.

5.1 Research Questions Revisited

At the start of this study few research questions were created in order to guide the work through out the study. These research questions are now revisited to see how they can be answered based on the content of this study.

Q1. How do organizations usually perceive software security?

From literature survey I have found that organizations are usually aware of the importance of software security and they are mostly interested in having secure software. They think of security as something that can be incorporated into the software after development. They consider security as a post development activity. In many organizations, security is considered as duty of network administrators. Organizations spent lot of money in purchasing best firewalls and antivirus and think that this outer shield is enough for making software secure.

Q2. Are there problems in the way organizations perceive software security? If yes, then what are those problems and how organizations can overcome those problems?

Yes, there are problems in the way many organizations perceive software security. From literature study I have found that organizations usually consider security as post development activity. The biggest problem is that, organizations do not understand the term software security. Organizations do not realize that "*Software security is an emergent property of a complete system, not a feature* [1]". They do not realize the fact real threats come from inside the software. Securing software through external shield is like curing the disease not the cause.

To overcome security problems organizations should understand the term 'software security' and realize that security is not something that can be added into the software after its development. Adding security by means of firewalls and antivirus programs is not enough. Organizations should understand that if they want to have

secure software then they need to develop secure software. To achieve this they need to learn, how they can develop secure software.

Q3. What should organizations do in order to develop secure software?

To develop secure software, the first thing organizations should do is to understand software security (Chapter 2 explains software security in detail). After understanding software security, the organizations need to do efforts towards building secure software. The first step towards building secure software is to educate the employees and achieve the trust and support of higher management. Without the trust and support of higher management it is highly unlikely that organizations will achieve the goal of building secure software. After educating employees, and gaining trust and support from higher management, organizations should start software development cycle with one additional goal in mind i.e. to minimize security flaws at each phase. This additional goal can be achieved by performing security activities in chapter 3.

Extra effort and time is required to perform security activities identified in chapter 3, but it does not demand any new setup or a completely new way of working. These activities are easily adjustable in different development methodologies (as briefly discussed in the start of discussion section). What organizations need to do is to analyze their current way of working and find out how the security activities can best fit into their current working setup. Incorporation of security activities in the development process of organizations will result in development of secure software.

5.2 Contribution

Many researchers are working on software security and have been coming up with ideas for developing secure software. Most of the research is scattered and to the best of my knowledge there is no existing comprehensive knowledge base that explains what is needed to build secure software in relation with the software development activities of the software development lifecycle. Based on literature study, this thesis identifies the security activities proposed by different researchers. The thesis adds what is needed more and then explains how these activities are related with the software development activities. In the knowledge domain this thesis adds single source for, understanding the term software security, its need, activities that need to be performed to build secure software, and how these activities are related with the software development activities of the software development lifecycle

6 CONCLUSION

This thesis presents security activities, in relation with the software development activities in the software development lifecycle with the purpose of developing secure software.

This thesis was motivated by the lack of comprehensive resources in the literature for the identification of security activities. The thesis shows how the security activities are related with software development activities in the software development lifecycle.

From literature survey it was found that the security problem in the software is growing. Due to increase in complexity, connectivity and extensibility, the software is becoming more vulnerable to attack. Organizations are aware of the importance of security but instead of dealing with the root cause, they make use of the firewalls and the antivirus software to secure software. In organizations, there is a need of both understanding 'software security' and the efforts for implementing software security.

To develop secure software, security must be given consideration in all the phases of the software development lifecycle. All of the software development phases should include one common goal, which is, to minimize security flaws. This common goal is achieved by performing security activities that identify and remove security flaws at all the stages of software development. All organizations can perform security activities, but for that they first need to analyze their development methodology and then tailor the security activities and the software development activities in such a way that the security activities become part of the development process.

7 FUTURE WORK

Based on the thesis I recommend following steps that should be performed in the future. These steps will add value to the thesis and help the thesis in gaining acceptance in the industry.

7.1 Perform Empirical Study

At the moment the thesis presents security activities without any supporting experiment conducted in the industry. Therefore the directions for future work primarily include an experiment with two or more organizations with different development methodologies performing the security activities presented in this thesis. This will serve as good basis for evaluating the effectiveness of the identified security activities.

7.2 Analyze Security Activities with Different Development Methodologies

The thesis presents security activities and shows how they are related with the software development activities. There is a need to analyze in detail what impact different development methodologies can have on these security activities. And what adjustments are required in security activities for each development methodology.

8 REFERENCES

[1] G. McGraw, "*Managing Software Security Risks*", IEEE Security & Privacy, Volume 2, Issue 2, Mar-Apr 2004, Page(s): 80-83.

[2] Crook R, Ince D, Lin L, and Nuseibeh B, "*Security Requirements Engineering: When Anti Requirements Hit the Fan*", Requirements Engineering, IEEE, 2002, Pages 203-205.

[3] G. McGraw, "*A Portal for Software Security*", Security & Privacy Magazine, IEEE, volume 3, issue 4, Jul-Aug, 2005, Page(s): 75-79

[4] G. McGraw, "*Software Security*" IEEE Security & Privacy, volume 2, issue 2, Mar-Apr, 2004, Page(s): 80-83

[5] Article at "*https://buildsecurityin.us-cert.gov/daisy/bsi/547.html*", Building Security In, Operated by Software Engineering Institute (SEI).

[6] G. McGraw, "*From the Groud Up: The DIMCAS Software Security Workshop*", Security & Privacy Magazine, IEEE, volume 1, issue 2, Mar-Apr, 2003, Page(s): 59-66.

[7] Glossary, "http://www.sei.cmu.edu/str/indexes/glossary/", Software Engineering Institute (SEI

[8] John D. McGregor, Clemson University and Luminary Software LLC, U.S.A, "*Secure Software*", Journal of Object Technology, published by ETH Zurich. Vol 4, No 4, May-June 2005

[9] Bengt Carlsson, School of Engineering, Blekinge Institute of Technology, Dejan Baca, Ericsson AB, "*Software Security Analysis, Execution Phase Audit*",

[10] James C.Foster, "Buffer *Overflow Attacks: Detect, Exploit, Prevent*", Chapter 1, 1st Edition, Syngress publishing.

[11] Crispin Cowan, Perry Wagle, Calton pu, Steve Beattie, Jonathan Walpole, "*Buffer Overflows: Attacks and Defense for the Vulnerability of the Decade*"

[12] Liang T.Chen, "*The Challenge of Race Conditions in Parallel Programming*", Sun Microsystems, July 21, 2006

[13] Robert H. B. Netzer, Barton P. Miller, "*What are Race Conditions? Some issues and Formalizations*", University of Wisconsin – Madison, IEEE explore.

[14] Hoglund, G. and G. McGraw (2004). "*Exploiting Software*", Boston, Addison-Wesley, pages 44.

[15] G. McGraw, "*Adopting an Enterprise Software Security Framework*", Security & Privacy Magazine, IEEE, volume 4, issue 2, Mar-Apr, 2006, Page(s): 84-87

[16] A.S Sodiya, S.A.Onashoga, and O.B.Ajayi, University of Agriculture, Abeokuta, Nigeria. "*Towards Building Secure Software Systems*". Issues in Information Science and Information Technology, Volume 3, 2006.

[17] "*Threat Model Your Security Risks*". Article at:
http://msdn.microsoft.com/msdnmag/issues/03/11/resourcefile/default.aspx

[18] Guttom Sindre, Andreas L. Opdahl, "*Eliciting Security Requirements by Misuse Cases*", IEEE Explore

[19] Gunar Peterson, "*Collboration in Secure Development Process, Part 1*", Information Security Bullettin, June 2004, Volume 9, Page 165.

[20] J. Rumbaugh, "*Getting Started: Using use cases to capture requirements*", Journal of Object Oriented Programming, September 1994, pp 8-23.

[21] Steve Lipner, Michael Howard, "*The Trustworthy Computing Security Development Lifecycle*", Microsoft Corporation. March 2005.

[22] Gunar Peterson, "*Collboration in Secure Development Process, Part 2*", Information Security Bullettin, June 2004, Volume 9, Page 210.

[23] Len Bass, Paul Clements, and Rick Kazman, "*Software Architecture in Practice"*, Second Edition. Addison Wesley 2003

[24] Peter Eeles, Senior IT Architect, IBM, "*What is a software architecture*", 15th Feb 2006

[25] Michael Howard and David LeBlanc, "*Writing Secure Code*", Second Edition, Microsoft Press, 2002.

[26] Michael Howard, "*A Look Inside the Security Development Lifecycle at Microsoft*", MSDN Magazine, November 2005.

[27] Michael Howard, "*A Process of Performing Security Code Reviews*", Security & Privacy Magazine, IEEE, volume 4, issue 4, Jul-Aug, 2006, Page(s): 74-79

[28] Kenneth R, "*Bridging the Gap between Software Development and Information Security*", Security & Privacy Magazine, IEEE, volume 3, issue 5, Sep-Oct, 2005, Page(s): 75-79

[29] Elfriede Dustin, "*The Secure Software Development Lifecycle*", Dev Source (sponsored by Microsoft), November 11, 2006

[30] J. Whittaker, "*Why Secure Applications Are Difficult to Write*", IEEE Security & Privacy Magazine, volume 1, issue 2, 2003, Page(s) 81-83

[31] Bruce Potter, "*Software Security Testing*", IEEE Security & Privacy Magazine, volume 2, issue 5, Sep-Oct, 2004, Page(s) 81-85

[32] H.H. Thompson, "*Why Security Testing is Hard*", IEEE Security & Privacy, vol. 1, no. 4, 2003, pp. 83–86.

[33] Kjetil Molokken-Ostvold and Magne Jorgensen, "*A Comparison of Software Project Overruns – Flexible versus Sequential Development Models*", IEEE Transactions on Software Engineering, volume 31, number 9, September 2005

[34] Michael Howard, "*Mitigate Security Risks by Minimizing the Code You Expose to Untrusted Users"*. MSDN Magazine, November 2004