Secure Software Development Model: A Guide for Secure Software Life Cycle

Malik Imran Daud

***Abstract*---Extreme programming (XP) is a modern approach for iterative development of software in which you never wait for the complete requirements and start development. Security is usually unnoticed during early phases of software life cycle. In this paper, our main objective is to focus on security requirements at each phase of software life cycle. In this regard, XP is a key solution that provides us with a guide with the ease to recheck our security requirements, if they are unnoticed at any step of software life cycle. Based on XP technique, a new model has been designed that focuses on the concept of iterative development of secure software. In addition, this paper is a guide for developers to develop secure software as most of the software developers are not trained for software security.**

***Index Terms* ---- Software Security, Software Life cycle, Extreme Programming (XP)**

I. INTRODUCTION

Software security is to engineer software in such a way that the required application functions uninterrupted and is able to nicely handle the security threats during malicious attacks. Security ensures that application works in a desired manner and to provide defense against security threats. In common practice, security is unnoticed in early phases of software life cycle (SLC). A good software engineering approach is to think about security right from beginning of SLC. Inadequate practice of software development can lead to insecure software [1]. According to [2], "software assurance is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner."

Extreme programming (XP) is an organized approach for developing software in an iterative manner. XP is considered best practice to improve the software quality by repeated feedback and changing requirements [3].

Manuscript received October 21, 2009. This work was supported in part by Foundation University Institute of Engineering and Management Sciences.

Malik Imran Daud is working as faculty member at Department of Telecommunication Engineering, Foundation University Institute of Engineering and Management Sciences (FUIEMS) Lalazar Rawalpindi, Pakistan (email: imrandaud@gmail.com).

Security engineers never wait for the perfect requirements, only initial requirements are gathered and developers start development.
During development system is presented to security analyst and security engineers for the recommendation and up gradation of the security requirements.

This research mainly focuses on the secure life cycle of software that requires a lot of thorough consideration. That includes security in *Requirements/Analysis, Design, Implementation and testing* phase. At each phase of SLC, security requirements are gathered and updated iteratively. Main focus of this technique is to monitor security requirements and identify security threats at each phase.

Section II describes an overview of software vulnerabilities whereas section III describes a new life cycle model and description of each life cycle model is explained in section IV.

II. SECURE SOFTWARE DEVELOPMENT

Most of the organizations process their confidential information using software systems via internet. A small bug in software can be exploited by hackers and confidential information can be stolen. Besides other problems of software development, security is becoming a major issue. According to CERT statistics [4] there has been considerable increase in vulnerabilities reported over the last few years, which are depicted in figure 1.

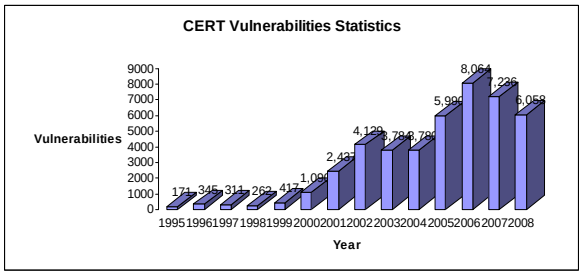

FIGURE 1: VULNERABILITIES STATISTICS

According to statistics shown in Figure 1, security has been taken as a serious challenge over the last two years. Accordingly new techniques and methods have been developed to cure software issues. This resulted in more secure software and vulnerabilities reported over the last

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

two years are comparatively less. Considering these statistics there is a need to develop such approach for software development that could guaranty security at each phase of software life cycle. [5] lists common software vulnerabilities, these vulnerabilities are mainly design and coding vulnerabilities which are unnoticed by the software engineers. For any secure software there are three main core properties which are *Confidentiality, Integrity and availability* that is our guideline for designing new approach.

III. ITERATIVE METHOD OF SOFTWARE LIFE CYCLE

Security itself is a complete life cycle of software development. Where as iterative method is considered more efficient and reliable approach for software development. You have few set of requirements and start development and iteratively new requirements are fulfilled. Blend of security and XP gives a new approach that is shown in figure 3. Figure 3 shows an iterative model of secure software life cycle (SSLC) based on extreme programming concept.

Requirements engineering is the main building block for any software development. Security engineers try to elicit security requirements by different methods, e.g. user stories, abuse cases, etc. Figure 4 lists all the main operation to be performed during SSLC. Based on the information provided by figure 4 we can derive following main sources to derive security requirements these are:

- *Functional Security Requirements*
- *Non functional Security Requirements*
- *Derived Security Requirements*
- *User stories*
- *Abuse cases*

During analysis phase we get security requirements from above listed sources. Most of the occasion requirements gathered from user stories and other sources are not well defined. These requirements can be refined by security functional requirements (SFR) module (Details are given in section *'IV-A'*).

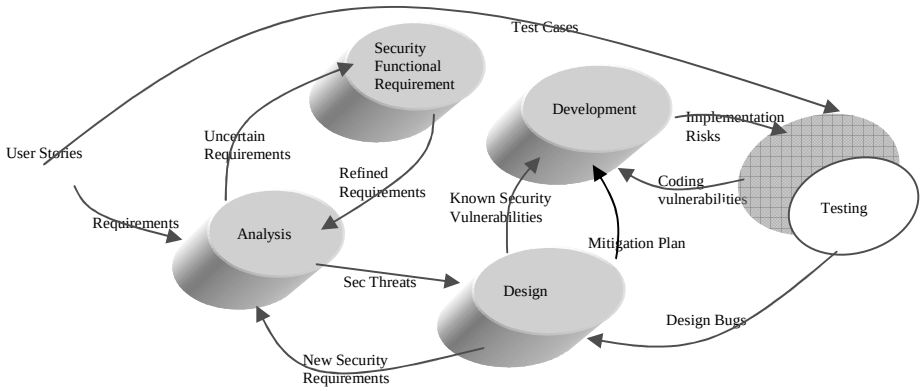

FIGURE 3: ITERATIVE LIFE CYCLE FOR SECURE SOFTWARE

Once the uncertain requirements are refined by SFR module, then we are ready to start designing our software. Design phase is important and requires more consideration in terms of security. Based on the information provided by analysis phase (Security Requirements by user stories and SFR) a threat model is developed. If security engineer feels some of the information is missing or some other security threats are possible then it goes back to analysis for the refinement of the security requirements. If security expert finds no problems, then a mitigation plan is designed to cater all those threats listed in threat model. Security vulnerabilities are identified during design phase and Table 2 gives a guideline to find such vulnerabilities. All the vulnerabilities that a software system may suffer from are documented and passed to development team. Developers start development by considering all vulnerabilities and their mitigation plan designed during design phase.

Once software is developed then it is handed over to testing team along with the documentation. During this phase different testing methodologies are used as discussed in section 'IV-C'. In this phase engineers try to find design or development bugs in software application. After that software application is ready for the deployment.

IV. SECURE SOFTWARE LIFE CYCLE – A MODEL

Software is vulnerable to attack, when some security lapses are overlooked during software life cycle. Software security unnoticed during early phases of life cycle is inherited to later phases; therefore one phase transfers its

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

vulnerabilities to the other phase. According to [6] software is vulnerable to threats that may occur during development of software. Security engineers may disrupt the software during software development life cycle either by intentional or unintentional modification of the *requirement's specification, design document, source code or test cases*. One should have a list of all major actions to be performed during life cycle of software. Therefore, to ensure software security following model has been designed that list all the actions to be performed during the life cycle of software.

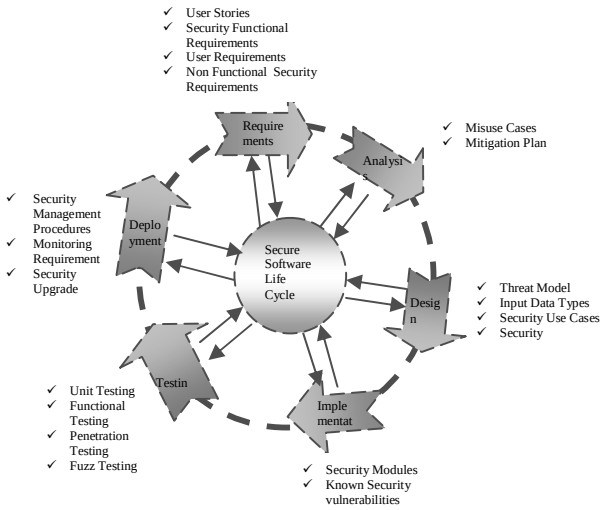

FIGURE 4: SECURE LIFE CYCLE OF SOFTWARE

Figure 4 depicts all the actions to be taken during the life cycle of software to guarantee security in software. This life cycle is iterative in nature; if some security modules are left at any phase of life cycle, then we can go back to that phase and fulfill those shortcomings. For example, if we are designing a cryptographic software that encipher and decipher the text. Suppose we are in design phase and left with few cryptographic requirements during requirement phase. We can go back to requirement phase and update those cryptographic requirements and can continue our design from same point from where we left. Each phase is discussed in following sections.

A. SECURITY ANALYSIS / REQUIREMENTS

For any major project, requirements engineering has always been critical for its success. Security requirements may fall into three main categories [7] these are*: i) Functional (Behavioral) security Requirements. ii) Non Functional security Requirements. iii) Derived security Requirements.* Functional requirements list all the functions that a system will perform. These requirements relate to input and output of a system and the relationship betweens input and output. These requirements also specify the actions to be performed for a specific input. Whereas non functional requirements list all the properties a system will possess like its environment where it will run like UNIX, Windows, etc. Derived requirements are those requirements which are derived from functional and other security requirements.

As far as software security requirements are concerned it comes in two different ways. One directly from user stories that can be user requirements and other security requirements are derived by the security engineers. User stories are an effective way to derive user requirements in efficient way from rapid changing real world requirements. Security engineers derive rest of the user requirements and these requirements are the security functional requirements. Common Criteria functional requirements [8] are the best source to derive such functional requirements. This is helpful for the consumer and developer both to identify security objective and security requirements.

It is important to anticipate abnormal behavior for secure and reliable software application. Therefore, security experts need to create use cases to mitigate those abnormal behaviors, i.e. misuse case. These are the cases, in which all those actions or processes of system that can be exploited by a misuser. Figure 5 show a relationship between use case and misuse case.

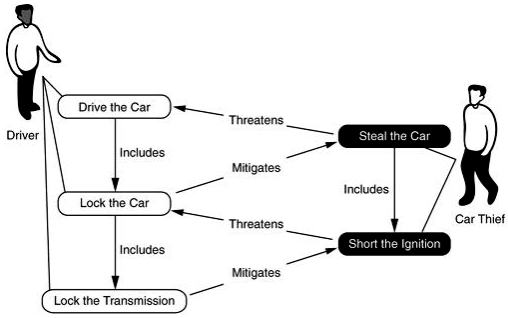

FIGURE 5: USE CASE AND MISUSE CASES [9]

SECURITY FUNCTIONAL REQUIREMENTS

Before defining security requirements, security engineers need to identify those parts of the software system that requires security. These parts of the software system are called *Target of Evaluation* (TOE). Once TOE is identified then finding security functional requirements (SFR) for those parts becomes simple. [8] lists different set of classes depending on the nature of application. Different set of SFRs can be chosen for the required TOE. Once required SFRs are chosen, then table can be designed to monitor its implementation in required software application. SFRs are chosen to counter threats in TOE of software system. For example; if we are trying to gather SFR of a web application; Table 1 lists related SFR's and their activity. There can be different TOE in a single software application; therefore different set of SFRs are collected for each TOE.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

TABLE 1: SFR ACTIVITY MODEL

Class	SFR	Description	SFR Levels	TOE		
				Client Server Communication	Digital Certificate	Authentication
FCO : Communication	FCO_NRO	Non Repudiation of Origin	FCO_NRO.1	✓	✓	
			FCO_NRO.2	✓	✓	
	FCO_NRR	Non Repudiation of Receipt	FCO_NRR.1	✓	✓	
			FCO_NRR.2	✓	✓	
FCS : Cryptographic Support	FCS_CKM	Cryptographic Key management	FCS_CKM.1		✓	✓
			FCS_CKM.2		✓	✓
			FCS_CKM.3		✓	✓
			FCS_CKM.4		✓	✓
	FCS_COP	Cryptographic Operation	FCS_COP.1	✓	✓	✓

B. SECURITY DESIGN & IMPLEMENTATION

Design phase shapes all the requirements into reality. This is a phase where, what and why of requirements become who, when, where, and how of the software to be [9]. Design phase plays very important role where you give design to security requirements. As listed in figure 4, it is significant to design a threat model for secure software application. Threat modeling is a technique to identify threats, vulnerabilities and their countermeasures. Once we have the security requirements and we have the data flow diagrams (DFDs), now there is need to identify the entry points and exit points to the system from DFDs. These are the points from where attacker can enter into the system. Once we have identified entry and exit points now identify all possible threats that an attacker can exploit from these points. Table 2 can be good source to find threats for particular application. Let's take an example of confidential data that needs to be stored. Security Engineer needs to identify all possible attacks by asking questions like: where to store this data, how to transfer data remotely, how attacker will manipulate data. These are the threats possible on sensitive data and their countermeasure are can be devised accordingly. Once we have identified possible attacks on software system then attack trees can be plotted to clear understanding of attacker's methodology. Figure 6 shows an example attack tree of attacks possible on confidential data.

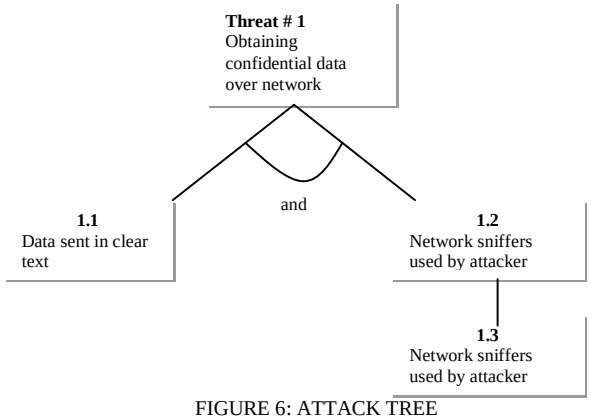

FIGURE 6: ATTACK TREE

Vulnerabilities analysis is also important part of threat modeling. Table 2 shows some common areas where vulnerabilities may occur. These vulnerabilities may occur at any Phase of software life cycle, but it is important to identify these vulnerabilities at design phase.

TABLE 2: COMMON VULNERABILITY AREAS

Vulnerability Area	Vulnerability Types	
Operating system(OS)	Buffer overflow(Stack, Heap), Null pointers, OS Resources deadlock, Exceptions etc	
Communication	Non repudiation of origin, Non repudiation of receipt etc	
Database/User Data	Invalid Data types, SQL injection, Cross Site Scripting, Rollback, Data integrity etc	
Cryptography	Key Management, Cryptographic operation, etc	
Access Control	Authentication	Access control policy, data authentication, information flow control policy etc
	Authorization	
Privacy	Privileges, Anonymity, pseudo anonymity etc	
Programming	Exception etc	

Vulnerability areas shown in table 2 can be taken as security use cases as well and their countermeasures are figured out. Once we have identified all the attacks and vulnerabilities now system is ready for implementation phase.

Developing robust and vulnerability free software is a challenging job. During implementation we have known security vulnerabilities and their countermeasures. [10] lists vulnerabilities and their countermeasures that can be taken into consideration while developing software.

C. SECURITY TESTING AND DEPLOYMENT

Security testing is vital and plays important role in identifying security flaws before the release of application. Security tester needs to think like an attacker and try to launch different attack to find bugs in software system. In order to check that software has met its security requirements we have two main types these are: *1) Functional Testing 2) Risk Based Testing* [11]. *Functional testing* deals with to test software application with functional requirements. Functional requirements define functional behavior of the software for a specific state, e.g.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

"if this condition occurs, then system should respond in that way". Functional testing may address all the threats and vulnerabilities identified in Table 2. *Risk based testing* deals with all states or behaviors that a system must not do. During software testing test plans are created for specific components of software that require security. Once we have all the information about security threats, vulnerabilities and their countermeasures then security tests are conducted. Testing techniques that may be followed can be 1) Penetration Testing. 2) Fuzz Testing. Penetration testing is performed to find vulnerabilities in software application. We have different types of penetration testing that include *Targeted Testing, External Testing, Internal Testing, Blind Testing and Double Blind Testing* as shown in table 3. Whereas in Fuzz testing a special tool known as Fuzz tester that is used to find vulnerabilities in software application.

TABLE 3: PENETRATION TESTING TYPES

Targeted testing	Testing conducted by IT testing team and penetration testing team.
External testing	Testing conducted on external servers and firewalls
Internal testing	Testing to check internal threats by authorized user.
Blind testing	All actions and procedures are examined that a real attacker can perform.
Double blind testing	Blind tests performed by few test engineers rest do not know about these types of tests.

During testing phase, if some of the security bugs are identified, then these bugs are reported to the concerned life cycle phase iteratively as shown in figure 3. After that software system is ready for deployment. Once deployment is complete then system is monitored for specific time for any bug. Security features are upgraded with the passage of time with security upgrades.

V. CONCLUSION & FUTURE WORK

Different software engineering approaches are followed for the design and development of software that includes the spiral model, waterfall model, agile methods and iterative approaches. These are efficient software engineering approaches, but security is neglected part and requires special consideration. Therefore, all these approaches needs security blend to make secure software engineering.

This paper is a comprehensive manual of software life cycle that explains a secure approach for software development. This paper can be a good guide for any security engineer. Secure model explained in this paper is iterative model based on extreme programming concept. Each phase of the software life cycle is explained as a step-by-step guide.

Model explained in section III can be further extended, whereas all sub activities at each phase can further be modeled. Like all the testing techniques can be ordered and their relationship can be modeled. Furthermore, this security model can be synchronized with the software engineering model and resulting model will be secure software engineering model for software development.

ACKNOWLEDGEMENT

I would like to thank Allah (SWT) almighty for the strength to conduct this research, my parents and all of the faculty members for their guidance and help while conducting this research here at Foundation University Islamabad (FUIEMS).

REFERENCES

[1]. C. Mann, "Why Software is so Bad" Technology Review (July/August 2002)
[2]. "National Information Assurance Glossary"; CNSS Instruction No. 4009 National Information Assurance Glossary
[3]. Don wells, Copyright © 1999, 2000, 2001[Modified: February 17, 2006], Extreme Programming, www.extremeprogramming.org
[4]. Carnegie Mellon University, Copyright © 1995-2009 [Modified: February 12, 2009], CERT, http://www.cert.org/stats/
[5]. M.A. Hadavi, H. M. Sangchi, V. S. Hamishagi, H. Shirazi, "Software Security; A Vulnerability-Activity Revisit" ARES Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, Pg 866-872, ISBN:978-0-7695-3102-1
[6]. Cappelli, Dawn, Trzeciak, Randall, & Moore, Andrew. "Insider Threats in the SDLC." Presentation at SEPG 2006. Carnegie Mellon University, Software Engineering Institute, 2006.
[7]. Paco Hope and Peter White, Cigital, Inc., Copyright © 2007, Software Security Requirements – the foundation for security, Cigital Inc, http://www.cigital.com
[8]. Common Criteria, Common Criteria: Part 2 Security Functional Components, Version 3.1, revision 2, September 2007.
[9]. Julia H. Allen; Sean Barnum; Robert J. Ellison; Gary McGraw; Nancy R. Mead, Addison Wesley Professional, Software Security Engineering- A guide for project managers, Pg 82, ISBN 978-0-321-50917-8.
[10]. D. Gilliam, T. Wolfe, J. Sherif, and M. Bishop, "Software Security Checklist for the Software Life Cycle," Proceedings of the 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprise pp. 243–248 (June 2003).
[11]. McGraw, Gary. Software Security: Building Security In. Boston, MA: Addison-Wesley, 2006.